Homework 2: Hybrid image, Image pyramid, Colorizing the Russian Empire

I-Sheng Fang
The Master's Degree Program in Robotics
National Yang Ming Chiao Tung University
`isfang.gdr09g@nctu.edu.tw`
Debug, Experiment, Discussion, Conclusion

Hsiang-Chun Yang
Institute of Multimedia Engineering
National Yang Ming Chiao Tung University
`yanghc.cs09g@nctu.edu.tw`
Implementation, Discussion.

Yu-Lin Yeh
Institute of Multimedia Engineering
National Yang Ming Chiao Tung University
`s995503@gmail.com`
Introduction, Experiment, Discussion.

Abstract

In this assignment, we have three tasks to implement, namely the hybrid image, image pyramid and colorizing the Russian Empire. In addition to compare the result with Gaussian filter and Ideal filter. Moreover, attempt to realize the various result of different sigma. We also collect some interesting to images to do more trial.

1 Introduction

The main target of hybrid image is to merge two images together in frequency domain. Pass two images through a high-pass filter and a low-pass filter separately and mix the filtered result to produce the hybrid image. This technique is used in many applications. For instance, animation processing, image stitching [1][2].

Image pyramid is a multi-scale representation of the image using multi-resolution to explain the structure of the image. This method is mainly used for image segmentation or compression [3][4]. There are some common image pyramids such as Gaussian pyramid and Laplacian pyramid.

In "Colorizing the Russian Empire" task, we have to colorize the Prokudin-Gorskii glass plate image, which records three exposures of every scene onto a glass plate using a red, green, and blue filter. Our goal is to find the best displacement between each color channel, then layer each channel together to create an RGB image [5].

2 Implementation Procedure

2.1 Hybrid image

Hybrid image is the sum of a low-pass filtered image and a high-pass filtered image. Filtering can be easily done by transforming the image into frequency domain and applying a filter to it. To obtain the hybrid image, combine two filtered spectrum together then compute the inverse Fourier transformation of it. There are two types of filters, ideal and Gaussian, which will be introduced in the next sections.

2.1.1 Ideal filter

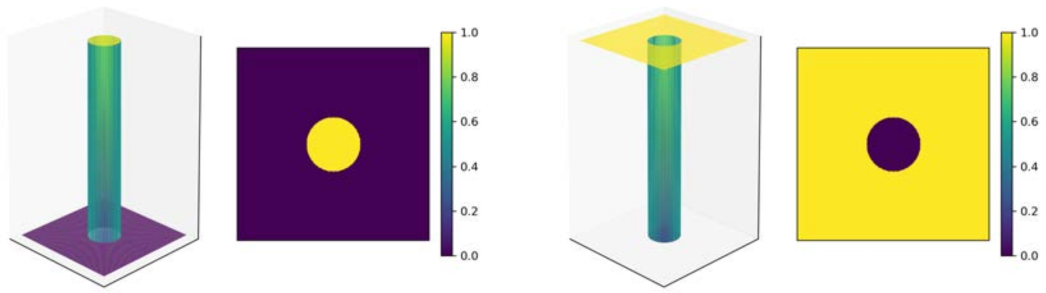
The low pass ideal filter is defined as

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \le D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$$

While the high pass ideal filter is defined as

$$H(u,v) = 1 - \text{low pass filter}$$

Note that $D(u,v)$ is the L2 distance between the origin and (u,v), and D_0 is the cutoff frequency. Figure 1 is the visualization of the ideal filter, with $u \in [-0.5, 0.5]$, $v \in [-0.5, 0.5]$, and $D_0 = 0.28$.



Low pass High pass

Figure 1: Ideal filter

2.1.2 Gaussian filter

The low pass Gaussian filter is defined as

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

While the high pass Gaussian filter is defined as

$$H(u,v) = 1 - \text{high pass filter}$$

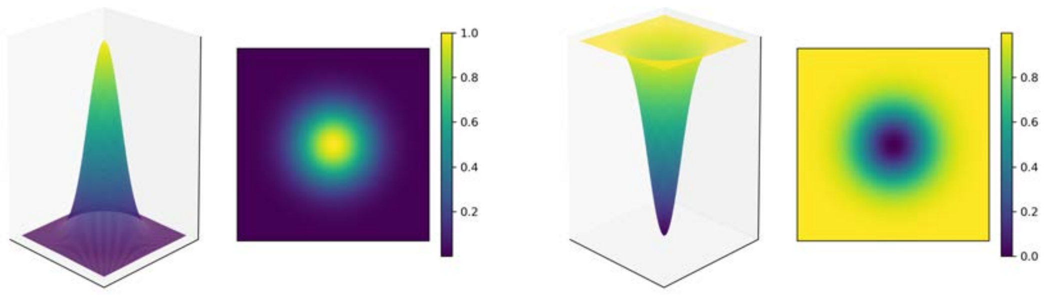
Note that $D(u,v)$ is the L2 distance between the origin and (u,v). Figure 2 is the visualization of the Gaussian filter, with $u \in [-0.5, 0.5]$, $v \in [-0.5, 0.5]$, and $D_0 = 0.28$

