ID:80585887
CS 5363
Dr. Olac Fuentes

I. Problem Description

For Lab 1 our task was to implement a series of operations with three different datasets that were given to us. Among those operations were the Histogram equalization, log correction, sigmoid correction, and gamma correction. In addition, we were also tasked with aligning images based on some predefined area that defined how should an image be shifted to produce more distinguishable/less blurry areas of high intensities in the image. Furthermore, our task also required to develop a heuristic approach to define the quality of an image for the alignment computations.

II. Algorithms Implemented

Problem 1

For this problem the first thing to do was basically to precompute the average of every dataset around the zeroth axis, since the images were sorted in the following fashion, (index_of_image, row_size, col_size). For this problem the algorithm basically involved adding every image into the same array. So, while the code could be expressed as a concatenation of three while loops that iterate over the whole dataset in this case I used the code provided by doctor Fuentes.

```
average = np.mean(dataset,axis=0)
```

Problem 2

For this problem we basically had to implement an algorithm to align the dataset, in order to do so I first pre-computed (as suggested by Dr. Fuentes) the brightest area of a certain size that was found the average image. By using the coordinates of the corner where such area started in the average image I proceeded to create an image that had the dimension of every image in the dataset, such image contained zeros as pixel values at every entry. Then, for every image in the dataset I computed the coordinates of the bounding box of the brightest area of same size as that which was found in the average image (Step 1). Using as a reference the coordinates of the brightest area of the average image I applied a shift of each of the images so that their individual brightest area matched coordinate-wise the coordinates of the brightest area in the average image.

By shifting each of the images inevitably there were areas of the original image that were out of bounds and such areas were “ignored”. Seen in another way this algorithm basically aligns the image and crops the areas that via alignment fall on indices that do not exist. As cautionary conditions I also included conditions that stop

the shift of an image if by too many pixels (I choose this threshold myself), this is in order to prevent the alignment of images whose brightest areas are very far away from the average image's brightest region. To even be more cautious this measure is applied independently to each of the axis, so if the shift in the X(cols) is too extreme but the shift in Y(rows) is not I would still ignore such image and continue aligning the rest of the dataset. This, as seen if the code is executed, reduces the number of images aligned from the what is pass as parameter to indicate the number of images to align. Finally, in order to scale down the values that are reached by adding hundreds of images I divide the image by the maximum argument in the array, this is so that when there transformations were applied no "weird" results were observable because of the values of the entries.

The implementation looks as follows:

```
def align(images, coordinates_average,size=100):
    print(images.shape)
    container = np.zeros((images.shape[1],images.shape[2]))
    x_avg,y_avg = coordinates_average
    print(coordinates_average)
    count=0
    for i in range(len(images)):
        x,y = compute_brightest_area(images[i],size,size)
        diff_x, diff_y = x_avg[0]-x[0],y_avg[0]-y[0]
        if diff_x == 0 :
            diff_x -= 1
        if diff_y == 0:
            diff_y -= 1
        if abs(diff_x) > 35 or abs(diff_y) > 35:
            continue
        if diff_x < 0 and diff_y < 0:
            container[:diff_y,:diff_x] += images[i,abs(diff_y):,abs(diff_x):]
            count+=1
        elif diff_x > 0 and diff_y > 0:
            container[diff_y:,diff_x:] += images[i,:-diff_y,:-diff_x]
            count+=1
        elif diff_x < 0 and diff_y > 0:
            container[diff_y:,:diff_x] += images[i,:-diff_y,abs(diff_x):]
            count+=1
        else: #diff_x >0 and diff_y < 0
            container[:diff_y,diff_x:] += images[i,abs(diff_y):,:-diff_x]
            count+=1
    print(f'Aligned {count} Images')
    plt.show()
    return container/np.amax(container)
```

