Group 27

Computer Vision – HW2

Task 1: Hybrid image

Introduction

A hybrid image is the sum of a low-pass filtered version of the one image and a high-pass filtered version of a second image.

Procedure

1. Let each input image be twice larger. Other pixels are filled with 0.
2. Multiply each input image by $(-1)^{x+y}$ to center the transform.
3. Compute Fourier transformation of input image(F(u,v))
4. Compute the filter function H(u,v)

Ideal low pass filter:

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

Gaussian low pass filter:

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

Ideal high pass filter:

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

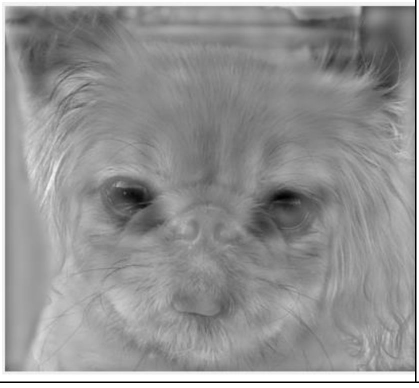
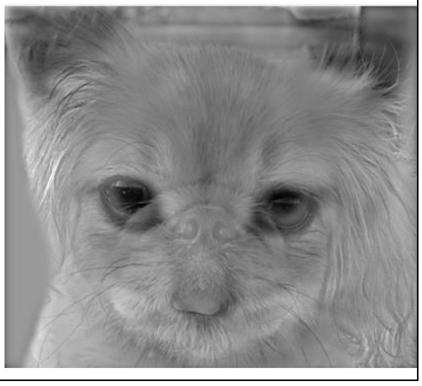
Gaussian high pass filter:

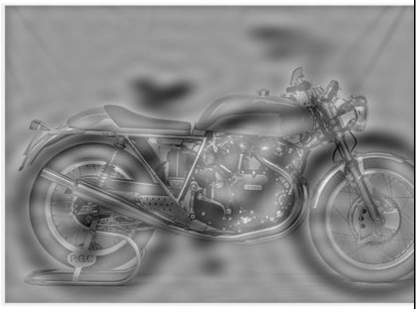
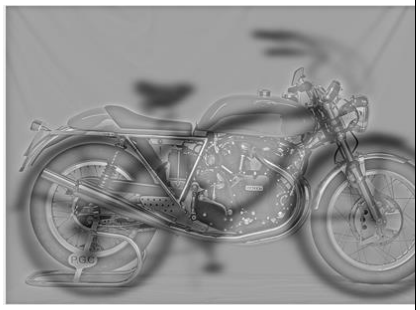
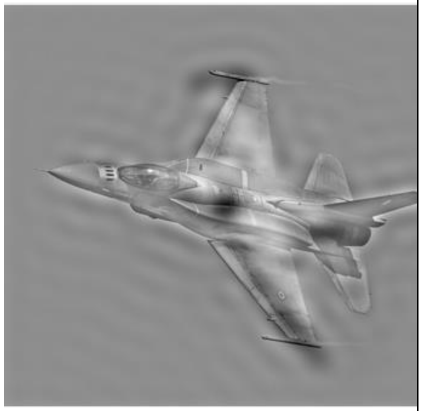
$$\mathrm{H(u,v)} = 1 - e^{\frac{-D^2(u,v)}{2D_0^2}}$$

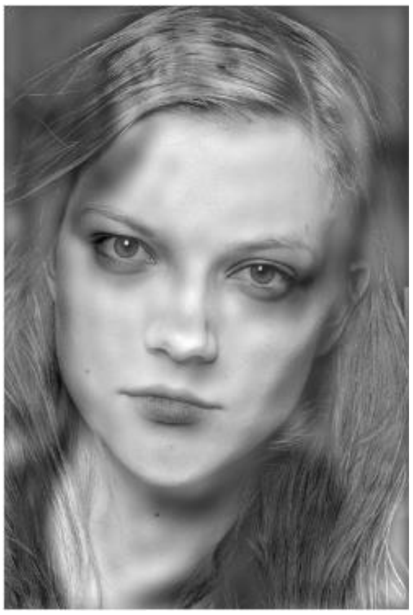
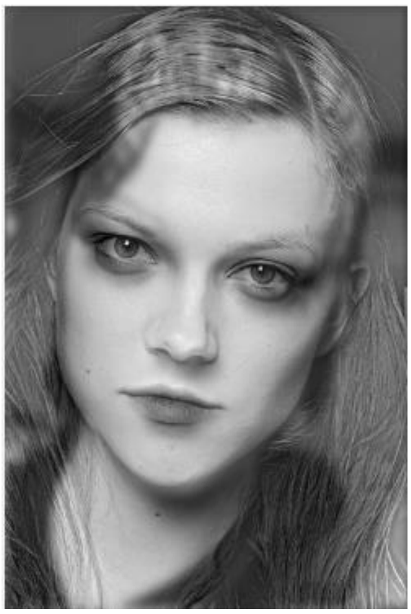
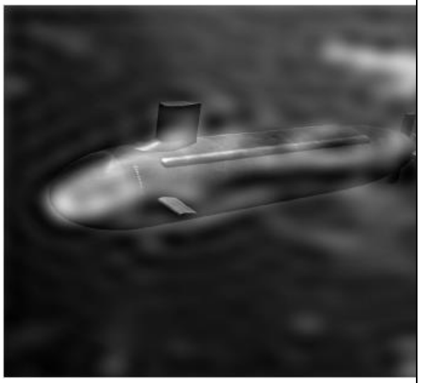
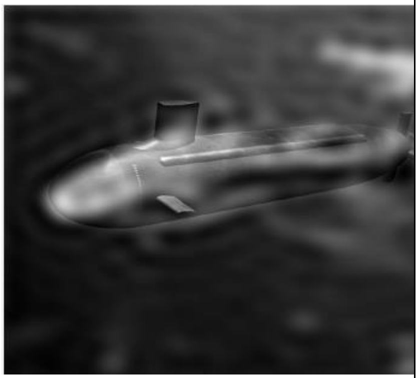
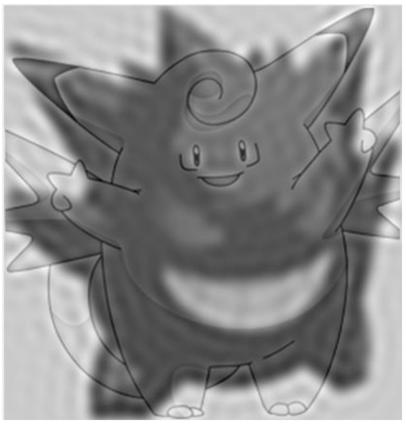
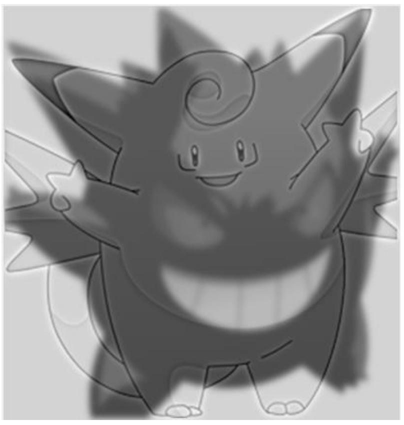
D_0 is cut-off frequency, $\mathrm{D(u,v)} = \sqrt{u^2 + v^2}$

