HW2 Report – Group2

Task 1 Hybrid image

Introduction

Our goal is to perform a hybrid image on an image pair. We can use either "Gaussian Filter" or "Ideal Filter" to blend the low-frequency components (blurry, large features) of one image with the high-frequency components (sharp, detailed features) of another. This technique illustrates how human vision interprets spatial frequencies, where the brain emphasizes different aspects of an image based on viewing distance.

The file "hybrid_image.py" is for this task, with the provided data, the result would be generated in the "../output/task1/data_output/" folder. Both results from Gaussian Filter and Ideal Filter would be shown.

Implementation procedure

The implementation of hybrid image is based on the slides provided. First, we would check if the 2 images have different sizes. If true, then we would call cv2.resize() to **resize both images** into

$$(min(y_1, y_2), min(x_1, x_2))$$

After that, we do:

Step 1. Multiply the input image by $(-1)^{x+y}$ to center the transform.

This step shifts the image's low-frequency stuff (the blurry, basic shapes) to the center of the image. It makes things easier later on when we apply filters, because the important parts will be right in the middle.

Step 2. Compute Fourier transformation of input image

This is where we break the image down into its frequency components using the Fourier Transform. Basically, it changes the image from a regular picture into a frequency-based version where we can easily separate the detailed parts from the blurry parts.

For this part, we call the np.fft.fft2() to do the Fourier transform. We have tried to build it from scratch, but the result didn't go well.

Step 3. Multiply F(u,v) by a filter function H(u,v).

We apply a filter to the frequency version of the image. If we want to keep the blurry parts, we use a low-pass filter. If we want to keep the fine details, we use a high-pass filter. This step helps us focus on just the part of the image we care about.

Step 4. Compute the inverse Fourier transformation of the result in Step 3.

After filtering, we need to convert the image back into its normal form. We do this by using the inverse Fourier Transform, which brings the filtered frequencies back to something we can look at.

Step 5. Obtain the real part of the result in Step 4.

When doing Fourier transforms, we end up with some complex numbers. We don't need the imaginary part, so here we just grab the real part to turn it back into a regular image that can be displayed.

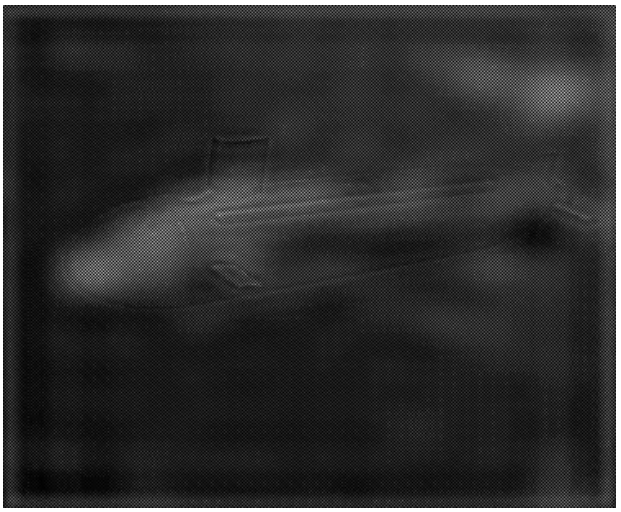
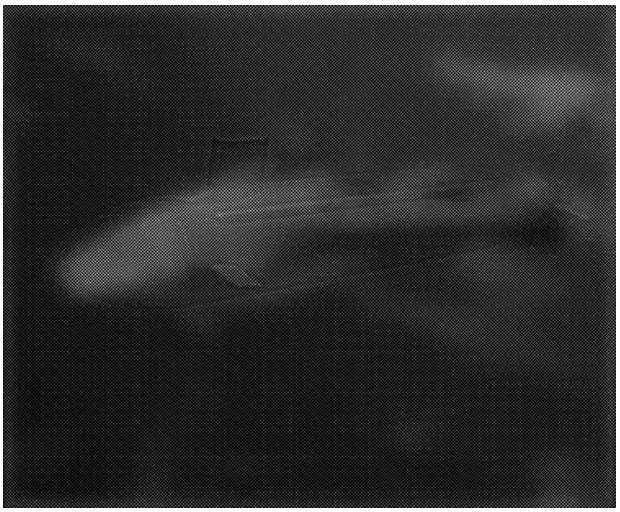
Step 6. Multiply the result in Step 5 by $(-1)^{x+y}$.