Problem 3

For this problem I implemented one of the ideas we collectively developed in class. Such idea involves selecting the best images based on an analysis of how disperse the concentration of pixel intensity between brightest regions of different sizes. For this algorithm I begin by computing the brightest area of 80x80 pixels and compute the sum of intensities in such area, then from the 80x80 area I find the area that is half size in every direction, so 40x40 and compute the summation of the intensities in such region, finally I repeat the process with a 20x20 region in the smaller 40x40 region and I, too, compute the sum of the intensities in such area. Finally, to score the image quality-wise I divide the sum of the 20x20 area by the sum of the 40x40 area and then finally I divide the resulting score by the summation of the 80x80 region. This process is heavily inspired in the idea of concentrations of pixel values, and again, the idea is that when a region is sparse it means the reading is poor, when a reading is focused and the division by the sums outputs a reasonable fraction means the brightest region is actually enclosing a small but very bright object, like a star.

The implementation of this code looks as follows:

```
def select_best(dataset):
    elements = {}
    fractions = []
    for img in range(len(dataset)):
        x,y = compute_brightest_area(dataset[img],80,80)
        c,r = x[0],y[0]
        sub_image_sum = np.sum(dataset[img,r:r+80,c:c+80])
        x,y = compute_brightest_area(dataset[img,r:r+80,c:c+80],40,40)
        c,r = x[0],y[0]
        smaller_sum = np.sum(dataset[img,r:r+40,c:c+40])
        x,y = compute_brightest_area(dataset[img,r:r+40,c:c+40],20,20)
        c,r = x[0],y[0]
        focus = np.sum(dataset[img,r:r+20,c:c+20])
        result = focus/smaller_sum/sub_image_sum
        elements[result] = img
        fractions.append(result)
    return fractions,elements
```

**The brightest region is found by making use of the summed area algorithm or what is also known as the integral image.*

Problem 4

For this problem I implemented the logarithmic correction of intensities which basically is: $img = log(img^k + 1)/log(k + 1)$, I implemented the Gamma correction of images which is basically: $img = img^\gamma$, I also implemented the sigmoid correction which is defined as: $img = 1/(1 + e^{k*(0.5-img)})$ and then normalized by subtracting the minimum intensity in the resulting image and then dividing by the highest intensity in the image after the subtraction. Finally, I also implemented the equalization of histograms via the SciPy packet. The implementation for all the previous corrections and the equalization looks as follows:

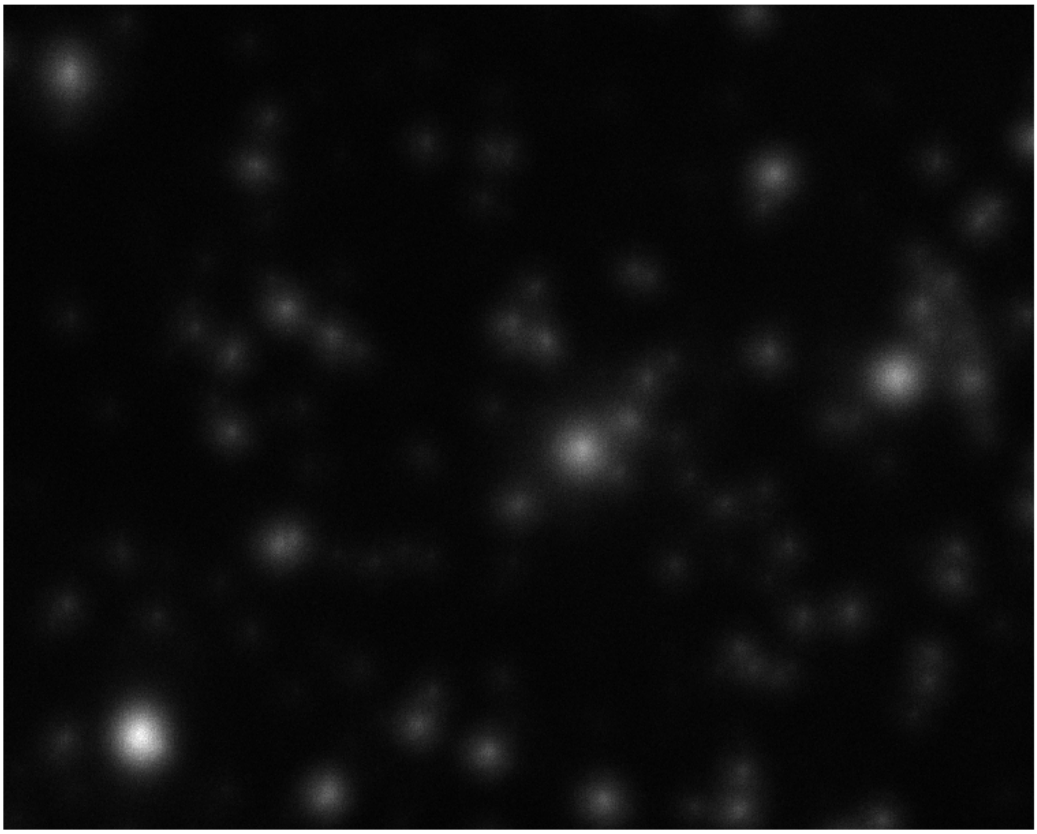
```
def equalize_historgram(img):
    im_h = rankdata(img.reshape(-1),method='max').reshape(img.shape)
    im_h = im_h/np.amax(im_h)
    return im_h

def gamma_correction(img,gamma):
    return img**gamma

def log_correction(img,k=1):
    return np.log((img**k)+1)/np.log(k+1)

def sigmoid(img,k=1):
    img = 1/(1 + np.exp(k*(0.5-img)))
    img = img-np.amin(img)
    img = img/np.amax(img)
    return img
```

