Ryan Miller
u1067596
Professor Whitaker
Digital Image Processing

Project 3 - Fourier Transforms and Image Mosaicing

Introduction

In the third project of CS 6640, I experiment with the use of the Fourier transform for image registration and filtering. In this report, I will present four tasks that were assigned for Project 3. The first task uses the fast fourier transform to analyze images and their power spectrums while experimenting with the application of low pass filters in the fourier domain. In the second task, I build a fourier based routine that calculates the phase correlation between two images. In the third task, I analyze the peaks in the phase correlation results of task 2 to determine if images passed through the routine overlap. The final task is a culmination of the three previous tasks, I build an algorithm that takes in a set of split images and builds a mosaic by piecing the images back together. Each task includes its own python script which is included in the submission along with the report. I will briefly give the logic behind my algorithms in this report but mainly report on the results of each individual task. A variety of images will be presented in the report that can be reproduced by the python scripts. After all tasks have been presented and analyzed, credit will be given to other individuals who have made this analysis possible.

Task 1 - FFTs

In the first task, I use the built in numpy Fast Fourier Transform function to compute the Fourier transform of various images and present their power spectrum. I develop two different low pass filters in the Fourier domain and analyze their effects in the spatial domain.

In this task, I analyze three different pictures of bridges. Using the “Task 1.py” file, the user is asked to specify a picture to be analyzed and can choose any of the bridges for analysis. The Fourier transform is then found using the built-in numpy fft function. To make filtering easier, the fftshift function is also employed to move the zero and subsequent low frequencies to the middle of the image. Since the low frequencies are clustered in the middle of the image, a low pass filter can easily be designed to keep the middle array of pixels and suppress the other pixels. A cheap and easy way to construct this filter is to create a matrix of the same size as the image being filtered and pad the middle of the image with a circle of 1’s with specified radius, making all other pixel values 0. Since the low frequencies are clustered in the middle of the image, this constructed image will act as a low pass filter when applied to the image under analysis. Since we are in the Fourier domain, this filter can be applied via multiplication.

This is the first filter I constructed, and is referred to as “My Low Pass Filter” in the following images. Since there is a discontinuity in the fourier domain, we expect to see ringing in the spatial domain. A picture of my filter is presented in Figure 1. Note, the radius of the circle can be changed to alter the cutoff frequency of the filter.

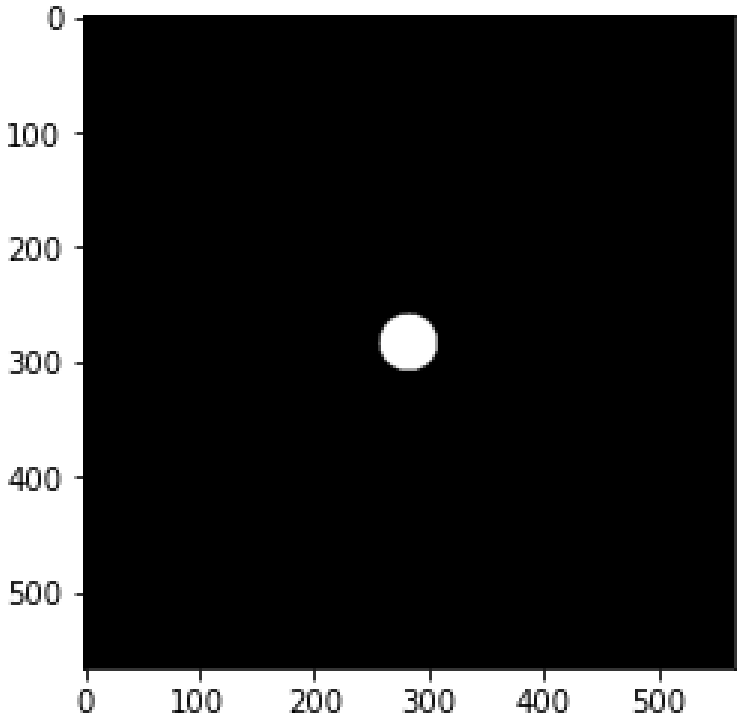

Figure 1: My Low Pass Filter, radius = 25

The second low pass filter I constructed and experimented with had a gradual decrease in intensity rather than the strict discontinuity as seen in Figure 1. This type of filter was chosen in an attempt to get rid of the ringing effects in the resulting filtered image. To create the smooth transition from high amplitude to low, I used the Kaiser Window function in numpy. This function will create an array of values using a special window function known as a Kaiser Window, where the steepness of the bell curve can be adjusted with the parameter alpha, as seen in Figure 2:

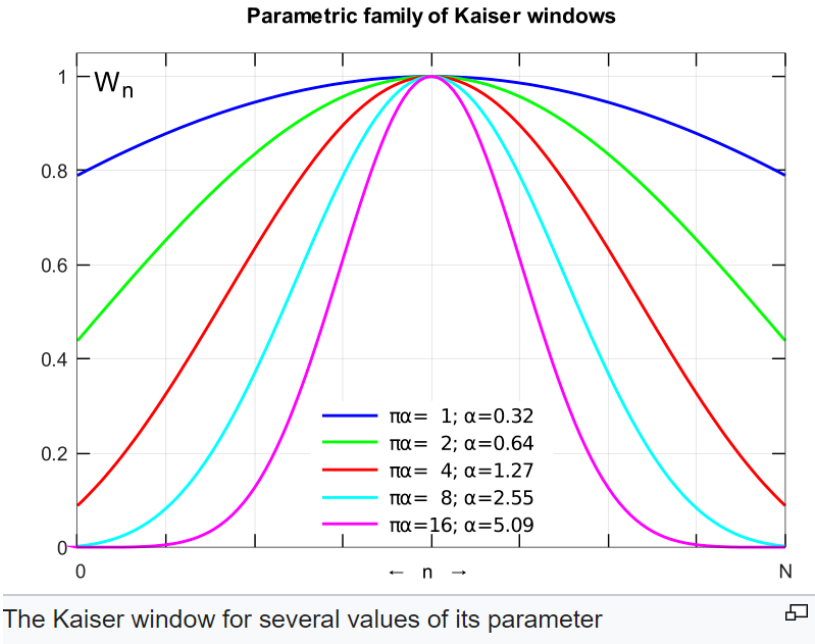

Figure 2: Kaiser Window (Credit: Wikipedia)

Many alpha parameters were tested and many will be presented in subsequent images, but it was found that an alpha value that was too large relative to the photo yielded blurry images after filtering. In the python implementation, a higher alpha value had the effect of decreasing the cutoff frequency. To make a 2D Kaiser Window, 2 separate 1D arrays were simply combined. For reference, see Figures 3 and 4 to see how changing the alpha parameter affects the filter geometry.