2.1.3 Filtering

Then we can filter the image with the following code.

```
ch = img.shape[2]
freq = [fftshift(fft2(img[:,:,c])) for c in range(ch)]
mag = [np.log(np.abs(f)+1e-5) for f in freq]
filtered_freq = [fftshift(fft2(img[:,:,c]))*fltr for c in range(ch)]
filtered_mag = [np.log(np.abs(f)+1e-5) for f in filtered_freq]
filtered_channel = [ifft2(ifftshift(f)).real for f in filtered_freq]
```

Where `fft2`, `ifft2`, `fftshift`, and `ifftshift` are functions provided by `Numpy`. After transforming the image from spatial domain to frequency domain, we multiply the frequency and the filter



Low pass High pass

Figure 2: Gaussian filter

element by element to do the filtering. And use inverse Fourier transformation to transform the filtered frequency back to the spatial domain. Notice that we also compute the magnitude spectrum for the original frequency and the filtered one.

2.1.4 Hybrid image

Once we get a low-pass filtered image and a high-pass filtered image, the final hybrid result can be obtained by adding these two images together. Please refer to `plot_hybrid` function in `HW2_1.py` for the implementation.

2.2 Image pyramid

The procedure of building a Gaussian pyramid is

- Set the finest scale layer to the original image
- For each layer, going from next to the finest to the coarsest
  Smooth the next finest layer with the Gaussian filter
  Obtain this layer by subsampling the smoothed result

After building the Gaussian pyramid, we can build the Laplacian pyramid by

- Set the coarsest scale layer to the top layer of the Gaussian pyramid
- For each layer, going from next to the coarsest to the finest
  Upsample the next coarsest layer
  Smoothing the upsampled result
  Obtain this layer by subtracting the layer before the next coarsest one with the smoothed result

In the following sections, we will give more detailed explanation of each step.

2.2.1 Smoothing

To smooth the image, we have to build the Gaussian filter first. By given the filter size n and standard deviation $sigma$, the 2D Gaussian filter is defined as

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

Where (x, y) is the 2D coordinate in the Gaussian filter, and the filter is centered at the origin.

The implementation of Gaussian filter is as follows. Notice that we normalize the filter at the end. If the summation of the filter is not equal to 1, the smoothed image may have the different brightness compared to the original one.

```python
def gaussian_filter(filter_size, sigma):
    kernel = np.zeros((filter_size, filter_size))
    center = filter_size // 2
    for i in range(filter_size):
        for j in range(filter_size):
            g = np.exp(-1*((center-i)**2+(center-j)**2)/(2*(sigma**2)))
            g /= 2 * np.pi * (sigma**2)
            kernel[i,j] = g
    kernel /= kernel.sum()
    return kernel
```

Then we can do the convolution between the original image and the Gaussian filter to smooth the image. Convolution can be done in many ways. Since matrix multiplication with sliding window may be time consuming, we apply Fourier Transform to both original image and the filter, and do the convolution in frequency domain. We use `fft2` and `ifft2` provided by `Numpy` to achieve this.

```python
def smooth(img, kernel):
    return ifft2(fft2(img, img.shape)*fft2(kernel, img.shape)).real
```

2.2.2 Subsampling

It is easy to subsample a image in `Python` by index slicing.

```python
img = img[::2,::2]
```

2.2.3 Gaussian pyramid

Build the Gaussian pyramid by combining section 2.2.1 and section 2.2.2 together.

```python
def gaussian_pyramid(img, num_layers, kernel):
    res = [np.array(img)]
    for i in range(num_layers-1):
        img = smooth(img, kernel)
        img = img[::2,::2]
        res.append(np.array(img))
    return res
```

2.2.4 Laplacian pyramid

After building the Gaussian pyramid, we can build the Laplacian pyramid by following the procedure in section 2.2. The implementation of Laplacian pyramid is as follows. Note that we use nearest neighbor tiling for upsampling.

```python
def laplacian_pyramid(g_pyramid, num_layers, kernel):
    res = [g_pyramid[-1]]
    for i in range(1, num_layers):
        upsample = g_pyramid[num_layers-i]
        upsample = upsample.repeat(2, axis=0).repeat(2, axis=1)
        if g_pyramid[num_layers-i-1].shape[0] % 2:
            upsample = upsample[:-1,:]
        if g_pyramid[num_layers-i-1].shape[1] % 2:
            upsample = upsample[:,:-1]
        upsample = smooth(upsample, kernel)
        res.append(g_pyramid[num_layers-i-1]-upsample)
    return res[::-1]
```

2.3 Colorizing the Russian Empire

The best displacement between two images, `img1` and `img1`, can be found by exhaustively searching over a set of possible displacements, computing the similarity between `img1` and of each displaced `img2`, and taking the one with the lowest difference. But exhaustive search will cost considerable amount of time when the search range or image size become too large. To tackle this problem, we have to apply a coarse-to-fine searching strategy using Gaussian pyramid. First, estimate the displacement of the coarsest images which is at the top of pyramid, then keep refining the displacement with the finer images until we reach the bottom of the pyramid.

