Image Processing - Exercise 3

Hadar Tal, hadar.tal, 207992728

1. Introduction

The goal of this exercise is to explore Gaussian and Laplacian pyramids, hierarchical image representations enabling efficient processing across multiple scales. These structures facilitate tasks like image blending and edge detection. Additionally, we aim to understand the roles of high and low frequencies in images, crucial for aspects like sharpness and contrast. This understanding enhances our ability to optimize processing techniques for diverse applications.

2. Algorithm

Description

Q1

1. Image Loading and Conversion to YIQ color space
2. Creation of Gaussian and Laplacian Pyramids - Gaussian pyramids are generated for each input image using the create_pyramid function.
3. Mask Processing - converting the mask to a binary representation and scaling it to match the dimensions of the Laplacian pyramid levels.
4. Blending based on Laplacian Pyramids.
5. Reconstruction and Conversion to RGB - reverses the pyramid decomposition process by upsampling and adding the levels to reconstruct the final image.

Q2

1. Image Loading and Conversion to grayscale.
2. Creation of Gaussian and Laplacian Pyramids.
3. Blending Based on Pyramid Levels.
4. Reconstruction of the Hybrid Image.

Implementation Details

Image Loading and Conversion to YIQ color space - it's better to work with the YIQ color space rather than RGB because the YIQ space separates the intensity (luminance) information from the color (chrominance) information. This separation allows for more effective manipulation of the image's brightness and contrast, which are crucial aspects in blending images seamlessly.
Creation of Gaussian and Laplacian Pyramids - Gaussian pyramids are generated for each input image using the create_pyramid function. This function decomposes an image into multiple levels, with each level representing a different scale of the image. The pyramid construction involves downsampling the image and applying a

Gaussian blur to each level to obtain a smoothed representation. Laplacian pyramids are derived from the Gaussian pyramids to represent the high-frequency components of the images. This is achieved by subtracting each level of the Gaussian pyramid from the next lower level.
Blending based on Laplacian Pyramids - For each level, the corresponding regions of the two input images are combined based on the mask values. The blending operation involves linearly interpolating between the Laplacian coefficients of the input images using the mask values.
Image Loading and Conversion to grayscale - it enhances the quality of the hybrid image. Although color blending is an option, I discovered that grayscale blending yields visually superior results.
Blending Based on Pyramid Levels: Blending is performed by selecting the Laplacian coefficients from the first input image up to the specified level and combining them with the Laplacian coefficients from the second input image from the specified level onwards.

Implemented from Scratch
Gaussian Kernel Calculation: The function get_gaussian_kernel calculates the Gaussian kernel matrix based on a given size. This functionality is implemented without relying on any external libraries and involves computing the binomial coefficients to generate the kernel.
Image Pyramid Construction: Functions such as create_pyramid, reduce_image, expand_image, blend_pyramids, reconstruct_image are all implemented to construct Gaussian and Laplacian image pyramids, perform image reduction and expansion, blend images based on pyramid levels, and reconstruct the final blended image. These functionalities are essential for hierarchical image processing and are custom implementations designed specifically for this task.

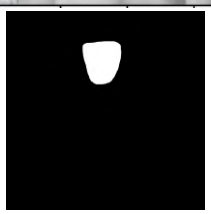
Usage of Existing Libraries
Image Loading and Manipulation: The code utilizes the Python Imaging Library (PIL) to load and convert images to grayscale.
Convolution and Blurring: The code uses the convolve2d function from the scipy.signal module to perform convolution operations and apply Gaussian blurring (convolve function). The convolve2d function efficiently computes the 2-dimensional convolution of an image with a kernel, which is essential for Gaussian blur operations. It's chosen because it is much faster than performing convolution operations directly in numpy, accelerating the overall image processing workflow.

