CS543 Assignment 2

Your Name: fanmin shi
Your NetId: fshi5

Part 1 Fourier-based Alignment:

You will provide the following for each of the six low-resolution and three high-resolution images:

- Final aligned output image
- Displacements for color channels
- Inverse Fourier transform output visualization for ***both*** channel alignments ***without*** preprocessing
- Inverse Fourier transform output visualization for ***both*** channel alignments ***with*** any sharpening or filter-based preprocessing you applied to color channels

A: Channel Offsets

Replace <C1>, <C2>, <C3> appropriately with B, G, R depending on which you use as the base channel. Provide offsets in the **original image coordinates** and be sure to account for any cropping or resizing you performed.

Low-resolution images (using channel B as base channel):

Image	G (h,w) offset	R (h,w) offset
00125v.jpg	(-1, 5)	(0, 10)
00149v.jpg	(2, 4)	(2, 10)
00153v.jpg	(3, 7)	(4, 14)
00351v.jpg	(0, 4)	(1, 13)
00398v.jpg	(3, 5)	(4, 12)
01112v.jpg	(0, 0)	(1, 5)

High-resolution images (using channel <C1> as base channel):

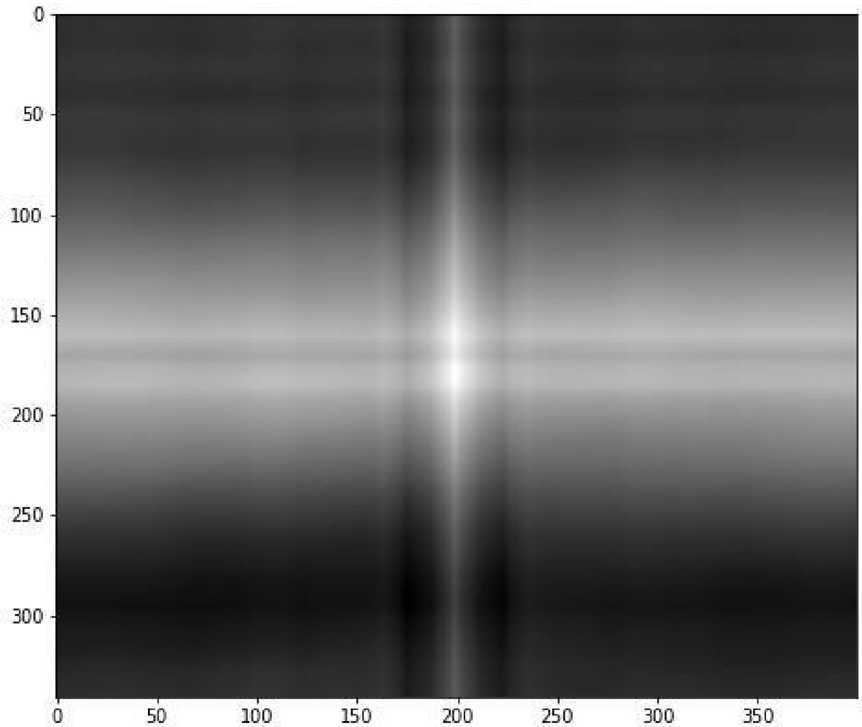
Image	G (h,w) offset	R (h,w) offset
01047u.tif	(21, 25)	(34, 72)
01657u.tif	(9, 57)	(12, 120)
01861a.tif	(40, 72)	(62, 148)

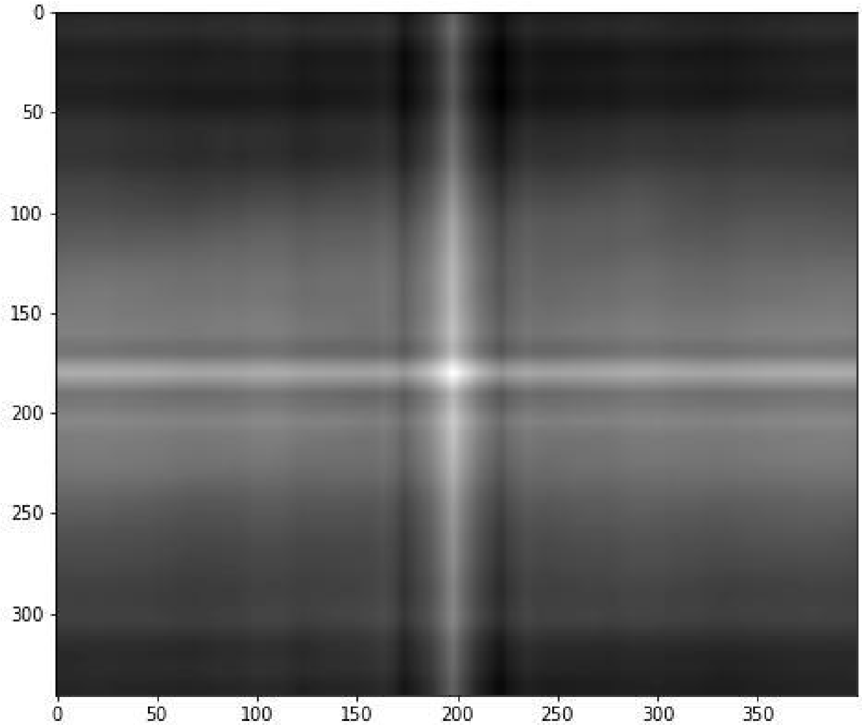
B: Output Visualizations

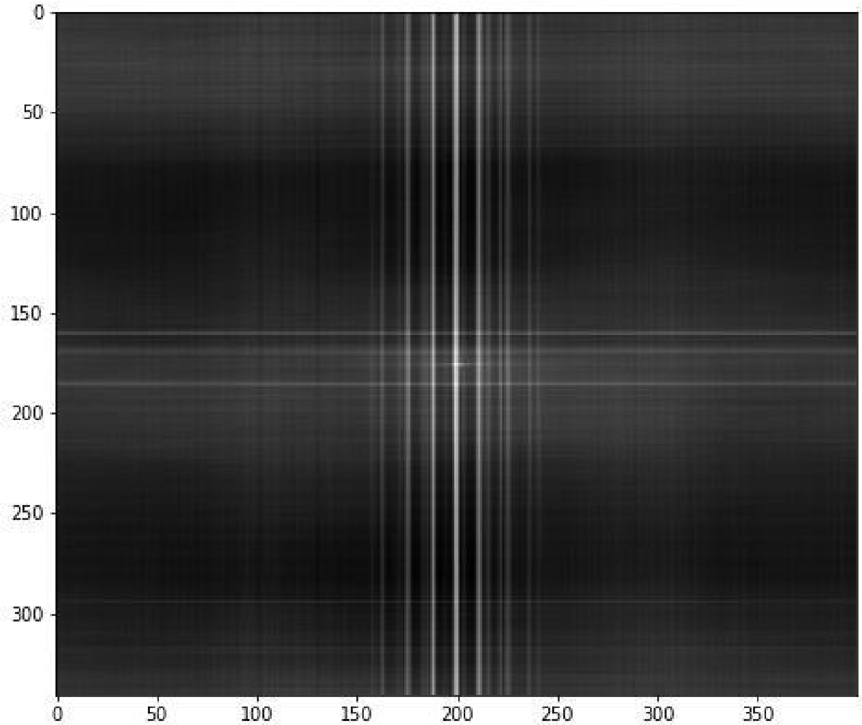
For each image, insert 5 outputs total (aligned image + 4 inverse Fourier transform visualizations) as described above. When you insert these outputs be sure to clearly label the inverse Fourier transform visualizations (e.g. “G to B alignment without preprocessing”).