III. Experimental Results

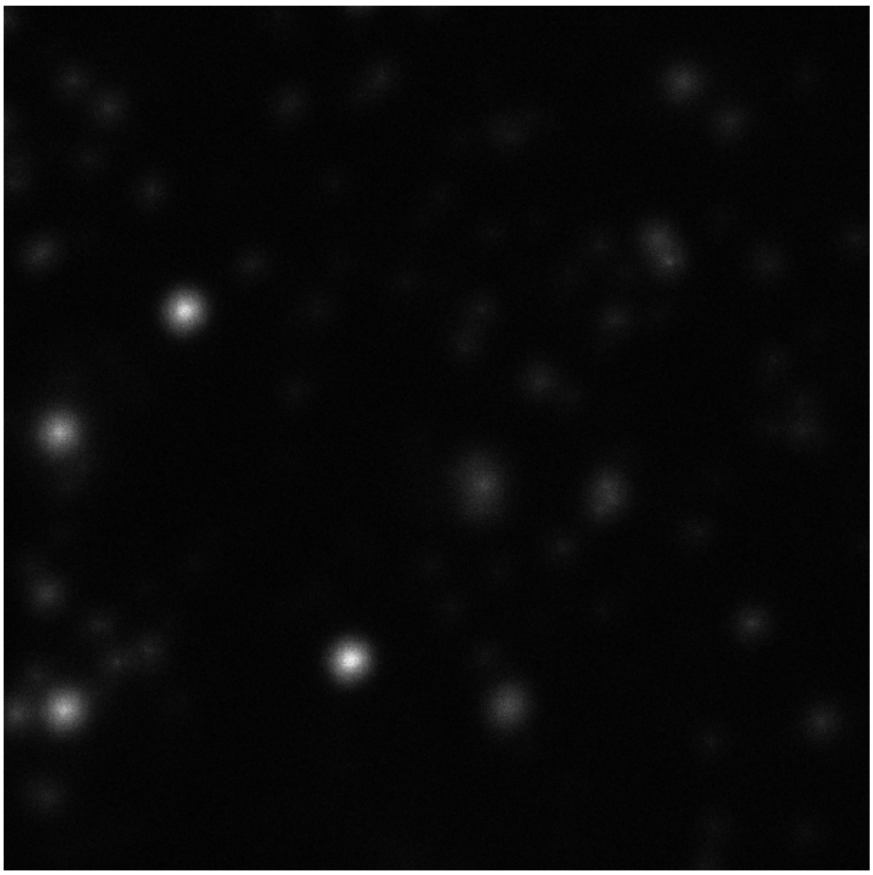
Problem 1:

- **Average images by dataset**

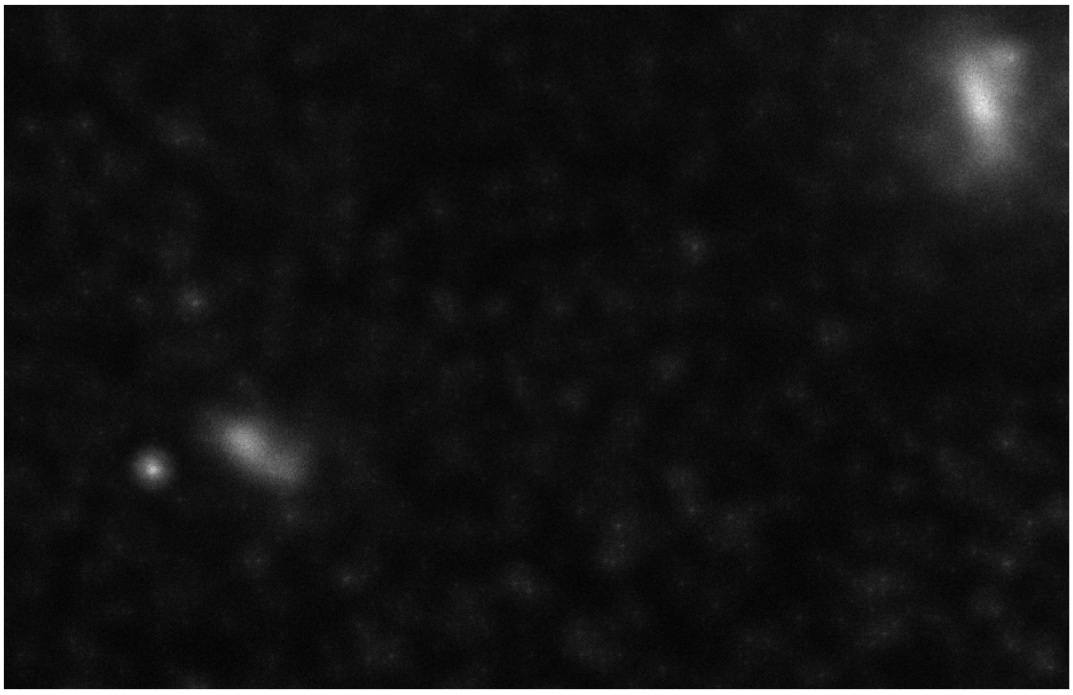
Average_image_set_ag.npz



Average_image_set_gl.npz