5. Multiply F(u,v) by a filter function H(u,v).
6. Compute the inverse Fourier transformation of 5.
7. Get the real part of 6.
8. Multiply the result in 7 by $(-1)^{x+y.}$
9. Add two images.

Experiment Results

D_0=30	Ideal	Gaussian
Cat & Dog		

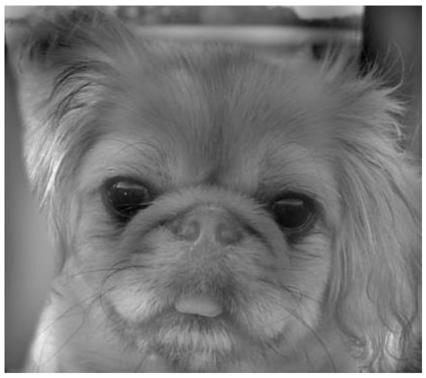
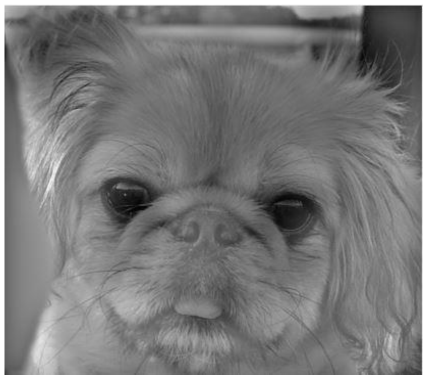
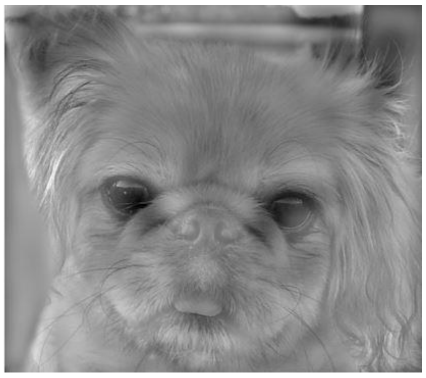
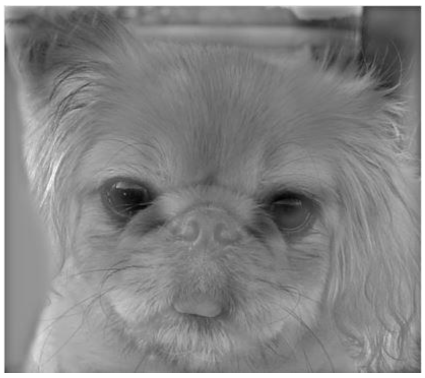
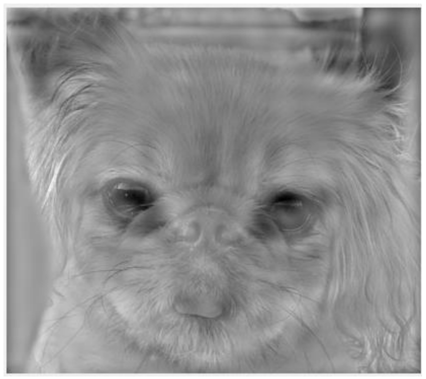
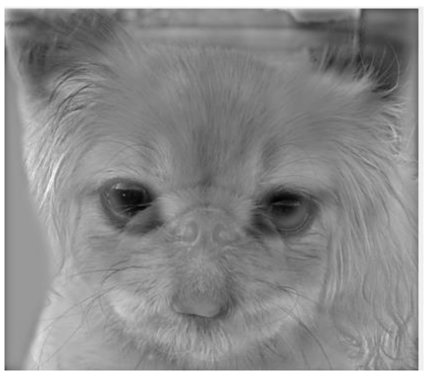
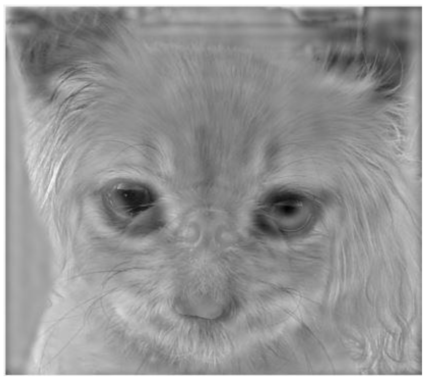
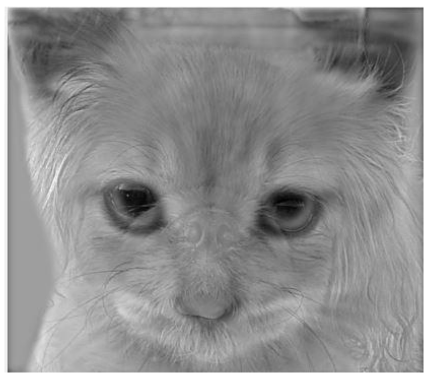
Afghan_girl_after & Afghan_girl_before		
Bicycle& motorcycle		
Bird & plane		
Einstein & marilyn		