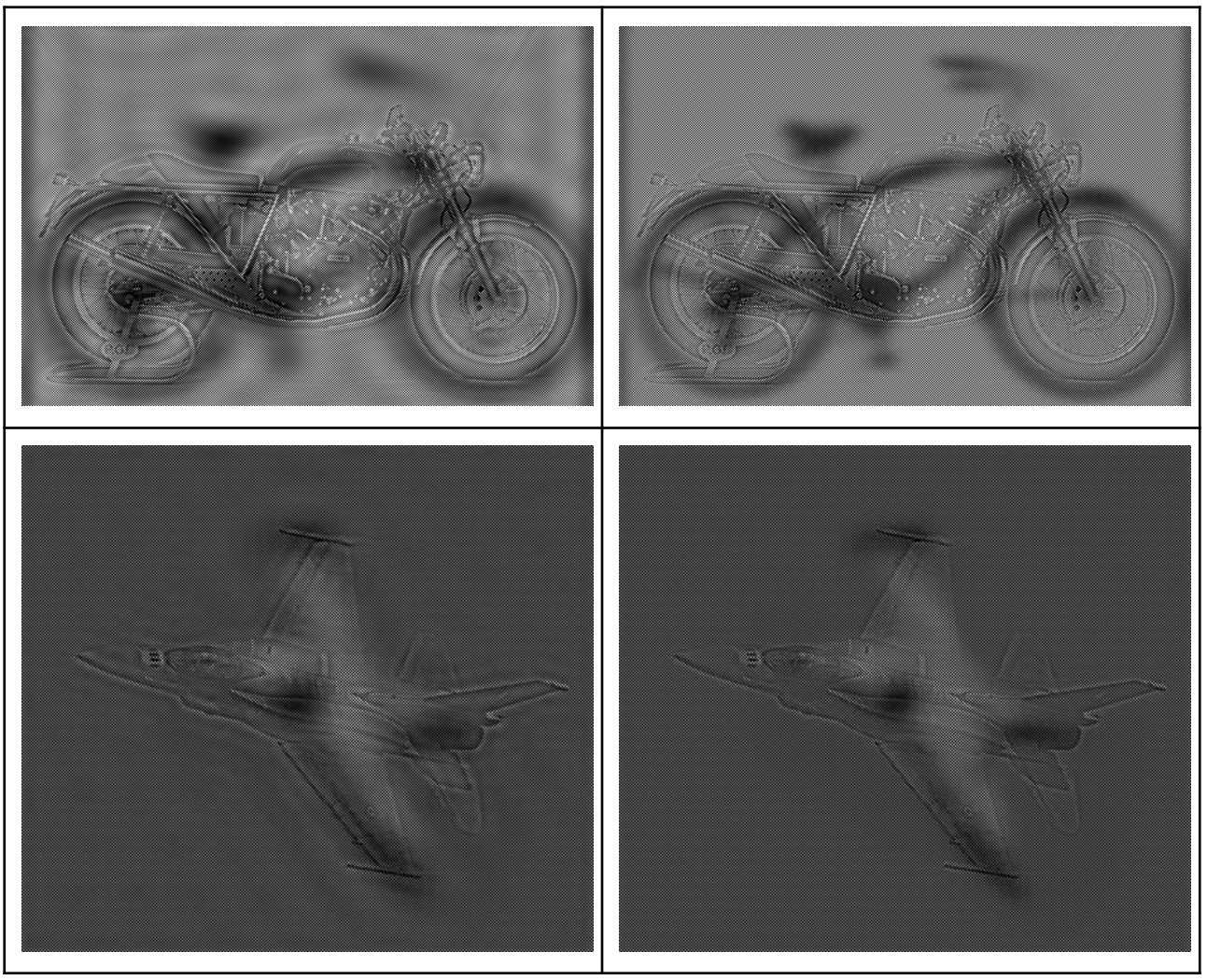
In this part, we just undo the shifting in Step 1.

Experimental results

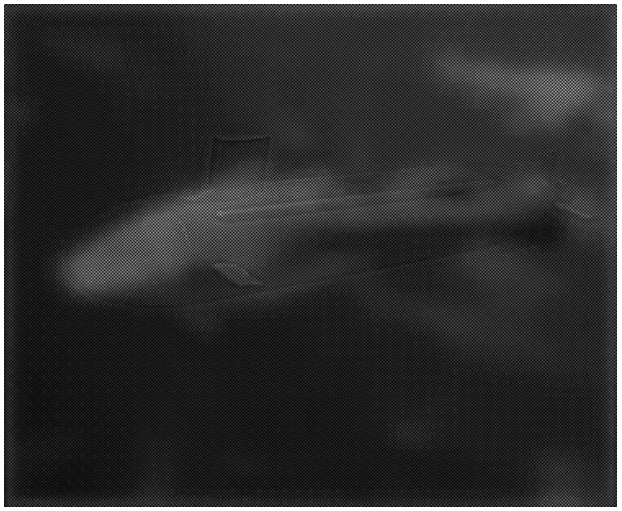
The results below are weird after being uploaded onto the google document. The original results would be saved in the "output/task1/" folder.