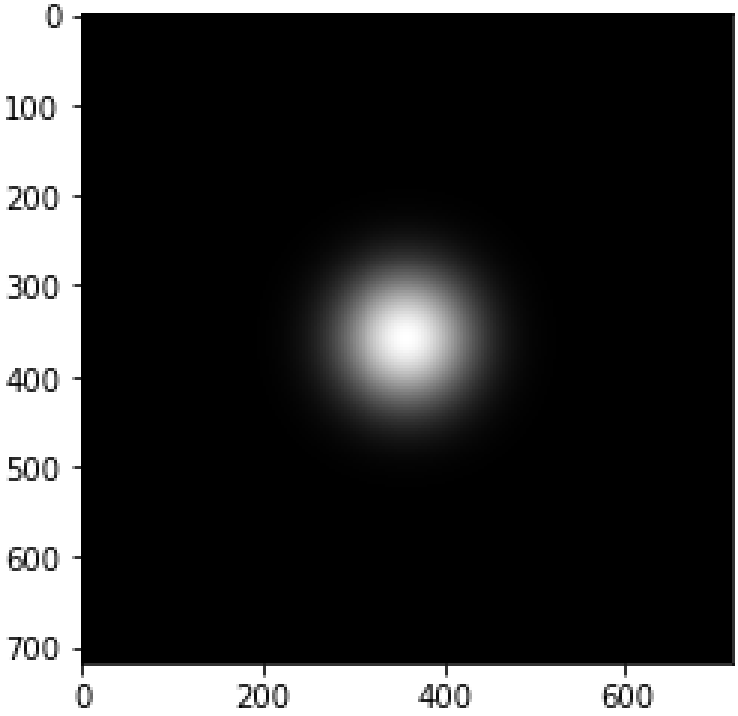

Figure 3: Kaiser Window, alpha = 100

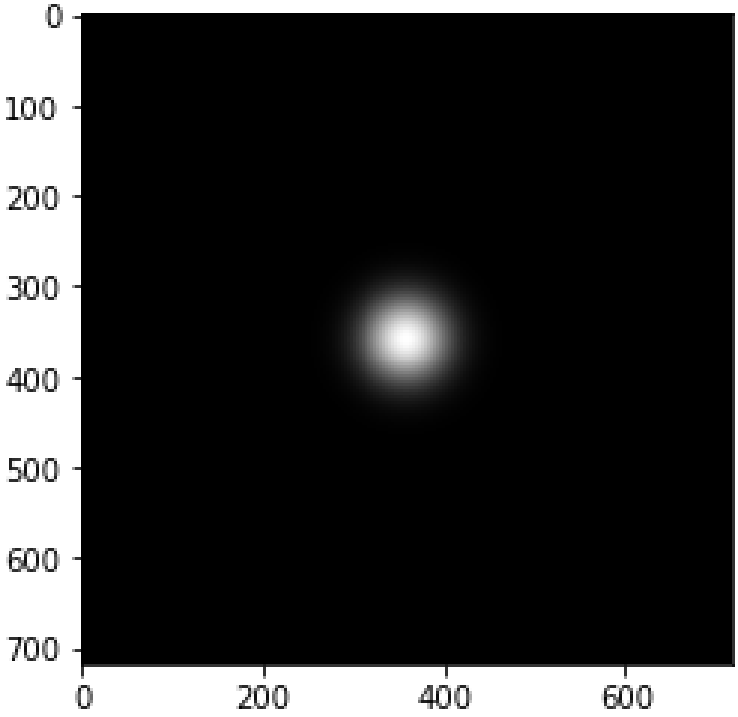

Figure 4: Kaiser Window, alpha = 250

As previously mentioned, the methodology for applying these filters is to multiply the fourier transform of the image under analysis by the filter in the fourier domain. Once the multiplication has been carried out, I took the inverse fourier transform to compare the original image against the filtered image. In the following Figures and results, I present figures with 6 subplots. Each figure will present the original image, the original fourier transform, the shifted fourier transform, the power spectrum, then the resulting image after applying both filters.

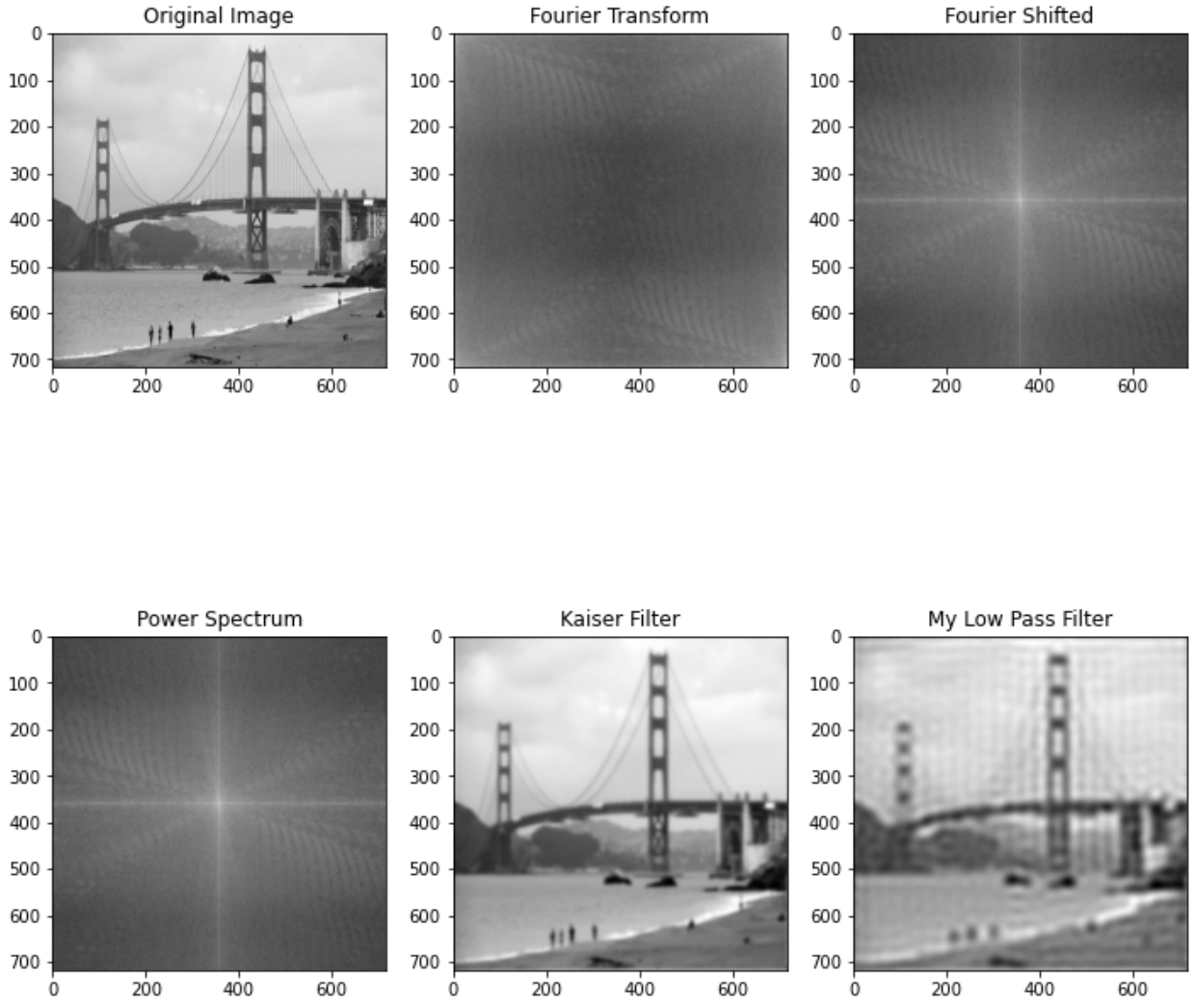

Figure 5: Results: Kaiser Filter Alpha = 250, Radius of "My Low Pass Filter" = 25

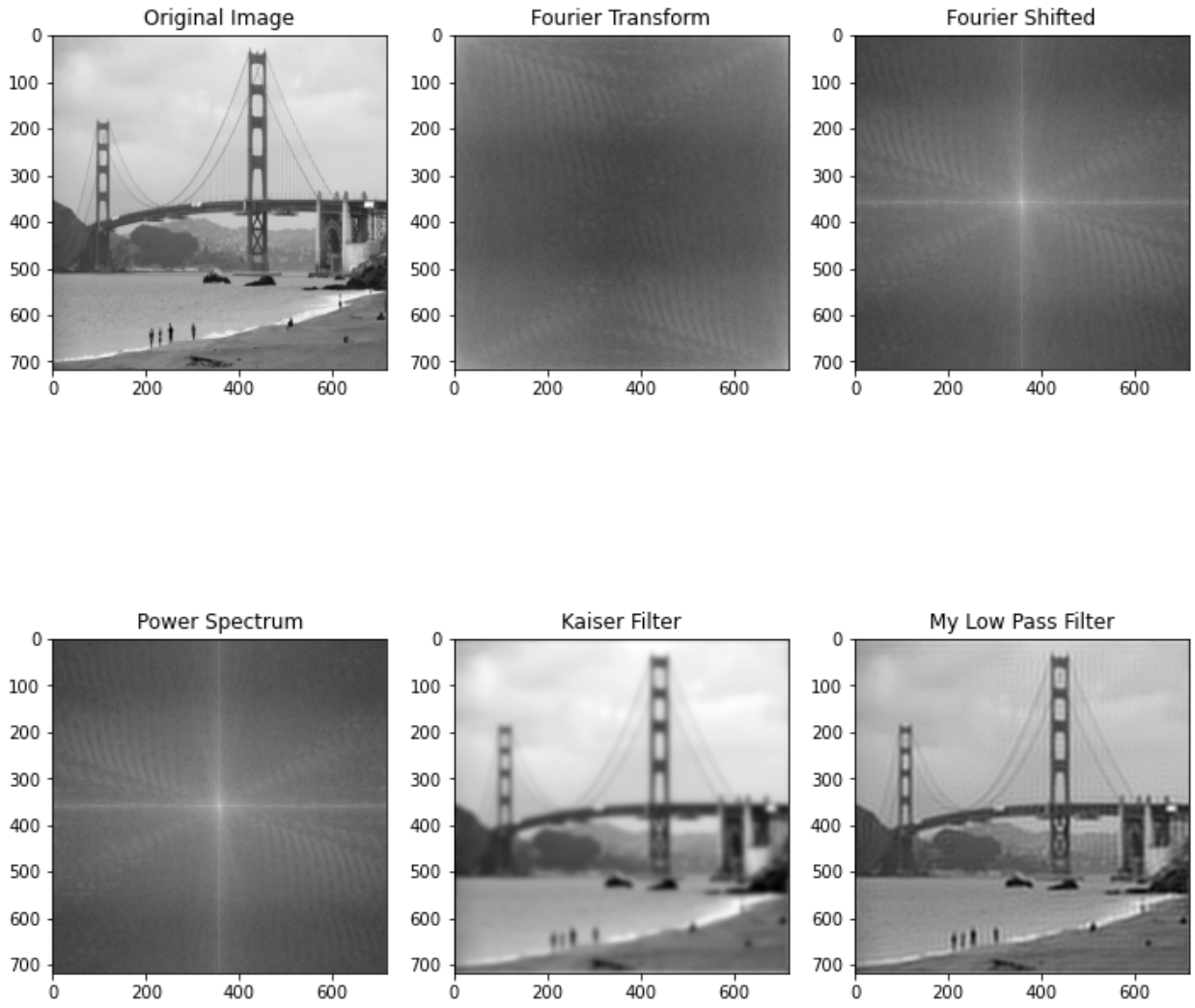

Figure 6: Results: Kaiser Filter Alpha = 500, Radius of “My Low Pass Filter” = 50