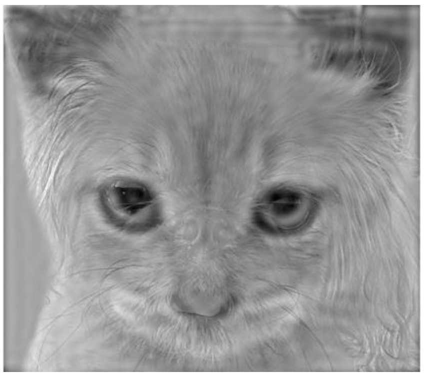
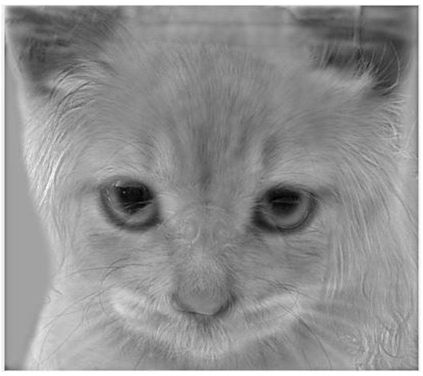
Makeup_after& Makeup_before		
Fish & submarine		
Our images		

With different D_0

$\mathbf{D_0}$	Ideal	Gaussian

Group 27

10		
20		
30		
40		

50		

Discussion

I think there is no problem in this task because the implement procedure is clearly stated in the PDF file.
The problem I encounter is to find other images. I think I have to find two pictures which each subject is in the similar location, or the result would look like two separated objects. And the solution is to select the proper images.

Conclusion

From the results, We can see the results with the Gaussian filter will be more smoother than those with the ideal one. Because every location in Gaussian filter has non-zero value, so the results will be smoother after calculation.
And the cut-off frequency will also influence results, so we must select a proper frequency, or the results will tilt to one side.

Task 2 Image Pyramid

A. Introduction

We practice how to build Gaussian pyramid, Laplacian pyramid, and show the result on frequency domain. We learn how to implement it from scratch in order to understand meanings of each step. We use python and jupyter notebook to implement this task.

B. Implementation Procedure

Step 1. Use Gaussian Filtering to Smooth the Picture

We decide to use 5*5 kernel size, mean equals to zero, and standard deviation equals to one to build our kernel. Our Gaussian filter follows $\mathrm{f(x,y)} = \frac{1}{2\pi} e^{-\frac{(x^2+y^2)}{2}}$. We further divide each element with a constant in the kernel to make sure that the sum of all elements is one. After building a fixed filter, we use it to construct a function called smooth to do convolution, which is the procedure of Gaussian filtering.

In smooth function, we first do padding. Instead of doing zero padding, we pad with the closest pixel's color, which is the color on the edge of a picture. This allow our edge not getting darker, but maintain the similar color after doing convolution. Then, we do convolution according to formula $\mathrm{G[i,j]} = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i-u, j-v]$. G[i, j] represents the output color at pixel[i,j]. H represent filter, and F stands for input image. k equals to 2 since our kernel size is 5*5.

Step 2. Down Sampling

We construct a function called pyrdown to do down sample after smoothing picture. This function allows images reduce to 1/4 times smaller pictures. We choose odd pixels and eliminate event pixels.

Step 3. Up Sampling

We duplicate each pixel four times to reconstruct images that have not done down sampling. This duplicate function we call it pyrup. After pyrup, we add a additional process. We, again, smooth the image using Gaussian filter. This makes images looks better and allowing them to have a higher resolution instead of just simply looks like having larger pixels.

Step 4. Laplacian

The process simply pixel wise minus images with up sampling images that is one-layer higher. Additionally, we just simply make sure that pixel value is legal.

