CV HW2: Image Resolution

資科工碩　309551178　秦紫頤
電資學士班　610021　鄭伯俞
清華　H092505　翁紹育

In this assignment, we are dealing with different representations of the image. The assignment splits into three parts. The first part is the hybrid image, the second part is the image pyramid, and the third part is colorizing the Russian Empire.

Part 1 : Hybrid Image Synthesis

Introduction

Hybrid images are based on the multiscale processing of images by the human visual system and are motivated by masking studies in visual perception. These images can create compelling displays in which the image appears to change as the viewing distance changes. Hybrid images combine the low spatial frequencies of one picture with the high spatial frequencies of another picture, producing an image with an interpretation that changes with viewing distance. A sharp image of high spatial frequencies can be seen from a short distance but viewed from a distance; low frequencies picture can be seen.

Implementation Procedure

Get the low-frequency component.

1. Get the first image from the pair
2. Get the shape (width, height, channel) of the image
3. Convert the image object to a NumPy array
4. Do the following steps a channel at a time -> RGB image has three channels
5. applying Fourier transform
    - get the frequency domain: 2D fast Fourier transform
    - shifting the zero-frequency component to the center of the frequency domain
6. Get the Gaussian low pass filter H
    - get the center point -> (height/2,width/2)

- Gaussian low pass filter: $H(u,v) = e^{-D^2(u,v)/2D_0^2}$, $D(u,v) = \sqrt{(u^2+v^2)}$
- set D_0 to 5, which is tested as the best value to blend low and high frequency

7. Get the low-frequency components $F * H$, F is the image after Fourier transform
8. Inverse Fourier transform
    - shift the zero-frequency component to the upright corner of the frequency domain
    - 2D inverse fast Fourier transform
9. Get the real part of after inverse Fourier transform -> by getting the absolute value
10. Step 4-8 is doing a channel at a time -> append the result at step 8 to the corresponding channel output image
11. Convert the NumPy array back to the image object after getting all the channel

Get the high-frequency component

- Step1: Get the second image from the pair

Step 2 - Step 6 same as the last section

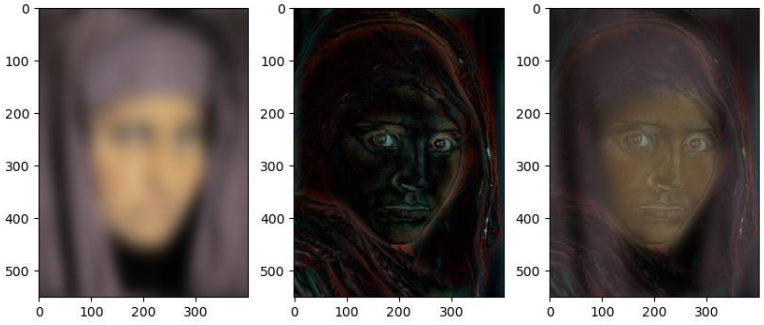
- Step7: For getting high pass filter -> 1-(Gaussian low pass filter)

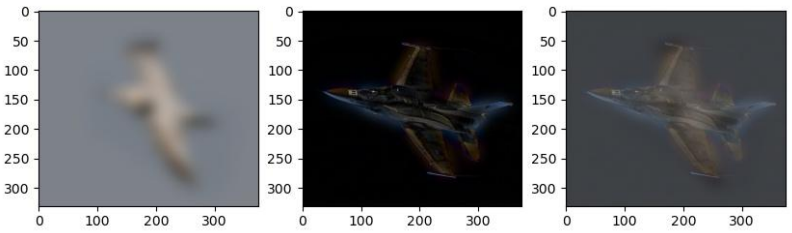
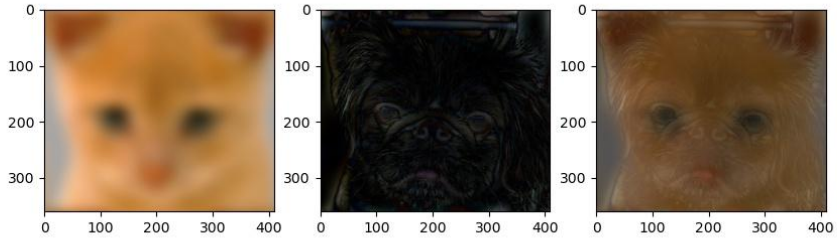
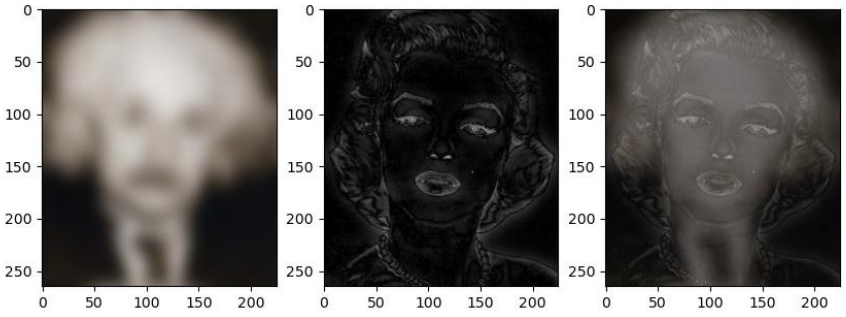
the rest is the same as Step 7 - Step 11 in the last section

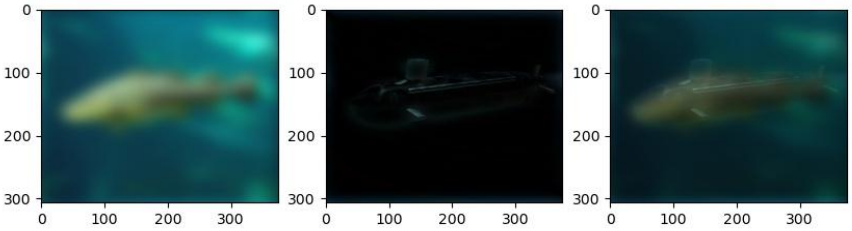
Blend the image

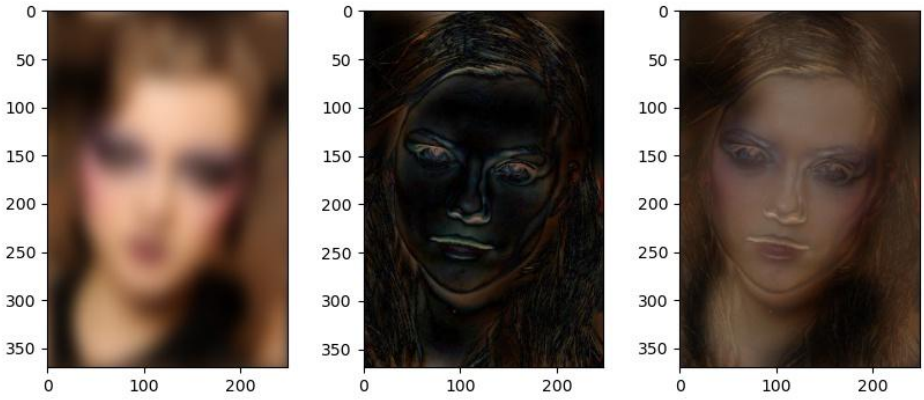
1. Resize the two images (low frequency, high frequency) to the same size
2. Blend them by `image.blend()`

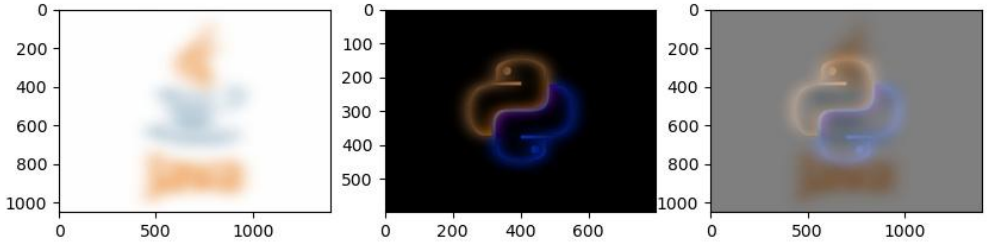
Experimental Results

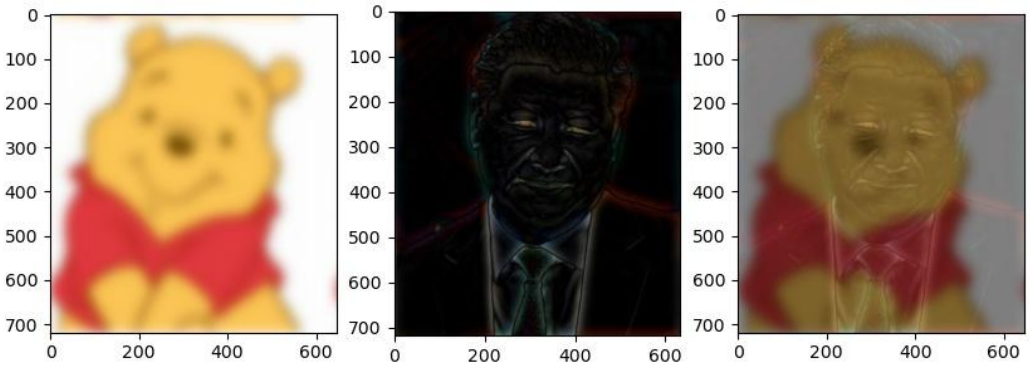










Discussion

We are dealing with color images with R, G, B 3 channels in every image in the hybrid image part. Our initial idea is to treat our data as a 3D cube and do the Fourier transform and applying the filter. At first, we test if Fourier transform will work in this setting or not. We do inverse Fourier transform right after Fourier transform and see if the image still looks the same. Luckily, it works fine. Then we do the Fourier transform, apply the filters, and do the inverse Fourier transform; the output is incorrect. However, if we deal with each channel respectively instead of 3 channels together, then the output is correct. We think the problem should be when applying filters. Nevertheless, in this step, we define a 3D cube filter same as the image and multiply the image (in frequency domain). How will this be wrong? Although we did not find the reason, at least the next time, we will know when dealing with a color image do one channel at a time.

Conclusion

In the hybrid image part, we learn how to deal with an image in the frequency domain. It will be simpler for some actions in the frequency domain than in the spatial domain (convolution in spatial domain, multiplication in frequency domain).

Part 2 : Image Pyramid

Introduction

Image pyramid is a technique that repeats smoothing and then subsampling. It is helpful in coarse-to-fine image tasks. In this task, we implement the Gaussian pyramid and Laplacian pyramid. We also show their corresponding magnitude spectrum for each image.

Implementation Procedure

1. Gaussian filter

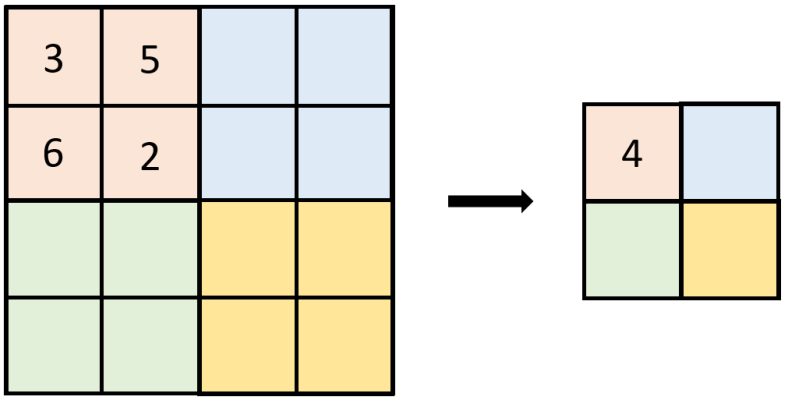
   For simplicity, we use 5x5 discrete approximation Gaussian kernel as our Gaussian filter. However, the total sum of the discrete kernel will not equal to one. Therefore, we normalize by dividing each element by sum of them.

$$\frac{1}{25}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

   Also, we pad the original image by half size of the kernel for successfully applying the Gaussian filter.

