

电子科技大学

实验报告

学生姓名：谢卿云 学号：2022010910017 指导教师：沈复民

一、实验项目名称：

Image Filtering and Hybrid Images

二、实验原理：

高低通滤波是数字图像处理领域的基础，它可以混合图像的生成。其核心原理在于理解图像的频率特性和卷积操作。

(一) 图像频率

图像可以被看作是二维信号，包含不同频率的信息。低频成分对应于图像中变化缓慢的区域，如平滑的背景和大的结构轮廓。高频成分对应于图像中变化快速的区域，如图像的边缘、纹理和细节。通过分离和处理这些不同频率的成分，我们可以实现图像的模糊、锐化或特征提取等操作。

(二) 图像滤波与卷积

图像滤波是通过卷积操作实现的。卷积是将一个小的矩阵（称为滤波器或卷积核）与图像的局部区域进行加权求和的过程，如图 1 所示

具体来说，滤波器在图像上滑动，在每个位置，将滤波器矩阵的元素与其覆盖的图像像素值相乘，并将结果相加，得到输出图像在该位置的像素值。

$$\begin{array}{|c|c|c|c|c|} \hline 1 & 1 & 1 & 1 & 1 \\ \hline -1 & 0 & -3 & 0 & 1 \\ \hline 2 & 1 & 1 & -1 & 0 \\ \hline 0 & -1 & 1 & 2 & 1 \\ \hline 1 & 2 & 1 & 1 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 0 & -2 & -1 \\ \hline 2 & 2 & 4 \\ \hline -1 & 0 & 0 \\ \hline \end{array}$$

图 1：卷积运算示意图

(三) 滤波器

低通滤波通过卷积一个低通滤波器（如高斯滤波器）实现，它可以平滑图像，去除高频噪声和细节，保留图像的整体结构和颜色。高通滤波的一种常用实现方式是：从原始图像中减去其低通滤波后的结果。低通滤波保留了图像的低频信息，去除高频信息。当用原始图像减去低频图像时，剩余的部分就是原始图像中的高频信息。这允许我们提取或增强图像的边缘和细节。

(四) 混合图像的生成

混合图像（Hybrid Image）是结合了两张不同图像的频率成分而创建的。其原理基于人类视觉系统的特性：在近距离观察时，人眼更容易感知图像的高频细节；在远距离观察时，则更容易感知图像的低频轮廓。通过将一张图像的低频成分与另一张图像的高频成分叠加合成，我们可以创建出一种在不同观察距离下呈现不同内容的图像。具体实现步骤如下：

1. 对第一张图像进行低通滤波，提取其低频成分。
2. 对第二张图像进行高通滤波，提取其高频成分（通过原始图像减去低通滤波结果）。
3. 将第一张图像的低频成分与第二张图像的高频成分相加，生成混合图像。
4. 对生成的混合图像像素值进行裁剪，确保其在有效的显示范围内。

通过调整低通滤波器（高斯滤波器）的截止频率，我们可以控制保留多少低频信息和高频信息，从而影响混合图像在不同距离下的视觉效果。

三、实验目的：

1. 理解图像中不同频率成分（高频和低频）的含义及其在图像中的表现。
2. 掌握图像滤波（卷积）的基本原理。
3. 学习如何利用滤波实现图像的高通滤波和低通滤波。
4. 理解混合图像（Hybrid Image）的生成原理，并实现将高频和低频图像融合创建混合图像的过程。
5. 通过对不同滤波器、截止频率等参数设置下图像滤波及混合的效果，直观感受频率域图像处理的应用。

四、实验内容：

1. 基于 skimage 中所包含的常见滤波器，对自选的图像进行滤波，直观地对比不同滤波器的效果
2. 实现高通滤波和低通滤波两种滤波形式，对图像进行高低通滤波，并且对图像分别在高低频进行融合。

五、实验步骤：

1. 实现 **my_imfilter** 函数：实现图像滤波的核心函数 **my_imfilter**。该函数接收一个图像和一个滤波器作为输入，并返回滤波后的图像。实现时需考虑以下要点：
2. 实现 **gen_hybrid_image** 函数：实现生成混合图像的函数 **gen_hybrid_image**。该函数接收两张输入图像和一个截止频率作为输入，并返回低频图像、高频图像和混合图像。
3. 进入 proj1.ipynb 笔记本，设置 **cutoff_frequency** 值为 7，选择自行车观察滤波情况，然后选择鱼和鱼雷图片生成低频、高频和混合图像。