2.3.1 Data preprocessing

Since there are white borders around digitized Prokudin-Gorskii glass plate image, the first step is to remove those noise. We use the following functions to remove the white borders.

```
def remove_white_border(img):
    white = 215
    while np.all(img[0,:]>=white):
        img = img[1:,:]
    while np.all(img[-1,:]>=white):
        img = img[:-1,:]
    while np.all(img[:,0]>=white):
        img = img[:,1:]
    while np.all(img[:,-1]>=white):
        img = img[:,:-1]
    return img
```

If there still exists a line of pixels around the image with all values greater than `white`, the line of pixels will be treated as the white border and stripped from the image. After that, the image will be split into three parts with the same height. The top $\frac{1}{3}$ of Prokudin-Gorskii glass plate image represents the exposure of the blue channel, while the middle $\frac{1}{3}$ and bottom $\frac{1}{3}$ represent exposures of the green and red channels respectively. And there are dark borders around each channel, so the next step is to remove those dark areas. To do so, we remove 3.5% of pixels from the top, bottom, right, and left of the image.

```
def remove_black_border(img):
    h, w = img.shape
    new_h, new_w = int(0.93*h), int(0.93*w)
    return img[h-new_h:new_h,w-new_w:new_w]
```

2.3.2 Alignment

Given the current estimation of the best displacements (x, y) and the searching range n. The possible displacements are $(x - n \sim x + n, y - n \sim y + n)$. We can obtain the best displacement between `img1` and `img2` by choosing the one with the lowest difference between `img1` and displaced `img2`. The implementation of alignment is as follows.

```python
def align(img1, img2, displ, search_range, measure):
    min_diff = float('inf')
    img1 = img1 / 255
    img2 = img2 / 255
    best_displ = [0, 0]
    for i in range(displ[0]-search_range, displ[0]+search_range):
        for j in range(displ[1]-search_range, displ[1]+search_range):
            shifted_img2 = shift(img2, [i, j])
            diff = measure(img1, shifted_img2)
            if diff < min_diff:
                min_diff = diff
                best_displ = [i, j]
    return best_displ
```

Where `displ` is the current estimation of the nest displacement, and `measure` is the metric to measure how well two images match. More details about `measure` will be given in section 2.3.3.

2.3.3 Metric

We have tried five different metrics to evaluate the difference between two images x and y.

- Euclidean distance: $\sqrt{\Sigma_{i=0}^{N}(x_i - y_i)^2}$
- Manhattan distance: $\Sigma_{i=0}^{N}|x_i - y_i|$
- Zero-normalized cross-correlation (ZNCC): $\frac{1}{n}\frac{(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y}$
- Normalized cross-correlation (NCC): $\frac{1}{n}\frac{xy}{\sigma_x\sigma_y}$
- Structural similarity (SSIM): $\frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)}$ where c_1 and c_2 are constants

For the implementations of those metrics, please refer to `euclidean`, `manhattan`, `zncc`, `ncc`, and `ssim` in `HW2_3.py`. When two images match well, ZNCC, NCC, and SSIM will become large. We simply multiply it by -1 so that it represents the difference between two images.

2.3.4 Colorize

The complete implementation of colorization is as follows.

```python
def colorize(r, g, b, pyramid_layer, measure):
    if pyramid_layer == -1:
        displ_r = align(b, r, [0, 0], 15, measure)
        displ_g = align(b, g, [0, 0], 15, measure)
    else:
        kernel = gaussian_filter(5, 0.7)
        g_py_r = gaussian_pyramid(r, pyramid_layer, kernel)[::-1]
        g_py_g = gaussian_pyramid(g, pyramid_layer, kernel)[::-1]
        g_py_b = gaussian_pyramid(b, pyramid_layer, kernel)[::-1]

        displ_r = align(g_py_b[0], g_py_r[0], [0, 0], 15, measure)
        displ_g = align(g_py_b[0], g_py_g[0], [0, 0], 15, measure)
        for i in range(1, len(g_pyramid_b)):
            displ_r = [d*2 for d in displ_r]
            displ_g = [d*2 for d in displ_g]
            displ_r = align(g_py_b[i], g_py_r[i], displ_r, 5, measure)
            displ_g = align(g_py_b[i], g_py_g[i], displ_g, 5, measure)
    shifted_r = shift(r, displ_r)
    shifted_r = shift(g, displ_g)
    result = [shifted_r, shifted_g, b]
    return np.stack(result, axis=2)
```

Where `gaussian_filter` and `gaussian_pyramid` are imported from `HW2_2.py`. `pyramid_layer` is the number of layers of Gaussian pyramid. Set `pyramid_layer` to -1 to disable the coarse-to-fine searching strategy.

3 Experimental Result

In this section, we show the results of three tasks and discuss the effect with different parameters.

3.1 Hybrid image

We test our implementation with the images TA provided with Gaussian filter and ideal filter in Figure 3, 4, 5 and 6. We also test hybrid image method with the badge of NYMU and NCTU and generate the badge of NYCU in Figure 7, 8.