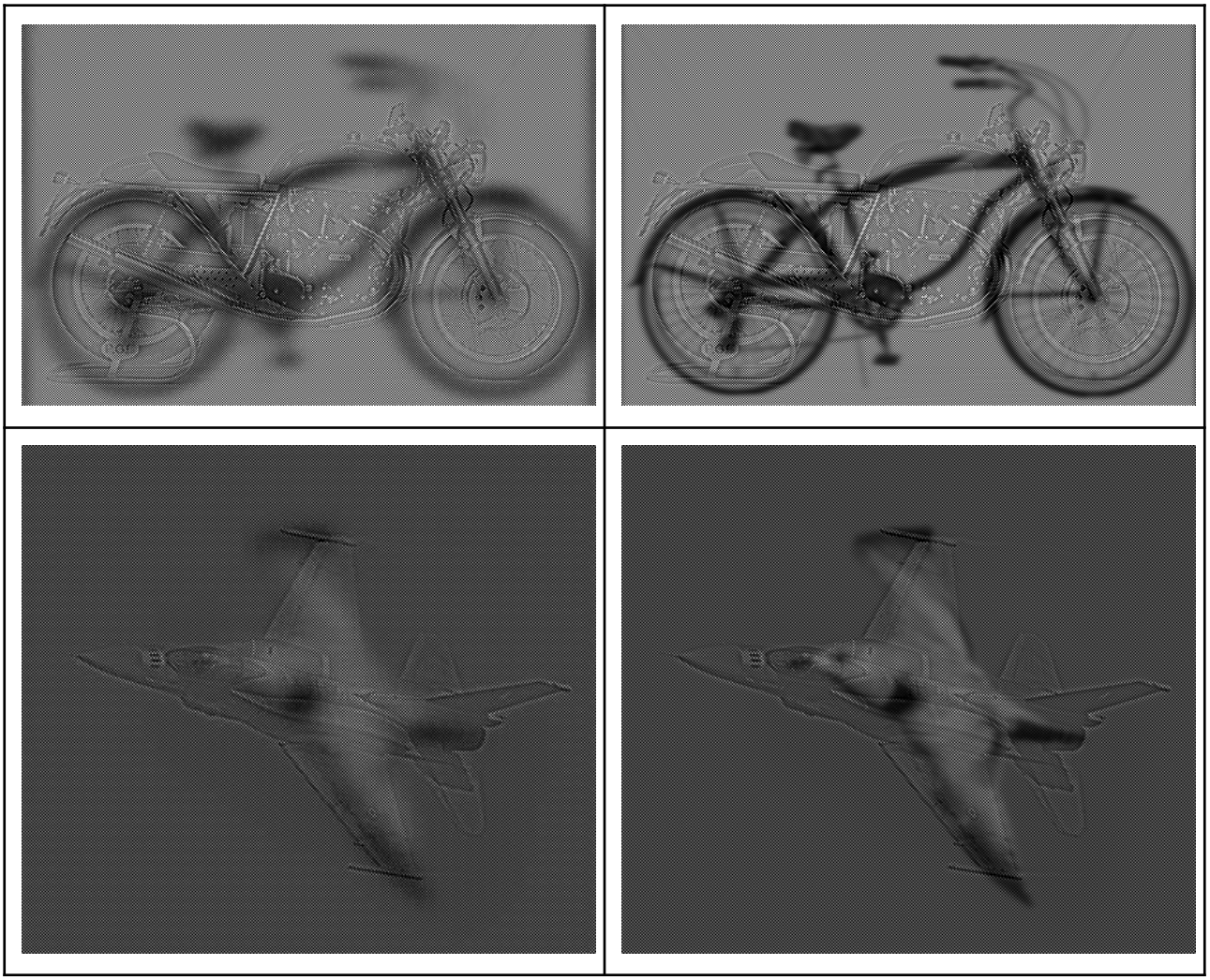
- Ideal/Gaussian Filter Comparisons

Ideal Filter results	Gaussian Filter results
	



- Cutoff Frequency Comparisons

low-pass $d = 10$, high-pass $d = 30$	low-pass $d = 30$, high-pass $d = 30$
	



Discussion

In the original results, we can see that the results generated by Ideal Filter are more noisy. But except that, we cannot actually tell the difference between the results generated by Ideal Filter and Gaussian Filter.