六、实验数据及结果分析：

(一) 观察滤波器

我们选取自行车图像进行 skimage 常见滤波器的观察，结果如图 2 所示，从上至下，从左至右依次是 blur,sobel,laplacian,high_pass 滤波器的结果。

(二) 融合图像

我们选择鱼和鱼雷图像作为融合函数的测试图像，分别通过高通和低通滤波器，结果如图 3 所示，它们的融合结果结果如图 4 所示

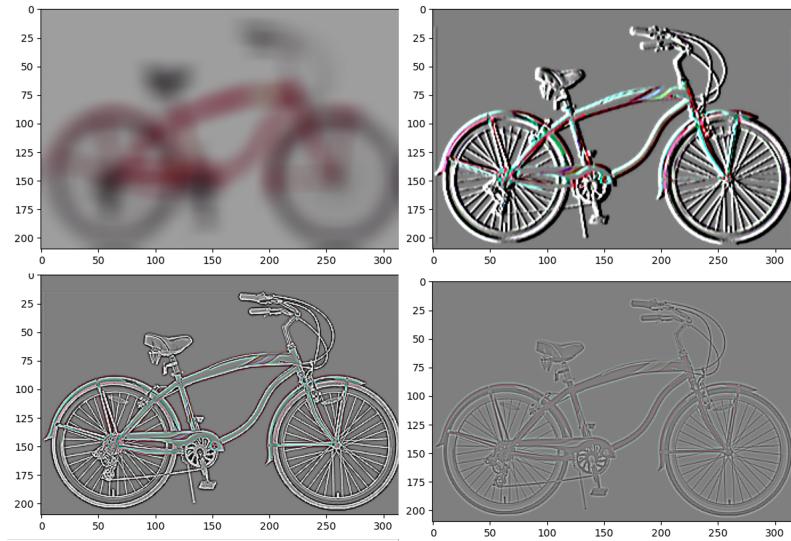


图 2: 常见滤波器处理自行车图片的结果

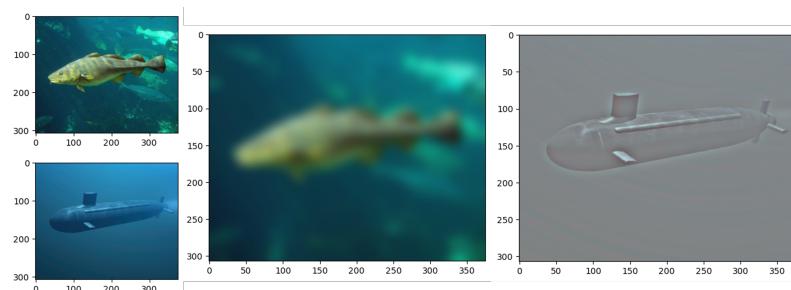


图 3: 鱼和鱼雷通过自定义的高低通滤波器的结果

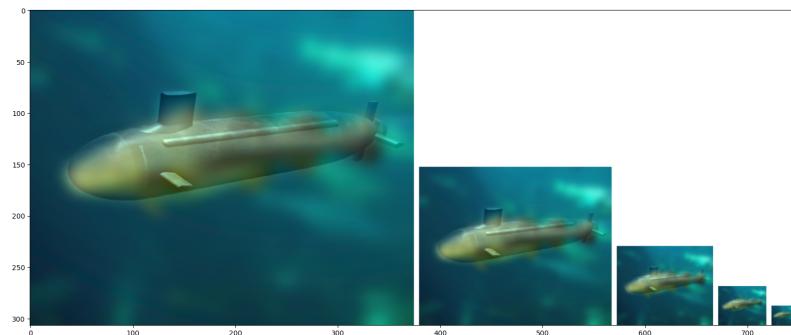


图 4: 鱼和鱼雷融合图像

七、实验结论：

从图 2 可以看到自行车的轮廓以不同方式显现出来，符合理论上我们对滤波器的认识；

从图 3 可以看到，当图片放大时，相对我们在近距离观察时，可以很明显地分辨它是鱼雷，但是当图片缩小时，相对我们在远距离观察时，可能会认为这是黄色的鱼，这样的结果符合我们对高低通滤波器效果的认知。这