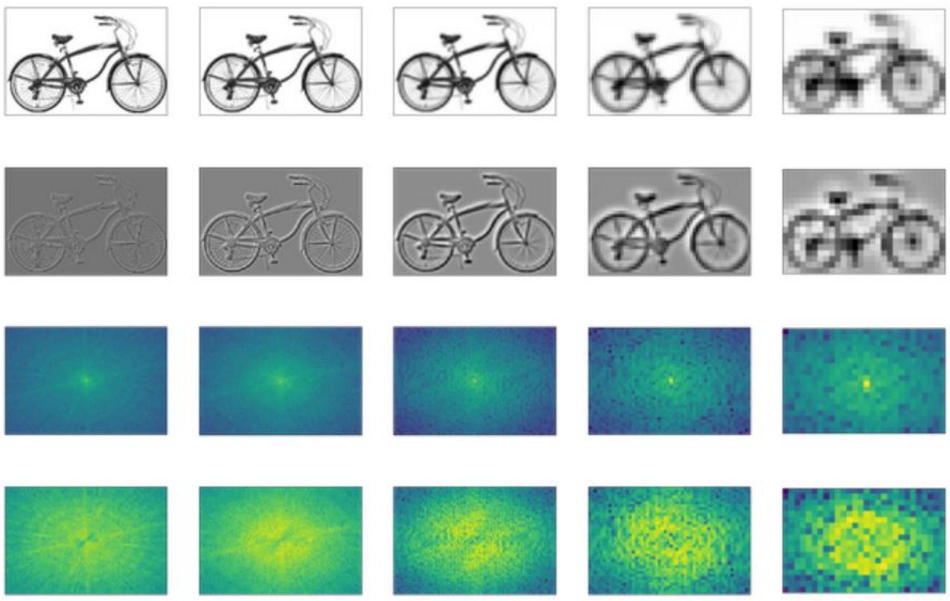
Step 5. Magnitude spectrum

We use build in function numpy.fft.fft2 to do discrete time Fourier transform. This function basically follows $\mathrm{F}(\mathrm{u}, \mathrm{v}) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[m,n] e^{-j2x(umx_o+vny_o)}$

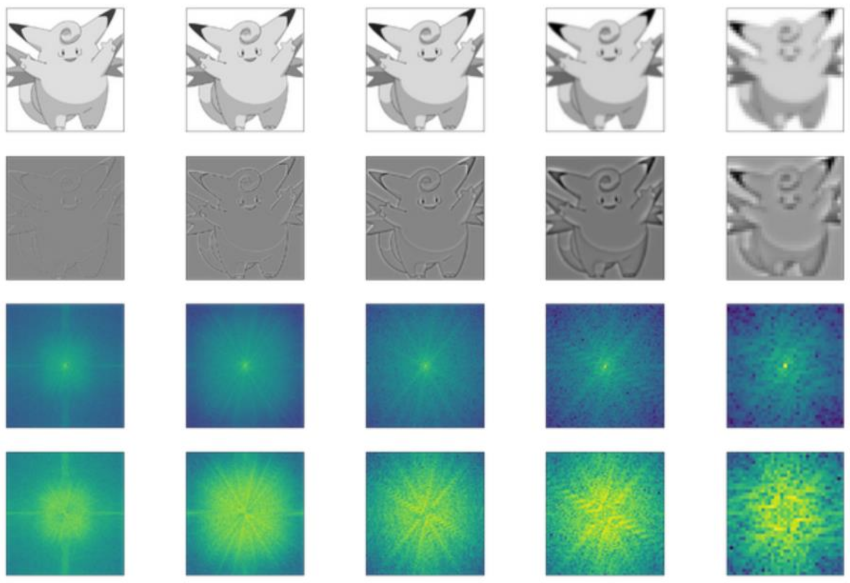
We then find out all frequency domain images for all picture in Gaussian pyramid and Laplacian pyramid.

C. Experimental Result

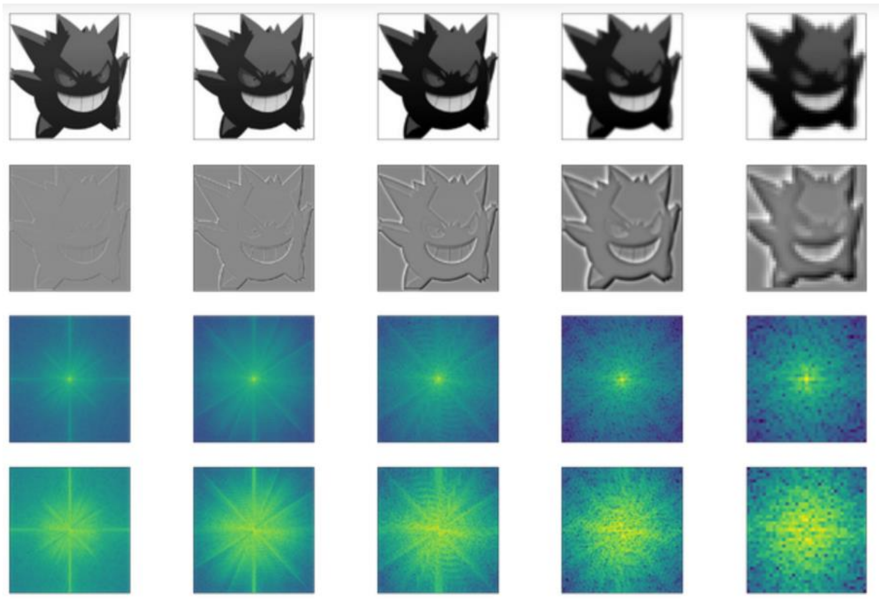
- The result of TA's data



- The result of our own data



- The result of our own data



1. We use cv2 build in function as the correct pyramid to check our result. We get the correct pyramid from our own code.

D. Discussion

1. Since Laplacian compress image value into 0-255, the background color(gray) is those with pixel value zero. Different Laplacian has different range of pixel value. The smallest grayscale values are different. This result in the different color for backgrounds.
2. The magnitude spectrum looks different from spectrum provided by TA. The conclusion is that all the differences only result from different color usage and build in plot method.

3. We use cv2 build in function to check our result. We find out that cv2 do smoothing again after up sampling. We try to compare the differences between doing smoothing or not after up sample images. We find out that images look a lot better with smoothing and Laplacian images would show more things being removed if we do smooth. We decided to follow cv2's method because the result looks better with it.