Comparing to the Gaussian filter results, the ideal filter results have more path artifacts, but more obvious edge. The higher cutoff ratio causes larger filter, which means lower variance, and more obvious edge.

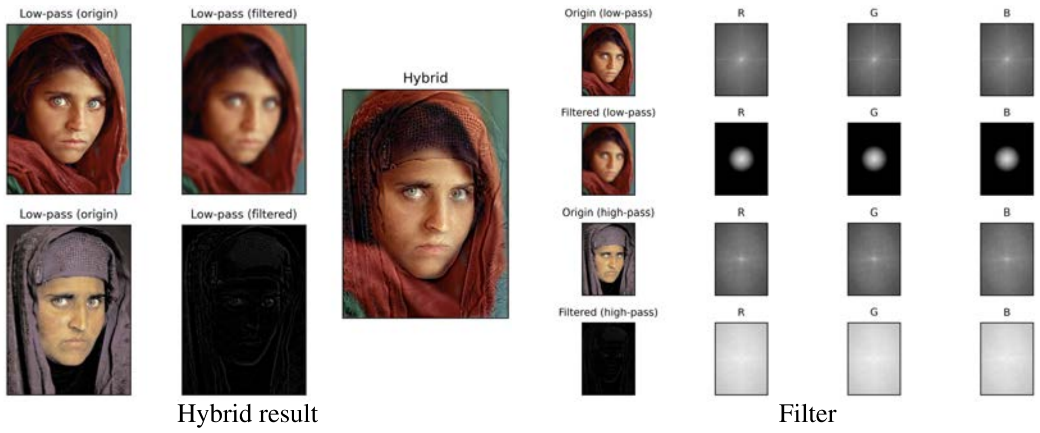


Figure 3: The results of Afghan girl of hybrid image and filter visualization with Gaussian filter and 0.08 cutoff ratio

method	time(sec)
Conv. w/ Freq.	0.0421
Conv. w/ Sliding Window	2.4954

Table 1: The average running second in different domain of convolution.

3.2 Image pyramid

We test our image pyramid implementation with the images TA provided in Figure 9, and the badge of NCTU in Figure 10. We also evaluate the running time of convolution in different domain in Table 1.

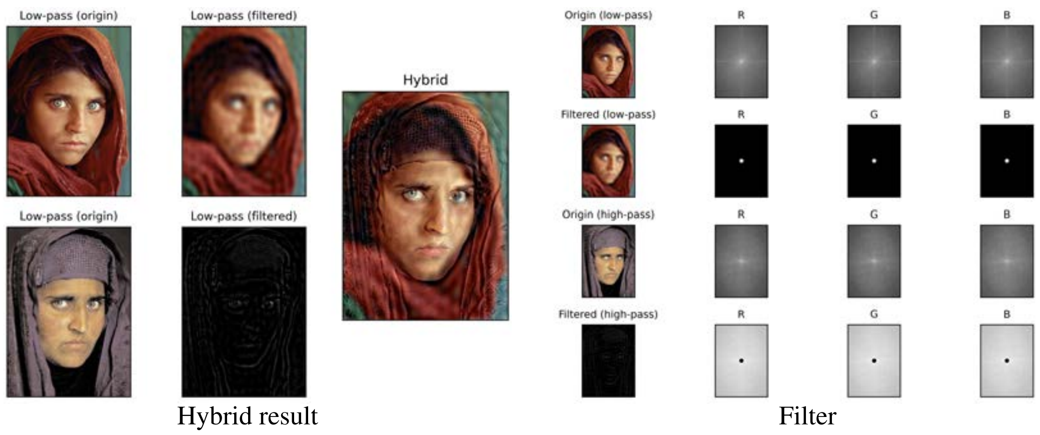


Figure 4: The results of Afghan girl of hybrid image and filter visualization with ideal filter and 0.08 cutoff ratio

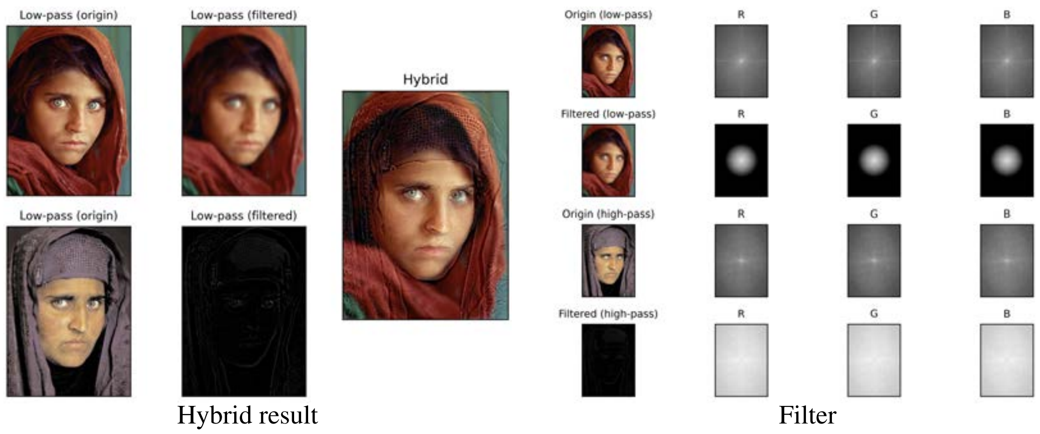


Figure 5: The results of Afghan girl of hybrid image and filter visualization with Gaussian filter and 0.1 cutoff ratio