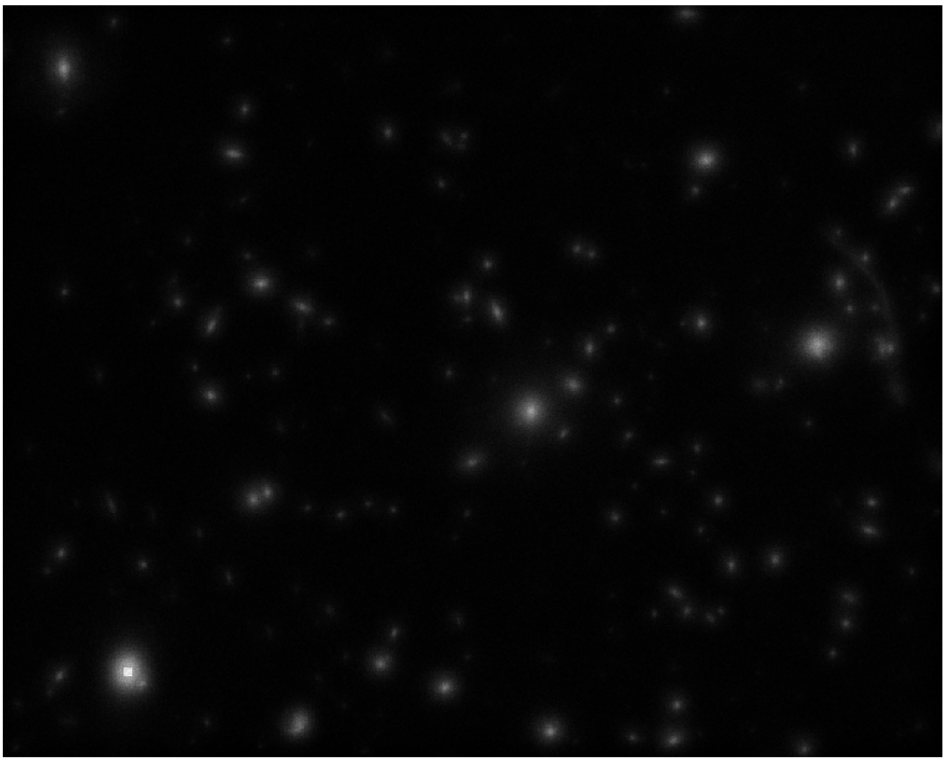
Average_image_set_mc.npz



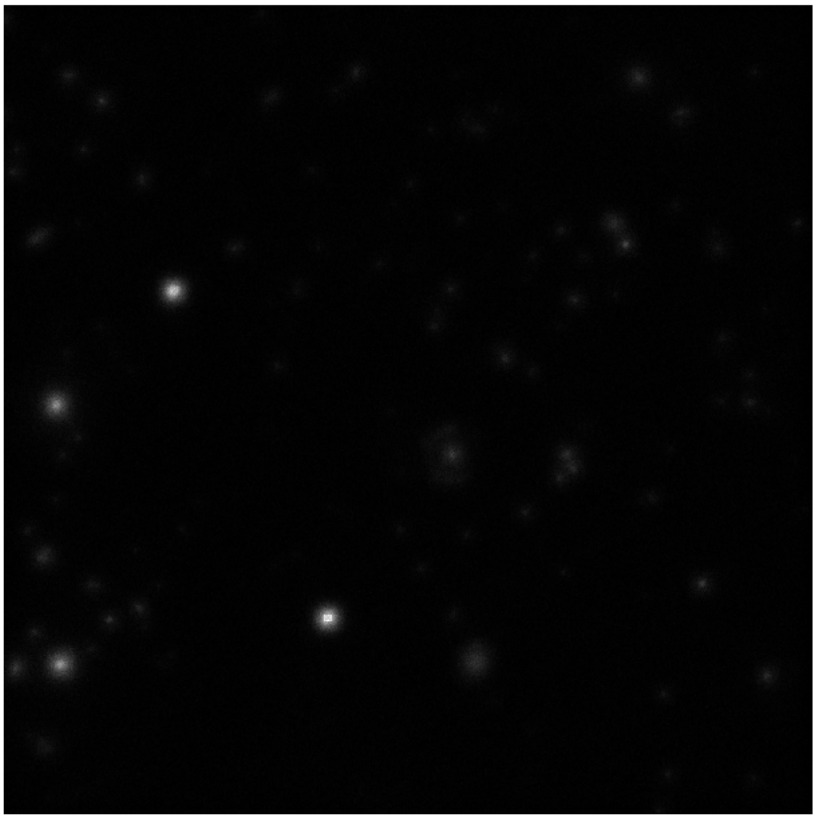
Problem 2, 3, & 4:

- **Aligned best 400 images**

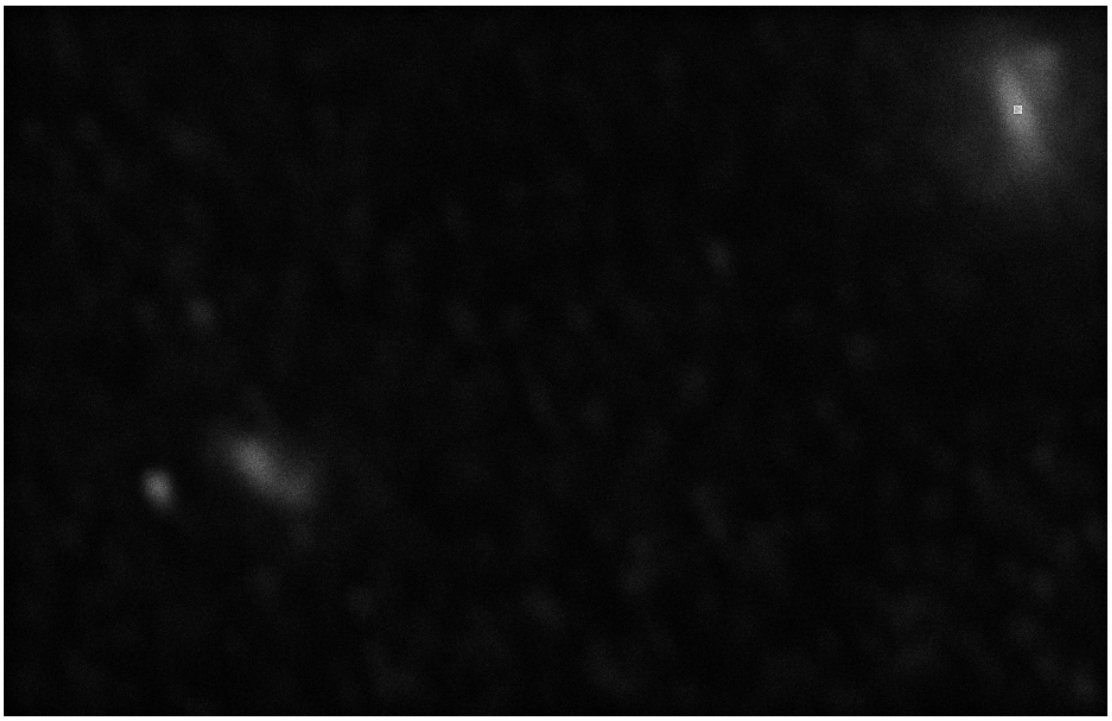
Aligned_image_set_ag.npz



Aligned_image_set_gl.npz

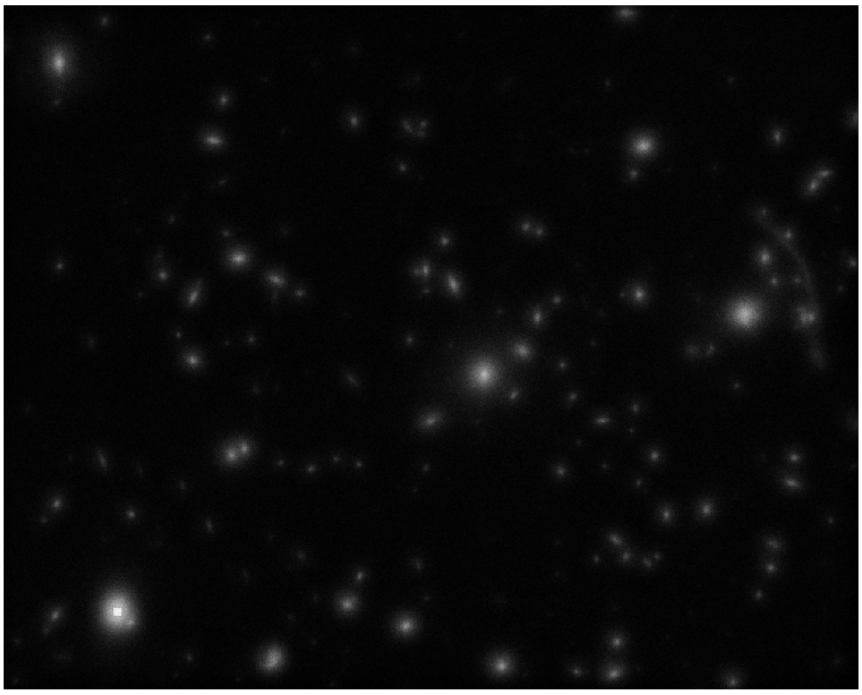