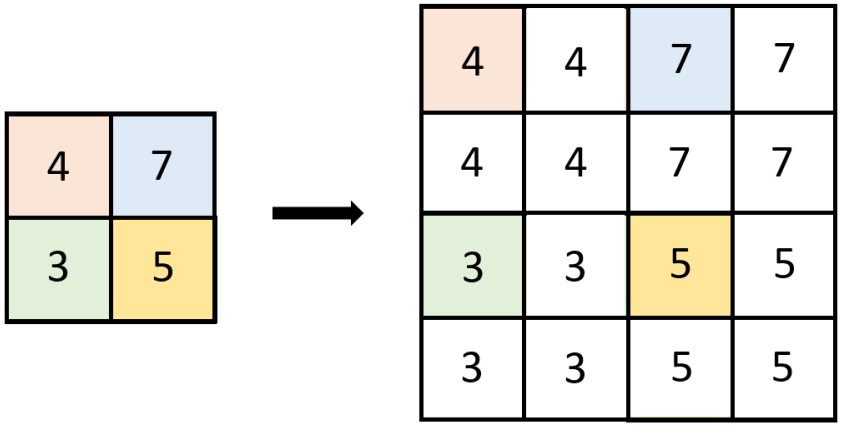
2. Subsampling

   The idea of subsampling is summing up the nearest four pixels together and then divide them by four to get the average.



3. Upsampling

For upsampling, resample by a factor of 2 with the nearest interpolation. We use a function of scipy - `scipy.ndimage.zoom` to finish this step.



4. Magnitude spectrum

   We use `np.fft.fft2` to derive the result of the Fourier transform. Furthermore, shift the zero frequency to the center by `np.fft.fftshift`.

5. Laplacian pyramid

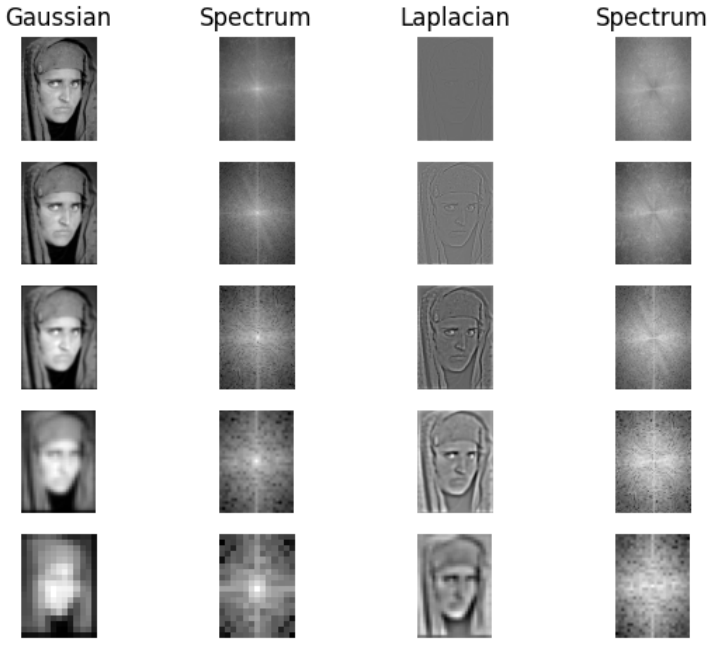
   In this step, we subtract the image after the Gaussian filter and downsampling from the original image to get the Laplacian pyramid.

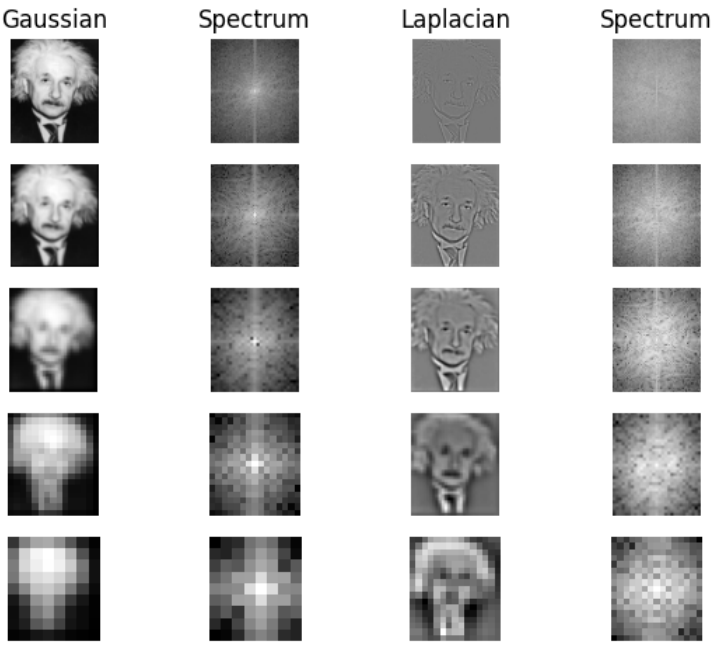
- Processing Pipeline

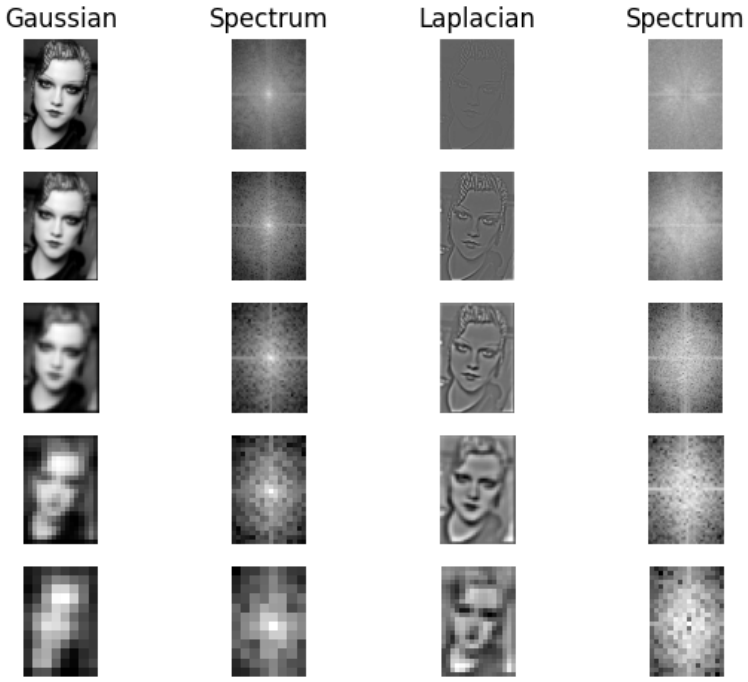
Gaussian pyramid → subsampling → upsampling → Laplacian pyramid