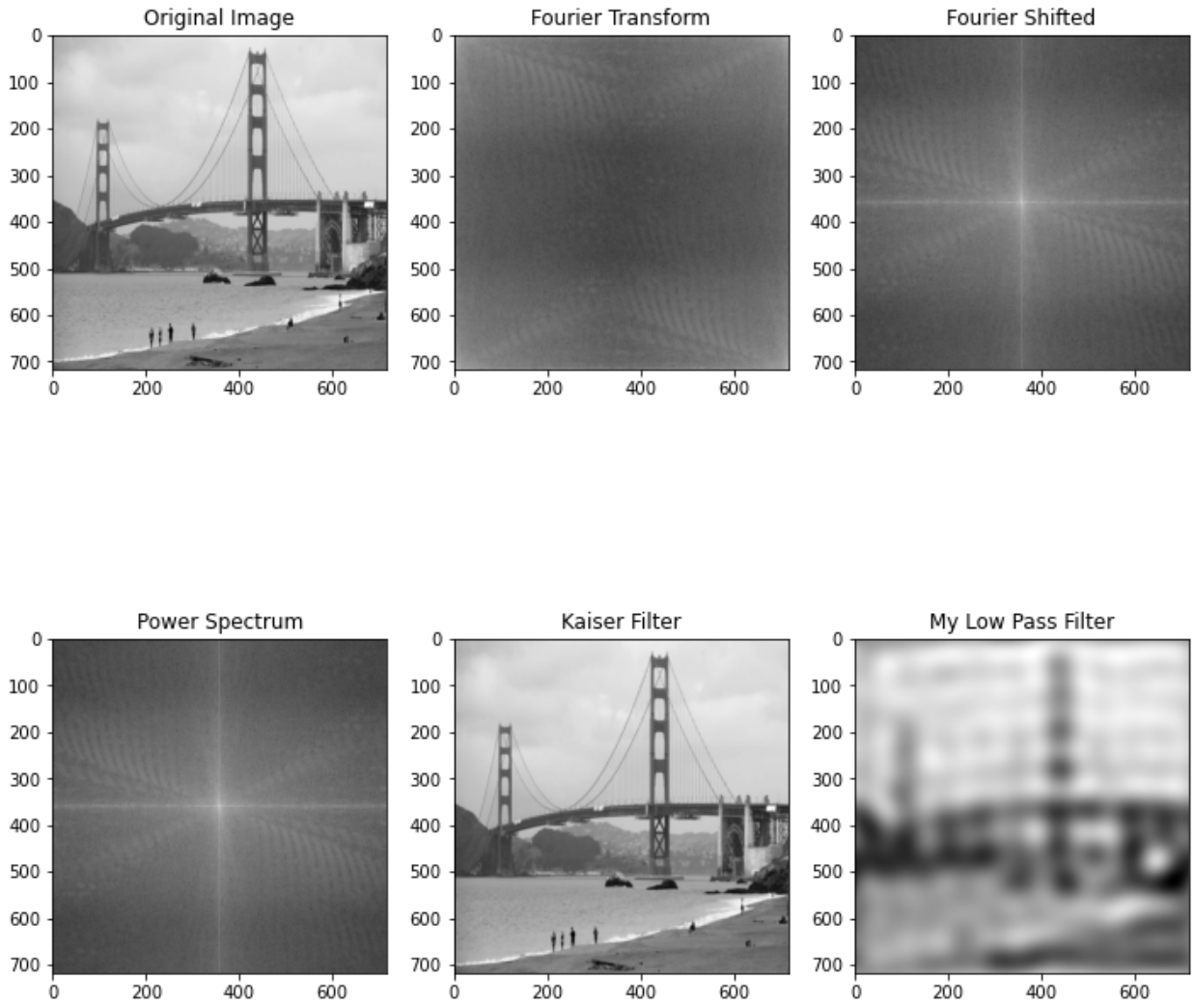

Figure 7: Results: Kaiser Filter Alpha = 10, Radius of “My Low Pass Filter” = 10

Let us first analyze the Fourier transform and power spectrum of the images. Since the original image is the same across Figures 5, 6, and 7, they share the same Fourier transform and power spectrum. The Fourier shifted image shows the Fourier transform of the image with low frequencies centered in the middle of the image. The low frequencies have relatively high intensity, which tells us that our original image has pixel values that change relatively slowly over the spatial domain. It must be noted that for the Fourier images presented above, the absolute value was taken in order to graph the results. Thus, to no surprise the power spectrum is very similar to that of the Fourier transform shifted.

In turning our analysis to the filters, we see if Figure 5 that my low pass filter introduces a great deal of ringing into the final image (as expected). From analyzing all the images, it appears that a smaller radius produces more ringing and more filtering and thus a blurrier image. Intuitively, this makes sense. As we decrease the cutoff frequency, we are doing more filtering, filtering out

a greater amount of “high frequency” signals and thus we get a blurrier image that less so resembles the original image. A great way to think about this is by thinking about what would happen if we were to take the limit of the cutoff frequency to infinity. In my implementation, this would be like taking the radius to infinity. If I were to do this, I would get the resulting original image back in the spatial domain after taking the inverse fourier transform. This is somewhat shown in FIgure 6 as the radius of the filter is 500 and we see the image is only just about 700 by 700. Thus, to no surprise, we do very little filtering and essentially get the original image back.

The Kaiser filter acts in a very similar manner but we notice that the ringing we were trying to suppress in the spatial domain is not seen for the Kaiser results. Since the gradient of the filter in the fourier domain is smooth, we do not get ringing in the spatial domain. In Figure 6, the alpha value of the Kaiser filter is higher and subsequently, we see the most blurriness. This constitutes the filter doing the most filtering and having a lower cutoff frequency than the other results. To ensure we do not only analyze results on a single image, I will present results on two more images but will not present written analysis as the analysis follows that of the subsequent images presented and analyzed.

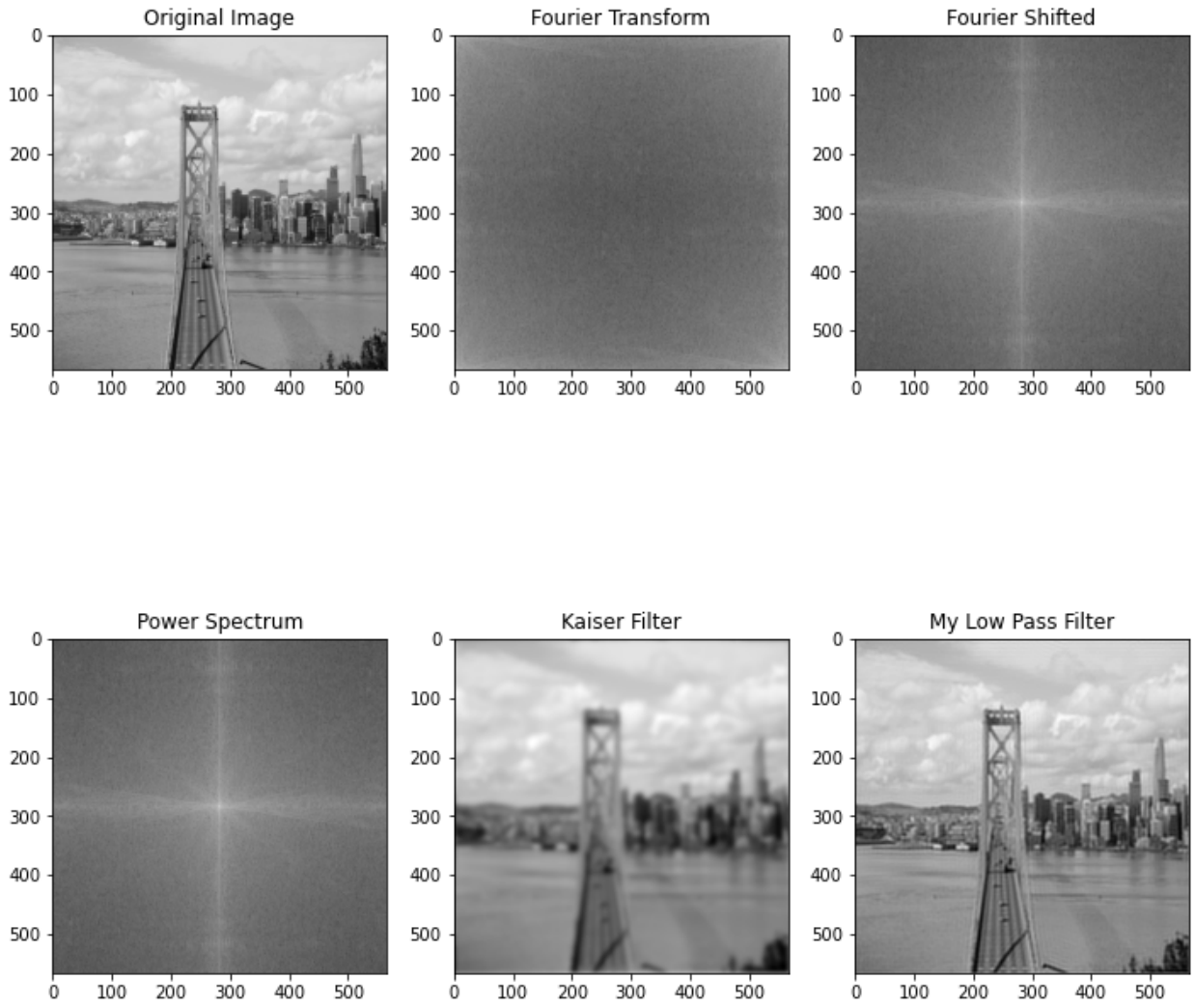

Figure 8: Results: Kaiser Filter Alpha = 350, Radius of “My Low Pass Filter” = 75