Cutoff frequency ***d*** controls how aggressively the filter affects the frequency components of the image. A **small *d*** means a more restrictive filter, while a **larger *d*** means a gentler filter, allowing more frequencies through.

In the experiments, we tried many different ***d*** and we found that for low-pass filters, we should use a **smaller *d*** or the image may be too sharp and it would make the high-pass filtered image not obvious to be seen.

Conclusion

In this project, we created hybrid images by blending the low-frequency components of one image with the high-frequency components of another using both Gaussian and Ideal filters. We observed that the Ideal filter introduces more noise due

to its sharp transitions, while the Gaussian filter produces smoother results. The cutoff frequency ***d*** plays a crucial role, where a smaller ***d*** helps control the blur in low-pass filters and prevents the high-pass details from being overshadowed. Overall, the experiment demonstrates how different filters and frequencies affect image perception, depending on viewing distance.

Task 2 Image pyramid

Introduction

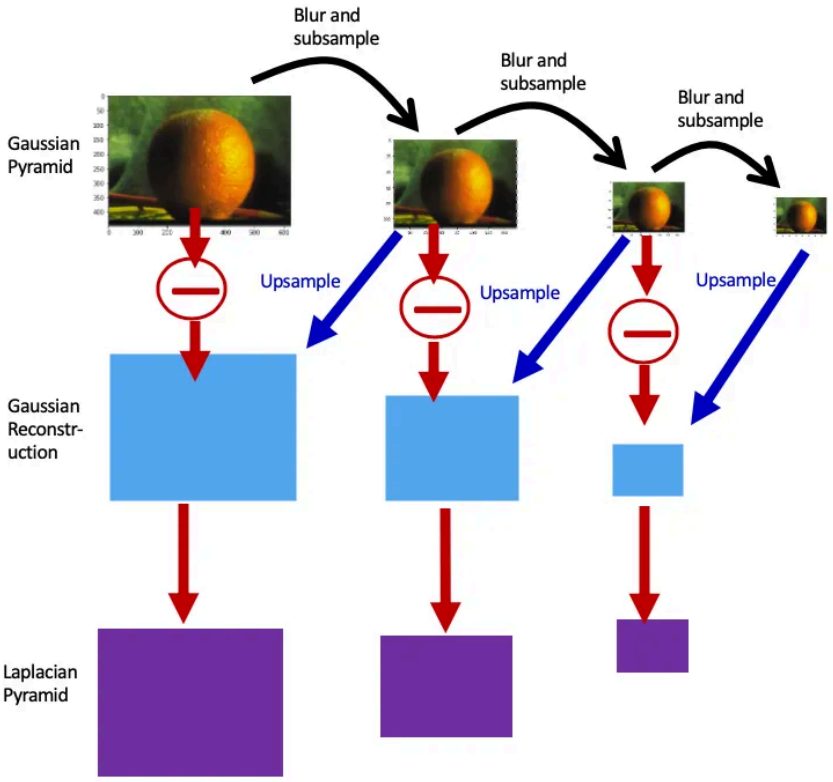
Image pyramid is a set of images with different resolutions, allowing for multi-scale analysis of images. The low-resolution images are blurry and lack detail, primarily capturing low-frequency information. In contrast, high-resolution images are more detailed and encompass both low and high frequencies.

Our goal is to implement an image pyramid, which is composed of Gaussian Pyramids and Laplacian Pyramids. In the end, we will show the result in the frequency domain.

The file “image_pyramid.py” is for this task, and the result would be put in the “../output/task2/data_output/” folder.

Implementation procedure

The picture below shows the relationship between Gaussian Pyramids and Laplacian Pyramids.



Step 1: Smooth images with a Gaussian

1. Gaussian Kernel:

Determine the size and standard deviation (σ) of the Gaussian kernel.

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Normalize the kernel to ensure the sum of all its values equals one.

2. Convolution:

Apply the Gaussian filter to the input image using convolution, represented as G=H*F, where G is the output image, H is the Gaussian kernel, and F is the input image.

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

The padding method used is copy edge, which helps maintain the image structure near the edges during convolution.

