Image Filtering and Hybrid Image Creation Algorithm

Bhattacharya Brothers
Darpan Bhattacharya (B2430044)
Soham Bhattacharya (B2430059)
Ramakrishna Mission Vivekananda Educational and Research Institute

February 10, 2025

1 Introduction

This report outlines the design and implementation of two functions: `my_imfilter` and `create_hybrid_image`. These functions perform image filtering and create hybrid images by combining low-frequency and high-frequency content from two input images. The process involves using a filter to smooth one image (low-pass filtering) and extract the high-frequency content from another image to create a hybrid effect.

2 Underlying Methodologies

2.1 `my_imfilter` Function

2.1.1 Objective

The `my_imfilter` function applies a convolution-based filter to an image. It performs a direct pixel-by-pixel convolution operation, where each pixel value in the output image is computed by applying the filter to the corresponding region of the input image.

2.1.2 Key Decisions

- **Filter Type:** The filter used is assumed to be a square kernel with odd dimensions. This allows for proper centering during convolution, as the middle element of the kernel corresponds to the target pixel.
- **Padding:** To handle border pixels during convolution, the input image is padded with zeros. The padding size is calculated as pad_size $= K//2$ where K is the kernel size.
- **Convolution Process:** A three-level nested loop is used:
  - The outer two loops iterate over the spatial dimensions of the image (height and width).
  - The innermost loop processes each color channel (RGB) independently.
- **Efficiency Considerations:** Though computationally expensive, the naive implementation meets task requirements and is acceptable for small-to-medium-sized images.

2.2 `create_hybrid_image` Function

2.2.1 Objective

The `create_hybrid_image` function combines the low-frequency content of one image and the high-frequency content of another to create a hybrid image.

2.2.2 Key Decisions

- **Input Image Validation:** Ensures both input images have the same dimensions.
- **Low-Pass Filtering:** The low-frequency content is obtained using `my_imfilter`.
- **High-Frequency Extraction:** Extracted by subtracting the filtered version from the original image.
- **Hybrid Image Creation:** Combines low-frequency content from one image and high-frequency content from another.
- **Clipping:** Ensures pixel values remain in the valid range $[0, 1]$.

3 Results Obtained from Testing

The implemented algorithms produced expected output images using the following input image.

Figure 1: Input Image

The following filters were tested:

- **Identity Filter:** `numpy.asarray([[0, 0, 0], [0, 1, 0], [0, 0, 0]])`
  Preserves the original image. The identity filter preserves the original image unchanged. The filter has a value of 1 at the center and 0 elsewhere, meaning that for each pixel, only the pixel value at the center of the filter's region is retained. The surrounding pixels are not considered, thus the image remains unmodified. This filter essentially performs no operation.

Figure 2: Identity Filter

- **Small Blur Filter:** `numpy.ones((3, 3))`
  The small blur filter is a simple 3x3 filter with equal weights (all 1s). This filter averages the pixel values in a 3x3 neighborhood around each pixel, resulting in a blurred version of the image. Since all values are equal, each pixel in the output image is a simple average of the surrounding pixels, leading to a mild smoothing effect.

Figure 3: Small Blur Filter

- **Large 1D Blur Filter (Gaussian Kernel):** `cv2.getGaussianKernel(ksize=25, sigma=10)`
  The large 1D Gaussian blur filter applies a Gaussian function to smooth the image. This filter blurs the image more significantly than the small blur filter because of its larger size (25x1) and higher sigma value (10), which makes the blurring effect more prominent and gradual. The result is a smooth image with reduced high-frequency content, where edges and fine details are softened.

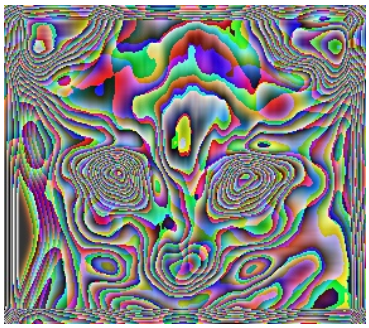

Figure 4: Large 1D Blur Filter

- **Oriented Sobel Filter:** `numpy.asarray([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])`
  The oriented Sobel filter is used for edge detection, specifically to detect horizontal gradients in the image. It highlights changes in intensity from left to right (or vice versa). The filter emphasizes vertical edges by calculating the difference in pixel values between the left and right sides of each pixel. The output image will show horizontal edges (where there are strong changes in intensity from left to right) and suppress other features.

