Fundamentals of Image Processing

# Question 1 – Hybrid Image

Objective of this exercise was to create a hybrid image by passing one image through low pass filter and another image through high pass filter to generate a hybrid image.

Following are some of results of this exercise:

A collage of different images of a fighter jet

AI-generated content may be incorrect.

**Figure 1**Original Image 1, Original Image 2, Hybrid image created by Low Pass Original image1+ High Pass original image 2.Cutoff frequency for low pass filter=8 and Cutoff frequency for high pass filter=3

A collage of different animals

AI-generated content may be incorrect.

**Figure 2** Original Image 1, Original Image 2, Hybrid image created by Low Pass Original image1+ High Pass original image 2.Cutoff frequency for low pass filter=4 and Cutoff frequency for high pass filter=5

A collage of different images of women

AI-generated content may be incorrect.

**Figure 3** Original Image 1, Original Image 2, Hybrid image created by Low Pass Original image1+ High Pass original image 2. Cutoff frequency for low pass filter=4 and Cutoff frequency for high pass filter=5

To achieve these results

1. **Low-Pass Filtering on Image 1:**
   * The image given by the user (Image 1) is blurred using a Gaussian blur filter.
   * This is done using the *low\_pass\_cv2()* function.
   * It takes two inputs:
     + The image itself.
     + The cutoff frequency (a number entered by the user).
   * The kernel size for the blur is calculated as *cutoff\_frequency \* 6 + 1.*
2. **High-Pass Filtering on Image 2:**
   * The second image (Image 2) is passed through a high-pass filter using the *high\_pass()* function.
   * This function also takes the image and a cutoff frequency as input.
   * Inside the function:  
     a. It first applies the same Gaussian blur using *low\_pass\_cv2().*b. Then, it subtracts the blurred image from the original image to keep only high-frequency details.
3. **User Input:**
   * When the script runs, the user is asked to enter the names (with extensions) of the two images and the two cutoff frequencies (for low-pass and high-pass filters).
   * These images should be placed in the *data* folder.
4. **Resizing Images:**
   * After loading, if the two images are not the same size, they are resized to match in height and width.
5. **Normalization:**
   * The pixel values of both images are converted from the range [0, 255] to [0, 1] by dividing by 255.
6. **Filtering:**
   * Image 1 is passed through the low-pass filter.
   * Image 2 is passed through the high-pass filter.
7. **Hybrid Image Creation:**
   * The hybrid image is made by adding the low-pass filtered image and high-pass filtered image together.
8. **Clipping Values:**
   * The pixel values of the hybrid image are clipped between 0 and 1 to keep them valid after adding.
9. **Fourier Transform Calculation:**
   * The script calculates the log magnitude of the 2D Fourier Transform (FFT) for:
     + Original Image 1
     + Low-pass filtered image
     + Original Image 2
     + High-pass filtered image
     + Hybrid image
10. **Multi-Scale Visualization:**
    * The *vis\_hybrid\_image()* function shows how the hybrid image looks at different scales (resolutions), by repeatedly shrinking the image and placing the results side by side.
11. **Plotting and Saving Results:**
    * The following plots are created and saved in an *output* folder:
      + Original images
      + Low-pass and high-pass filtered images
      + Their corresponding FFT (frequency domain) images
      + The final hybrid image and its FFT
      + The scaled (multi-resolution) versions of the hybrid image
      + A combined view of original images and hybrid image side by side
12. **Output Directory:**
    * All the plots are saved as .jpg files in a folder named output inside the script directory.

Let’s deep dive into Figure 2.

|  |
| --- |
| A collage of images of a dog  AI-generated content may be incorrect.  **Figure 4** Original image, Low passed version of original image and their respective FFT |
| A collage of images of a cat  AI-generated content may be incorrect.  **Figure 5** Original image, High passed filtered version of original image and their respective FFT |

|  |
| --- |
| **A close-up of a cat  AI-generated content may be incorrect.**  **Figure 6** Scaled version of Hybrid Image |
| A close-up of a cat  AI-generated content may be incorrect.  **Figure 7** FFT of Hybrid Image |

From figure 4, we can see that the FFT of the original image (bottom-left) contains a wide spread of frequency components, including both high and low frequencies, which is expected from a detailed natural image. In contrast, the FFT of the low-pass filtered image (bottom-right) appears significantly darker in the outer regions. This shows that most high-frequency components (fine details and noise) have been suppressed. Only a narrow cross of bright lines remains, representing the dominant low-frequency content and some remaining structure along horizontal and vertical edges. This visual change confirms that applying a low-pass filter effectively smooths the image by attenuating high-frequency information, leading to a blurred appearance and reduced detail.

In the second figure, we observe that the FFT of the original image (bottom-left) shows a central bright spot with a circular distribution, representing a mixture of both low and high-frequency components that contribute to the full image detail. When a high-pass filter is applied to the image, as shown in the High-Pass Filtered (cutoff=5.0) (top-right), the image becomes much more focused on edges and fine details, while the larger regions of uniform color are suppressed. This is reflected in the FFT of the high-pass filtered image (bottom-right), where the central bright spot is significantly reduced, and high-frequency components (edges and fine texture) dominate, visible as bright points spreading from the center in multiple directions. This change indicates that the high-pass filter has removed the low-frequency components, leaving behind only the sharp details, which results in the sharp edges and more defined structure seen in the high-pass filtered image.