3. Subsampling:

Reduce the size of the output image to half in both width and height dimensions by selecting every second pixel, resulting in a lower-resolution image.

Step 2: Laplacian

1. Upsampling

Implement upsampling using Nearest Neighbor Interpolation to increase the resolution of the subsampled image.

2. Laplacian Pyramid

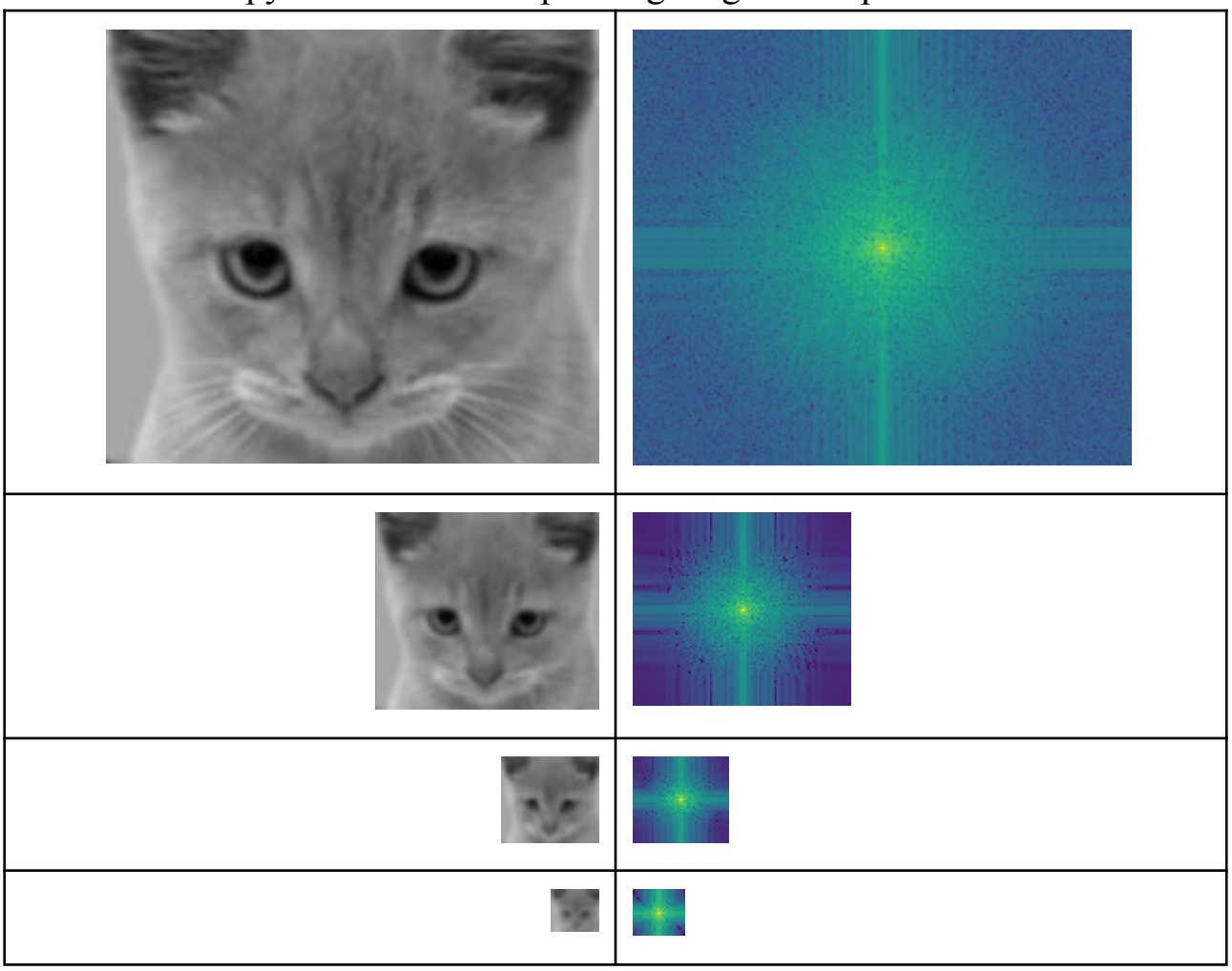
Construct a Laplacian pyramid by pixel-wise subtracting the blurred, upsampled version of the image layer from the original layer. This process highlights the details and edges in the image.