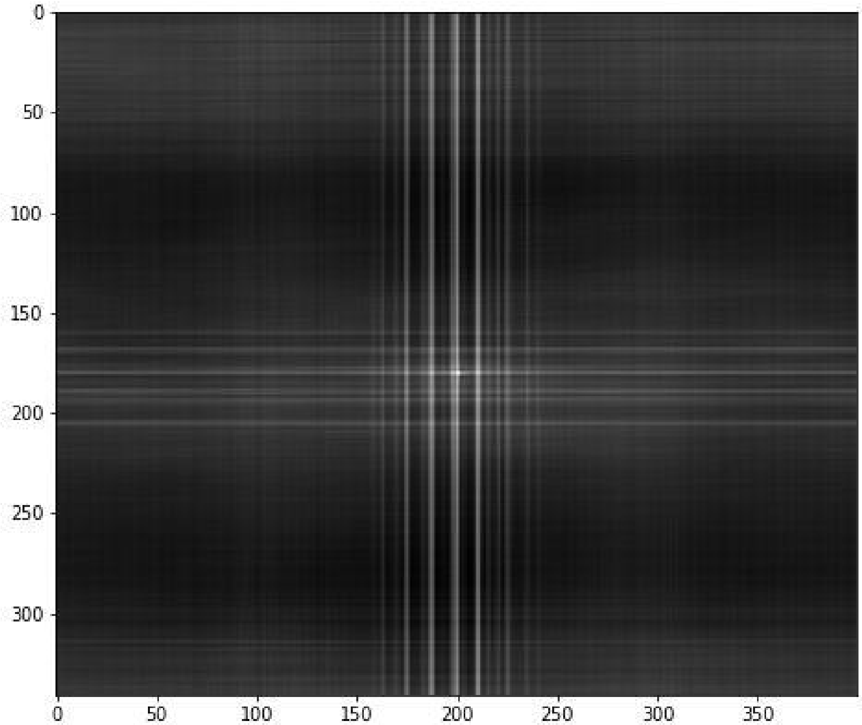
00125v.jpg

Aligned	

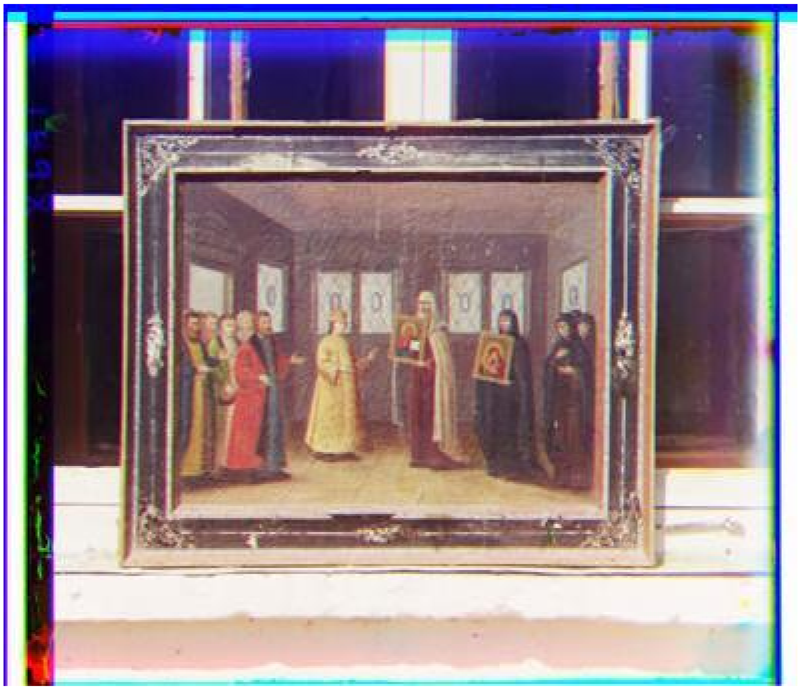
G -> B No Processing	

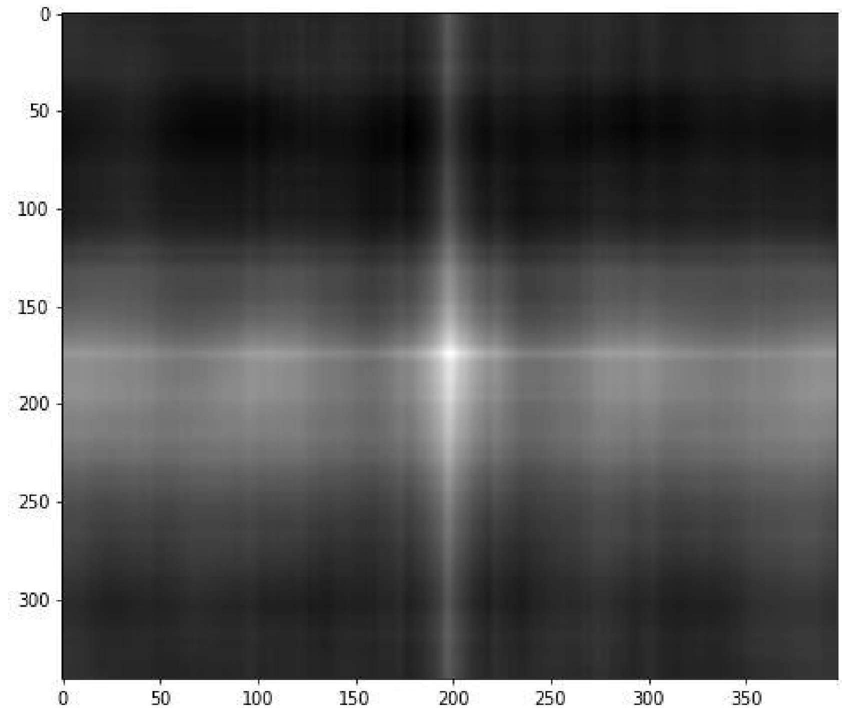
R -> B No Processing	

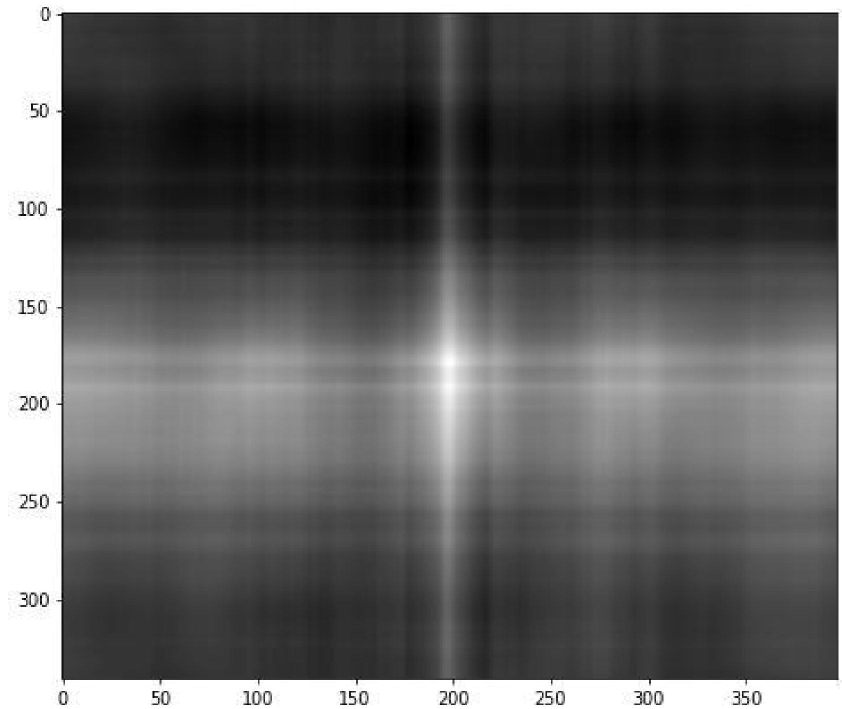
G -> B Processing	

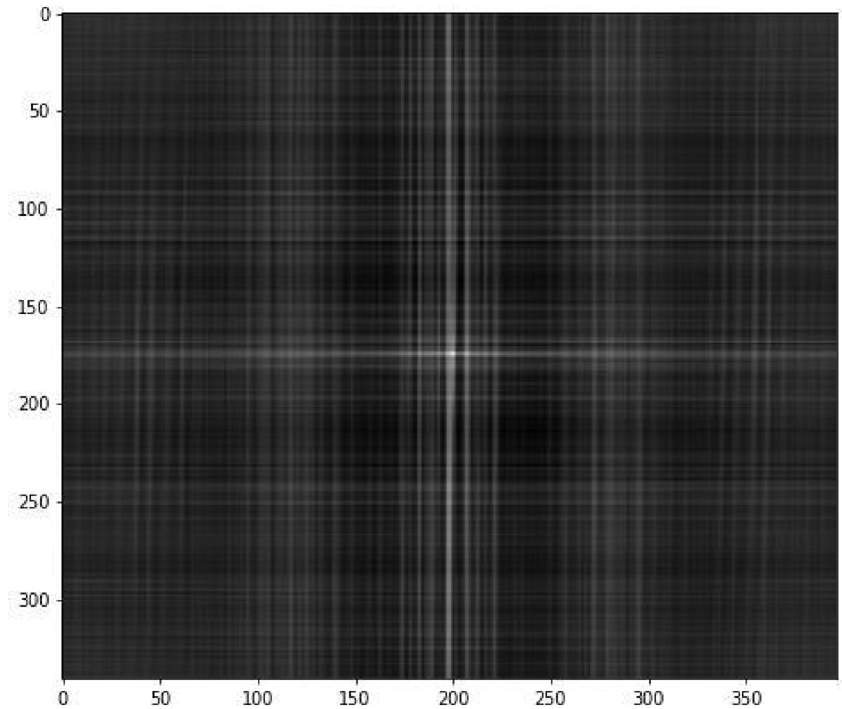
R -> B Processing	

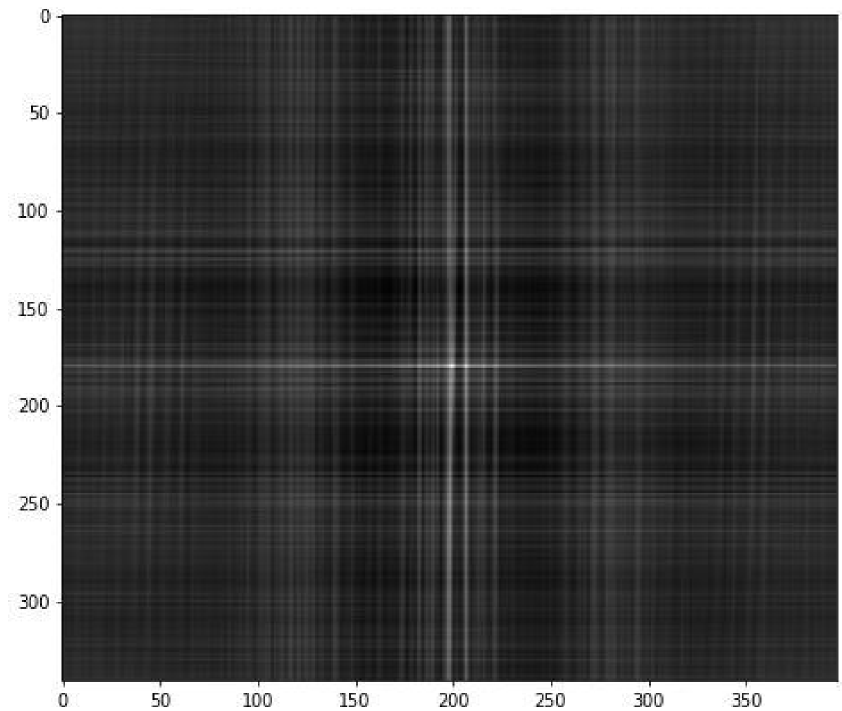
00149v.jpg

Aligned	

G -> B No Processing	

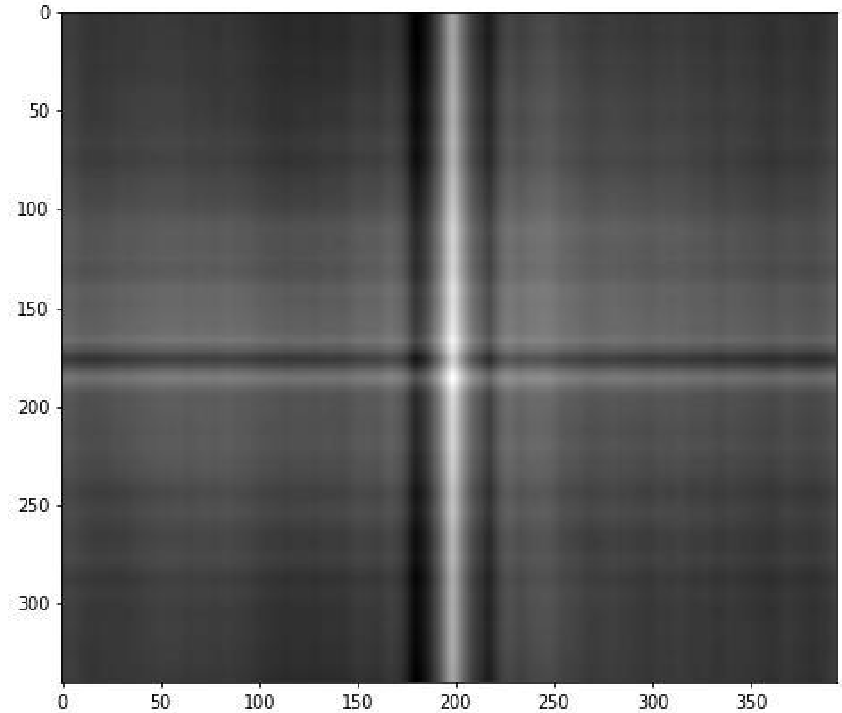
R -> B No Processing	

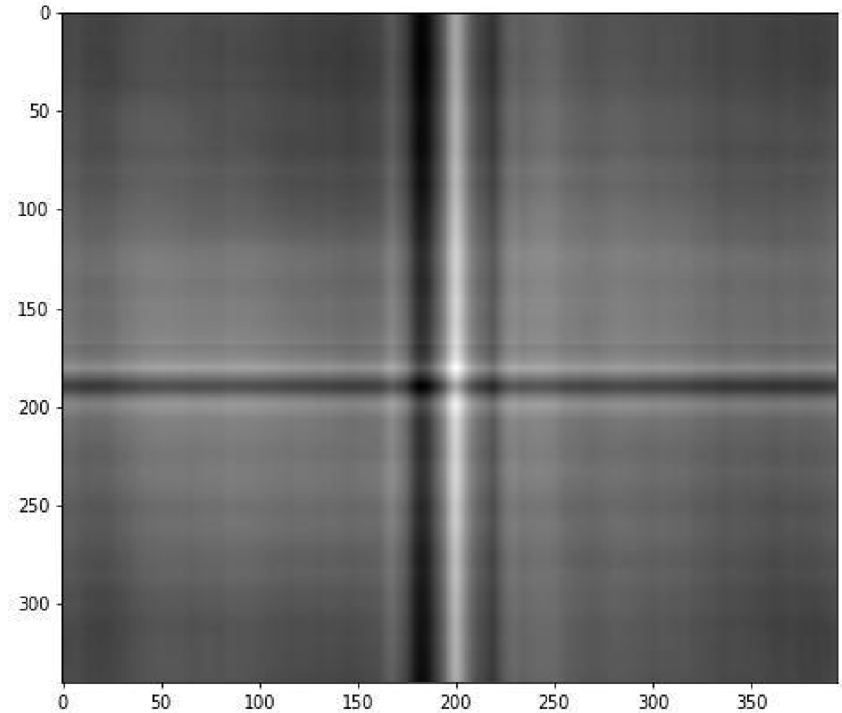
G -> B Processing	

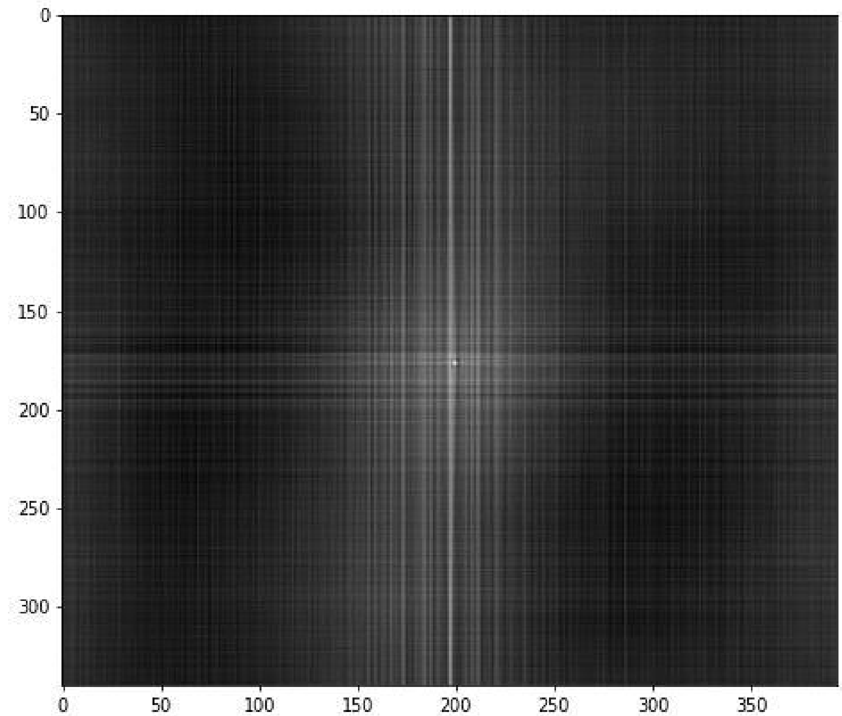