Experimental Results

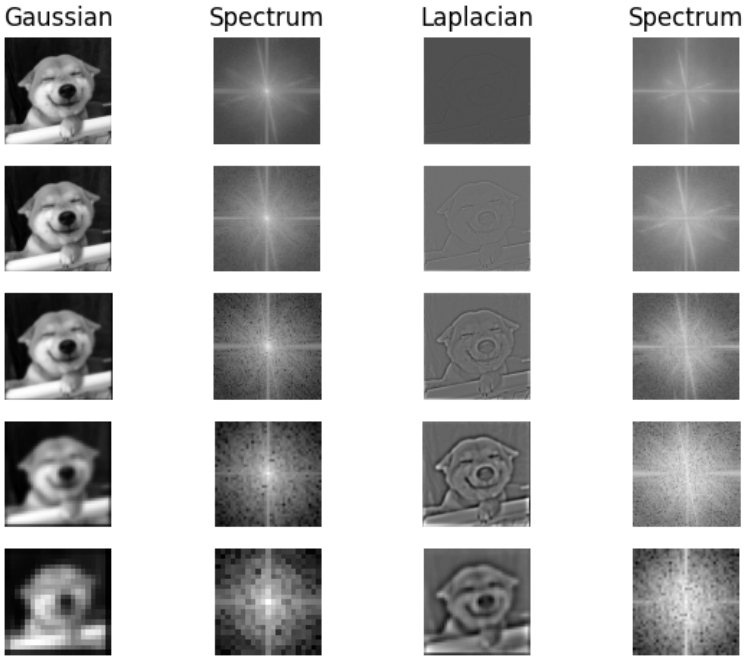
- TA's data

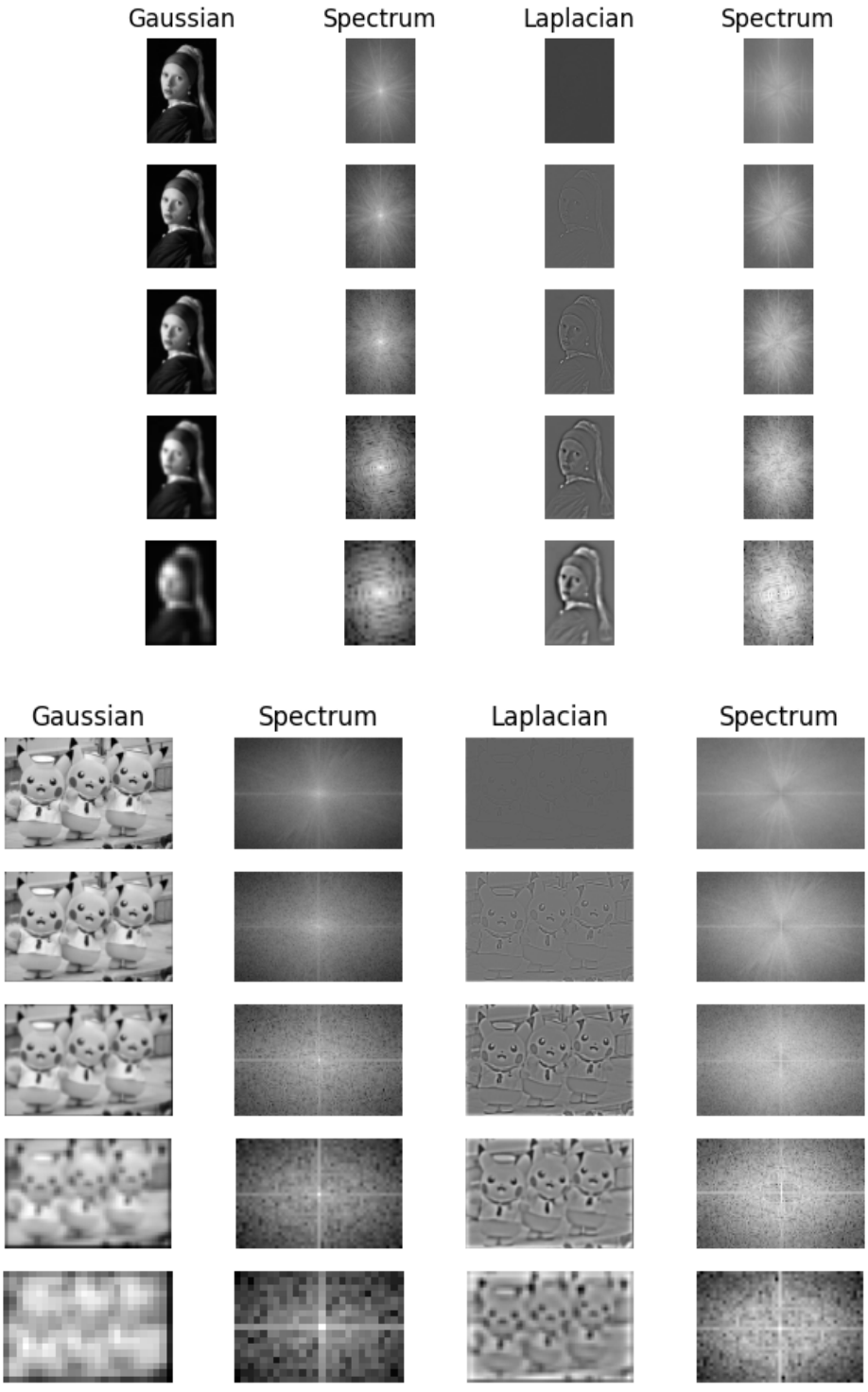

Gaussian
Spectrum
Laplacian
Spectrum


Gaussian
Spectrum
Laplacian
Spectrum


Gaussian
Spectrum
Laplacian
Spectrum

- Our data


Gaussian
Spectrum
Laplacian
Spectrum



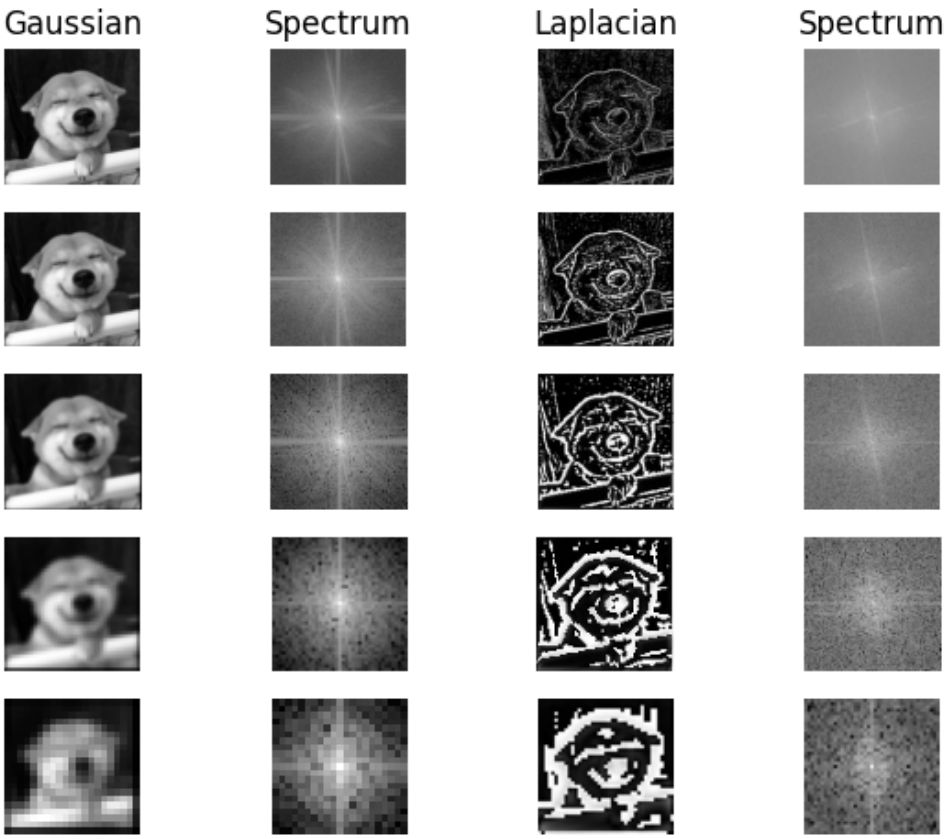
Discussion

In the image pyramid part, we encounter the error message after subsampling, which tells us that the value range is wrong. We found out that some of the values may not be integers after taking the average during debugging. Therefore, we apply `astype(np.uint8)` to the result. Meanwhile, we also add

`astype(np.uint8)` after upsampling. Furthermore, the result shows below.

`np.subtract(old,gaussian_filter(upsample(new)).astype(np.uint8)[0:old.shape[0],0:old.shape[1]])`

$$\begin{bmatrix} 252 & 255 & 7 & \dots & 55 & 163 & 191 \\ 254 & 251 & 250 & \dots & 11 & 123 & 163 \\ 254 & 248 & 242 & \dots & 246 & 104 & 149 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ 80 & 48 & 19 & \dots & 189 & 154 & 169 \\ 86 & 64 & 41 & \dots & 200 & 167 & 180 \\ 116 & 92 & 80 & \dots & 227 & 194 & 200 \end{bmatrix}$$



We can observe that the Laplacian pyramid is with a black background. We are not quite sure if this be a proper presentation or not. Hence, we change the code as below.

`np.subtract(old,`
`gaussian_filter(upsample(new))[0:old.shape[0],0:old.shape[1]])`

$$\begin{bmatrix} -4 & -1 & 7 & \dots & 55 & 163 & 191 \\ -2 & -5 & -6 & \dots & 11 & 123 & 163 \\ -2 & -2 & -14 & \dots & -10 & 104 & 149 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ 80 & 48 & 19 & \dots & -67 & 154 & 169 \\ 86 & 64 & 41 & \dots & -56 & 167 & 180 \\ 116 & 92 & 80 & \dots & -29 & 194 & 200 \end{bmatrix}$$

It is clear that after subtracting, there will be some negative value. Compared to the result above, we think this one is much closer to the definition of the Laplacian pyramid. Therefore, this is our final version. Also, padding is a proper technique to ensure every element in the image has gone through the convolution process.

Conclusion

We implement image pyramid in this assignment. It shows that the Gaussian pyramid is a low pass filter; each time, it will make the image blurrier. In contrast, the Laplacian pyramid is a high pass filter; the shape in the image is preserved.

Part 3: Colorizing the Russian Empire

Problem Context

Before the 20th century, color photography had not yet become widespread - developments in the field were still rudimentary, at best Sergei Mikhailovich Prokudin-Gorskii (1863-1944), a Russian photographer, was the one responsible for spearheading work in color photography with his implementation of the Three-color principle. In this homework, we will use image processing techniques to automatically colorize the glass plate images taken by Prokudin-Goskii. In each image, a special camera is used to record the scene with three exposures: a red, a green and a blue filter. The process of colorization is simple. We extract the three color channel images, lay them on top of each other, and align them so they form a single RGB color image.

Introduction

- In this section, we need to write a script that **automatically** produce color images from the digitized Prokudin- Gorskii glass plate images
- We are asked to assume that **only x, y translation are needed** to align different color channels of the images, which turns out to be insufficient for aligning large images (discussed later).
- We implement a two-stage alignment process based on the Canny edge detector, our method is highly efficient because we only correlate between binary edge images.

- For large images of size (MxN), edge detecting and correlating becomes computationally expensive. We avoid this problem by first resizing the input images to (300x300), we then find x, y translation for aligning these smaller images. Finally, to align the large images of original size, we scaled the previous translation by the resizing ratio

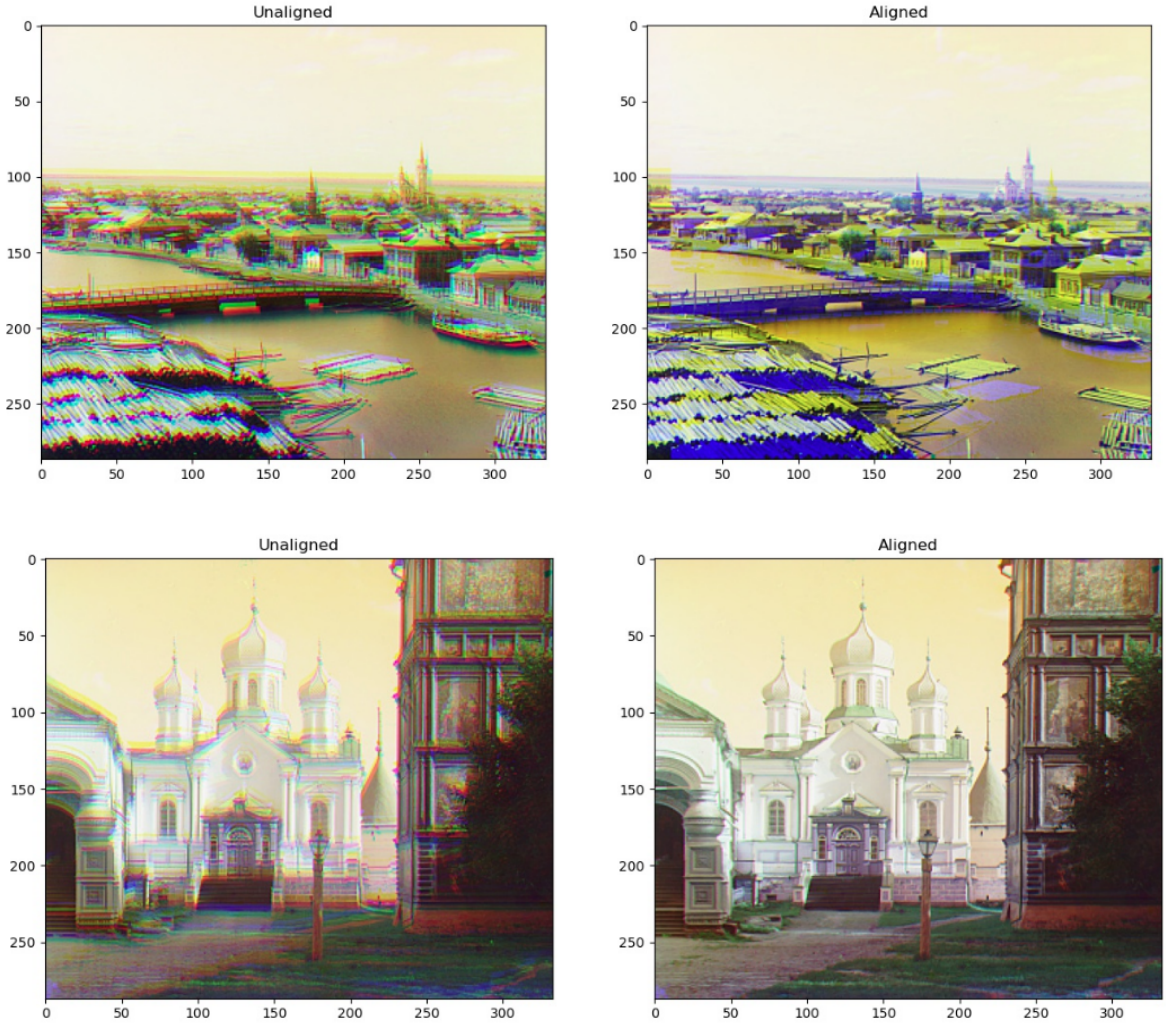
$$X_{original} = x_{resized} \times \lfloor \frac{M}{300} \rfloor$$
$$Y_{original} = y_{resized} \times \lfloor \frac{N}{300} \rfloor$$

- While the resizing mechanism speeds things up, it fails to align the original images accurately due to the rounding effect of scaling. To solve this, we crop a small region of the roughly aligned real-size image, and **do the alignment process on the small region again**. The resulting X, Y translation of the small region is then applied to the whole image to yield results of higher quality.

Naive Approach

We describe here the naive approach to this problem.
First define the shifting range of the image, the value I set is [-20,20]. Then we search for every possibility of shifting permutations in the horizontal and vertical directions. To measure the goodness of the alignment, we use the sum of square differences to get the distance of B/R channel and G channel. We set the best shifting offset to the one that has the minimum distance. Then we merge the three channels after alignment to the color image for output. This method works best for the small images. If the image is too large, this method is inefficient and sometimes not correct.