E. Conclusion

1. By observation, with Gaussian filtering no matter prior or post makes images look a lot better. It helps solve aliasing problem. An image looks the same after down sampling if we scale them to the same smaller size. Thus, it is important to gaussian filtering if there is a need to modify the size of an image.
2. Magnitude spectrum shows that Gaussian filter is a low-pass filter. Higher frequencies are removed fist. This also reflect on images. Details are being remove first. The highest level of Gaussian pyramid only remains basic contour.
3. We use realistic example to confirm the changes in frequency domain in different layers of pyramid. It looks the same as schematic diagram in the lecture.
4. One way to make different Laplacian image comparable is to turn of the auto normalization in plot function. However, this leads to the problem that we can not visualize negative value this way. As a result, we do not turn off auto normalization while plotting results.

Task 3 Colorizing the Russian Empire

A. Introduction

The goal is automatically to produce a color image from the digitized Prokudin-Gorskii glass plate images with as few visual artifacts as possible. We learn how to extract the three color channels from the glass plate, then place and align one above the other so that the combination forms a single RGB color image. And we should make the procedure fast and efficient despite the high resolution images.

B. Implementation Procedure

Step 1. Preprocessing

First of all, we split the glass plate images into three equal parts based on the height to get the three channels. Then we crop 10% from each side to remove the black edges.

Step 2. Single-scale image alignment for jpg

In small images, we construct a function called shift(), which calculates image A should shift how much pixel to match image B. To align two small images, we exhaustively shift one of the images plus or minus 15 rows and columns, computing the sum of squared differences (SSD) between the two images for each shift. The shift that produces the minimum SSD is the best alignment. We roll R channel and G channel into B channel by using shift function, and then stack the three channels to get the color image.

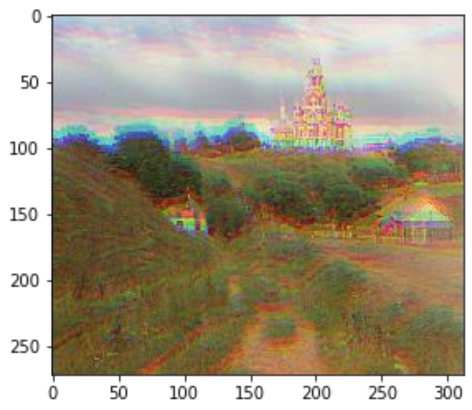
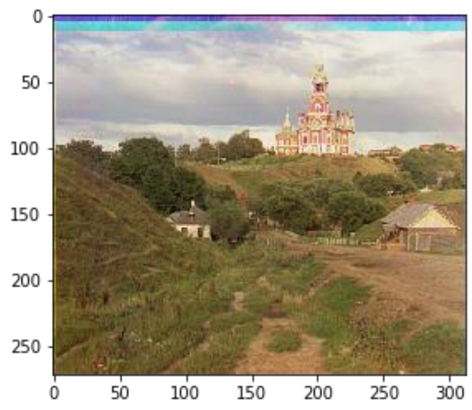
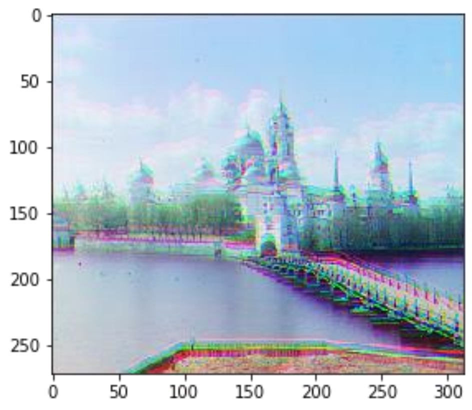
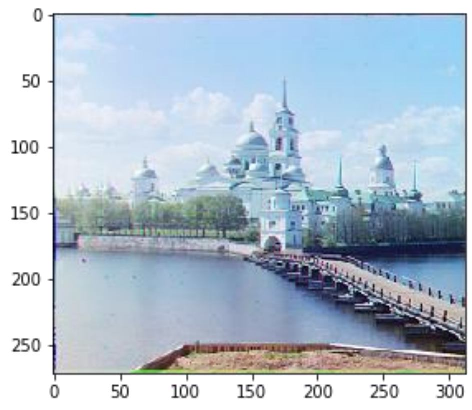
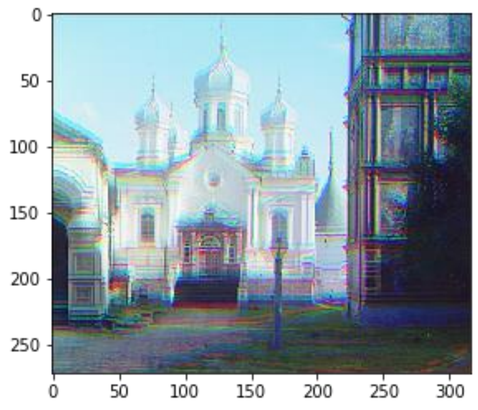
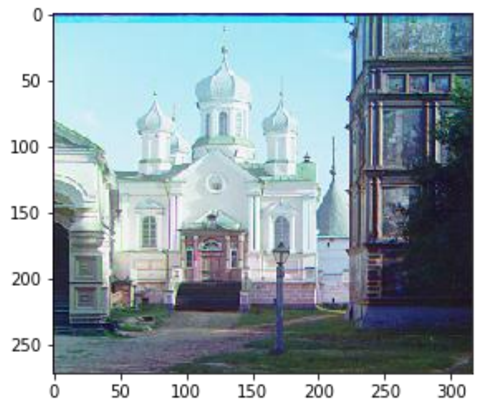
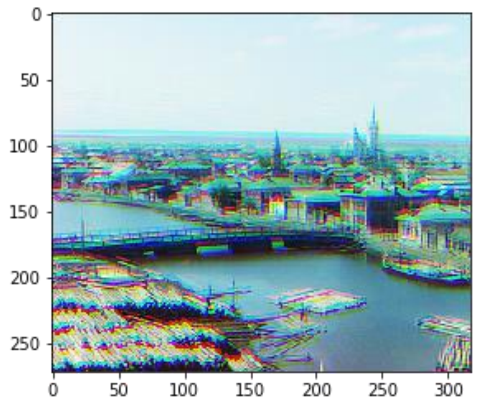
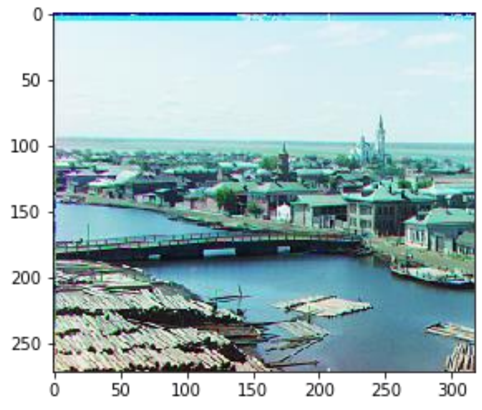
Step 3. Multi-scale image alignment for tif