Figure 5: Oriented Sobel Filter

- **High-Pass Laplacian Filter:** `numpy.asarray([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])`
  The oriented Sobel filter is used for edge detection, specifically to detect horizontal gradients in the image. It highlights changes in intensity from left to right (or vice versa). The filter emphasizes

vertical edges by calculating the difference in pixel values between the left and right sides of each pixel. The output image will show horizontal edges (where there are strong changes in intensity from left to right) and suppress other features.

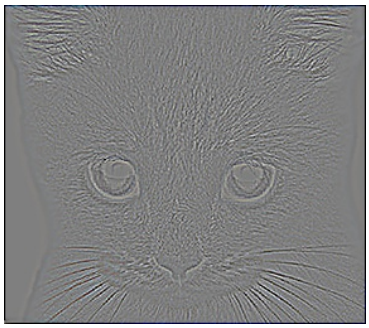

Figure 6: High-Pass Laplacian Filter

4 Hybrid Image Production

A Gaussian low-pass filter with a cutoff frequency of 7 was used to extract low-frequency content. The high-frequency content was obtained by subtracting the low-pass filtered version from the original image. The hybrid image is a combination of these components.
The notebook used for the purpose of generating hybrid images is `proj1.ipynb`.
The following are the images utilised hybrid image generation:

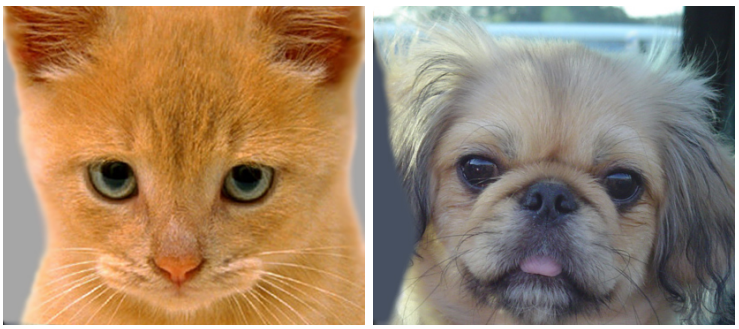

Figure 7: Input images for Hybrid Image generation

The python code generates a Gaussian low-pass filter with a cutoff frequency of 7 by creating a 1D Gaussian kernel using cv2.getGaussianKernel() with a kernel size of cutoff_frequency*4+1 (29 in this case) and a sigma (standard deviation) of 7. This kernel is then converted into a 2D filter by multiplying it with its transpose. The filter is used to extract the low-frequency content from an image by smoothing it, removing sharp edges and fine details. To get the high-frequency content, the low-frequency version of another image is subtracted from the original, highlighting edges and rapid intensity changes. By combining the low frequencies from one image and the high frequencies from another, a hybrid image is created, where the background is dominated by one image and the details are from the other.
Output images received are as follows:

Figure 8: (From left to right) Low-frequency, high-frequency & Generated hybrid image

Figure 9: Generated Hybrid Scaled Image

5 Individual Contributions

- **Soham Bhattacharya** used his understanding of linear algebra to better grasp the workings of kernels and the convolution operation. The process of applying a filter to an image, as well as the use of Gaussian filters for frequency domain manipulation, became clearer through concepts like matrix multiplication and eigenvectors. This deepened the understanding of how mathematical models like convolution and Gaussian smoothing are applied in image processing.
- **Darpan Bhattacharya** focused on optimizing the computational efficiency of the algorithm. He used his knowledge of algorithm design and data structures to reduce the time complexity of the convolution process, ensuring the code could handle larger images and filters without compromising performance. By applying concepts like caching and vectorization, he made the image filtering process faster and more scalable.

Together, our collaboration allowed us to blend mathematical rigor with computational efficiency, improving both our algorithmic and practical skills in the process.

6 Conclusion

The presented algorithm effectively implements image filtering and hybrid image creation. The `my_imfilter` function performs convolution manually, while the `create_hybrid_image` function combines frequency components from two images, ensuring visually interesting results. The approach ensures that the final hybrid image maintains visually interesting features, with smooth low-frequency components and detailed high-frequency components.