We think this method isn't the best one to align color channels because the method only uses SSD to measure the alignment which is not enough. But this method is efficient in small images and the result isn't perfect. Some of the channels output images don't align that well, we think this is because the method is too simple. We didn't consider details like edges or corners.
Result of Naive approach is shown below :

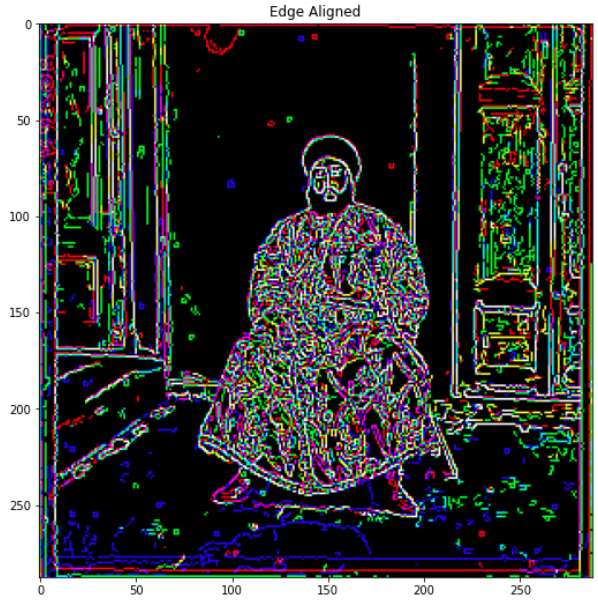


Implementation Procedure

We only briefly show the implementation idea here, for more detailed explanation, please refers to ***"HW2-3.ipynb"***

1. First we resize the image to (300x300), then apply the Canny Edge algorithm on each channel.

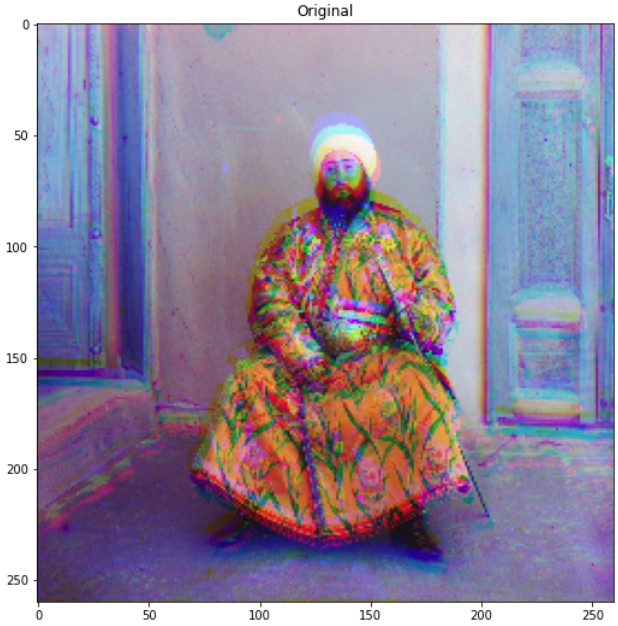
2. We correlate the binary edge images with `bitwise_and` and align the 3 color channels using the correlation result.



Left : Aligned edge image. The white lines correspond to edges that are presented in all 3 color channels.

Down Left : Resized image before alignment.

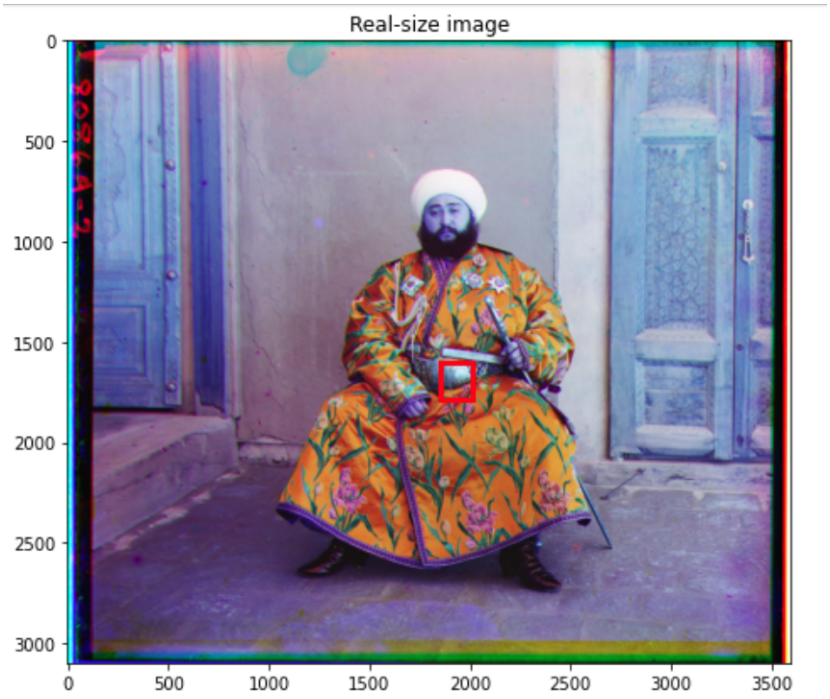
Down Right : Result after aligning the resized image using the x, y translation offset of the edge alignment.

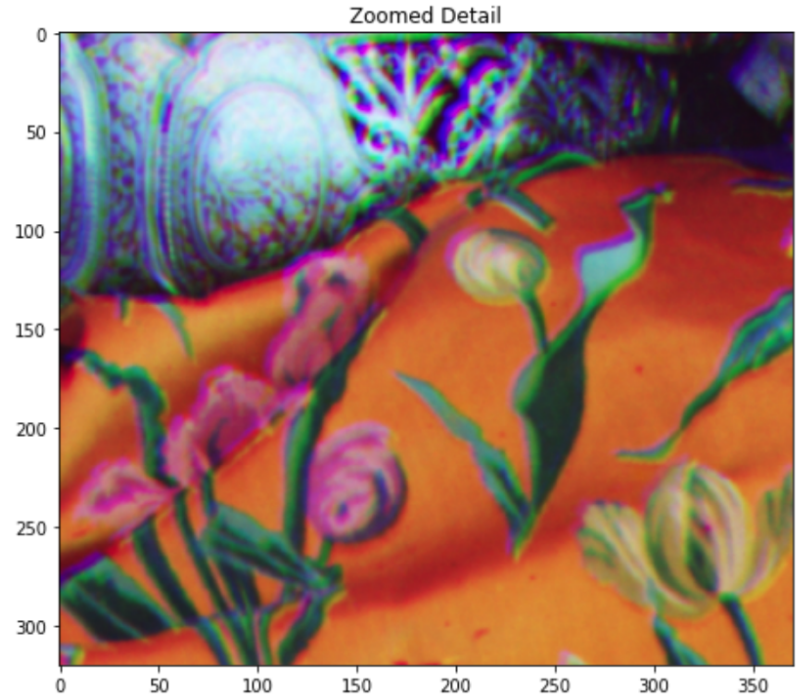


3. Now that we have done the alignment process on a small (300x300) image, we need to apply the result to the real-size image. A naive approach is to simply scale the translation offsets to fit the original image size as follows.

$$X_{original} = x_{resized} \times \lfloor \frac{M}{300} \rfloor$$
$$Y_{original} = y_{resized} \times \lfloor \frac{N}{300} \rfloor$$

The result looks nice at first glance, but if we zoom in the image (we show the zoomed version of the red rect region below), we will find tiny defects because of the error introduced by scaling .





4. To further enhance the aligning accuracy of a real-size image, we crop a small rectangle region from the image and repeat the above aligning steps on that region. (ie. find Canny Edge -> find x,y translation for

aligning the edges), we then apply the offsets of that region to the whole real-sized image. The assumption behind this approach is that the translation needed for alignment is the same for all regions of an image, in the discussion section, we will see that this assumption is not entirely true.

The key step of this approach is to choose the **"right region"** from the original image (ex. We don't want to choose regions containing only the sky). For edge alignment to work, we want to choose a region that is full of matching edges.

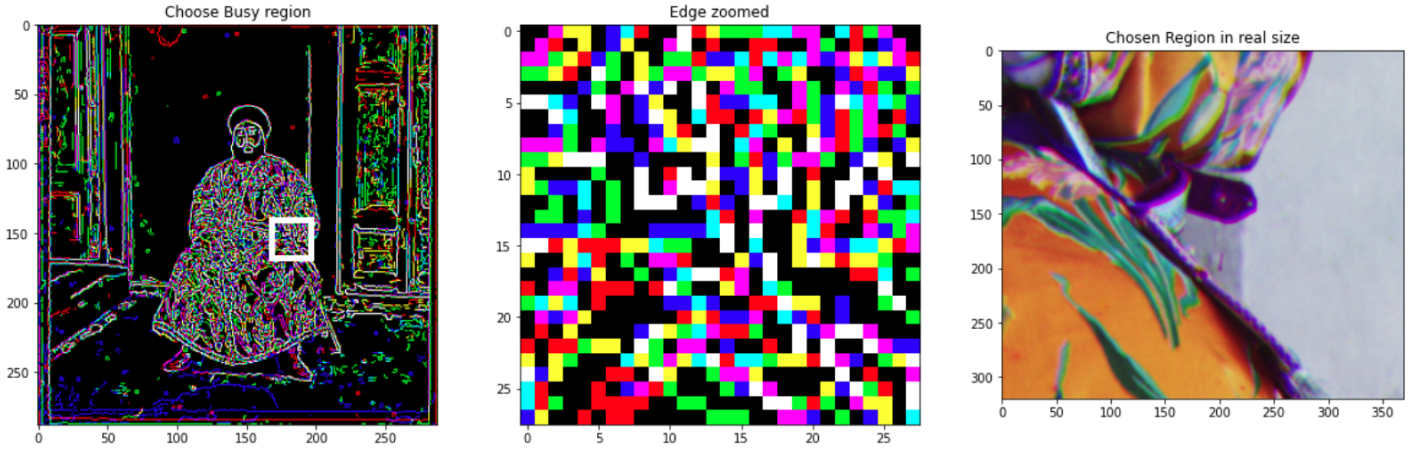
To find out good regions, we use the information of the previously generated 300x300 edge image as a guidance. We divide the real-size image into a 10x10 square grid and choose the one that has the highest **"edge_score"**. The definition of edge_score is as follows:

$$Edge\_Score = \Sigma_{all\_pixels}[\texttt{bitwise-and(re, ge, be)} + \texttt{re} + \texttt{ge} + \texttt{be}]$$

where `re, ge, be` represent the binary edge image of each color channel.

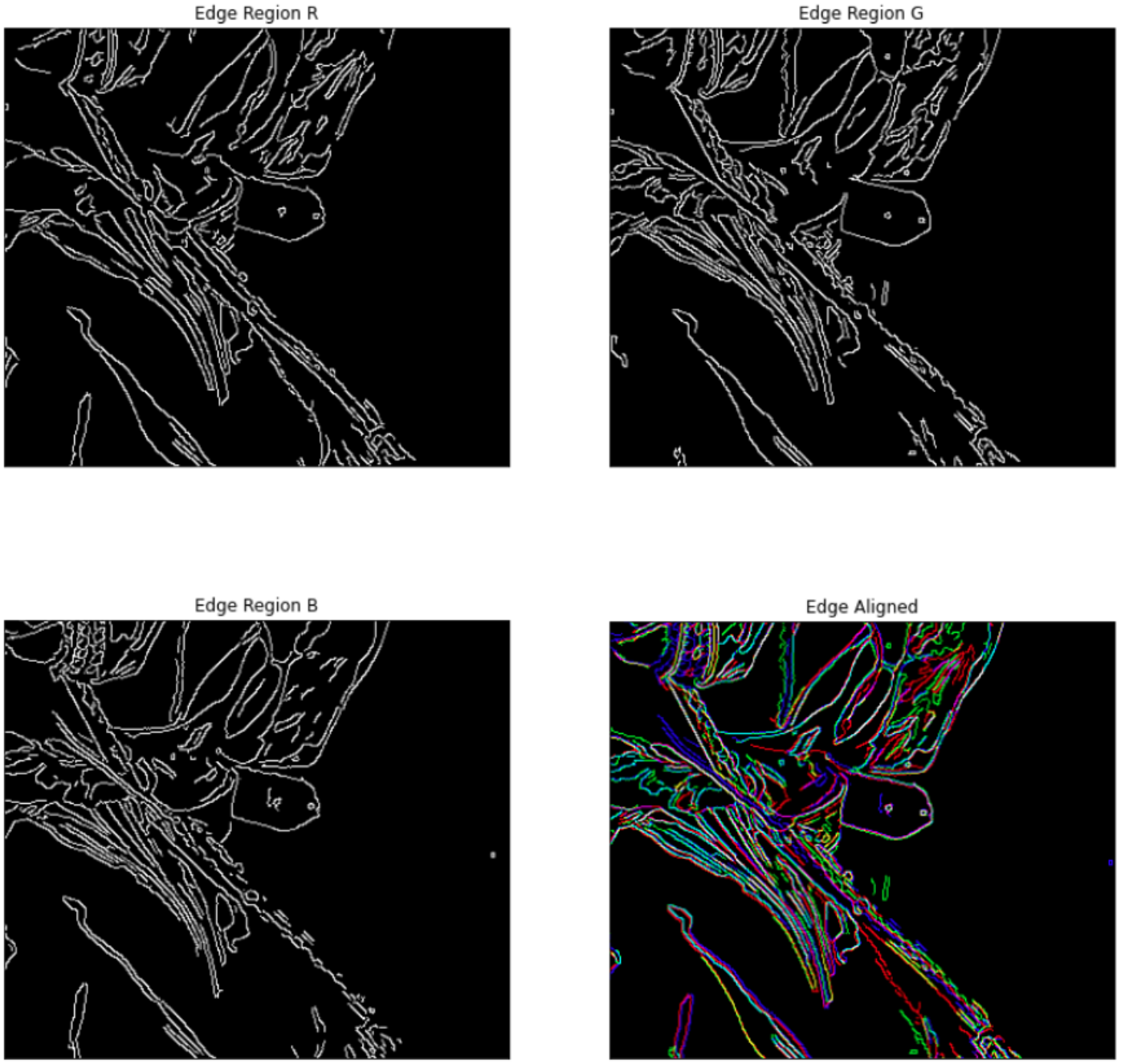
Here is the region chosen in the example.

```
busy region: (5, 6)
```