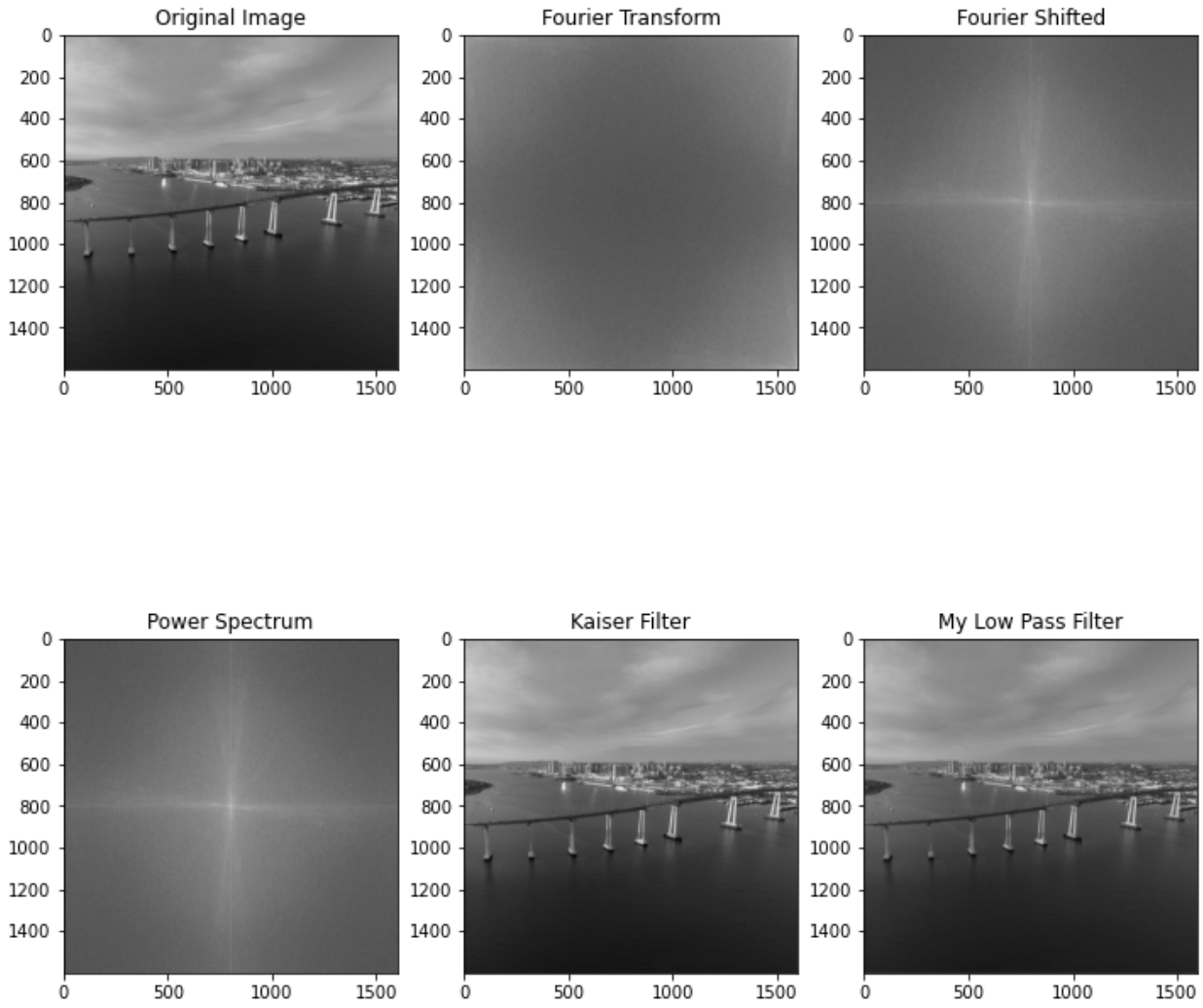

Figure 9: Results: Kaiser Filter Alpha = 1, Radius of “My Low Pass Filter” = 150

Task 2 - Phase Correlation

In the next task, I utilize key properties of the Fourier transform to find the phase correlation between images. From a theoretical point of view, the phase correlation estimates the relative translation between two images using the cross power spectrum. The cross power spectrum is found by converting both images to the Fourier domain and multiplying them with one image's conjugate result. Normalizing this result and taking the inverse fourier transform leaves us with an image which has a peak in values at the location of this translation. A low pass filter can also be applied to reduce various edge effects in the analysis.

$$p(x,y) = \mathbb{F}^{-1}\left[H(u,v)\frac{F^*(u,v)G(u,v)}{|F^*(u,v)G(u,v)|}\right]$$

Figure 10: Formula for phase correlation

Although a phase correlation function can be implemented using the above function, an issue arises that has to do with wrapping. Since phase correlation allows for wrap-around, the actual physical location of the peak in phase has some ambiguity in its true meaning. To work around this, the phase correlation can be found by finding the maximum value of the resulting matrix from the Equation presented in Figure 10 and then analyzing the individual 4 spatial subdomains to analyze which subdomain has the highest normalized correlation. This subdomain with the highest normalized correlation is the true orientation of the overlap. A visual image is presented in Figure 11 highlighting this idea.

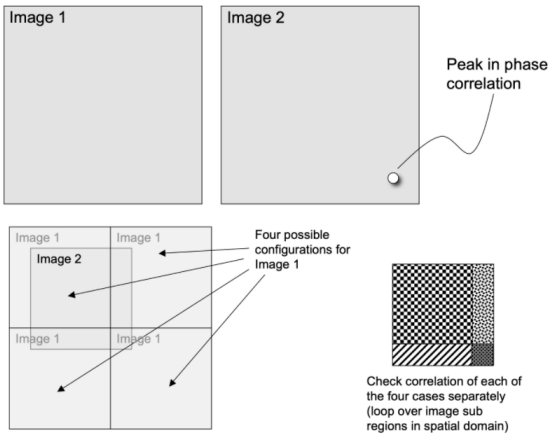

Figure 11: Visual representation of wrapping

Although we have not yet implemented the mosaicing aspect of this project where this idea of wrapping plays a major role in the ability to perform the task, I implement this portion of the project in this task, Task 2, so that I can utilize the function defined for later parts of the project.

To complete this task in python, I first ask the user for the input they would like to analyze. I have prepared 3 sets of 2 images where 1 image is translated relative to the first. Both of these images are then passed through a phase correlation function. In this function, the operations presented in Figure 10 are carried out: the fourier transform is taken of both images, then the conjugate for one image is taken and multiplied with that of the other before taking the normalized value. The Kaiser Low Pass filter is applied in the fourier domain to try and suppress noise in the resulting image that may have some minor effects on our finding of the true peak. Now the resulting phase correlation matrix has been formulated, but as presented before, we do not know if wrapping is occurring or not. To test for this, four subdomains are defined, called in the code as “TopR”, “TopL”, “BotR”, and “BotL”. In each case, the original inputs are resized and shaped to match the portion of the subdomain in question.

The normalized correlation is then taken between the input images for that particular subdomain and stored in a matrix. At the end of the conditional statements splitting the subdomains, the best fitting subdomain is found by taking the maximum overall correlation value. Based on the index of this value, the wrapping effect is defined by the strings presented earlier “TopR” etc. At the very end, the resulting phase correlation matrix, x value for the peak, y values for the peak, and location (wrapping details) are output. In the following images presented for results, since it is hard to makeout a single pixel/group of pixels with high values, the corresponding x and y values of the peak and location of wrapping are also output.