R -> B Processing	

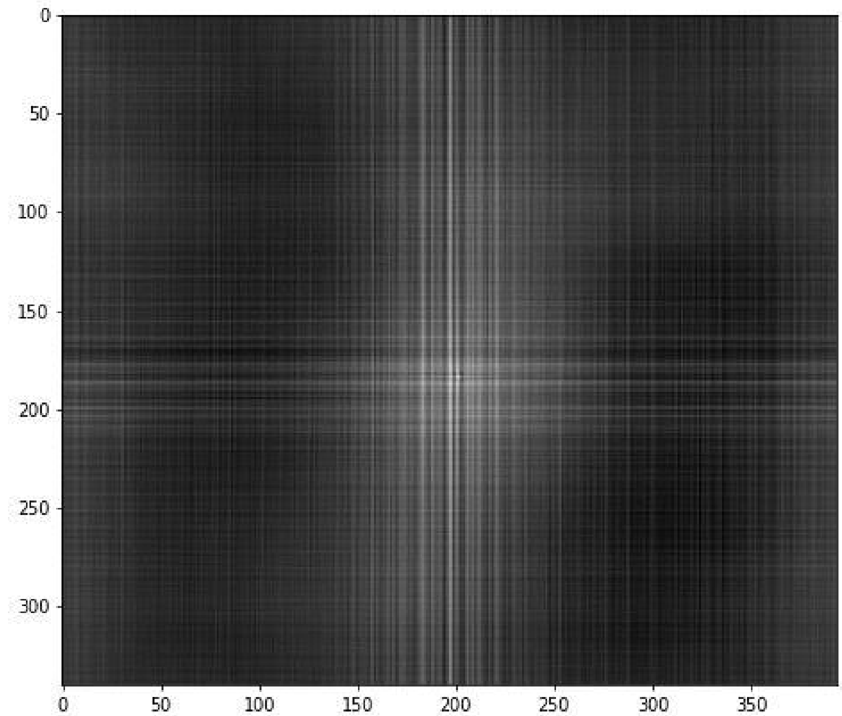
00153v.jpg

Aligned	

G -> B No Processing	

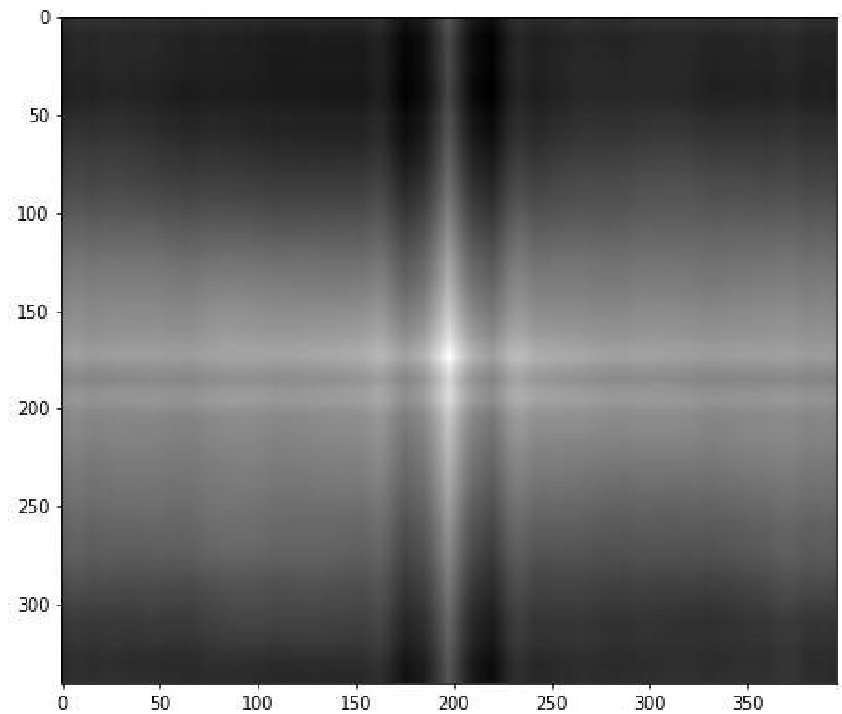
R -> B No Processing	

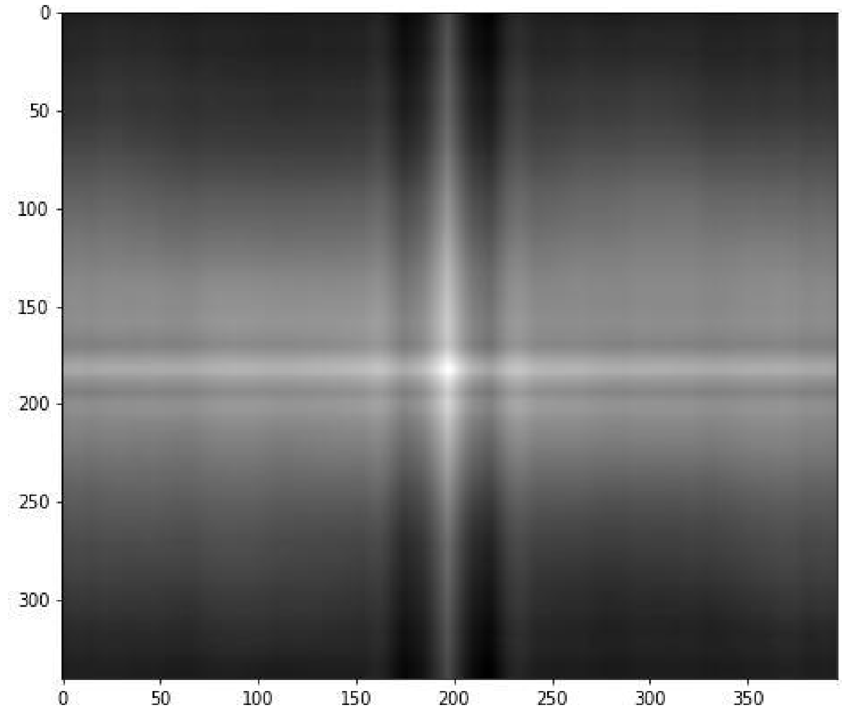
G -> B Processing	

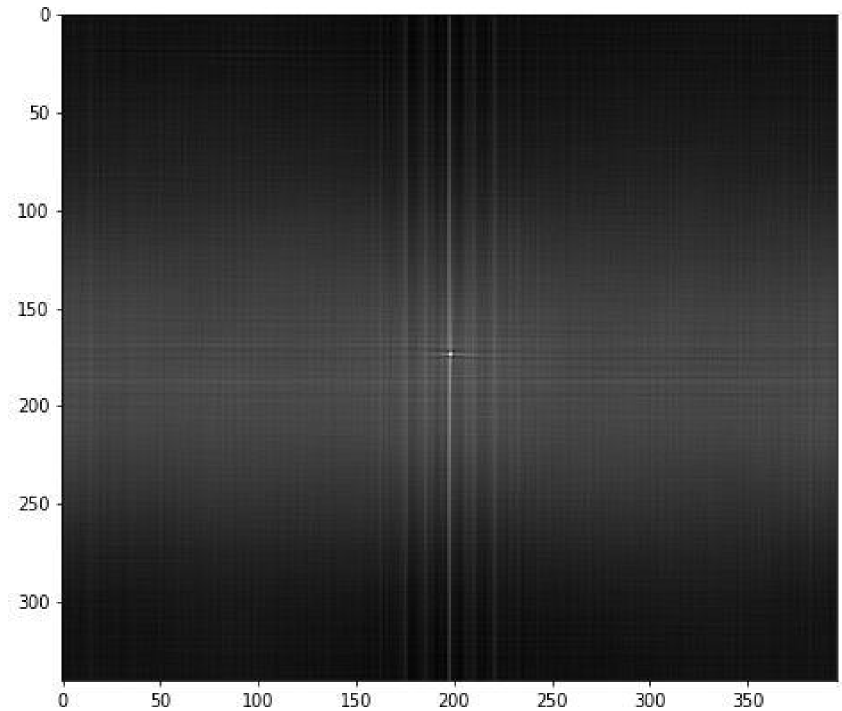
R -> B Processing	

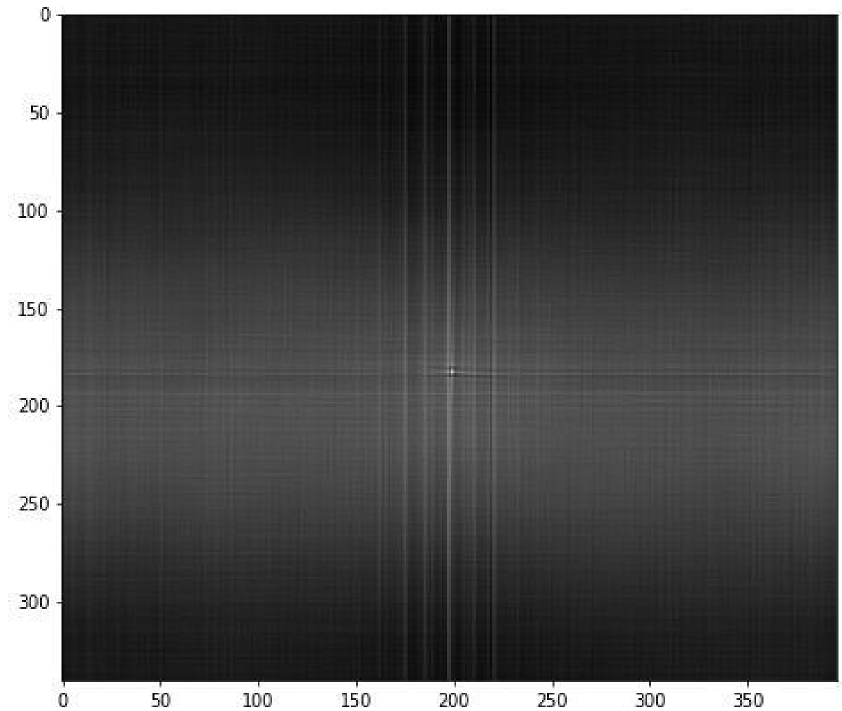
00351v.jpg

Aligned	

G -> B No Processing	

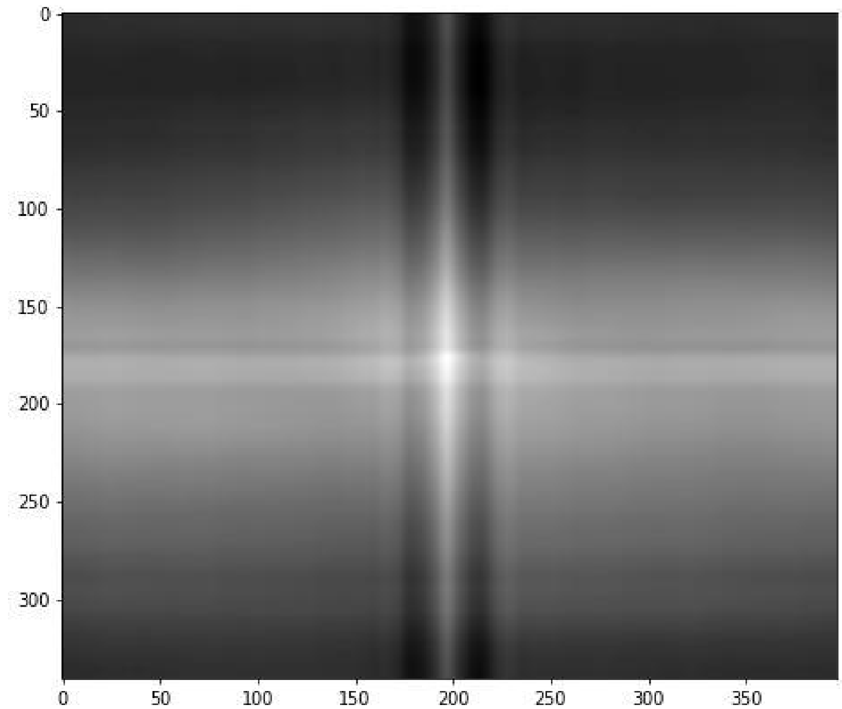
R -> B No Processing	

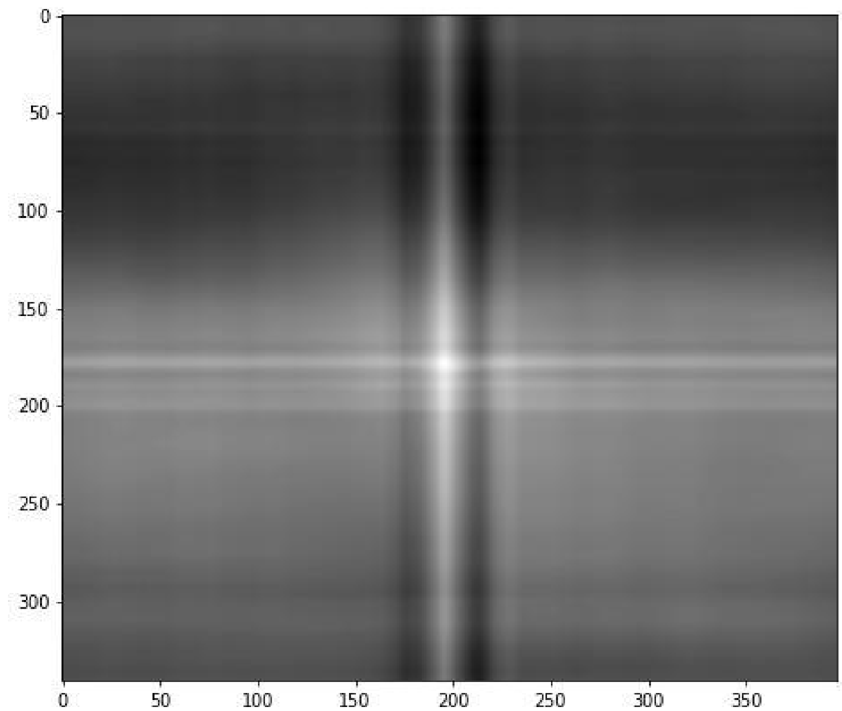
G -> B Processing	

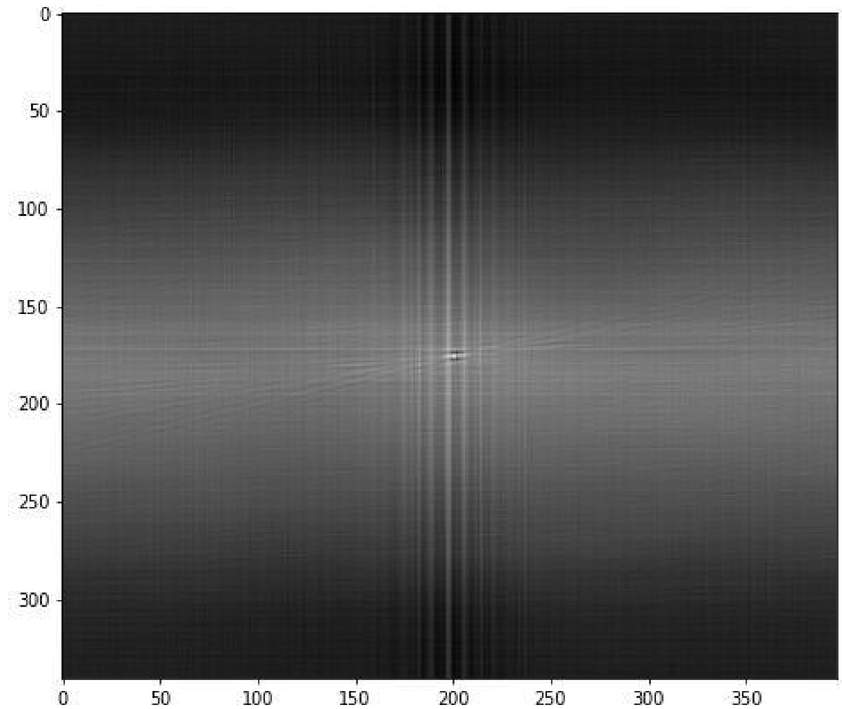
R -> B Processing	