Aligned_image_set_mc.npz

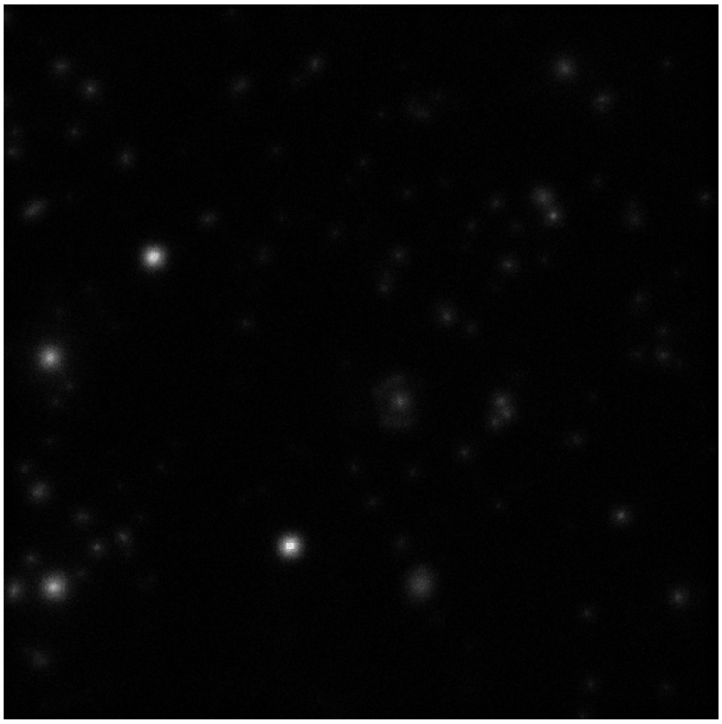


- **Aligned best 400 images with log correction**

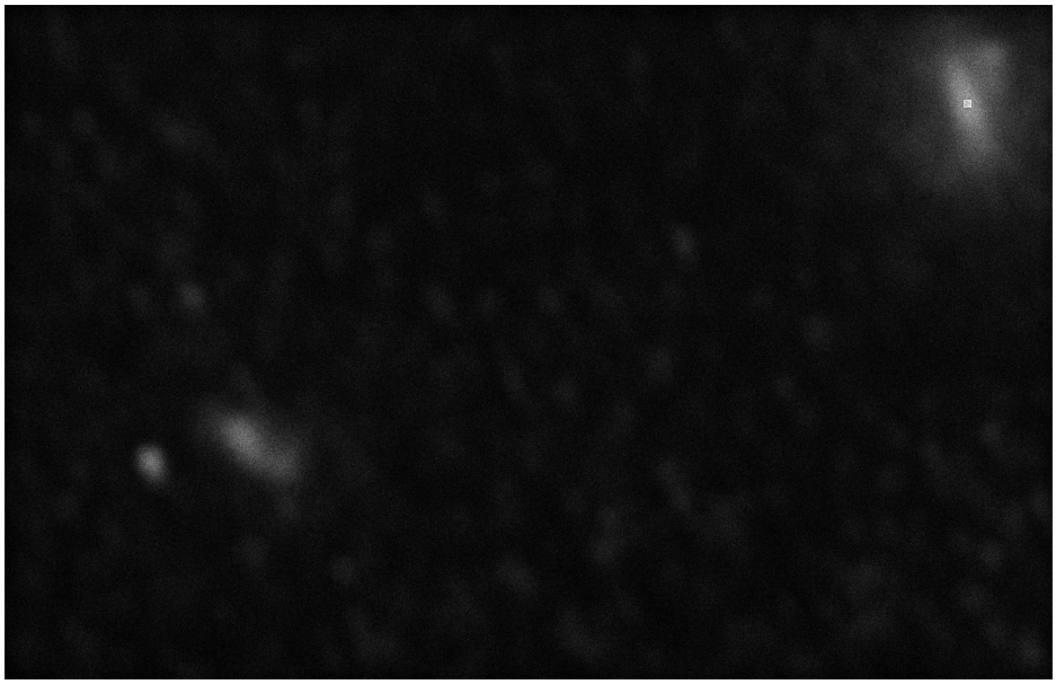