以上结果说明我们的实验比较成功。

八、总结及心得体会：

首先，我在图像滤波、卷积原理及图像频率分析方面加深了理解。通过亲自动手实现 `my_imfilter` 函数，我不仅掌握了卷积操作如何通过滤波器提取图像特征的原理，还更加清楚地认识到图像中的高频和低频信息分别代表什么，以及它们在图像感知中的作用。

其次，我在编程实践和问题解决的能力上有所提高。实验过程中，实现卷积操作需要仔细处理索引、边界填充（反射填充）以及多通道图像的处理，这些细节的实现锻炼了我严谨的编程习惯。遇到的潜在问题，例如数值溢出或边界处理不当，通过调试和查阅资料（如 `numpy.pad` 的使用），我成功克服了这些困难，这显著提升了我分析和解决图像处理实际问题的能力。

此外，我还学习到了参数调优和结果分析的重要性。在生成混合图像时，不同 `cutoff_frequency` 的选择直接影响了高低频成分的分离效果和最终混合图像在不同距离下的视觉呈现。

总的来说，这次实验不仅巩固了我对图像处理基本理论知识的掌握，特别是频率域处理的概念，更重要的是，通过从零开始实现核心算法，极大地提升了我的实践操作能力、代码调试能力以及对复杂图像处理流程的端到端理解。同时，通过对不同参数效果的分析，也培养了我科学实验中必不可少的分析和总结能力。

九、对本实验过程及方法的改进建议：

基于本次实验的实现和体验，以下是一些对实验方法或代码的改进建议：

1. 优化 `my_imfilter` 函数的效率：当前 `my_imfilter` 函数采用的是直接的空间域卷积实现，对于较大的图像或滤波器，计算开销较大。可以考虑引入更高效的卷积方法，比如在频域上使用快速傅里叶变换；

2. 增强 `gen_hybrid_image` 函数的灵活性：目前的 `gen_hybrid_image` 函数硬编码使用了高斯滤波器进行高低通滤波。可以改进为允许用户选择或传入不同类型的滤波器，以探索不同滤波器对混合图像效果的影响。
3. 边界处理方式的多样化：`my_imfilter` 中只实现了反射填充。可以增加对其他边界填充方式的支持，如零填充（zero padding）、边缘复制填充（replicate padding）等，并在实验中比较不同填充方式对滤波结果，特别是图像边缘的影响。