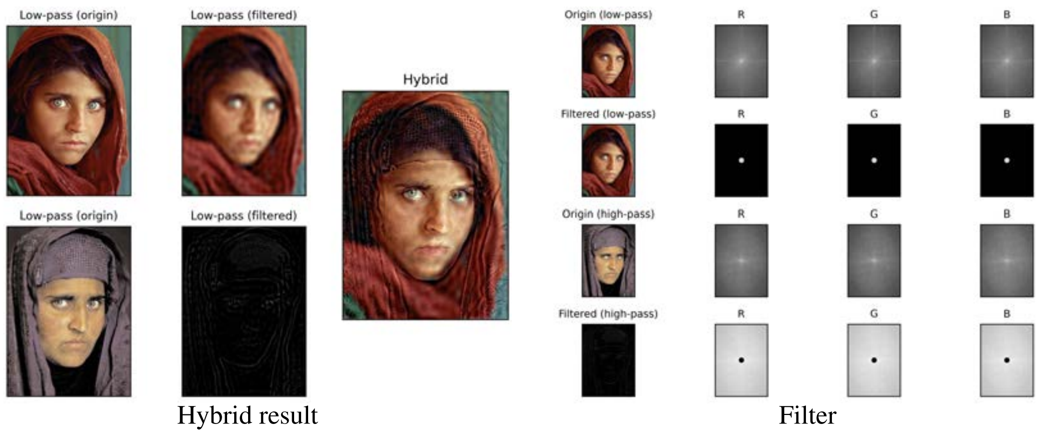


Figure 6: The results of Afghan girl of hybrid image and filter visualization with ideal filter and 0.1 cutoff ratio

From these results, we confirm that Laplacian pyramid preserves more content information than Gaussian pyramid. Therefore, Laplacian pyramid has the ability to reconstruct original image. We also discover that the lower sigma value cause more polarized results.

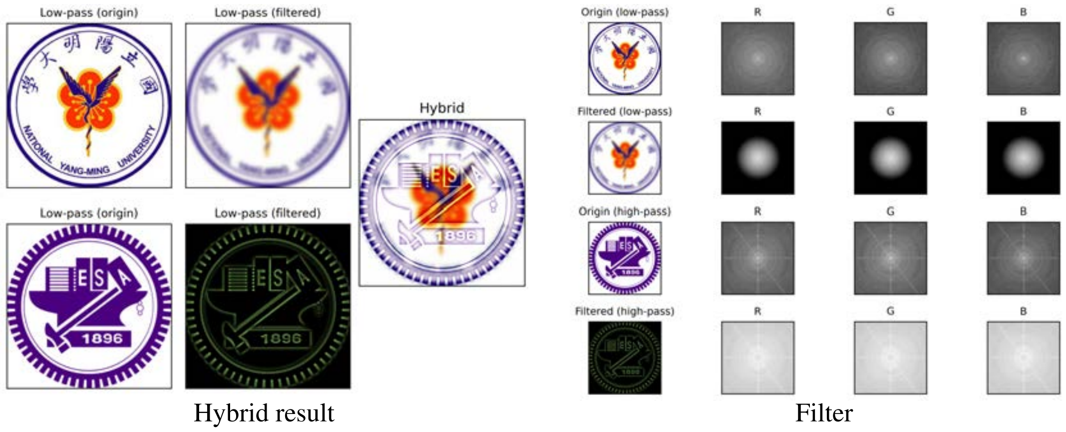


Figure 7: The badge of NYCU which is the hybrid result between the badge of NYMU and NCTU with Gaussian filter, and its filter visualization

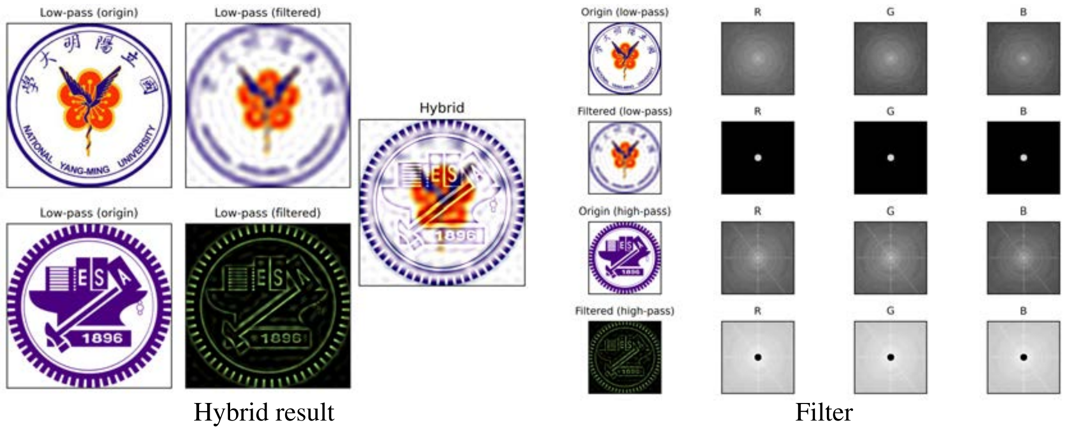


Figure 8: The badge of NYCU which is the hybrid result between the badge of NYMU and NCTU with ideal filter, and its filter visualization