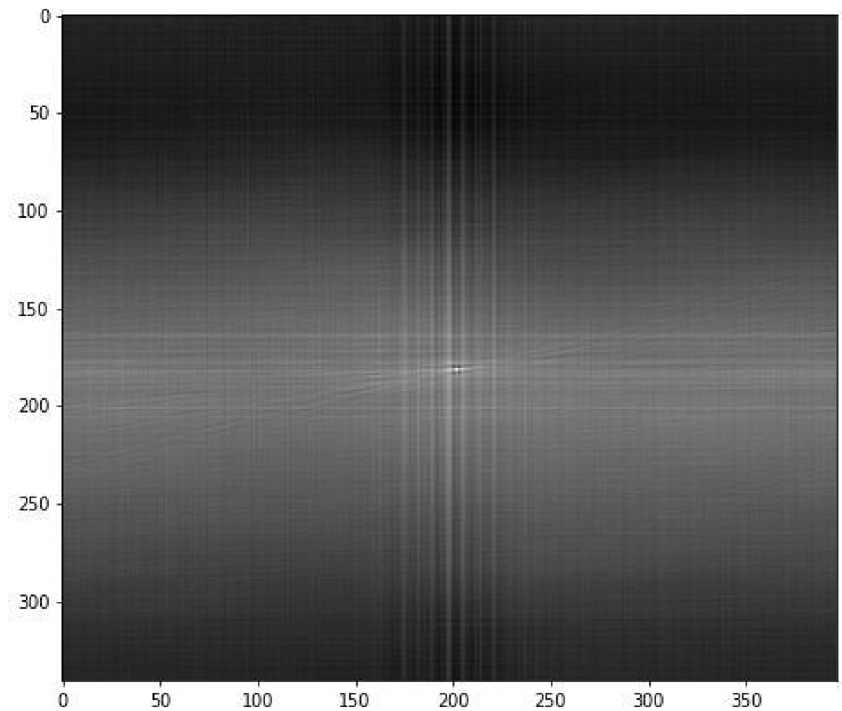
00398v.jpg

Aligned	

G -> B No Processing	

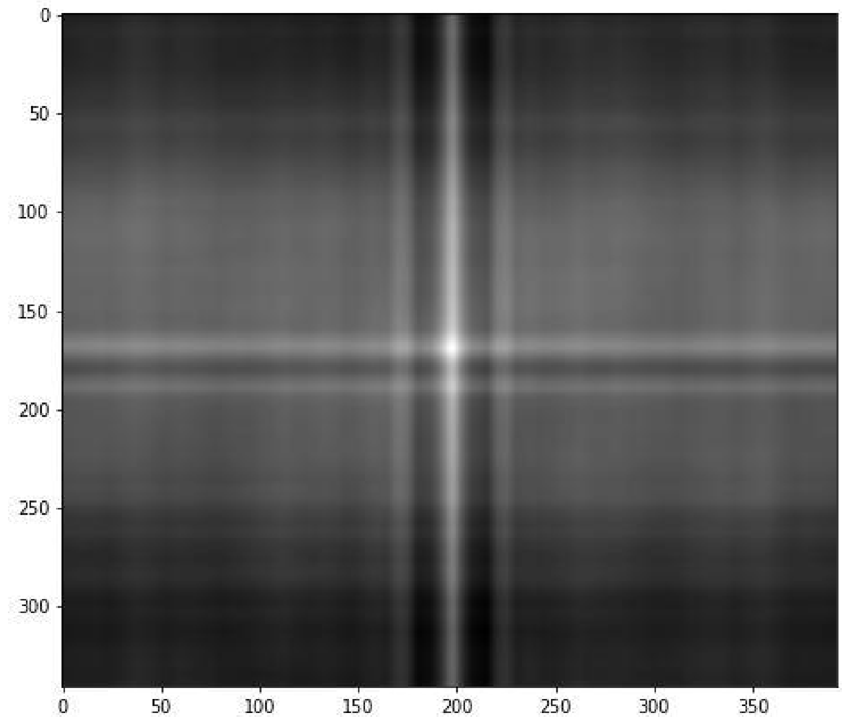
R -> B No Processing	

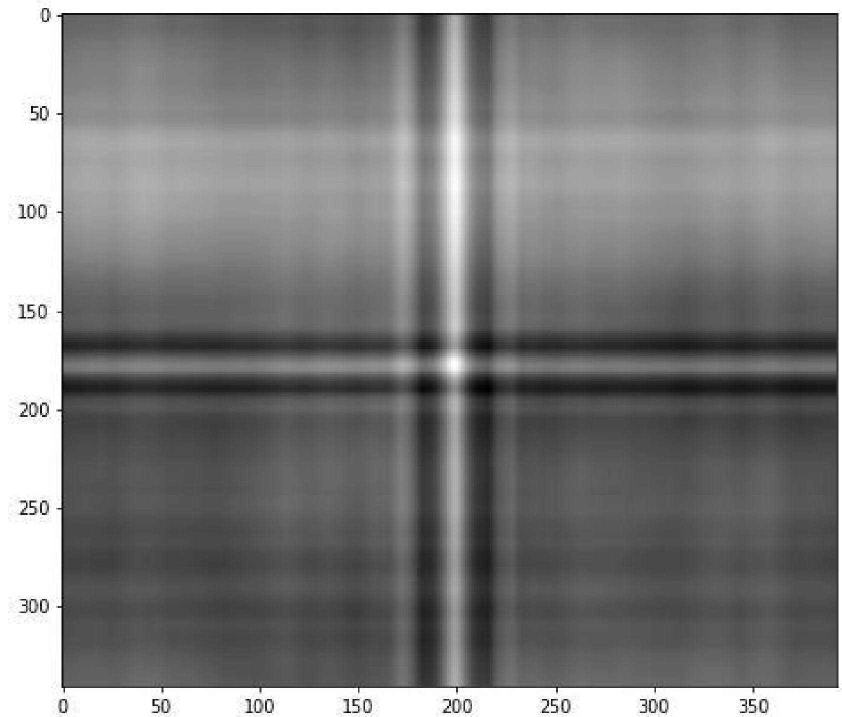
G -> B Processing	

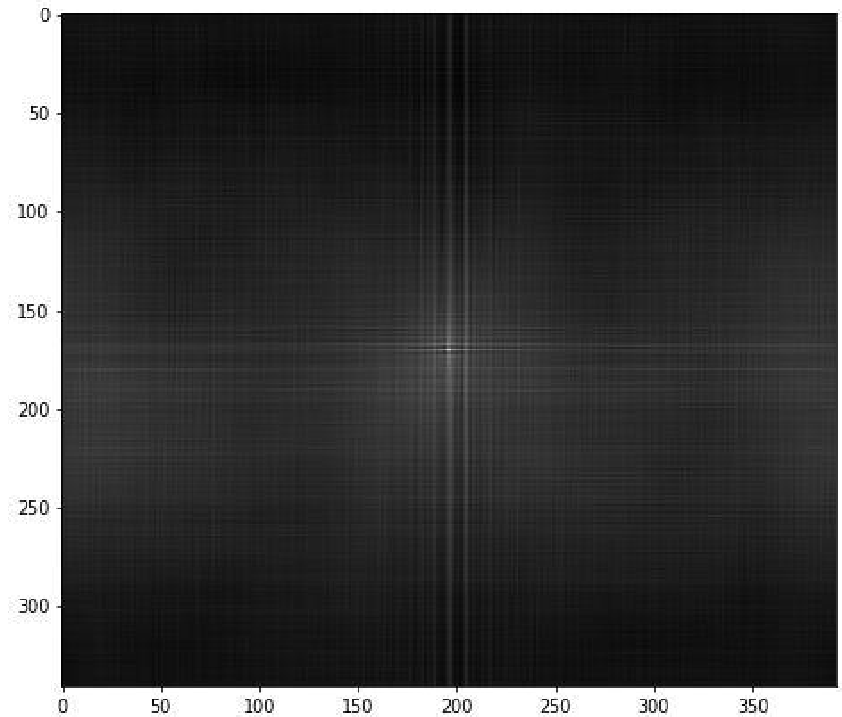
R -> B Processing	

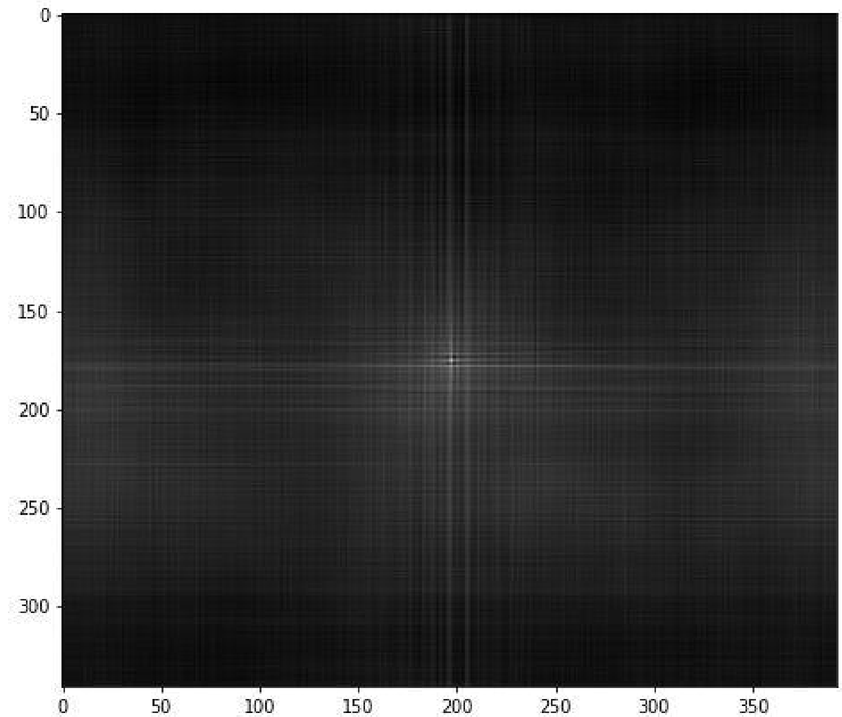
01112v.jpg

Aligned	

G -> B No Processing	

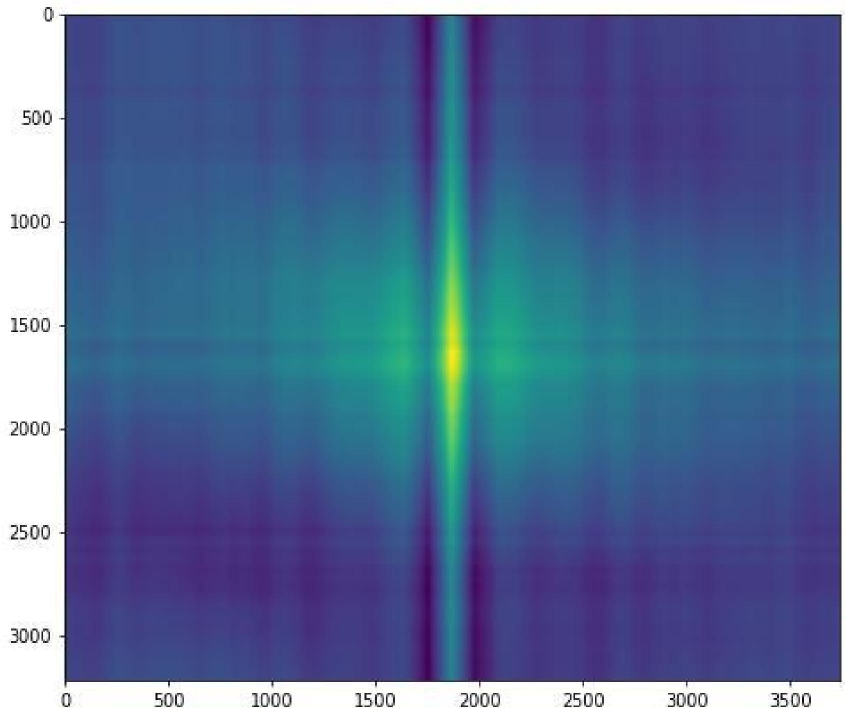
R -> B No Processing	

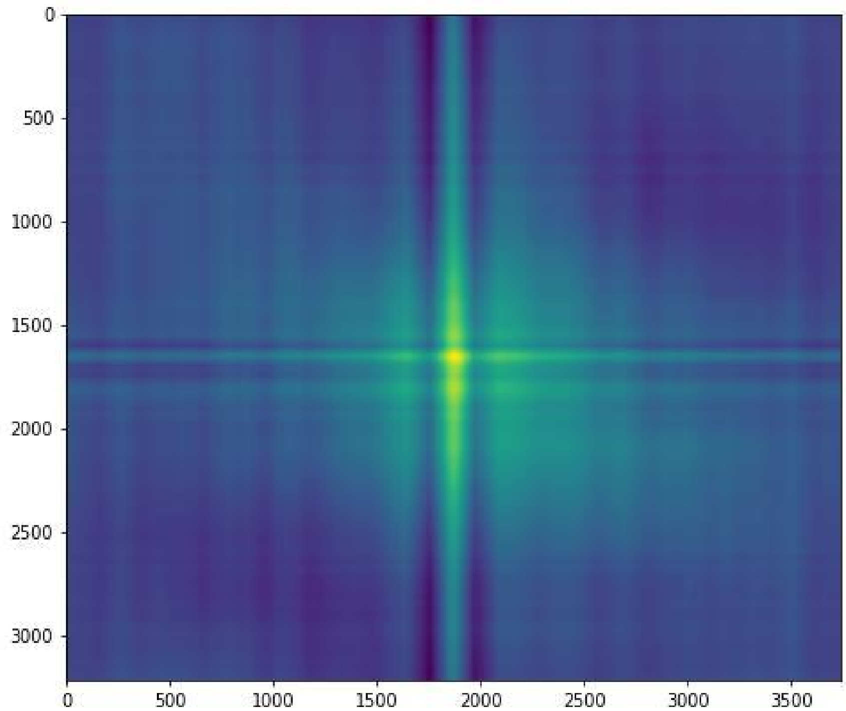
G -> B Processing	

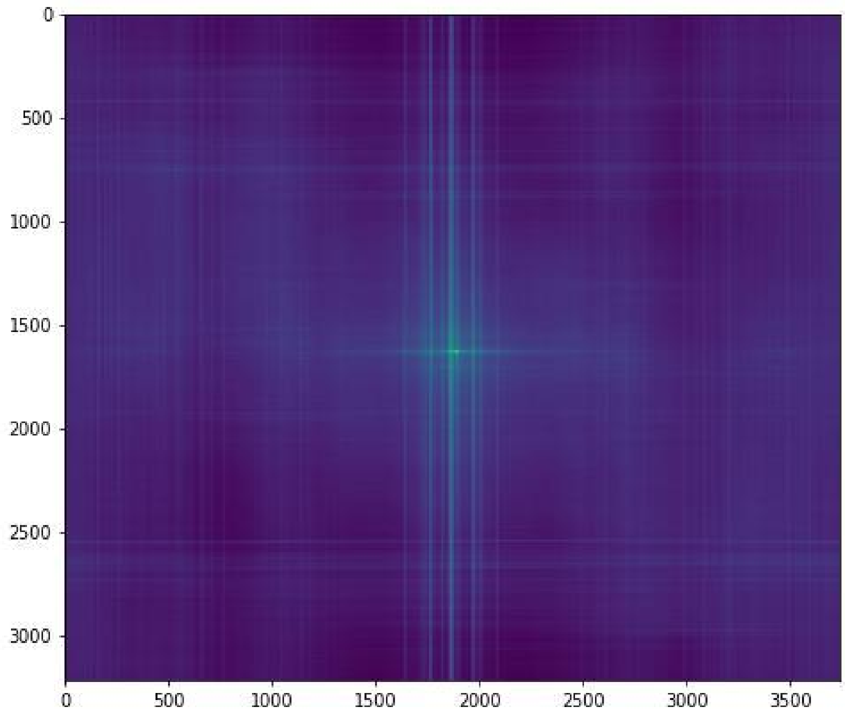
R -> B Processing	

01047u.tif

Aligned	