Step 3: Magnitude spectrum

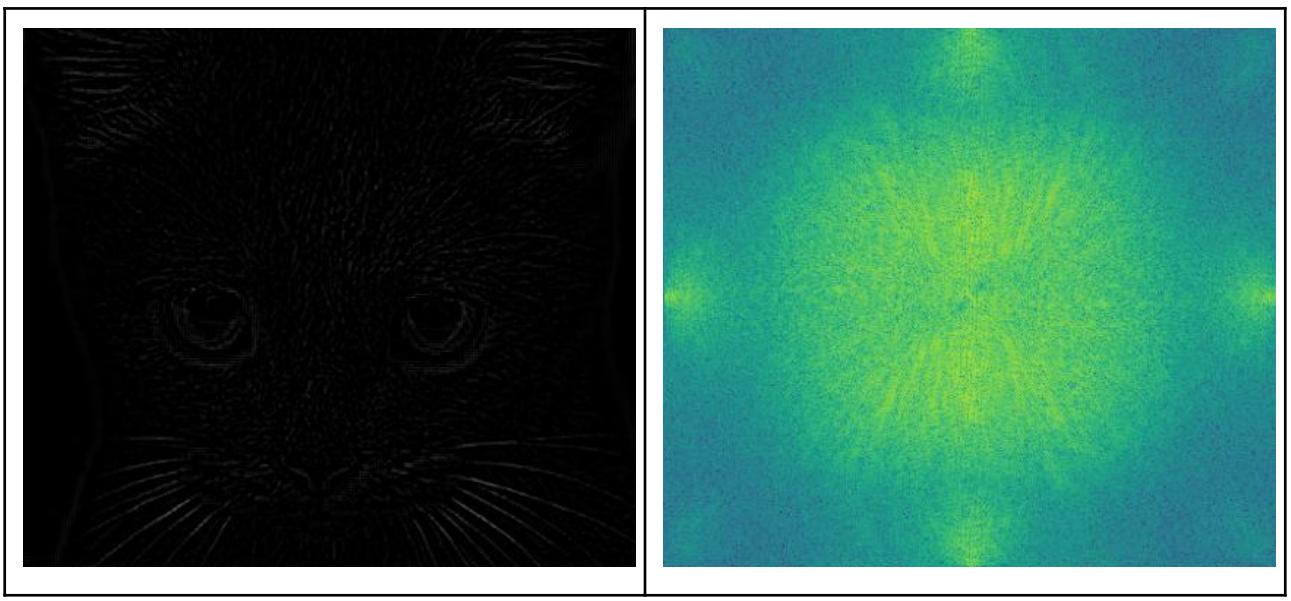
Utilize Fourier Transformation to represent the two pyramids in the frequency domain. Display the corresponding magnitude spectrum of the two pyramids.

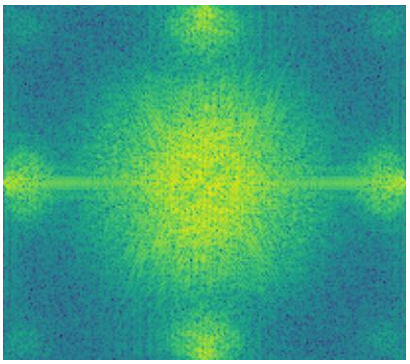
Experimental results (data)

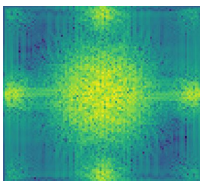
- Gaussian pyramid and corresponding magnitude spectrum

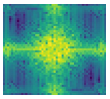


- Laplacian pyramid and corresponding magnitude spectrum



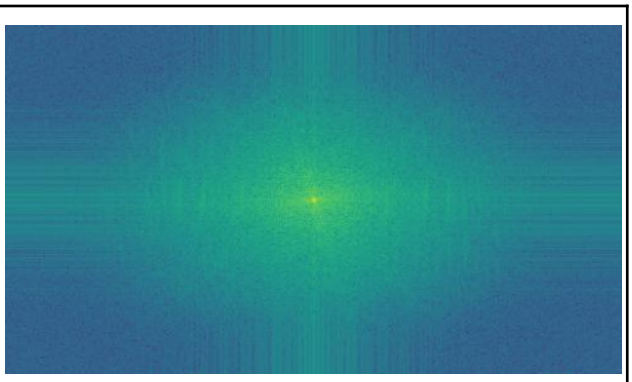
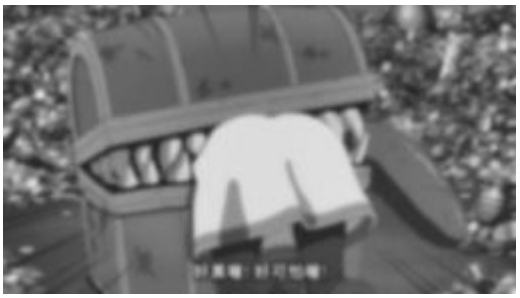
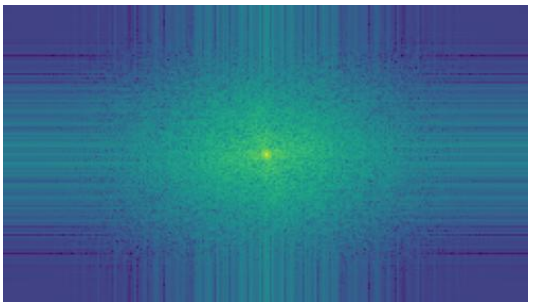