3.3 Colorizing the Russian Empire

We test our image colorizing implementation with the images TA provided in Figure 11. We also evaluate the running time of convolution in different domain in Table 2. We do not discover the difference with most of the parameters like pyramid depth, filter size, filter variance and similarity metric. Only the choice of base channel in some images effects the results as Figure 12 shown.

method	time(sec)
Conv w/ Freq.	0.9314
Conv w/ Sliding Window	425.4569

Table 2: The average running second in different domain of convolution.

4 Discussion

In "Image pyramid" task, we discover that doing convolution in the frequency domain has 40 times computation efficiency as Table 1 shown. In "Colorizing the Russian Empire" task, we discover that the image, "Emir", must choose the green channel as the base channel for alignment. Otherwise, it has a large miss alignment as Figure 12 shown. We notice that "Emir" has a large intensity difference between each channel as Figure 13 shown. We believe it is the main reason causing the

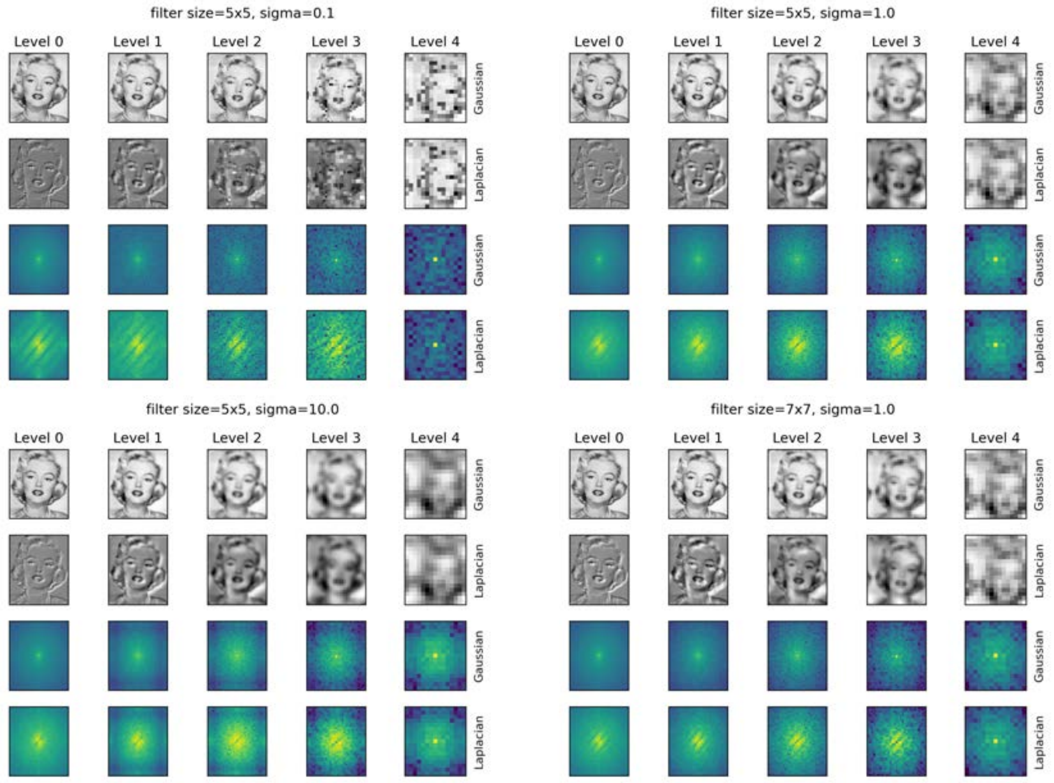


Figure 9: The image pyramid with different filter sizes and variances

miss alignment. We also discover that doing convolution in the frequency domain has 450 times computation efficiency as Table 2 shown.

5 Conclusion

In this homework, we realize that filtering operation in the frequency domain has computational efficiency. We also learn that it is impractical to evaluate similarity only with the intensity of each channel. It is important to exact representation of higher-level structure for comparing similarity. In conclusion, this experience will inspire us for image representation research in the future.