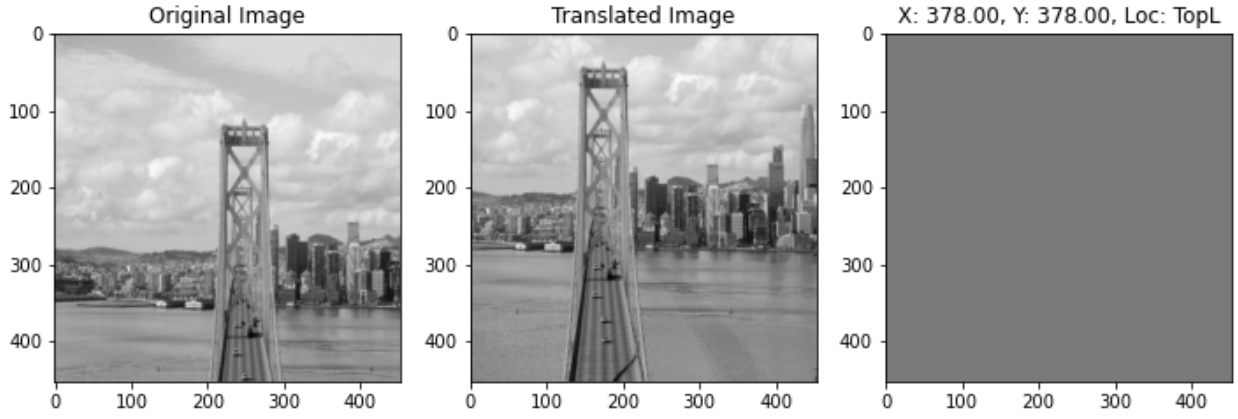

Figure 12: Phase Correlation results for Bay Bridge images

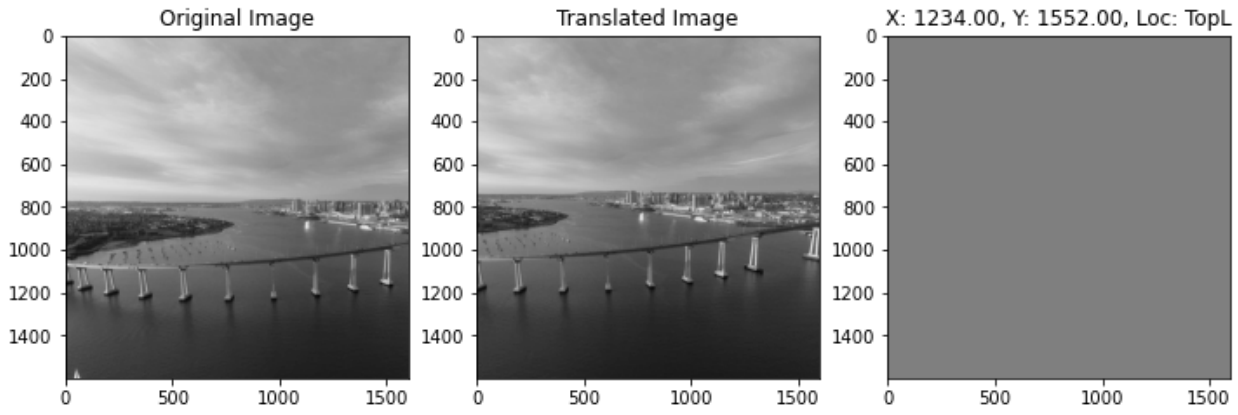

Figure 13: Phase Correlation results for Corona Bridge images

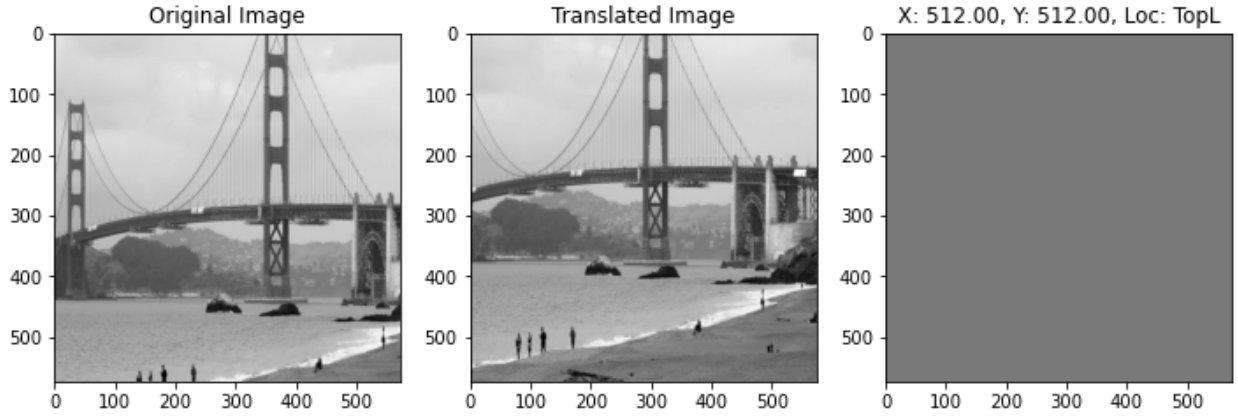

Figure 14: Phase Correlation results for Golden Gate Bridge images

It must be noted that since I had to physically crop the images myself to result in 2 images with same size that were shifted, the directions of the shift are very similar and thus “TopL” is the output location for all of the images. In the mosaicing example, we will see how changes in this location play into the change in location of the one translated image relative to another.

Task 3 - Peak Finding

In the next task of the project, I utilize previous work in the other tasks to develop a set of checkers to test if two images overlap or not. From a hint from the professor in the handout, I utilize the magnitude of the peak found using the phase correlation equation implemented in Task 2. Although I originally thought that this would be enough to accurately characterize the overlap (or lack thereof) of all of my images in question, I quickly found I was mistaken. Of the 3 different sets of images tested, for one set of images, even though there was sufficient overlap, the peak magnitude fell to a value that could not be distinguished from the non overlapping images. Thus, I knew I needed another criterion to base this decision off of. After working through the mosaicing algorithm, I realized that two images that overlapped had a very high normal correlation value in the subdomain of overlapping segments. Thus, I used a very similar function that I implemented in Task 4 and Task 2 to take in two images, split them up based on the separate possible wrapping orientations after having found the translation in x and y via phase correlation.

After splitting up the subdomains and calculating the normalized correlation between the subdomains, I realized that if images overlapped, these normalized correlation values for that specific subdomain chosen to be the right orientation were very high. Thus, instead of only using the magnitude of the peak correlation, I also implemented a condition for if the certain overlapping subdomain normalized correlation was above a certain threshold. With these conditions, I was able to accurately predict if all of my sets of images overlapped or not. To find the right threshold value for the normalized correlation, I ran a series of test images through the conditionals to see what typical values of overlapping and non-overlapping images produced for the normalized correlation. It was found that a normalized correlation value above 0.7 only came from overlapping images. After setting this threshold, I tested the conditions by passing my original set of images through the script. The script works by asking the user for a first image and then a second image. The first image can be chosen based on what the user would like to see, but the second image chosen dictates whether the two images have overlap or not. If the same architectural feature is chosen in the second image as the first, the images overlap. The images being analyzed are bridges, thus, if the user selects “Golden Gate” for both the first and second image, the two images selected overlap. With 3 sets of images, I plan an experiment to test all possible outcomes. First, we test to ensure the overlapping images are found to overlap. The output for this task is simply a message telling the user the images overlap or do not overlap, thus I will present the results in tabular form:

Image 1	Image 2	Overlap?	Model Prediction	Correct (Y/N)
1 (Golden Gate)	1 (Golden Gate)	Yes	Yes	**Y**
1 (Golden Gate)	2 (Bay Bridge)	No	No	**Y**
1 (Golden Gate)	3 (Corona Bridge)	No	No	**Y**
2 (Bay Bridge)	3 (Corona Bridge)	No	No	**Y**
2 (Bay Bridge)	2 (Bay Bridge)	Yes	Yes	**Y**
3 (Corona Bridge)	3 (Corona Bridge)	Yes	Yes	**Y**
3 (Corona Bridge)	1 (Golden Gate)	No	No	**Y**
3 (Corona Bridge)	2 (Bay Bridge)	No	No	**Y**

Table 1: Experimental Results for Task 3 - Finding Overlap

For reference, the images being analyzed in Table 1 are presented in the following Figures.

Figure 15: Bay Bridge used in first image pick

Figure 16: Bay Bridge used in second image pick

Figure 17: Corona Bridge used in first image pick

Figure 18: Corona Bridge used in second image pick

Figure 19: Golden Gate Bridge used in first image pick

Figure 20: Golden Gate Bridge used in second image pick

It is clear that the conditionals and thresholds I have set up work very well for classifying if images overlap or do not overlap. It is verified to work.

Although it works as intended, one must continue to test the model if a different dataset was used. This is a very common idea in machine learning, but there is a chance of overfitting where the model set up to predict overlap or not is too highly fit to the data that I have used and could possibly not be a great model for predicting overlap for a different dataset. Thus, the model must continue to be dynamic as it is put into different situations with different datasets.

Task 4 - Mosaic Building

Putting all of the previous tasks together, I complete the last task: creating an Image Mosaic. To create a mosaic, we are given a set of images that have some overlap in some orientation. The algorithm for creating the mosaic is as follows: take in all images a set an image to be the anchor in a blank canvas that is large enough to handle edge cases (all images line up in a single column or row). Cross correlation is then used to compare values for image pairs after alignment to see best possible overlapping images that are in the active set. The anchor image is set into the active set and iteration is carried out to find the best non-active matching image to add to the canvas. Previously defined phase correlation equations are used to find the x and y values of the translation which allow for the correct placement of the image if the global coordinates are kept track of during each iteration.