G -> B No Processing	

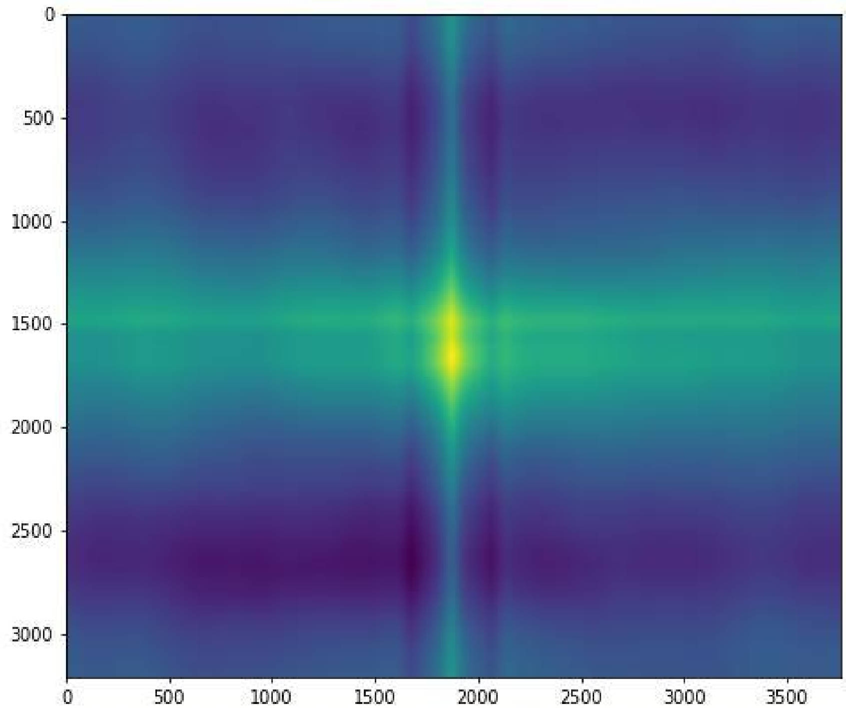
R -> B No Processing	

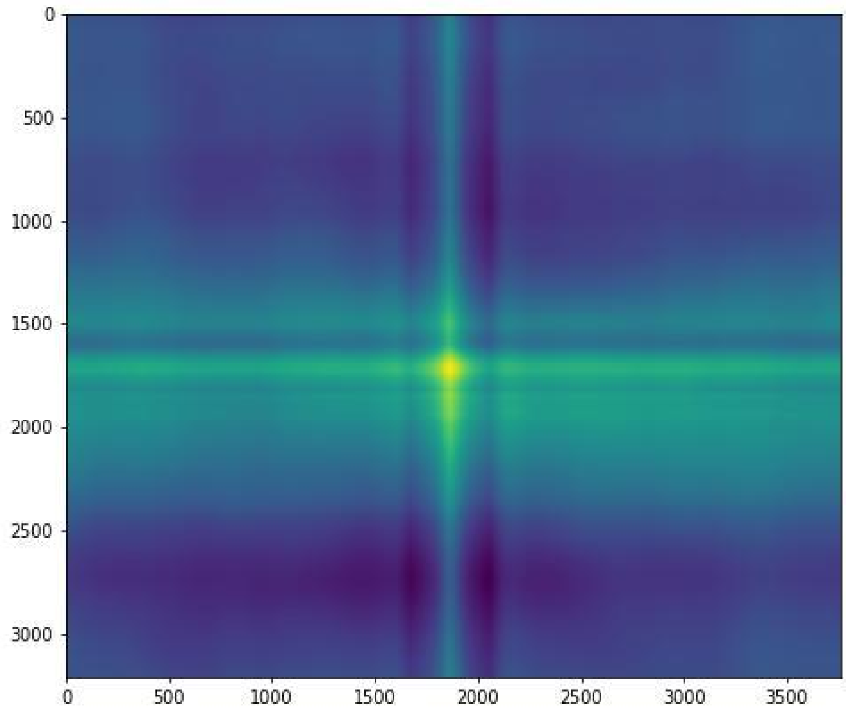
G -> B Processing	

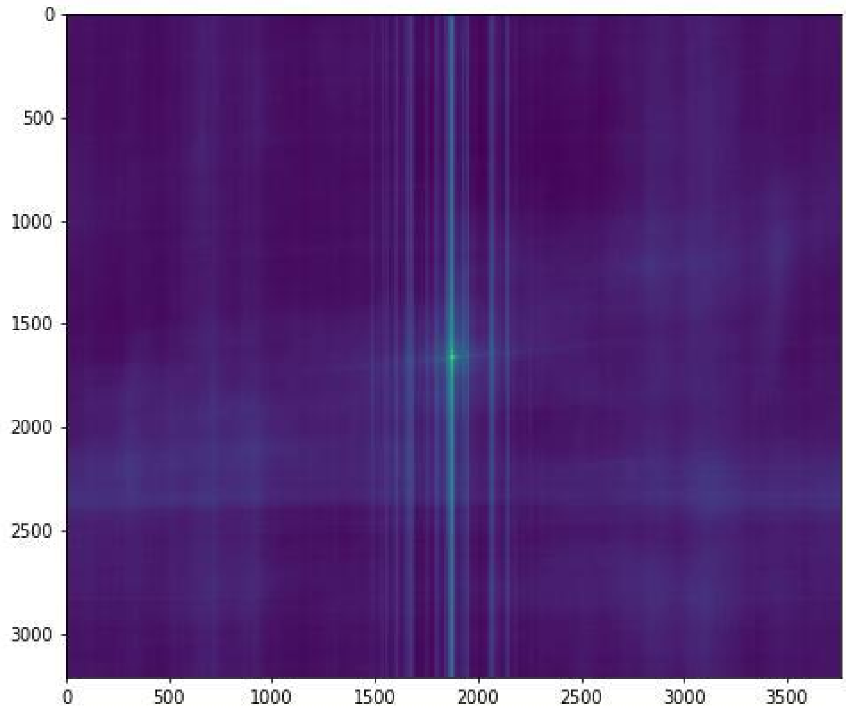
R -> B Processing	

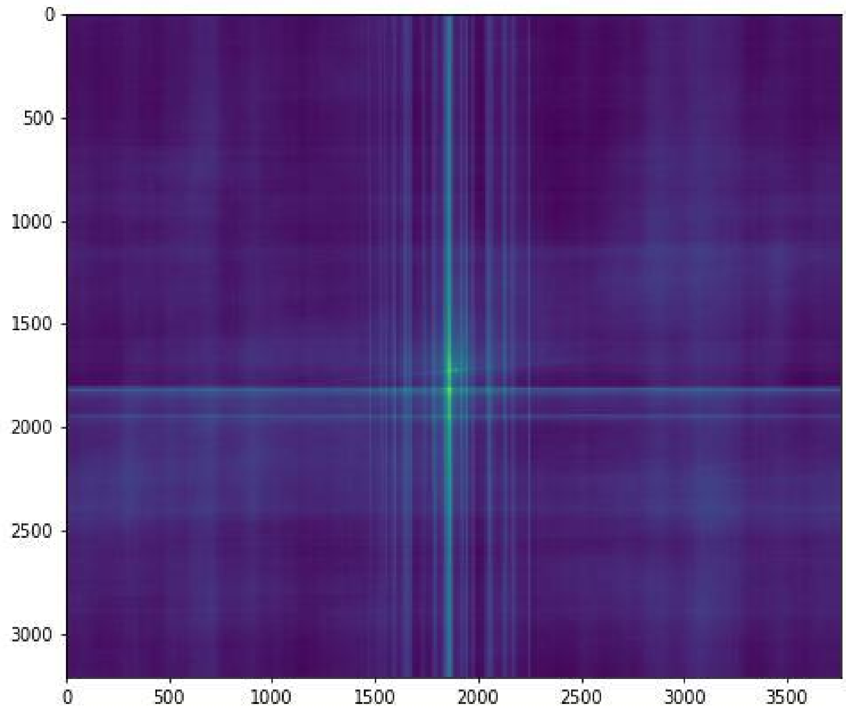
01657u.tif

Aligned	

G -> B No Processing	

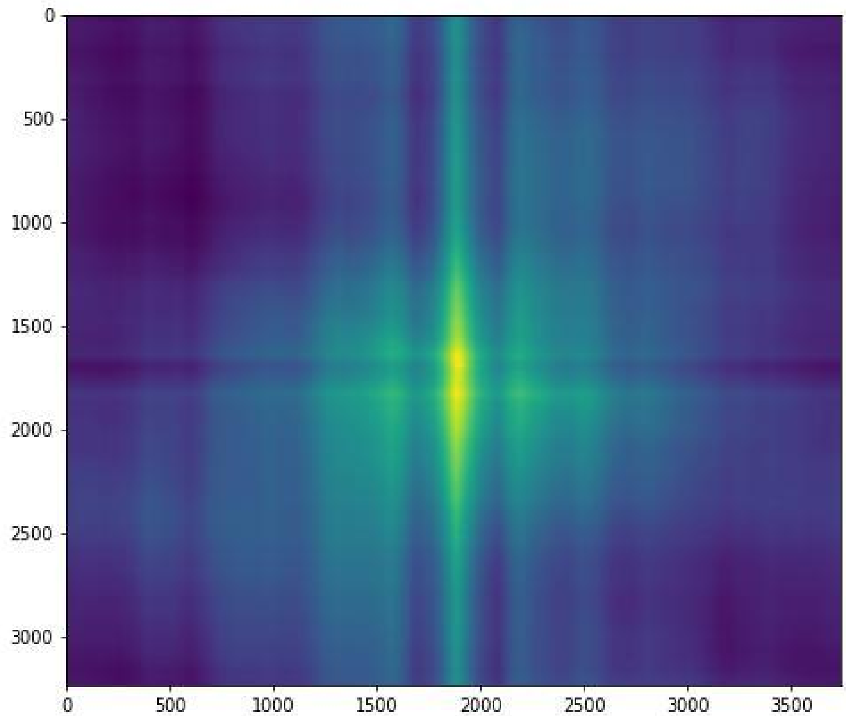
R -> B No Processing	

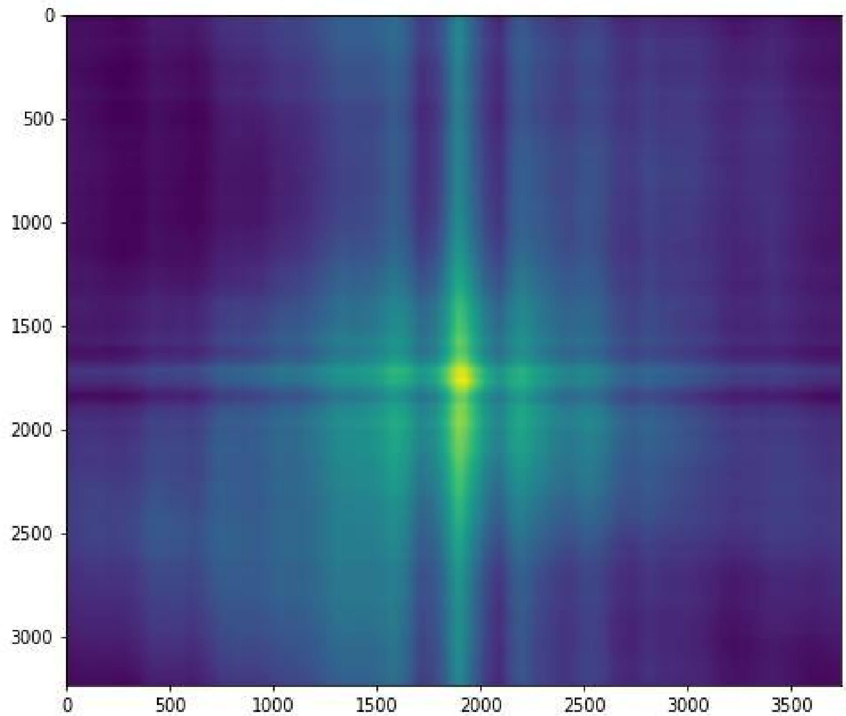
G -> B Processing	

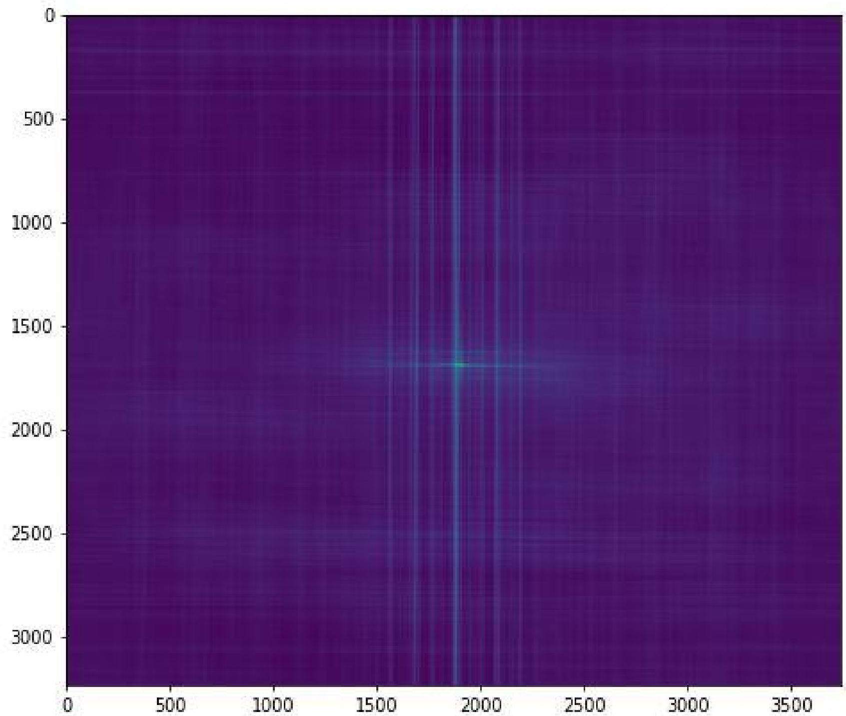
R -> B Processing	

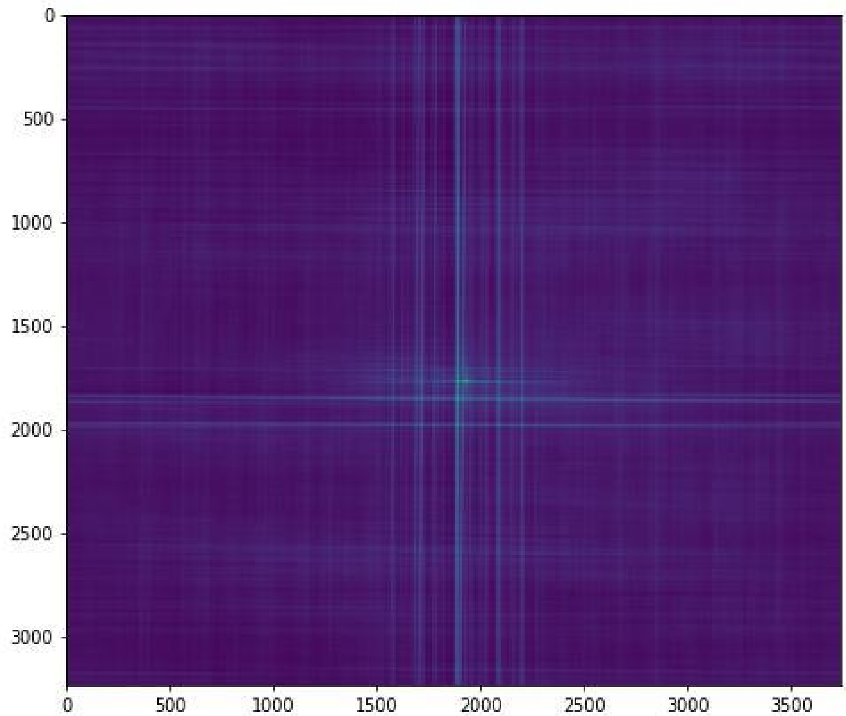
01861a.tif

Aligned	

G -> B No Processing	

R -> B No Processing	

G -> B Processing	

R -> B Processing	

C: Discussion and Runtime Comparison

Discussion of any preprocessing you used on the color channels to improve alignment and how it changed the outputs

For smaller images, I used the predefined `ImageFilter` to compute enhanced edges. It is sufficient to find the best alignment.

```
im_edge = im..filter(ImageFilter.FIND_EDGES)
im_edge = im_edge.filter(ImageFilter.EDGE_ENHANCE)
```

However, for larger images. This didn't quite work because too many noises are being generated by the filter. To make the filter more granular, I use `gaussian_laplace(b,` `sigma=2)` to find edges. I tried a few sigma values from 32 - > 2. I found that the small sigma generates the best alignment. Larger sigma value creates mis-alignment especially for the lady picture because it is harder to find the edges if sigma is too big.

Measurement of Fourier-based alignment runtime for high-resolution images (you can use the python time module again). How does the runtime of the Fourier-based alignment compare to the basic and multiscale alignment you used in Assignment 1?