# Question 2 Image Pyramids

Following is a brief explanation of what image Pyramids are and the approach taken while writing the code.

## **Constructing a Laplacian Pyramid from a Gaussian Pyramid:**

1. **Build the Gaussian Pyramid:**
   * Start with the original image (.
   * Apply a Gaussian blur.
   * Down sample the blurred image (reduce resolution by a factor of 2) ().
   * Repeat the process to generate multiple levels ()
2. Build Laplacian Pyramid:
   * For each level i from 0 to n-1.
     + Upsampled to the size of (Linear interpolation used in the script)
     + Apply Gaussian blur to the Upsampled image.
     + Subtract the Upsampled image from :
     + The final level of the Laplacian pyramid is just Gn​, the smallest Gaussian level, since it cannot be decomposed further.
3. Reconstruct the original image
   * The original image can be reconstructed by using the Laplacian pyramid by reversing the process:

## Code Description

1. **Function** *pyramidsGL()*
   * Returns two lists: the **Gaussian Pyramid** and the **Laplacian Pyramid** for the given image.
   * It uses:
     + *cv2.pyrDown()* to **downsample and apply Gaussian blur** while building the Gaussian Pyramid.
     + *cv2.pyrUp()* to **upsample** the lower level of the Gaussian pyramid. This upsampled image is then subtracted it from the current level to get the Laplacian Pyramid.
   * Takes **two arguments**:
     + The **input image** (in float format, range [0, 1]).
     + The **number of pyramid levels** to generate.
2. **Function** *displayPyramids()*
   * Creates a  **figure** to visualize both Gaussian and Laplacian pyramids.
   * Takes **two arguments**:
     + A list of **Gaussian pyramid levels**.
     + A list of **Laplacian pyramid levels**.
   * It **displays the Gaussian pyramid in the first row** and the **corresponding Laplacian pyramid in the second row**, showing how image detail evolves across scales.
3. **Function** displayFFT()
   * Displays the **log magnitude of the 2D Fourier Transform (FFT)** of each image in a given list.
   * Takes one argument:
     + A **list of images** (e.g., Gaussian or Laplacian pyramids).
   * For each image:
     + It converts the image to grayscale if necessary.
     + Applies *np.fft.fft2()* and *np.fft.fftshift()* to get the frequency representation.
     + Plots the **log-transformed magnitude** spectrum to reveal frequency content.
4. **Function** *reconstruct\_from\_Laplacian\_pyramid()*
   * Reconstructs the final **blended image** from its Laplacian pyramid.
   * Takes one argument:
     + A list of **Laplacian pyramid levels** (from coarsest to finest).
   * Starting from the smallest level:
     + Upsamples it using *cv2.pyrUp()* and adds the next Laplacian level.
   * Repeats this until the original resolution is reached.
   * Returns the **reconstructed blended image**.
5. **In execution, the current script directory is obtained and an output path for storing results is created if it doesn't already exist.**
6. **Images for blending (**jet.jpeg **and** cloud.jpg**) and a mask (**mask.jpg**) are read. They are resized to the same dimensions, converted to**float32**, and normalized by dividing pixel values by 255.**
7. **The grayscale input mask is converted to a 3-channel format by stacking it along the third axis, making it suitable for RGB blending.**
8. **The number of pyramid levels is set (e.g., 5), and Gaussian and Laplacian pyramids are generated for both input images using the**pyramidsGL()**function. A Gaussian pyramid of the 3D mask is also generated.**
9. **The image pyramids for both images are visualized using the**displayPyramids()**function and saved as PNG files in the output directory.**
10. **The FFT (log magnitude spectrum) of each level of the Laplacian and Gaussian pyramids for both images is computed and displayed using**displayFFT()**. These visualizations help understand frequency components at each scale and are saved to the output path.**
11. **Blending is performed at each pyramid level using the corresponding Gaussian mask level:**blended = Gm[i] \* L1[i] + (1 - Gm[i]) \* L2[i], where Gm[i] is ith Gaussian pyramids of mask. **The result is a blended Laplacian pyramid stored in a list.**
12. **The final blended image is reconstructed from the blended Laplacian pyramid using the**reconstruct\_from\_Laplacian\_pyramid() **function.**

Following are the results of this exercise.

A collage of images of a plane flying

AI-generated content may be incorrect.

**Figure 8** Gaussian G[n] and Laplacian L[n]Image Pyramid for Image 1

A collage of clouds in the sky

AI-generated content may be incorrect.

**Figure 9** Gaussian G[n] and Laplacian L[n] Image Pyramid for Image 2

G[4] and L[4] are same images, but due to normalization their intensities are appearing different

A collage of images of a blue and green color

AI-generated content may be incorrect.

**Figure 10** FFT analysis of Gaussian Pyramid of Image 1

A close-up of several images

AI-generated content may be incorrect.

**Figure 11** FFT analysis of Laplacian Pyramid of Image 1

A screenshot of a computer generated image

AI-generated content may be incorrect.

**Figure 12** FFT analysis of Gaussian Pyramid of Image 2

A close-up of several images

AI-generated content may be incorrect.

**Figure 13**  FFT analysis of Laplacian Pyramid of Image 1

A collage of a plane flying in the sky

AI-generated content may be incorrect.

**Figure 14** Reconstructed image with Laplacian Pyramid