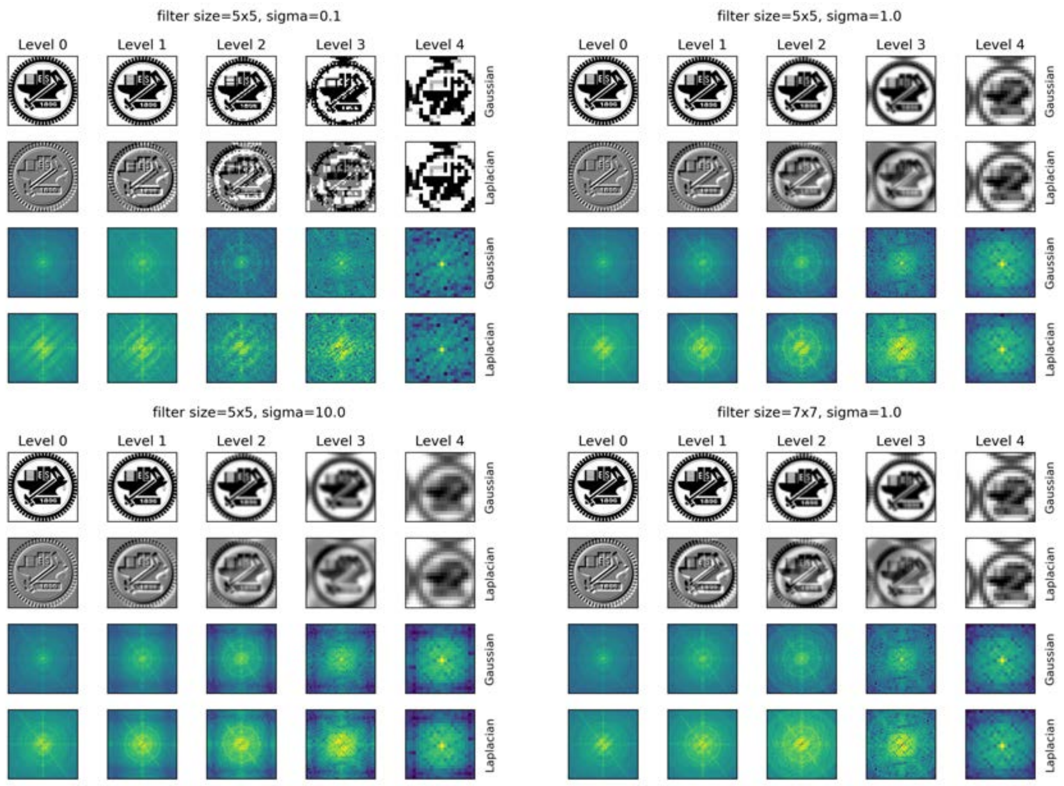


Figure 10: The image pyramid with different filter sizes and variances

References

[1] H. Kostas & E. Serafim N &M. Nikolaos and K. Aggelos K (1998) Hybrid image segmentation using watersheds and fast region merging *IEEE Transactions on image processing*

[2] Z. Nanrun &Z. Aidi & W. Jianhua & P. Dongju &Y. Yixian (2014) Novel hybrid image compression–encryption algorithm based on compressive sensing *Optik*

[3] A. Edward H &A. Charles H &B. James R &B. Peter J &O. Joan M (1984) Pyramid methods in image processing *RCA engineer*

[4] C. Hua-mei &V. Pramod K (2000) A pyramid approach for multimodality image registration based on mutual information *Proceedings of the third international conference on Information Fusion*

[5] P. Suporn (2005) Automatic Digicromatography: Colorizing the Images of the Russian Empire *Accessed May*