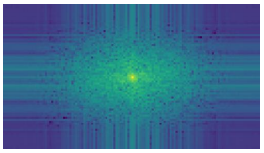


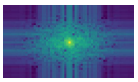


Experimental results (my data)

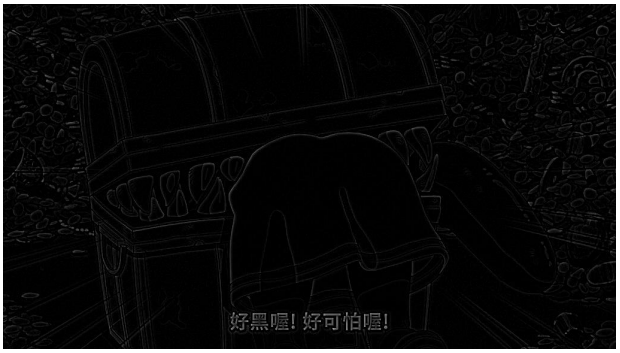
- Gaussian pyramid and corresponding magnitude spectrum

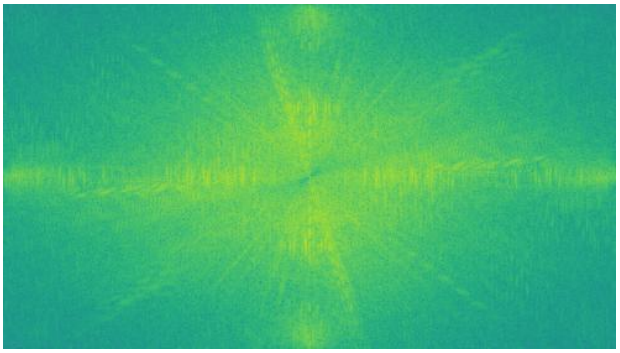




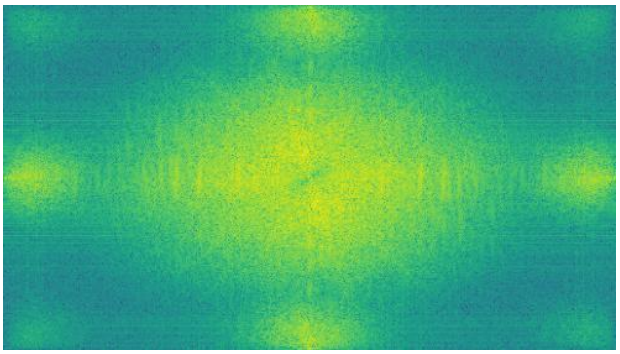


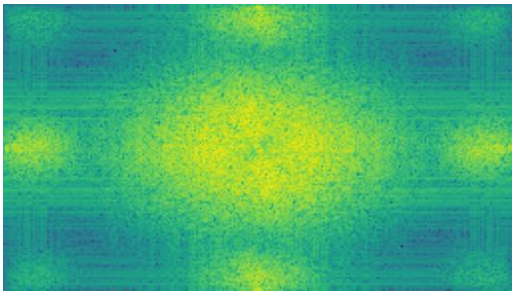



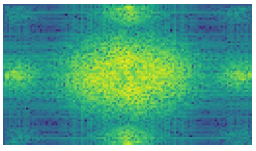
- Laplacian pyramid and corresponding magnitude spectrum









Discussion

Adjusting the Gaussian filter parameters, specifically the filter size and σ, took some time. A larger filter size provided more smoothing but risked losing important details, while a smaller size preserved details but was less effective in reducing noise.