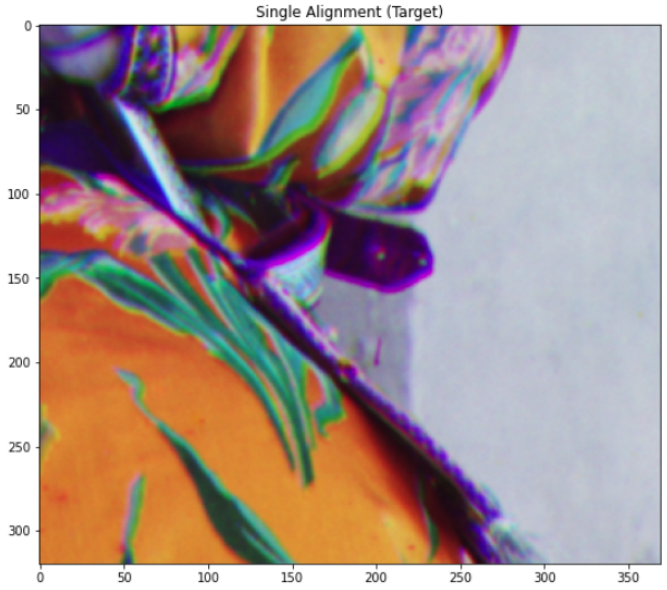
We then crop the resize image and do edge alignment in the chosen region:

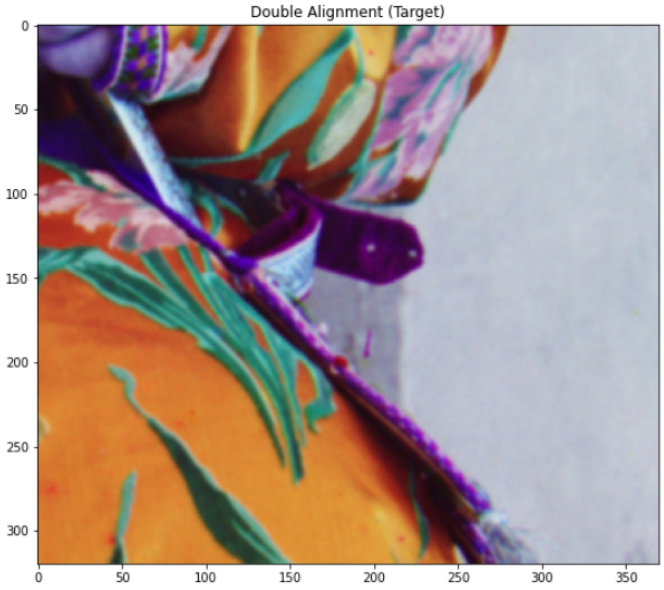
```
offset gr=(-3, 0), offset gb = (-4, -4)
```

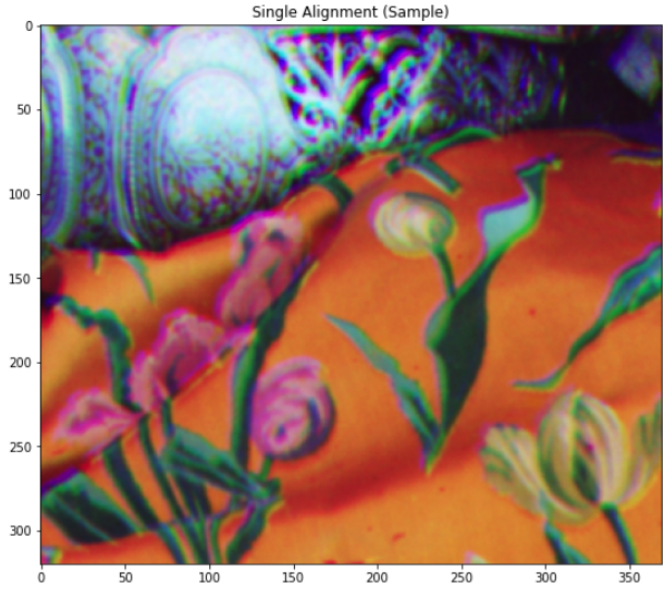


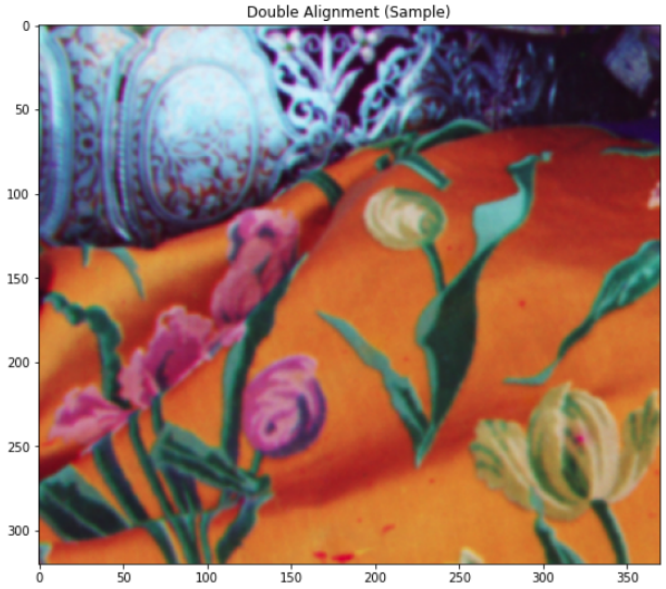
Note that because the image has already been aligned one time by the scaled translation of (300x300) image, we only need to search a small range for possible offsets.

The effect of second alignment is significant as shown below:







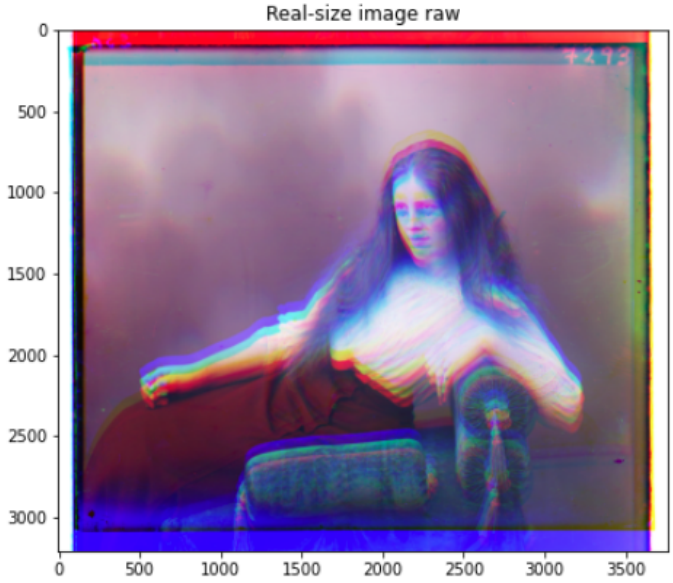


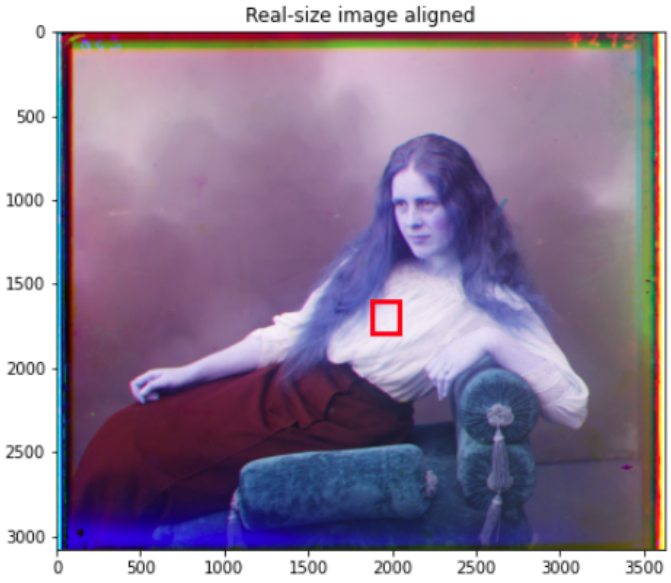
Experimental Results:

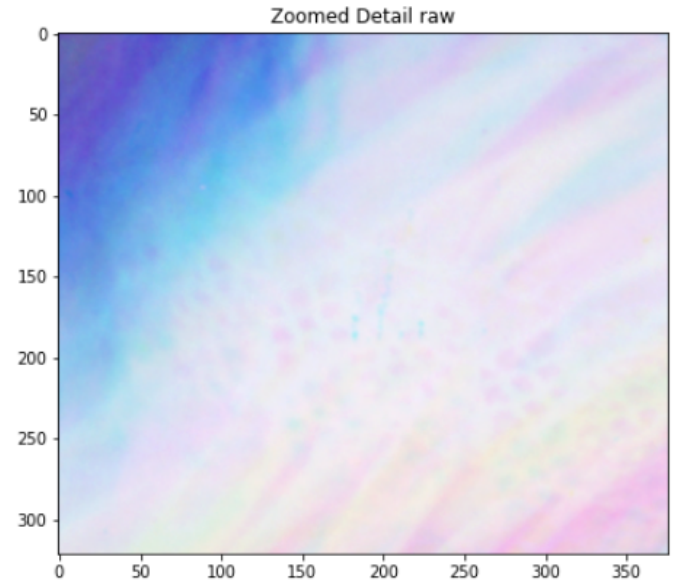
If we only save the resulting image in file (without plotting intermediate results), our method costs about 1 second to process each image (on my laptop).