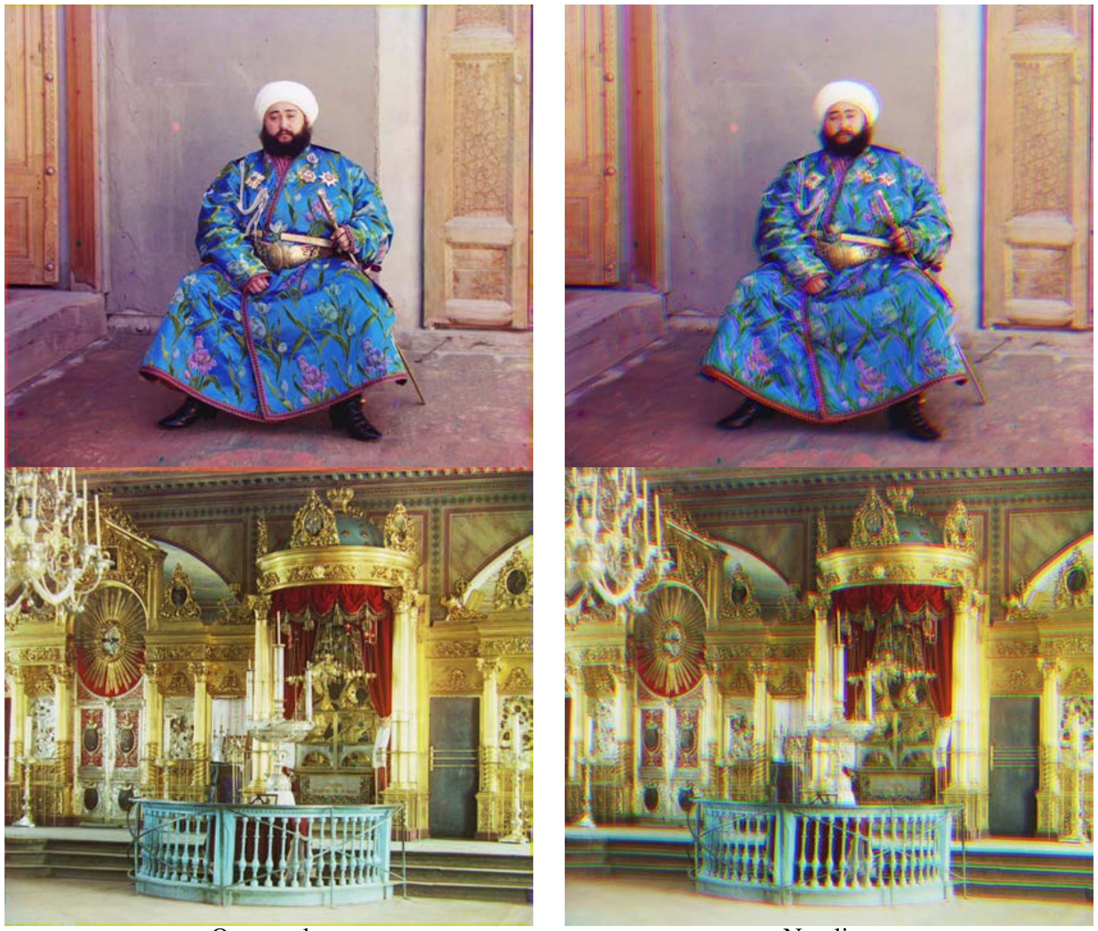


Figure 11: The colorization result with our method and alignment-free approach

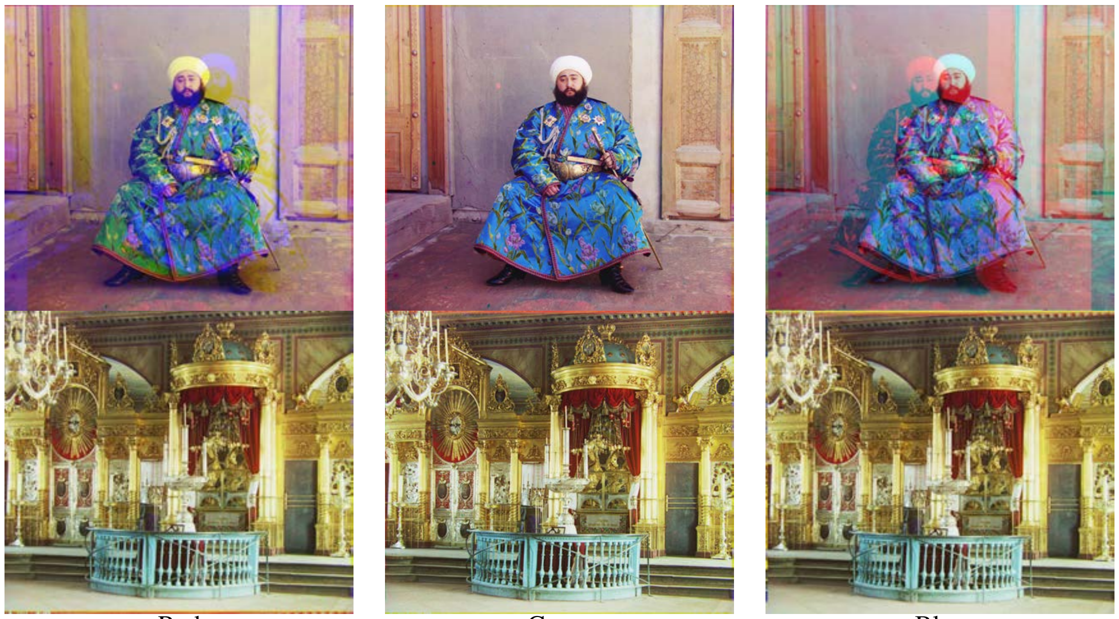
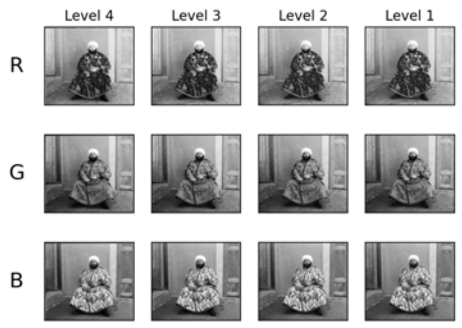
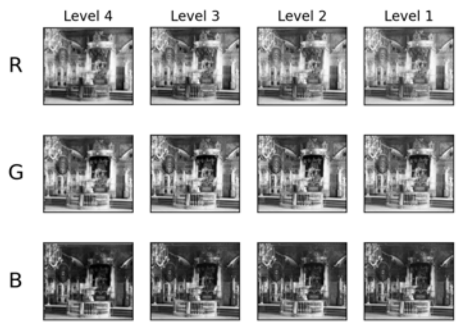


Figure 12: The colorization result with different base channels



Emir



Icon

Figure 13: The image pyramid of each channel. The "Emir" has a large intensity difference between different channels