In high resolution images, it is extremely slow and inefficient to perform the window method over a large set of displacements. Therefore, we apply the pyramid method. We firstly use Sobel operator to get the edges and then iteratively downscale the image by local mean with size f*f and use a window size to align the coarse image (the method in step 2). The smaller f, the smaller window size. We get a shift and multiply it by f in each scale. Finally, we add up the shift to get the total shift of the channel to the other. We roll R channel and G channel into B channel by that shift amounts, and then stack the three channels to get the color image.

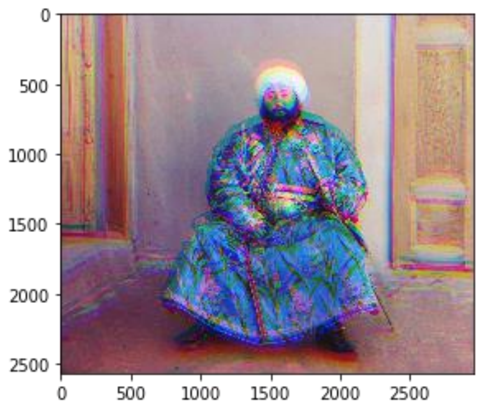
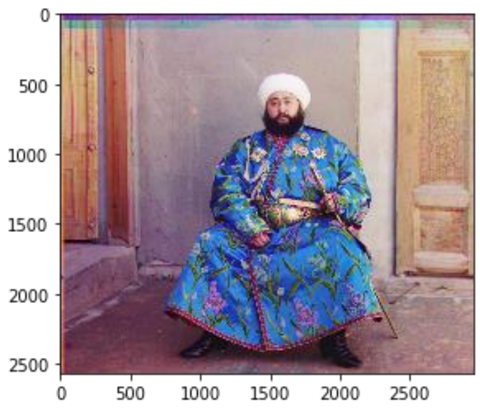
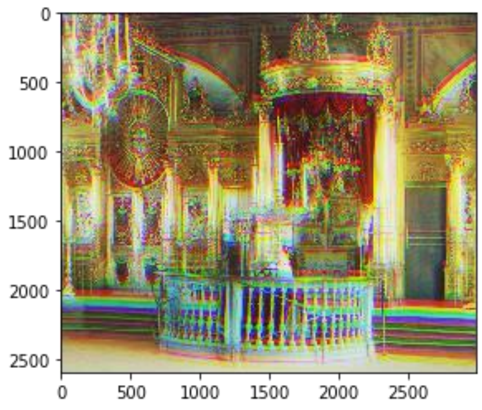
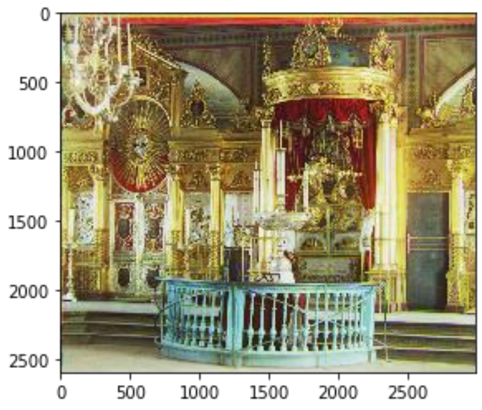
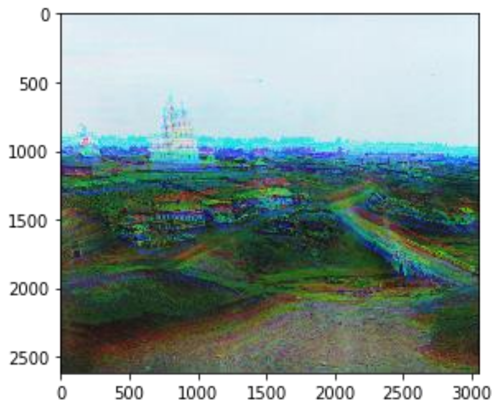
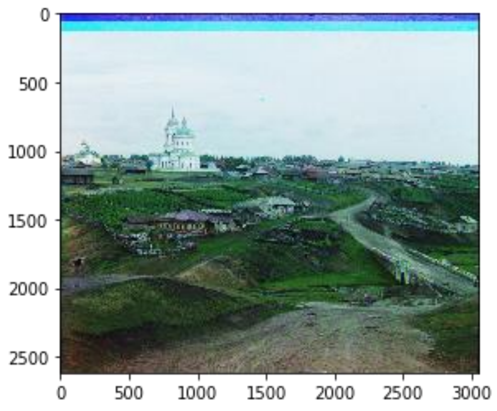
C. Experimental Result