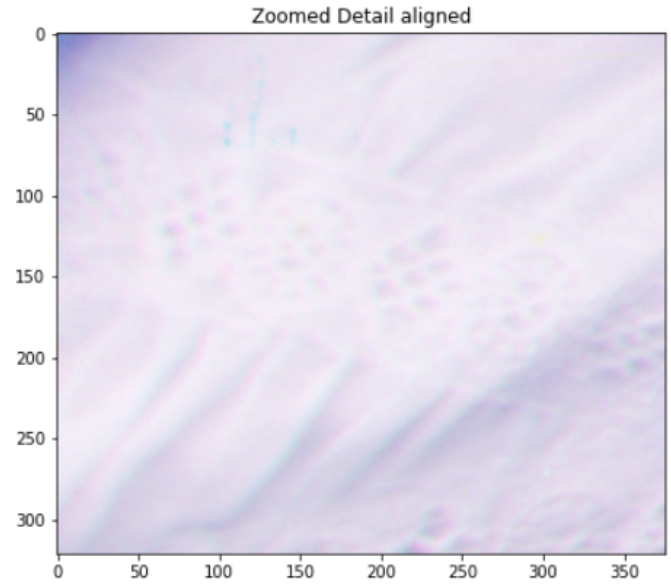
`lady.tif :`

First Alignment :

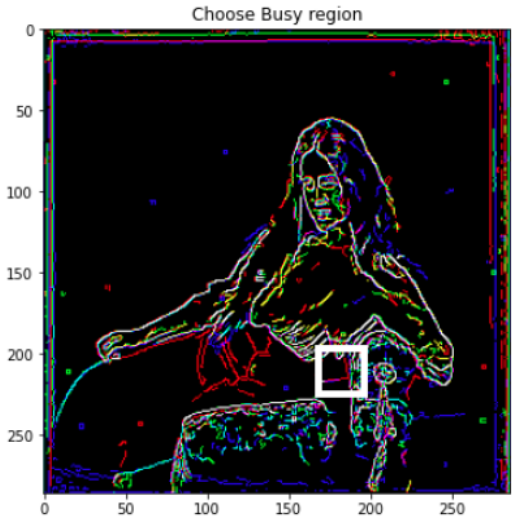


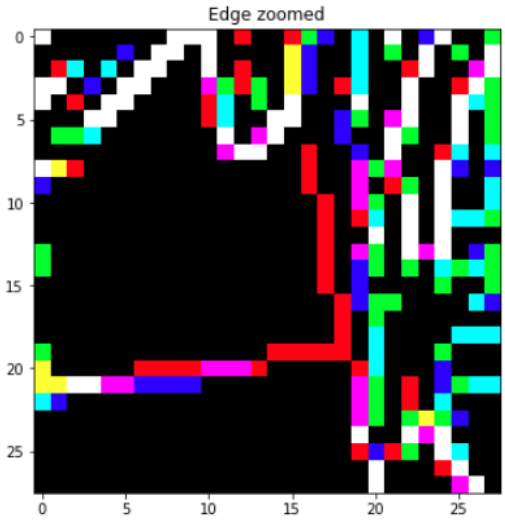




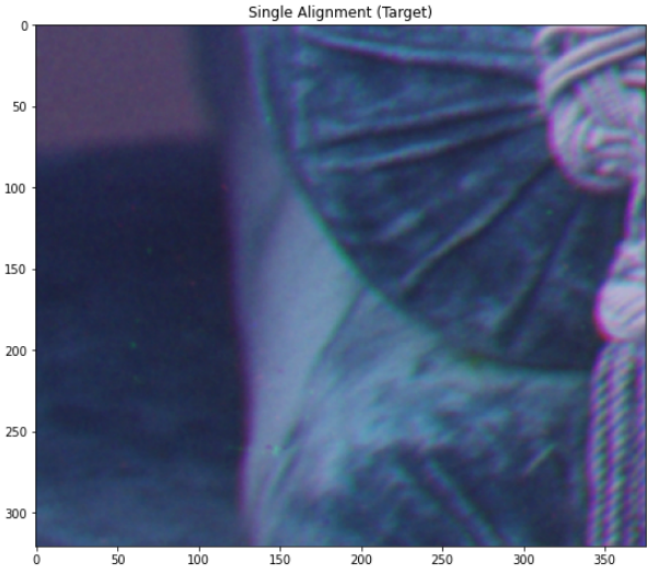


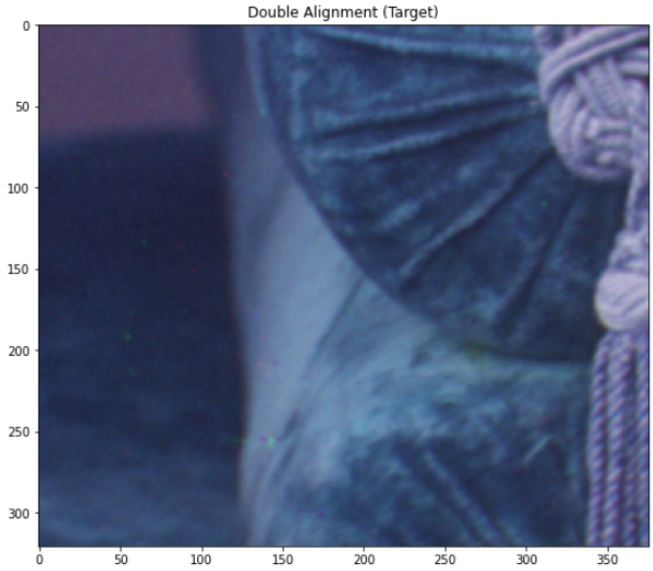
Choose edge-rich region

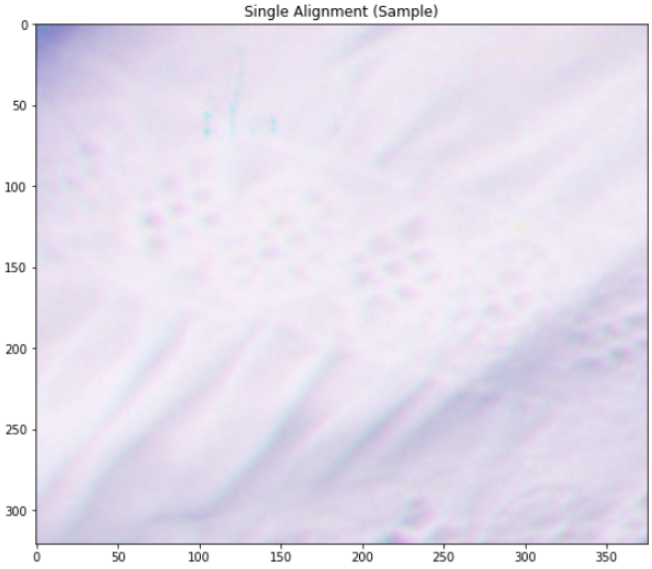

Choose Busy region
0
50
100
150
200
250
0
50
100
150
200
250

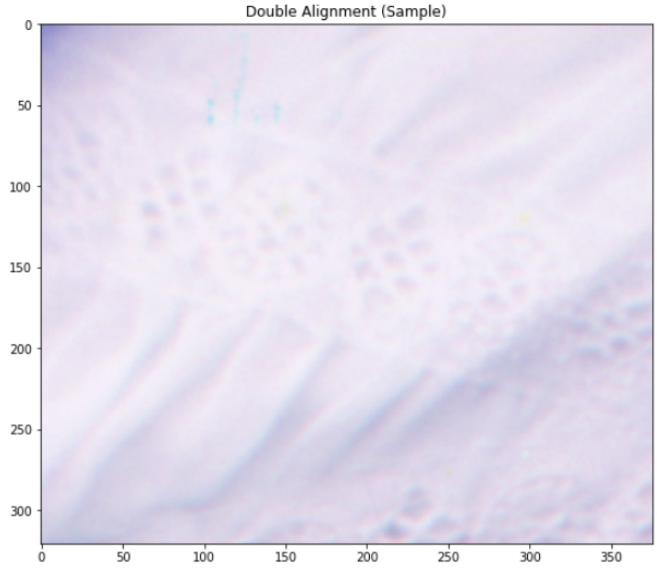

Edge zoomed
0
5
10
15
20
25
0
5
10
15
20
25

Double alignment result:



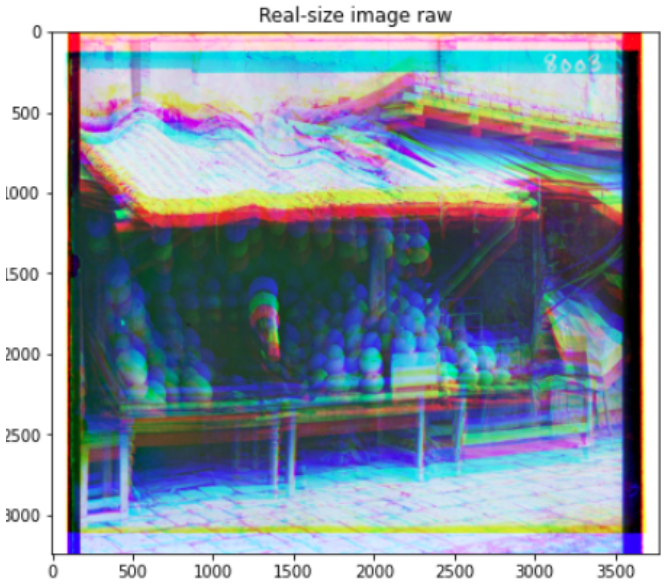


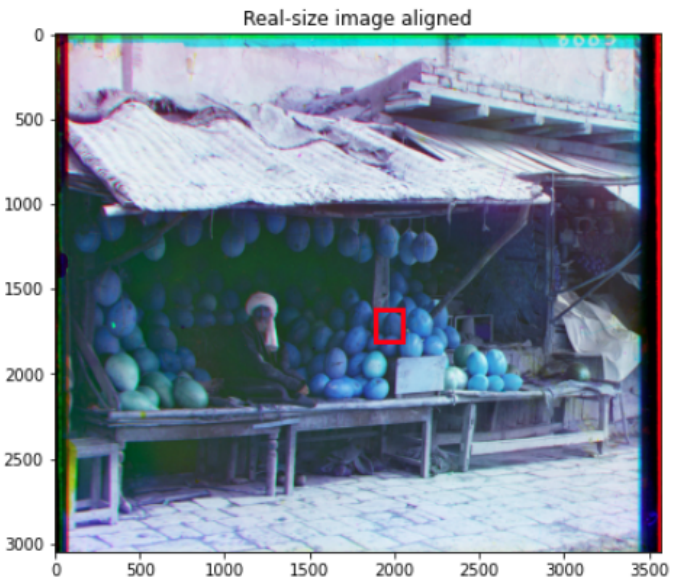


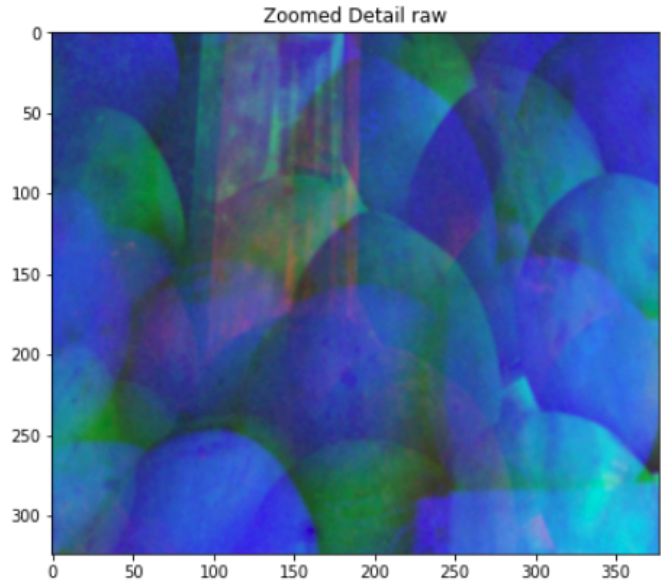


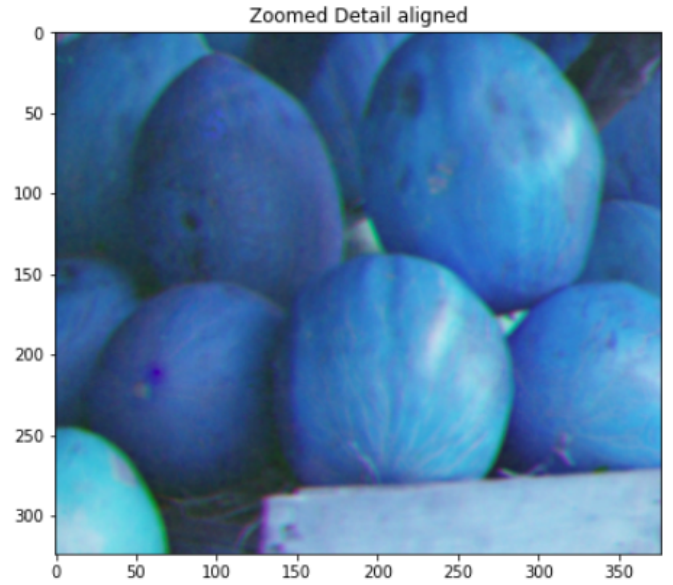
mellon.tif

First Alignment :