There are a few key points to this algorithm (the weeds) that I must highlight before presenting results. There are essentially two coordinate frames to keep track of in these examples. First, the coordinate frame of the image to be input into the canvas, and then that of the canvas itself. To input a new image into the canvas, I used a simple nested for loop to iterate across the image starting from the top left corner of the image and working down to the bottom right. Thus, after new images are selected to be entered, my global coordinate frame must move to that of the new top left corner of the incoming image so that my canvas coordinates are correct for the incoming image. This subsequent translation must also be taken into account to get the local incoming image coordinates. For example, you might be starting to put a new image in the x,y coordinate of 1200, 1200 in the canvas but this might correlate to 0,0 for the individual image. These small details are simply bookkeeping but must be taken into account when considering the logic behind the algorithm.

A low pass Kaiser Window filter, as described previously in this report, was also used in the phase correlation to filter out higher frequency noise. I will go more in depth into this detail later in the report and report on the effect of changing the Kaiser Filter alpha parameter on the final results. The final canvas was taken to be very large, accounting for worst possible arrangement cases, but was subsequently cut down for visual purposes. The mosaic algorithm was tested on two data sets: cells and a bridge. The cell data set was provided by the Professor whereas I constructed the bridge data set by taking a bridge picture and breaking it down into 8 different photos. The results of both data sets are presented below:

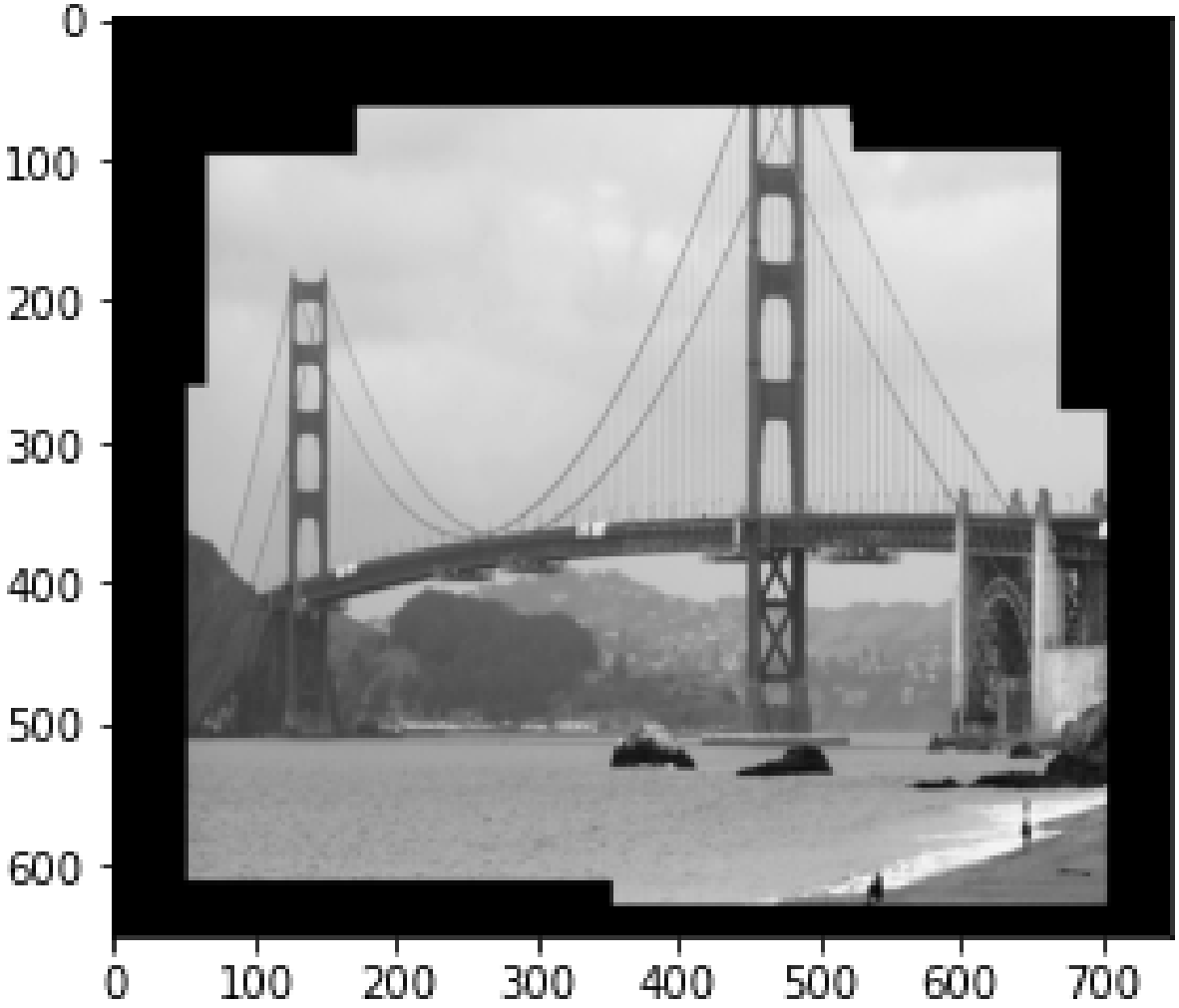

Figure 22: Golden Gate Bridge constructed from 8 different tiles

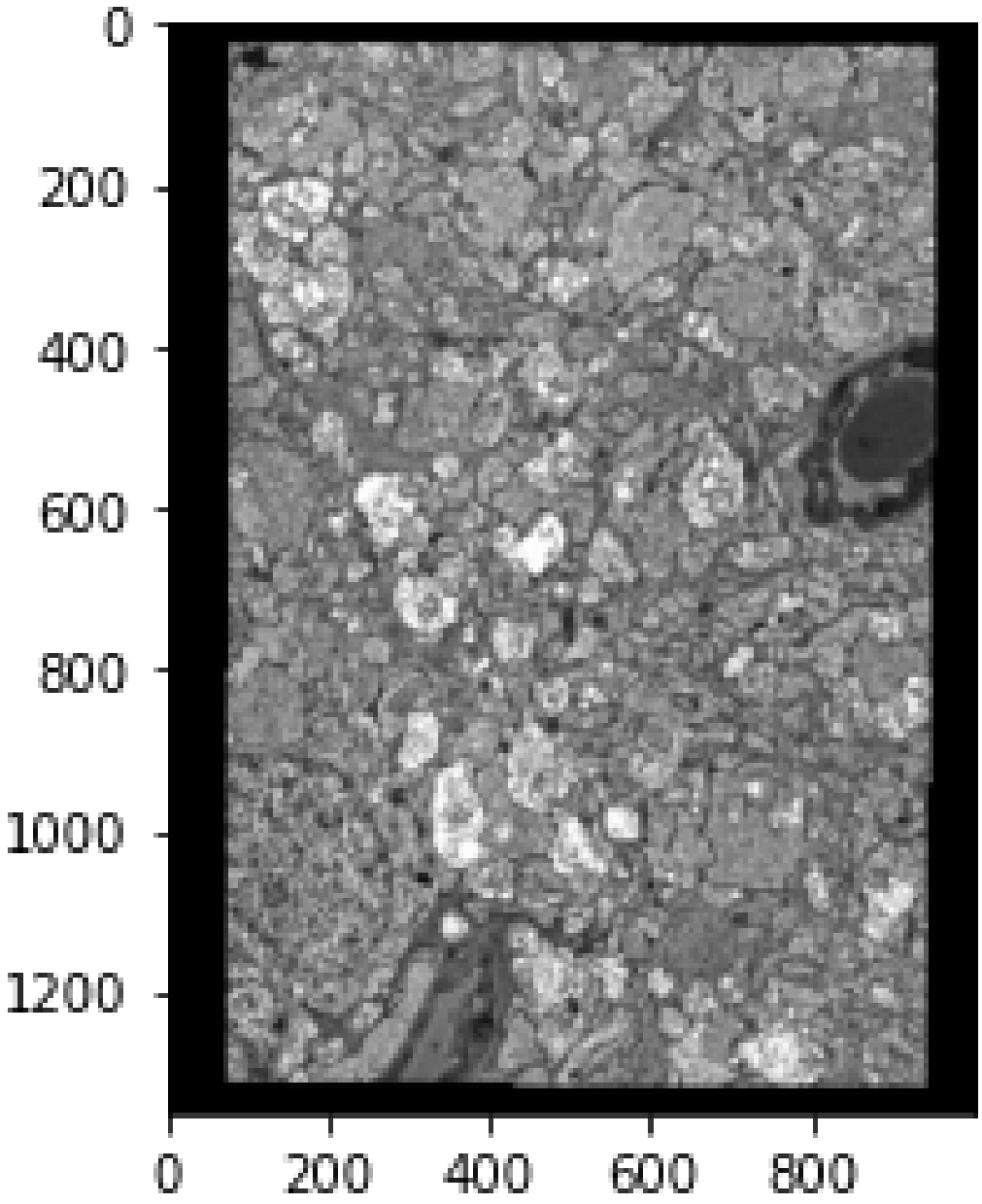

Figure 23: Cells constructed from 6 different tiles, Kaiser Filter, alpha = 100

As shown from FIgures 22 and 23, the algorithm works as intended and accomplishes the goal on two different data sets. Note, both of these examples use the Kaiser Filter in their phase correlation functions with alpha set to 100. If we change this parameter to 400, we observe no noticeable change in the Golden Gate Bridge Mosaic. It was also observed that no noticeable change was seen with that of the filter set to a lower value. This makes sense intuitively as the filter is just helping us find the peak and is not presented in the final image.