Hyperparameters
1. max_level (blend_images function) - This parameter controls the depth of blending in the Laplacian pyramid. Higher values of max_level indicate a deeper blending process, where more levels from the Laplacian pyramid are considered for blending. Increasing max_level allows for blending at lower frequencies, potentially resulting in smoother transitions between the input images. However, setting a higher max_level

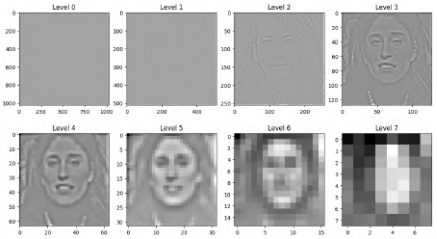
may also lead to a greater impact on the resulting image, particularly if one image dominates in size or importance. Therefore, the choice of max_level should be balanced to achieve the desired blending effect without sacrificing important visual features. While in the course we did not explicitly set this parameter, I found during implementation that adjusting it could significantly affect the blending outcome.
2. kernel_size (blend_images function) - defines the size of the Gaussian kernel for blurring during image expansion and reduction. Larger kernel sizes result in a smoother blur effect by considering more neighboring pixels during convolution. However, excessively large kernels may overly smooth the image, sacrificing important details, and incur higher computational costs.
3. max_level_1 (blend_pyramids_based_on_level function) - Determines the highest level of the Laplacian pyramid from the first input image included in the blended pyramid. Levels up to max_level_1 are sourced from pyramid1, while levels beyond are from pyramid2. This parameter affects the blended image's appearance by prioritizing high-frequency components from img1, preserving its characteristics such as fine details and edges.

In Q1, the primary objective is to seamlessly blend two images using a mask to determine each image's contribution at various levels of the Gaussian pyramid. This process aims to create a smooth transition between the two images while maintaining their overall visual integrity. On the other hand, in Q2, the goal is to generate a hybrid image that can be perceived differently based on viewing distance. This is achieved by blending two input images at varying levels of detail. The resulting hybrid image combines low-frequency components from one image with high-frequency components from the other.

3. Results

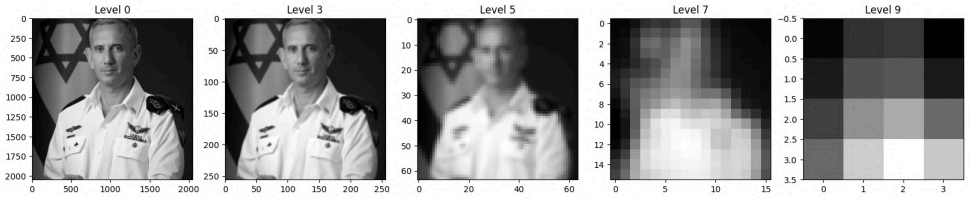


In the left image, where max_level is set to 9, the blending of Shmuel's face appears seamless within the original image. Conversely, when blending occurs across all levels in the right bottom image, Shmuel's face becomes overly dark, resulting in a less favorable outcome. Despite being amusing, blending the cat's face with HUJI's symbol in the right image reveals noticeable boundaries due to the symbol's single-color (white) background.
On the left - Hybrid image (max_level_1 = 3) and its corresponding Laplacian pyramid. At right - Hybrid images of Marilyn Einstein and Albert Monroe.

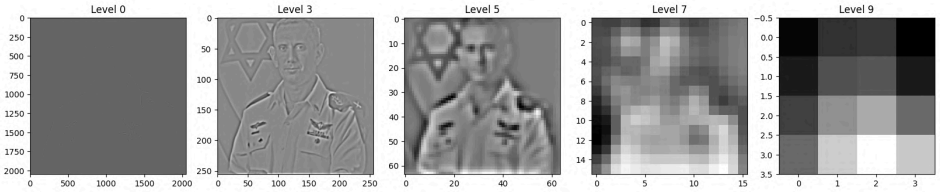



4. Pyramids

Gaussian pyramid - we can observe the gradual reduction in image resolution at each level, resulting in a series of images representing the original image at different scales.



Laplacian pyramid - we can observe the high-frequency components or details of an image at different scales, as each level represents the difference between the corresponding level in the Gaussian pyramid and a version of the image that has been upsampled and smoothed.



Conclusion

In summary, this exercise has deepened my understanding of Gaussian and Laplacian pyramids, offering efficient methods for image processing at various scales. By grasping the significance of high and low frequencies, I've honed techniques for seamless blending and gained insights into image characteristics.