Group 27

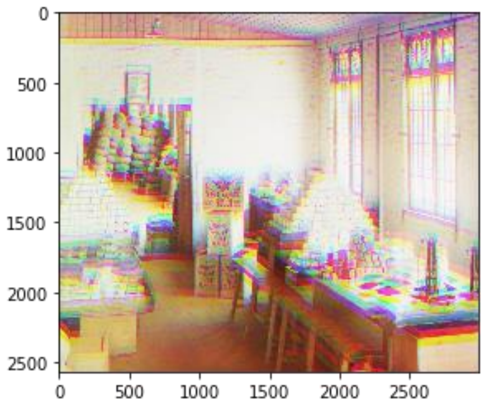
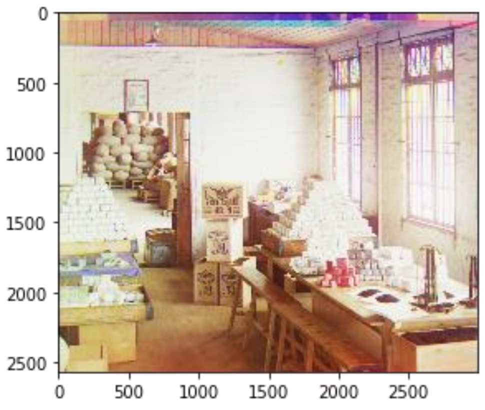
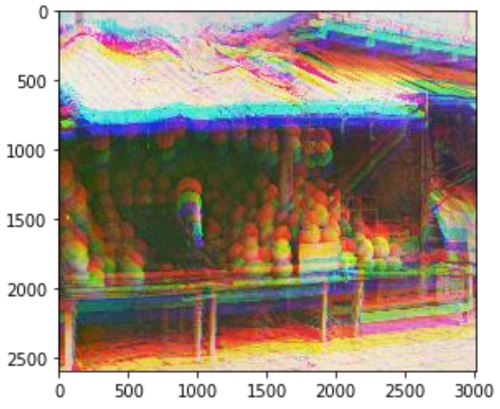
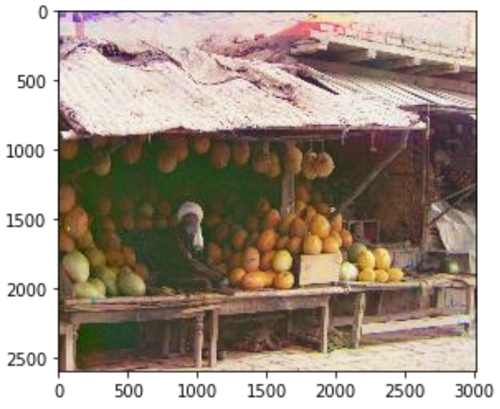
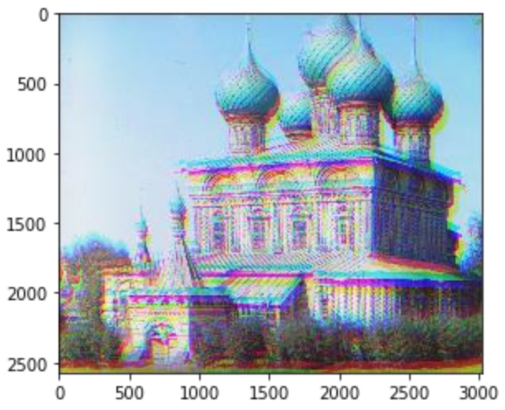
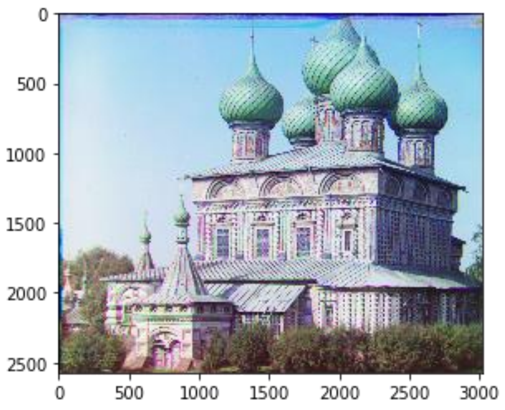
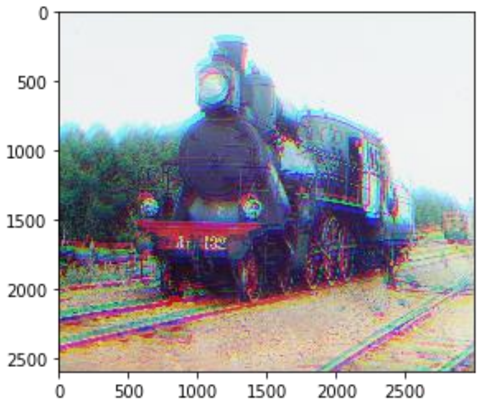
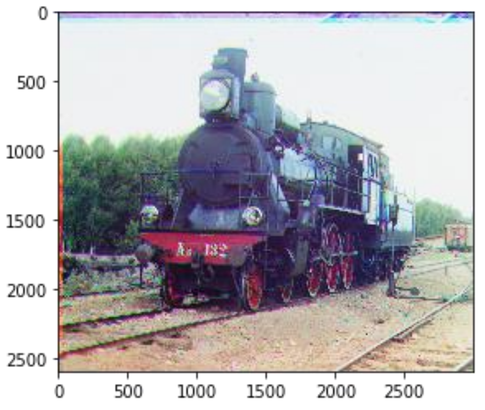
- Single-scale alignment image for .jpg