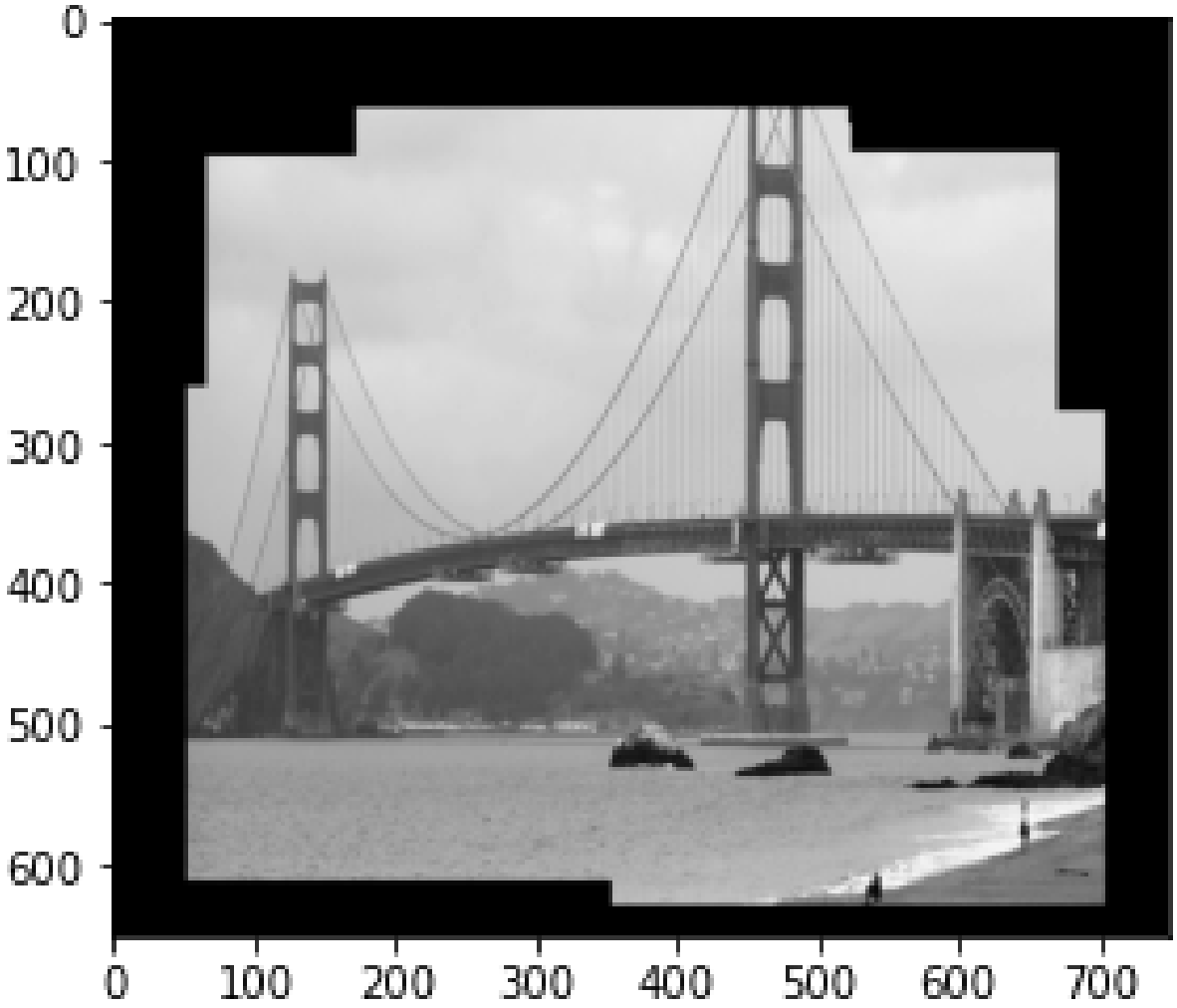

Figure 24: Golden Gate Bridge Mosaic, Kaiser Filter alpha = 400

Now, if I were to use my own filter as described in earlier tasks that has a hard cutoff frequency, there is potentially a point where the filter would filter out the frequencies needed to identify the peaks in the phase correlation which would lead to errors in the final mosaic construction.

Interestingly enough, with a large enough Kaiser filter alpha value, it does indeed mess up the final mosaic. I present Figure 25 where the Kaiser alpha parameter is set to 250 (up from 100 as shown in Figure 23), and the algorithm incorrectly puts back together the mosaic. As mentioned before, this is most likely due to the added filtering out of information needed in phase correlation.

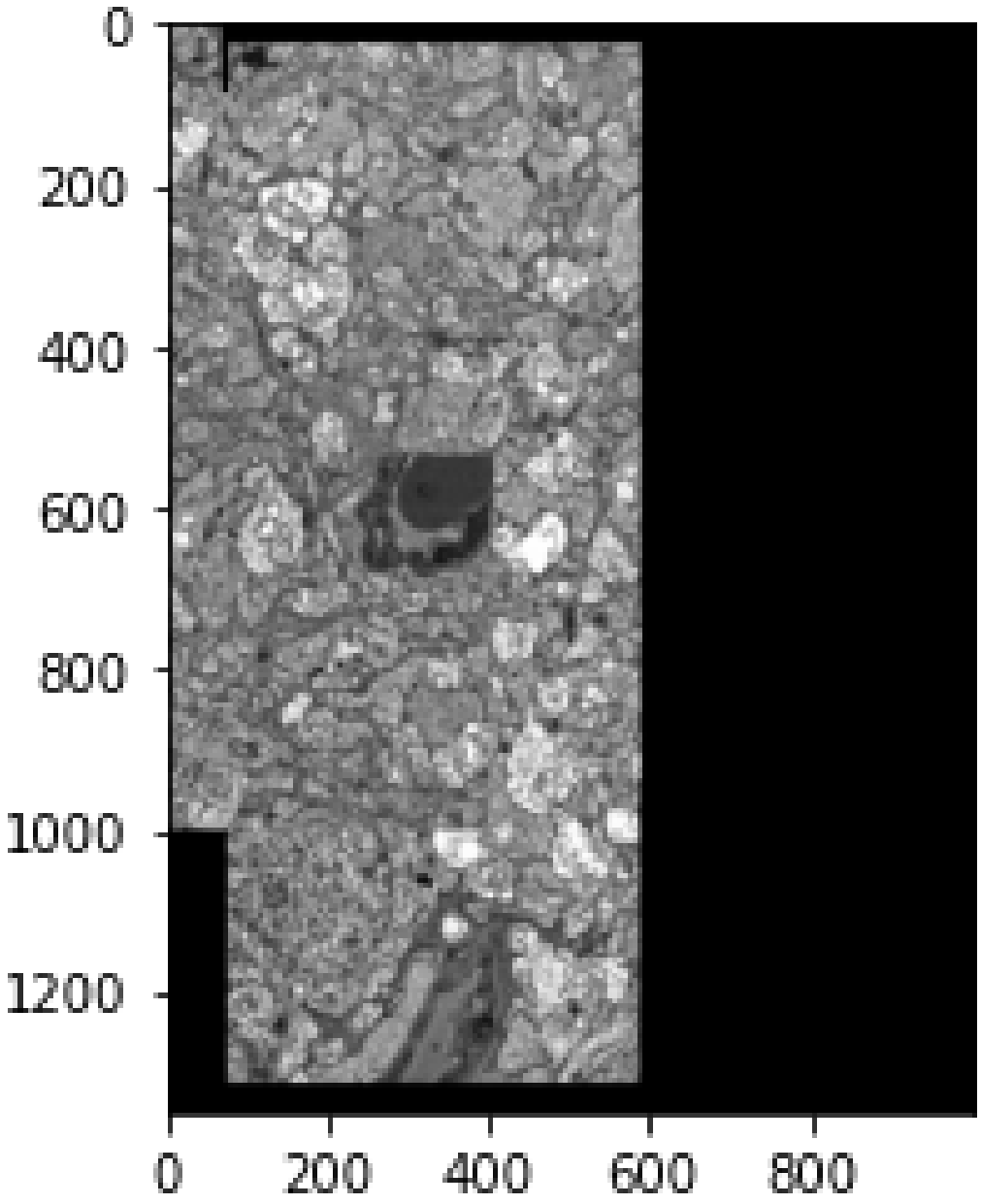

Figure 25: Cells, Kaiser Filter alpha = 250

SImilar results were seen for the usage of “my own filter” as defined in earlier parts. With a small radius, 25, the filter actually does more than we want and the algorithm incorrectly pieces the tiles back together: Figure 26.