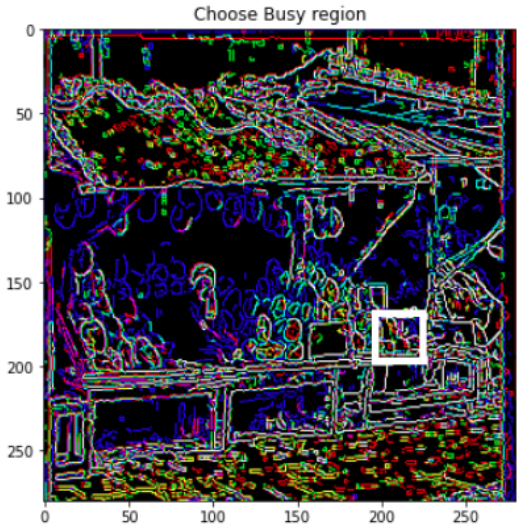


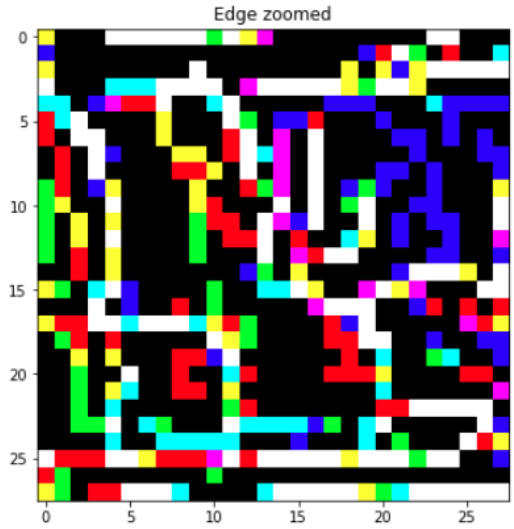




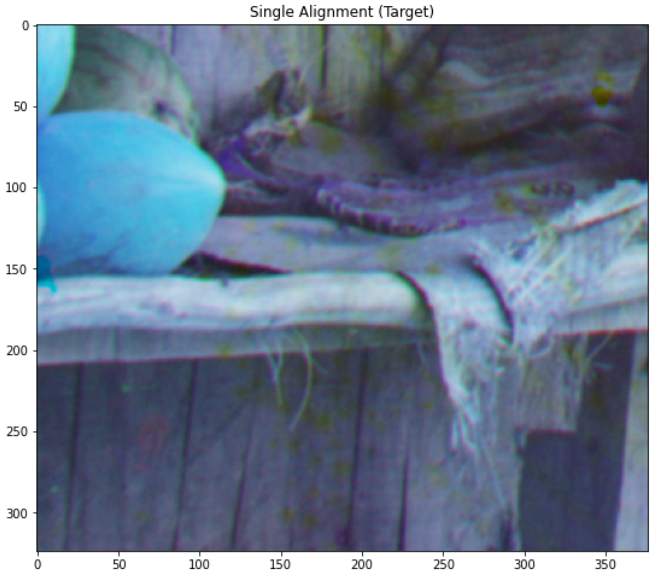


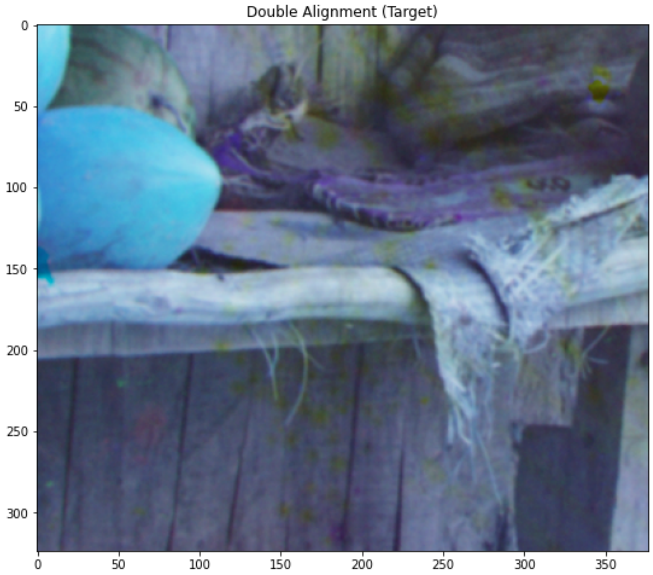
Choose edge-rich region

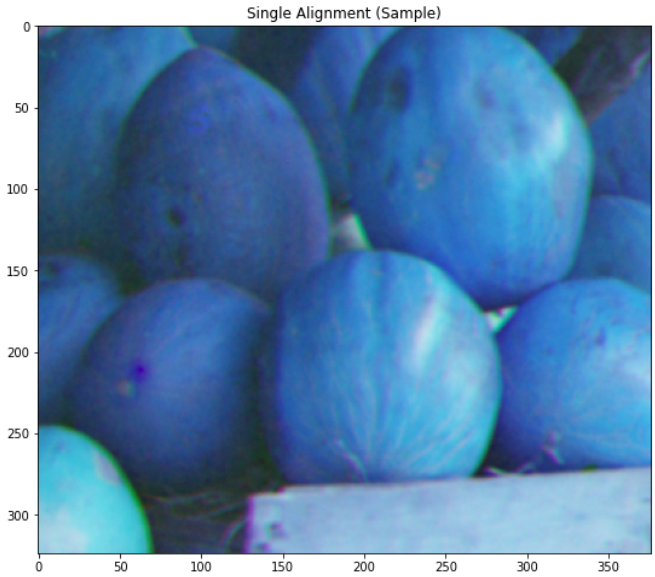

Choose Busy region
0
50
100
150
200
250
0
50
100
150
200
250

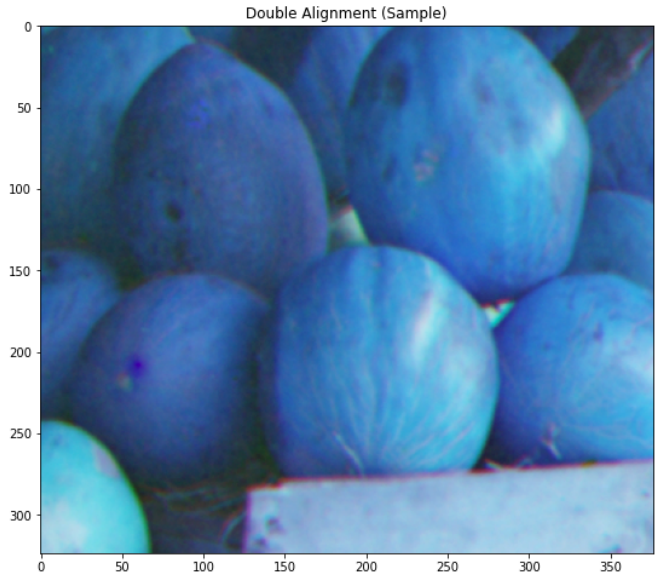

Edge zoomed
0
5
10
15
20
25
0
5
10
15
20
25

Double alignment result:



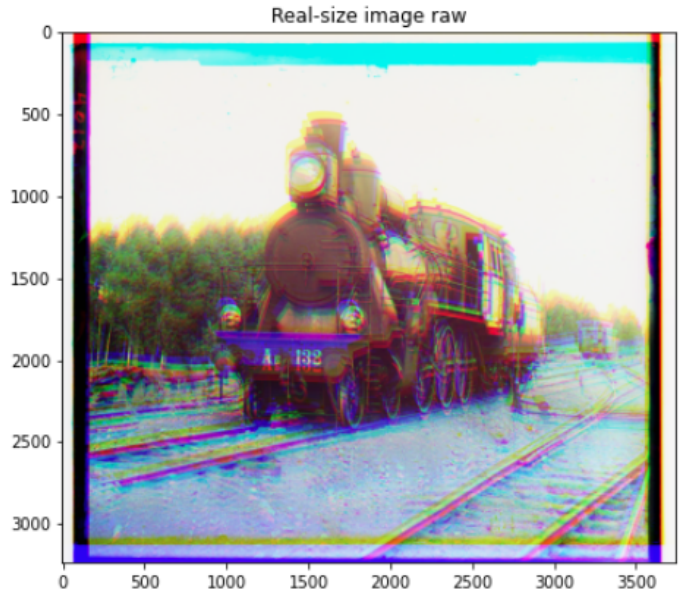


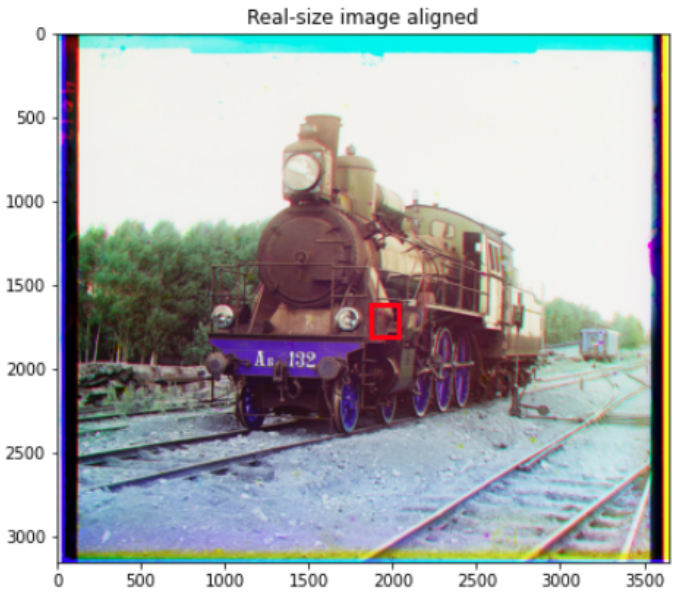


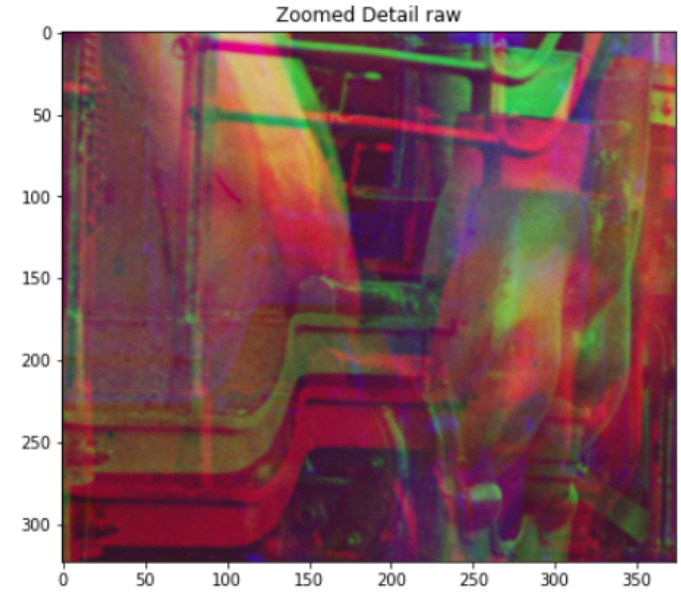


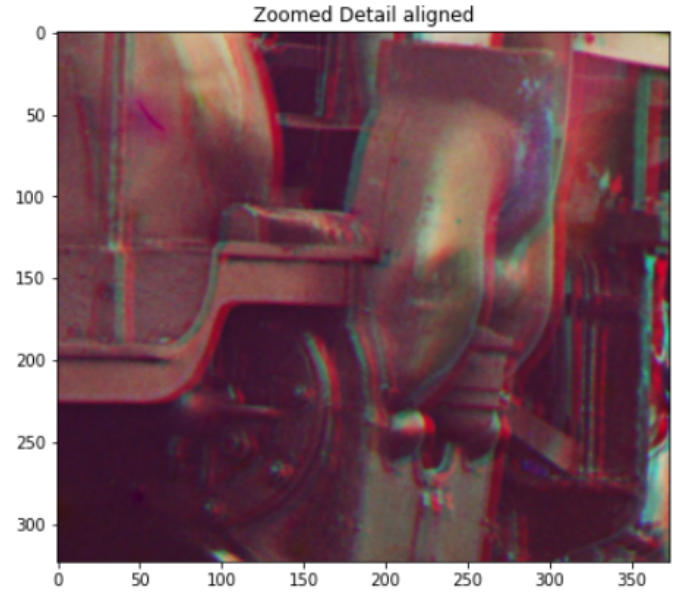
train.tif

First Alignment :