Similarly, a higher σ gave more smoothing but blurred edges, while a lower σ maintained edges but didn't smooth as much.

Conclusion

In this project, the implementation of the image pyramid, including Gaussian and Laplacian pyramids, provided valuable insights into multi-resolution image processing. By progressively smoothing, downsampling, and upsampling images, we were able to efficiently capture and analyze different levels of detail.

Task 3 Colorizing the Russian Empire

Introduction

Our goal is to automatically generate a color image from digitized Prokudin-Gorskii glass plate images. These glass plate images capture three exposures of each scene on a single plate using red, green, and blue filters. By dividing the image into three equal sections and aligning the green and red channels with the blue channel, we can reconstruct the original color image. However, the challenge lies in performing this process fast and efficiently, especially when dealing with large full-size glass plate images.

Implementation procedure

1. **Remove surrounding border**
   We found that if we don't remove the surrounding black borders first, the BGR is unable to align with a great result. So we need to crop the image first to get the central image.

2. **Splitting image to BGR**
   The input image we are working with contains three exposures of the same scene stacked vertically. Each of these exposures corresponds to one of the color channels: blue (B), green (G), and red (R). Here we should remember the order isn't RGB.

   Each image height equals to img height divided three(height = int(h / 3)).

   - first slice : img[0:height, :] extracts the top third of the image, which corresponds to the blue channel

- second slice img[height:2*height, :] extracts the middle third of the image, corresponding to the green channel
- third slice img[2*height:3*height, :] extracts the bottom third of the image, which is the red channel

3. **Align image - shifting**

After splitting the image into the B, G, and R channels, we perform alignment by shifting the green and red channels to match the blue channel.We use **Normalized Cross-Correlation (NCC)** as the method to align the channels. This technique calculates the similarity between two images and helps us find the optimal shift for aligning the channels. In our case, we treat the blue channel as the reference and shift the green and red channels accordingly to achieve proper alignment.And also did the **Edge Detection** to improve the accuracy of alignment, reducing the influence of noise or irrelevant background details.

If the image type is '.tif', we need to do the **subsampling** first. Make sure the process can be fast and efficient. We choose 1/10 to sub sampling the image, and scale back to the original full-resolution when applying the shifts.

Next, we **calculate the optimal shift** for the green and red channels by comparing them to the blue channel. We perform this by iterating over a predefined range of possible shifts and using NCC to determine the similarity between the shifted channels and the reference blue channel.

`align()` calculates the NCC for each possible shift and returns the best shift values `(shift_red and shift_green)` that align the channels most accurately

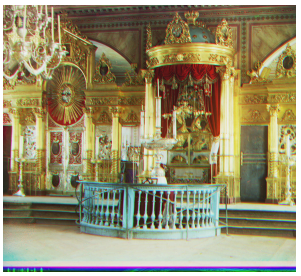
4. **Save image**

Once the red and green channels are aligned with the blue channel, we use `np.dstack()` to combine the three channels into a single color image. This function stacks the red, green, and blue channels along the third axis, forming an RGB image.Since OpenCV uses BGR as its default color format for saving images, we need to convert the RGB image to BGR before saving it. This is done using the cv2.cvtColor() function. Finally, we save images by cv2.imwrite().

Experimental results

- data from resource













- data from ourselves(resource website)





Discussion

- **Precision**

Although we try to make sure every image can be aligned with high quality, using the same crop method may not be suitable for all images. I try different crop size numbers, but if we automatically choose the crop size images will not be the same size, it will cause that we can't align two different images with different sizes. The trade-off is that some images are not as precise as others.(right side example)

- **Efficiency**

  Total execution time with data from the package resource(13 images) is around 28 seconds. We can change the subsample rate to improve the execution time, but it will slow down the precision. So how to choose the rate is the key problem. Here I tried a different rate to compare the trade-off between efficiency and precision, at the end we choose 10 be the rate.

Conclusion

We developed a method to align and colorize Prokudin-Gorskii glass plate images by shifting the green and red channels to match the blue channel using Normalized Cross-Correlation (NCC). Key steps like border removal and

subsampling improved both accuracy and speed, with a subsample rate of 1/10 offering a good balance. Improvements could include refining the cropping method to handle different image sizes better and exploring more advanced alignment algorithms for greater precision.This technique could be useful in historical photo restoration, digitization projects, and enhancing archival images for modern use.

Task1 Hybrid image	紀宇烜
Task2 Image pyramid	羅孟妍
Task3 Colorizing the Russian Empire	張芸烜