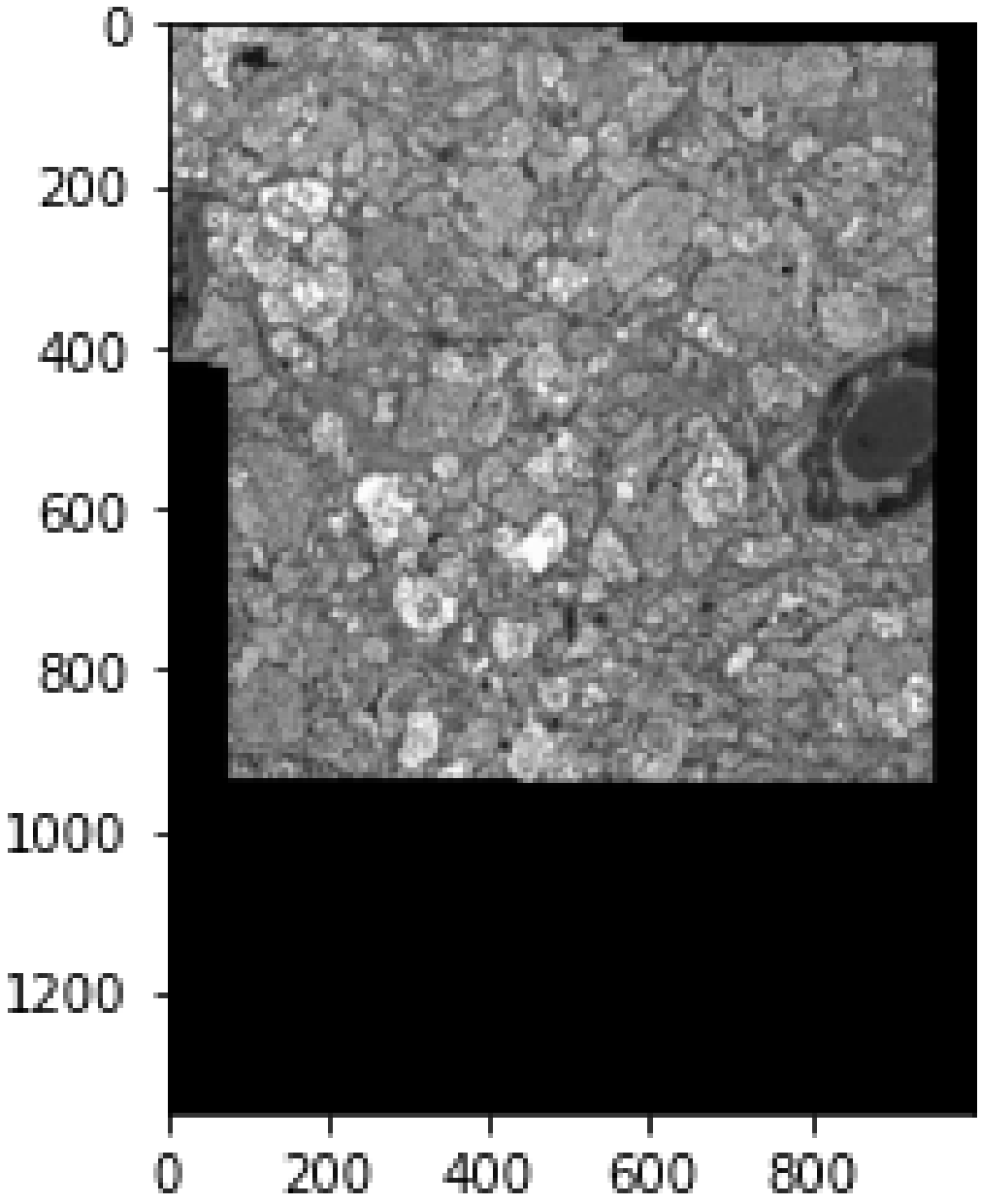

Figure 26: Cells, “My own Filter” Radius = 25

In the end, the algorithm worked as intended and we were able to piece together mosaic tiles but the effect of filtering had to be kept in mind to ensure we did not suppress the signals that we needed to use to analyze spatially. Again, in FIgure 27, I present the final cell mosaic for reference with sufficient amount of filtering.

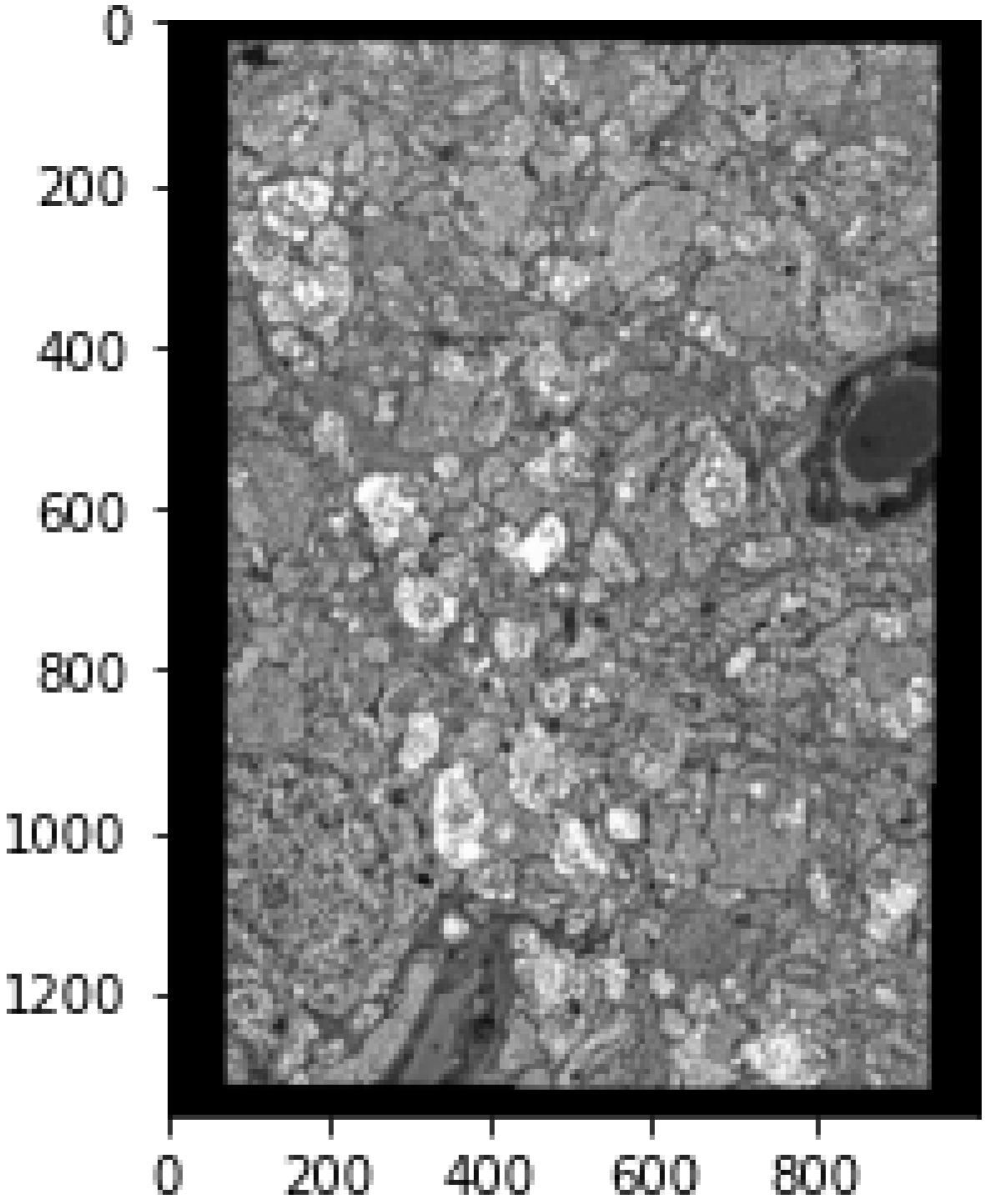

Figure 27: Cells constructed with acceptable amount of filtering

Conclusion

This project introduced the Fourier Transform in the realm of image processing and helped us analyze the theoretical concepts by carrying out a multitude of smaller tasks with basis in the Fourier domain. A constant theme of this semester has been image filtering and this project allowed us to experiment with filtering in the fourier domain and its effects in the spatial domain using the inverse fourier transform. We also utilized the power spectrum and correlation to analyze areas of high frequency between multiple images and the meaning behind this peak in magnitude. In the end, we pieced everything together and developed an algorithm that takes in a set of image pieces and accurately assembles the original, unsplit image, using methodologies and functions with roots based on the Fourier transform.

Credit

I must credit the following people and online websites for making this project possible:

- Dr. Ross Whitaker for teaching me the theory and giving me hints behind the mosaic algorithm
- TA Tushar for helping walk me through flow chart of the mosaic algorithm before I started
- Image cropping online tool: Download file | iLoveIMG
- https://stackoverflow.com/questions/30230592/loading-all-images-using-imread-from-a-given-folder
- https://stackoverflow.com/questions/2771021/is-there-an-image-phase-correlation-library-available-for-python
- https://numpy.org/doc/stable/reference/generated/numpy.kaiser.html
- Kaiser window - Wikipedia