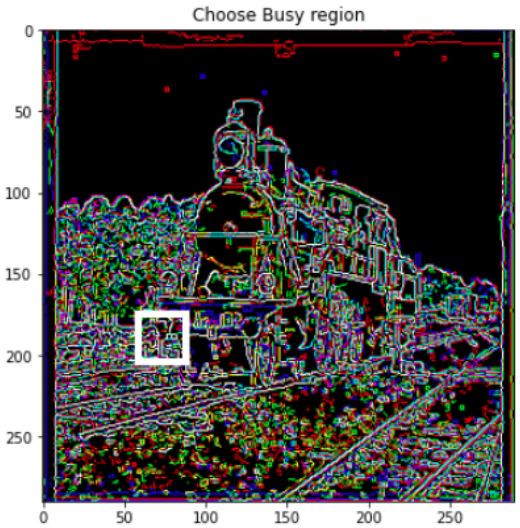


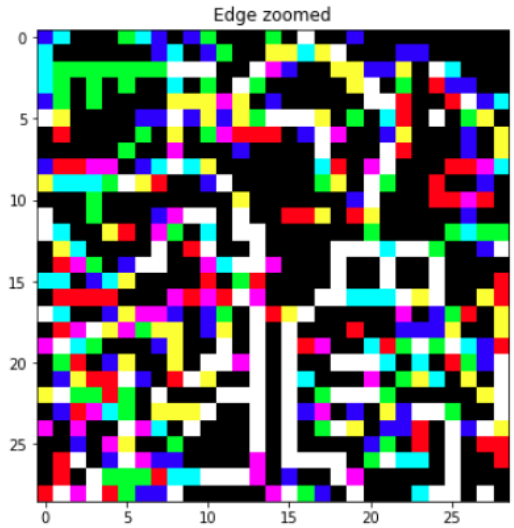




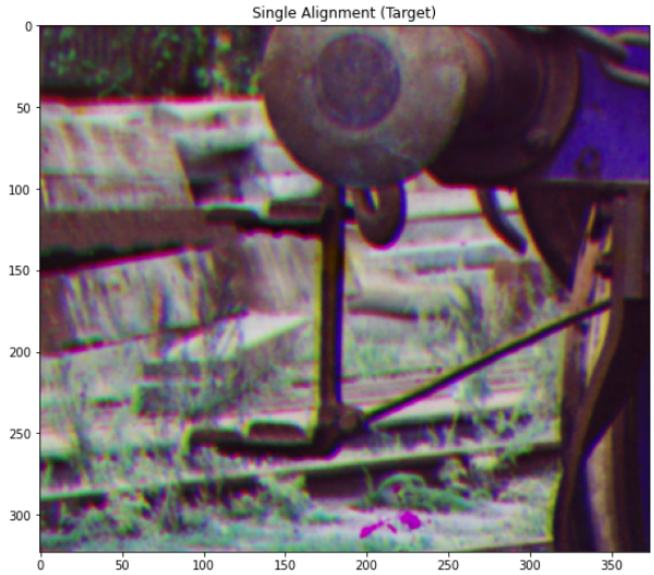


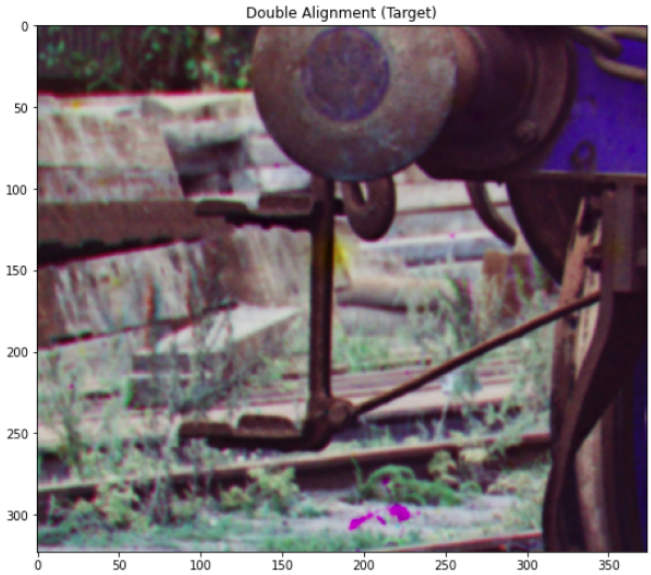
Choose edge-rich region

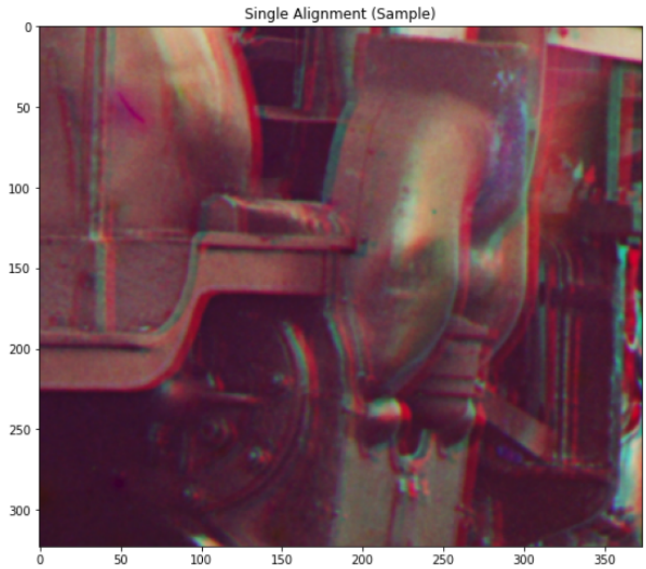


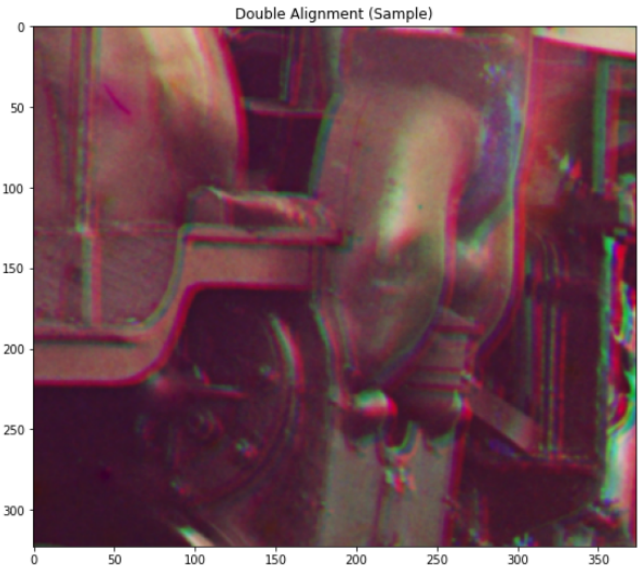


Double alignment result:



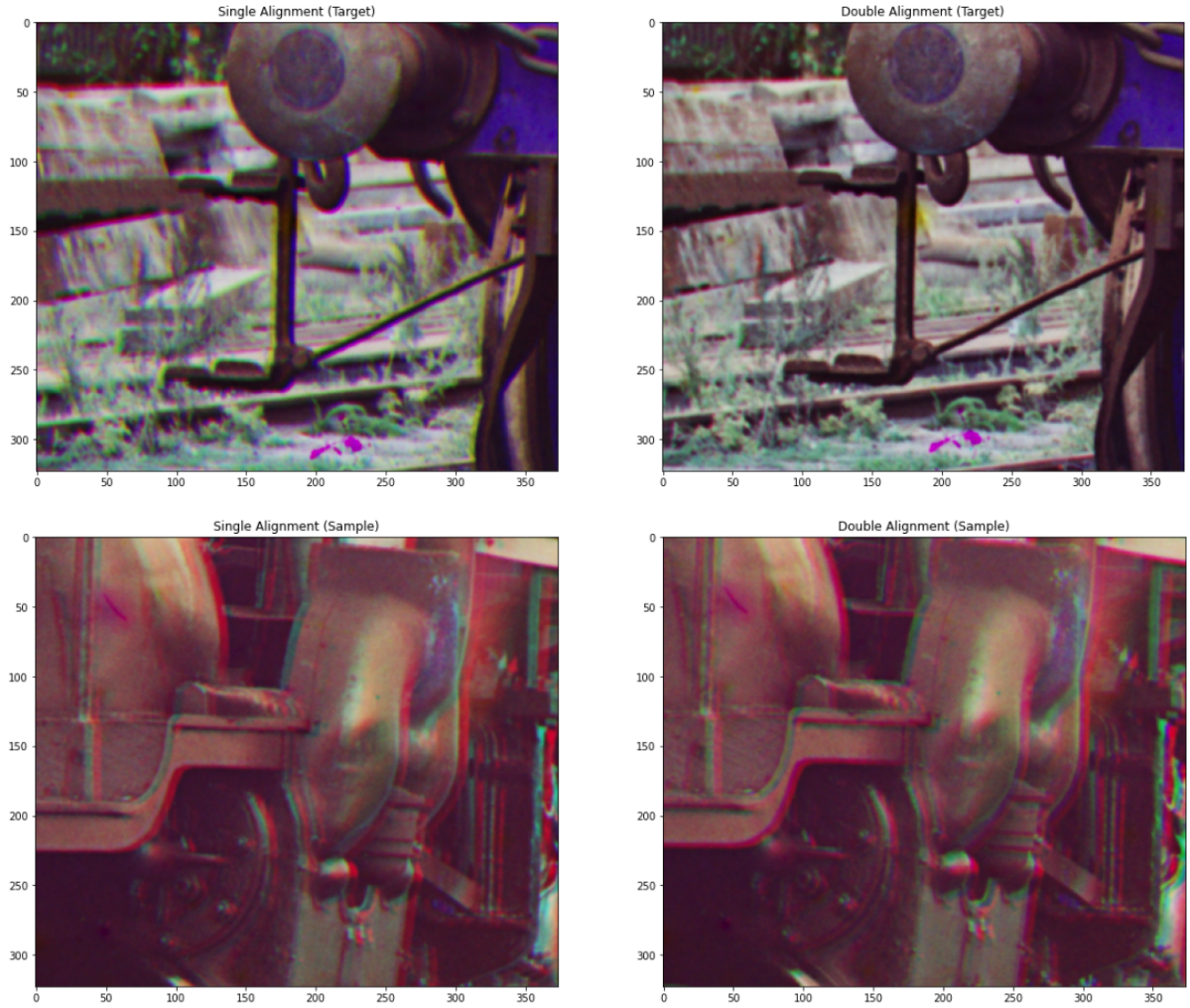






Discussion

In the result of train.tif, we see that although we have aligned the target region perfectly, the color channels of the sample region remain unmatched. We conclude that it is not possible to align “train.tif” perfectly with merely translation. The channels are not only shifted, but also scaled or rotated relatively to each other.



A possible solution may be to search for rotation and scaling in the alignment process, extend our matching model from translations to affine transformations on 2D planes. Another simpler solution would be to align each small region of real-size image separately, using different translation offsets, the trade off is that it takes more computing time and may cause discontinuity at the border of each aligning region.

Conclusion

In Part3, we develop an efficient and accurate alignment algorithm that automatically generates colored pictures from the input channel images. For large images, our result is satisfying at our chosen target region, but we cannot guarantee the quality of result at other regions of the same images because the input image may be rotated or distorted, which makes it difficult to align using merely translation.

Work Assignment Plan

秦紫頤: Hybrid Image, Colorizing the Russian Empire (Naive Approach), Report
鄭伯俞: Colorizing the Russian Empire, Report
翁紹育: Image Pyramid, Report