Algorithm running time

- **Single Alignment**: 749.4506392478943 seconds
- **Multiscale Alignment**: 20.944703817367554 seconds
- **Fourier-based alignment:** 17 seconds.

Fourier-based alignment is the fastest and beats the **Multiscale Alignment** running time like 10%.

Part 2 Scale-Space Blob Detection:

You will provide the following for ***8 different examples*** (4 provided, 4 of your own):

- original image
- output of your circle detector on the image
- running time for the "efficient" implementation on this image
- running time for the "inefficient" implementation on this image

You will provide the following as further discussion overall:

- Explanation of any "interesting" implementation choices that you made.
- Discussion of optimal parameter values or ones you have tried

I use the same implementations for all images and here are their descriptions
Interesting Implementations:

- Instead of using squared laplace response to give a positive response, I use **absolute value**. The absolute value gives a larger response and I found it easier to determine the threshold value. e.g I use .05, .1, .15 for absolute value instead of 0.0005 from the squared response.
- I use from `skimage.feature import peak_local_max` to find the local maximum. The output **local_maximals** is a 2D boolean array with the same dimension as the image. If there is a peak at x,y, then the value will be 1. This makes finding max local value across scales much easier.

```
sigma = int(math.pow(sig, (i+1)*k))
lim = np.abs(math.pow(sigma, 2) * gaussian_laplace(im,
```

```
sigma=sigma))
local_maximals = peak_local_max(lim, min_distance=tmp,
indices=False)
```

- For downsample images, I need a way to “up-sample” the laplace response to the original image size. So that I can determine the value for the peaks. I found the resize function especially helpful. It will do the interpolation automatically which saves me the trouble to do it myself.

```
s = 1/k**(i+1)
rim = rescale(im, s, anti_aliasing=False)
lim = np.abs(math.pow(tmp, 2) * gaussian_laplace(rim,
sigma=sig))
rlim = resize(lim, (im.shape[0], im.shape[1]),
anti_aliasing=False)
```

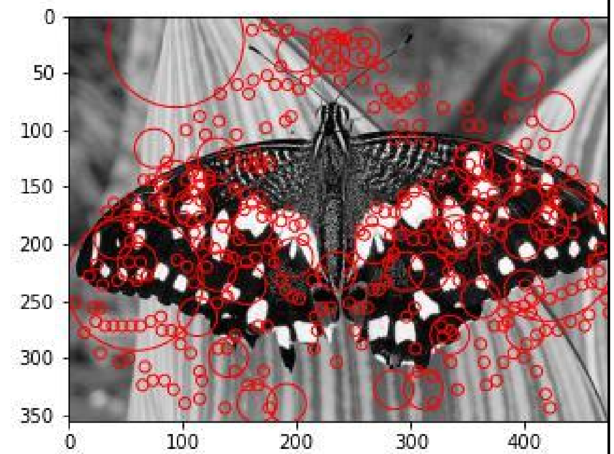
- For downsample images, I need a way to “up-sample” the `local_maximals` boolean array where 1 indicates a peak. I tried to use the resize function but I noticed that it creates extra 1 in the final larger boolean array due to interpolation. So I wrote my function to resize the boolean array.

I use the following method to determine the optimal value for each pic.

- Eyeball the response array and noticed most of value stays between .04 -> 1
- So I tried threshold values in a binary search manner; 1, .5, .25, .125, … to see which values create the best blob deletction.
- Record the best value that matches the sample output

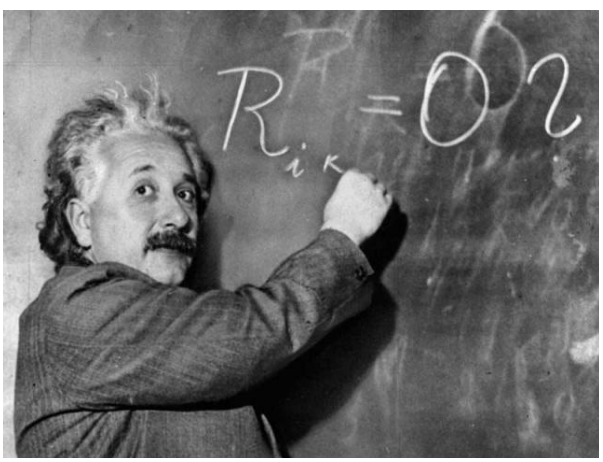
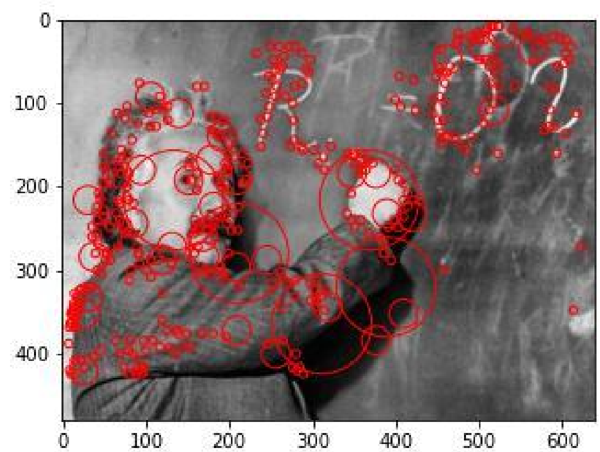
Example 1:

Original	Blobs Detected

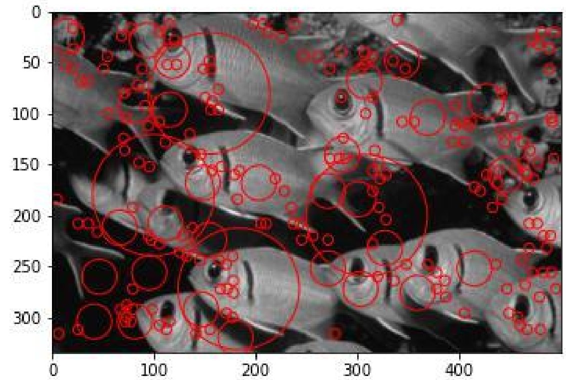
Efficient runtime: .33 seconds
Inefficient runtime: 8.58 seconds
Optimal parameters: .15

Example 2:

Original	Blob Detected
	

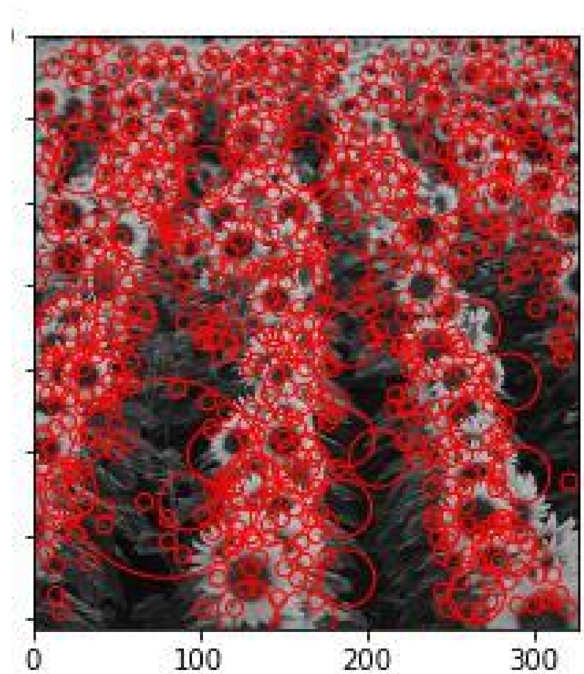
Efficient runtime: .54 seconds
Inefficient runtime: 15.22 seconds
Optimal parameters: .10

Example 3:

Original	Blob Detected
	

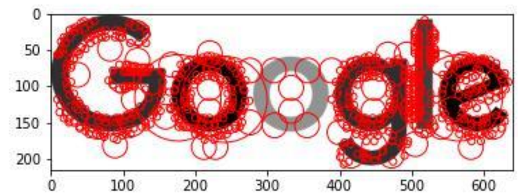
Efficient runtime: .33 seconds
Inefficient runtime: 8.06 seconds
Optimal parameters: .08

Example 4:

Original	Blob detected
	

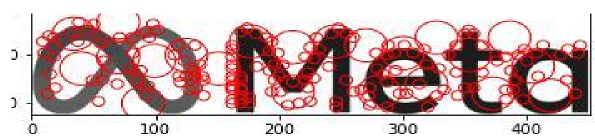
Efficient runtime: .25 seconds
Inefficient runtime: 5.28 seconds
Optimal parameters: .09

Example 5:

Original	Blobs Detected
	

Efficient runtime: .9 seconds
Inefficient runtime: 9.4 seconds
Optimal parameters: .1

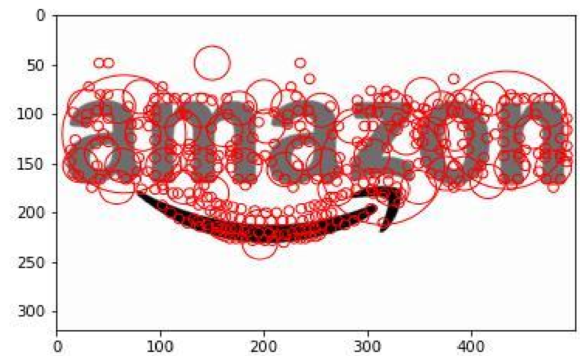
Example 6:

Original	Blobs Detected
	

Efficient runtime: .13 seconds
Inefficient runtime: 2.7 seconds
Optimal parameters: .1

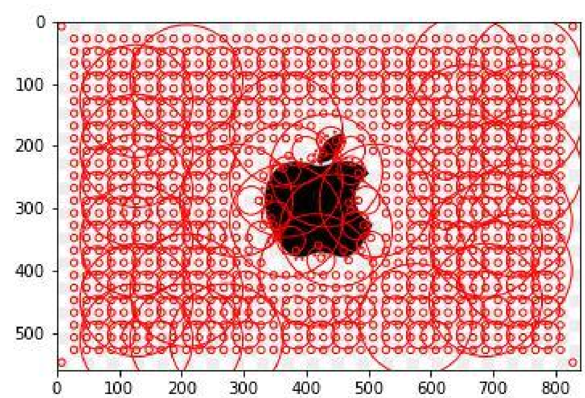
Example 7:

Original	Blobs Detected

Efficient runtime: 2.4 seconds
Inefficient runtime: 15.6 seconds
Optimal parameters: .12

Example 8:

Original	Blobs Detected
	

Efficient runtime: 1 seconds
Inefficient runtime: 23 seconds
Optimal parameters: .2

Bonus:

Blob-Detection Extra Credit

- Discussion and results of any extensions or bonus features you have implemented for Blob-Detection