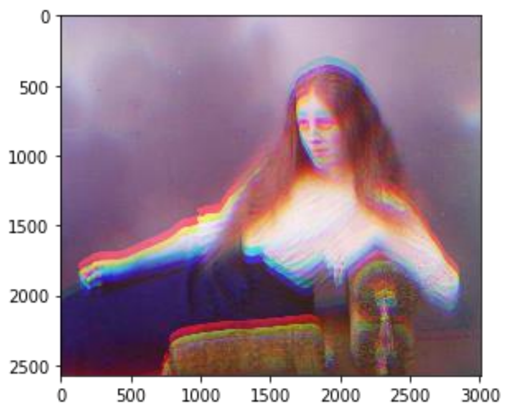
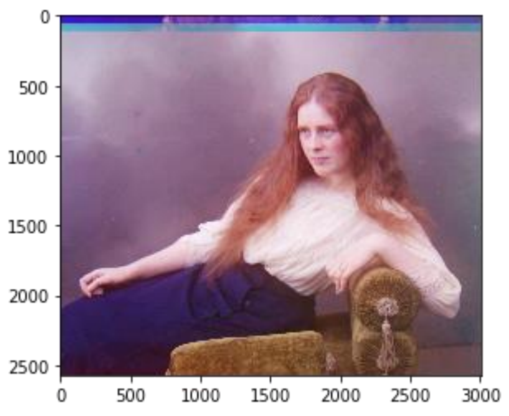
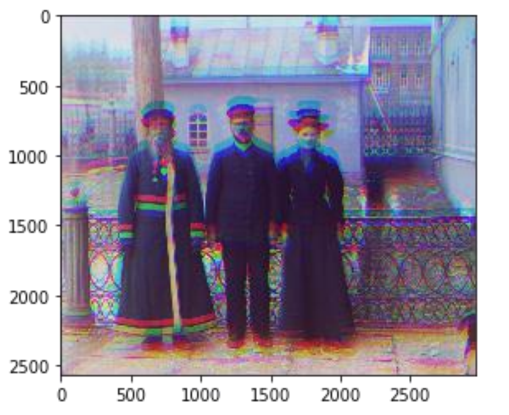
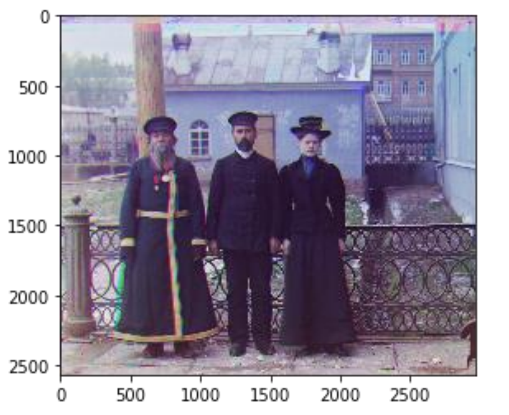
Before align	After align
	
	
	
	

- Multi-scale alignment image for .tif

Before align	After align
	
	
	

Group 27

D. Discussion

1. When doing Sobel operator, it takes lots of time. The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical. That is,

$$\boldsymbol{G_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \boldsymbol{A} \text{ and } \boldsymbol{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \boldsymbol{A}$$

if we use the kernel do the convolution, we have four loops and get O(ghmn) time, where image size is g*h and kernel size is m*n. This operator causes huge time to calculate. We try to divide the kernel into two 1-dim vectors. That is,

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * [-1 \quad 0 \quad 1]$$

$$\text{and} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * [1 \quad 2 \quad 1]$$

After that, we have only three loops and get O(gh(m+n)) time.

Even though we have speed up Sobel operator, it still takes about 1000 seconds at high resolution image.
So we keep thinking to deal with that problem. Eventually, we come up with a technique to solve it by using advantage of Numpy array. Speedup is about 1000. It takes less than 2 seconds at high resolution image. In our code, you can find out the different between sobel_old_version and sobel_new_version.

2. When we get the $\boldsymbol{G_x}$ and $\boldsymbol{G_y}$, we have three method to combine them.
   i. $\mathrm{G} = G_x + G_y$
   ii. $\mathrm{G} = |G_x| + |G_y|$ (L1 distance)
   iii. $\mathrm{G} = \sqrt{G_x^2 + G_y^2}$ (L2 distance)

   We choose first method since it looks better.
3. The .tif images have large resolution, so we use the pyramid method to deal with it.
4. We implement the function - downscale-local-mean, and it take most of the execution time. The downscale-local-mean takes 150 seconds when f = 1, about 40 seconds when f = 2, about 10 seconds when f = 5, about 2 seconds when f = 10, and about 1 seconds when f = 20. And the pyramid method time except downscale_local_mean time takes about 10 seconds. When we call the package of Python to do that, downscale_local_mean takes about 5 seconds. But the total time still less than using only the single-scale alignment method and it takes about 200 seconds for each image.

E. Conclusion

1. We have two kinds of image format type - .jpg and .tif, and we deal them with single-scale image alignment method and multi-scale image alignment method, respectively.
2. Divide Sobel operator into two 1-dim vector to speed up the execution time and pad with the copy border method in the beginning.
3. Choose $\mathrm{G} = G_x + G_y$ for Sobel operator.
4. Use sum of squared differences (SSD) method to get the shift between two images.
5. Use pyramid method to implement the colorizing of high resolution images.
6. When doing the downscale, we use local mean with window size f*f.

F. Work Assignment

Group 27