报告评分：

指导教师签字：

附录一 核心代码

如代码 1 所示。

代码 1: student.py

```
1 # Project Image Filtering and Hybrid Images Stencil Code
2 # Based on previous and current work
3 # by James Hays for CSCI 1430 @ Brown and
4 # CS 4495/6476 @ Georgia Tech
5 import numpy as np
6 from numpy import pi, exp, sqrt
7 from skimage import io, img_as_ubyte, img_as_float32
8 from skimage.transform import rescale
9 import math
10
11 def my_imfilter(image, filter):
12     """
13         Your function should meet the requirements laid out on the project
14             webpage.
15         Apply a filter to an image. Return the filtered image.
16         Inputs:
17         - image -> numpy nd-array of dim (m, n, c)
18         - filter -> numpy nd-array of odd dim (k, l)
19         Returns
20         - filtered_image -> numpy nd-array of dim (m, n, c)
21         Errors if:
22         - filter has any even dimension -> raise an Exception with a suitable
23             error message.
24     """
25
26     image_height, image_width = image.shape[:2]
27     filter_height, filter_width = filter.shape
28     num_channels = 1 if image.ndim == 2 else image.shape[2]
29
30     # Calculate padding amounts
31     pad_height = filter_height // 2
32     pad_width = filter_width // 2
33     padded_image = np.pad(image, ((pad_height, pad_height), (pad_width,
34                                   pad_width), (0, 0) if num_channels > 1 else (0, 0)), mode='reflect')
35
36     # filter process: convolution
37     filtered_image = np.zeros_like(image)
38     flipped_filter = np.flip(filter, axis=(0, 1))
39     for c in range(num_channels):
40         for i in range(image_height):
41             for j in range(image_width):
42                 image_patch = padded_image[i:i+filter_height, j:j+
43                                             filter_width, c] if num_channels > 1 else padded_image[i:
44                                             i+filter_height, j:j+filter_width]
45                 filtered_image[i, j, c] = np.sum(image_patch *
46                     flipped_filter) if num_channels > 1 else np.sum(
47                         image_patch * flipped_filter)
48     return filtered_image
49
50
51 def gen_hybrid_image(image1, image2, cutoff_frequency):
52     """
53         Inputs:
54         - image1 -> The image from which to take the low frequencies.
55         - image2 -> The image from which to take the high frequencies.
56         - cutoff_frequency -> The standard deviation, in pixels, of the
57             Gaussian
58                             blur that will remove high frequencies.
59
60         Task:
61         - Use my_imfilter to create 'low_frequencies' and 'high_frequencies'.
62         - Combine them to create 'hybrid_image'.
63     """
64
65
66
67
68
69
70
71
72
73
74
75
```

```

56 """
57
58 assert image1.shape[0] == image2.shape[0]
59 assert image1.shape[1] == image2.shape[1]
60 assert image1.shape[2] == image2.shape[2]
61
62 # Steps:
63 # (1) Remove the high frequencies from image1 by blurring it. The amount
64 # of blur that works best will vary with different image pairs
65 # generate a 1x(2k+1) gaussian kernel with mean=0 and sigma = s, see
66 # https://stackoverflow.com/questions/17190649/how-to-obtain-a-gaussian
67 # -filter-in-python
68 s, k = cutoff_frequency, int(cutoff_frequency*2)
69 probs = np.asarray([exp(-z*z/(2*s*s))/sqrt(2*pi*s*s) for z in range(-k,k
70 +1)], dtype=np.float32)
71 kernel = np.outer(probs, probs)
72
73 # Your code here:
74 # low_frequencies = None # Replace with your implementation
75 low_frequencies = my_imfilter(image1, kernel)
76
77 # (2) Remove the low frequencies from image2. The easiest way to do this
78 # is to
79 # subtract a blurred version of image2 from the original version of
80 # image2.
81 # This will give you an image centered at zero with negative values.
82 # Your code here #
83 # high_frequencies = None # Replace with your implementation
84 image2_low_frequencies = my_imfilter(image2, kernel)
85 high_frequencies = image2 - image2_low_frequencies
86
87 # (3) Combine the high frequencies and low frequencies
88 # Your code here #
89 # hybrid_image = None
90 hybrid_image = low_frequencies + high_frequencies
91
92 # (4) At this point, you need to be aware that values larger than 1.0
93 # or less than 0.0 may cause issues in the functions in Python for
94 # saving
95 # images to disk. These are called in proj1_part2 after the call to
96 # gen_hybrid_image().
97 # One option is to clip (also called clamp) all values below 0.0 to 0.0,
98 # and all values larger than 1.0 to 1.0.
99 hybrid_image = np.clip(hybrid_image, 0, 1)
100
101 return low_frequencies, high_frequencies, hybrid_image
102
103 def vis_hybrid_image(hybrid_image):
104 """
105     Visualize a hybrid image by progressively downsampling the image and
106     concatenating all of the images together.
107 """
108 scales = 5
109 scale_factor = [0.5, 0.5, 1]
110 padding = 5
111 original_height = hybrid_image.shape[0]
112 num_colors = 1 if hybrid_image.ndim == 2 else 3
113
114 output = np.copy(hybrid_image)
115 cur_image = np.copy(hybrid_image)
116 for scale in range(2, scales+1):
117     # add padding
118     output = np.hstack((output, np.ones((original_height, padding,
119                                         num_colors),
120                                         dtype=np.float32)))
121     # downsample image
122     cur_image = rescale(cur_image, scale_factor, mode='reflect')
123     # pad the top to append to the output
124     pad = np.ones((original_height-cur_image.shape[0], cur_image.shape[1],

```

```
118     num_colors), dtype=np.float32)
119     tmp = np.vstack((pad, cur_image))
120     output = np.hstack((output, tmp))
121     return output
122
123 def load_image(path):
124     return img_as_float32(io.imread(path))
125
126 def save_image(path, im):
127     return io.imsave(path, img_as_ubyte(im.copy()))
```