Aligned-Log_image_set_ag.npz



Aligned-Log_image_set_gl.npz

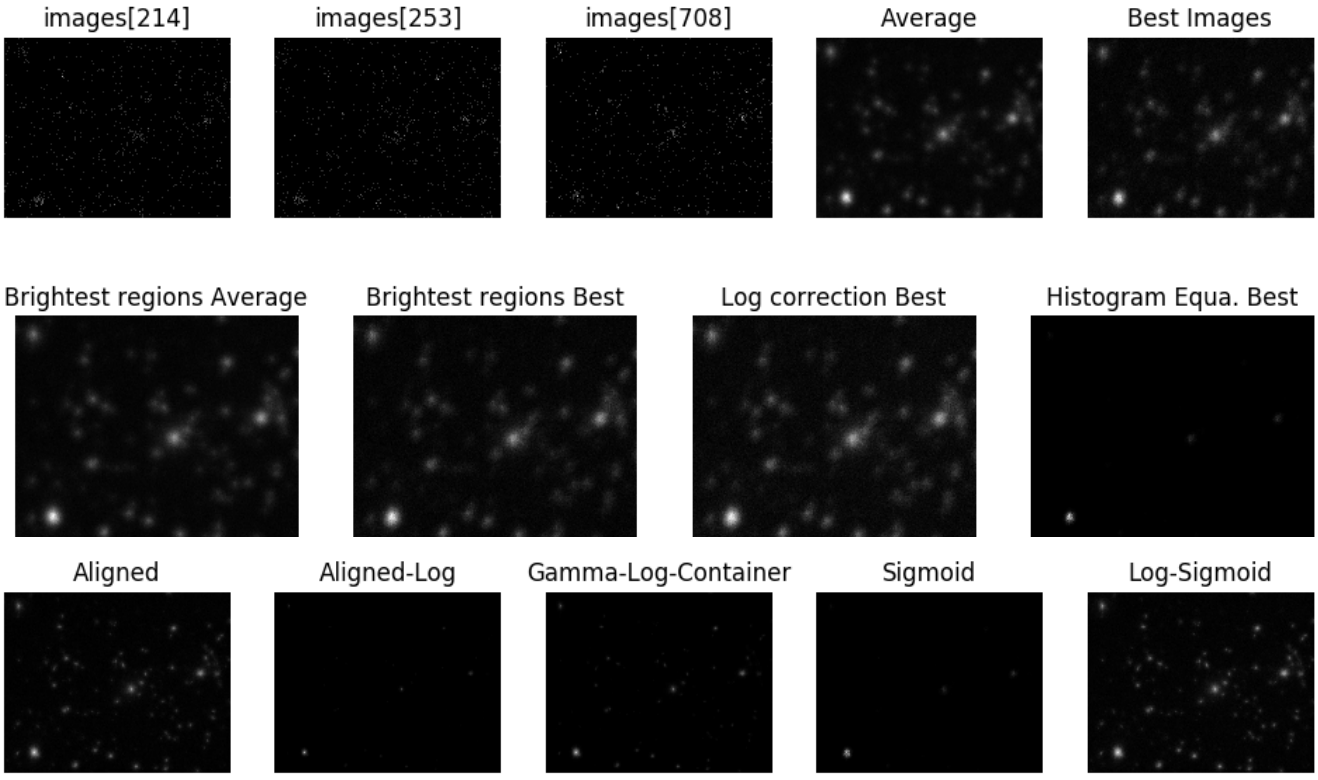


Aligned-Log_image_set_mc.npz

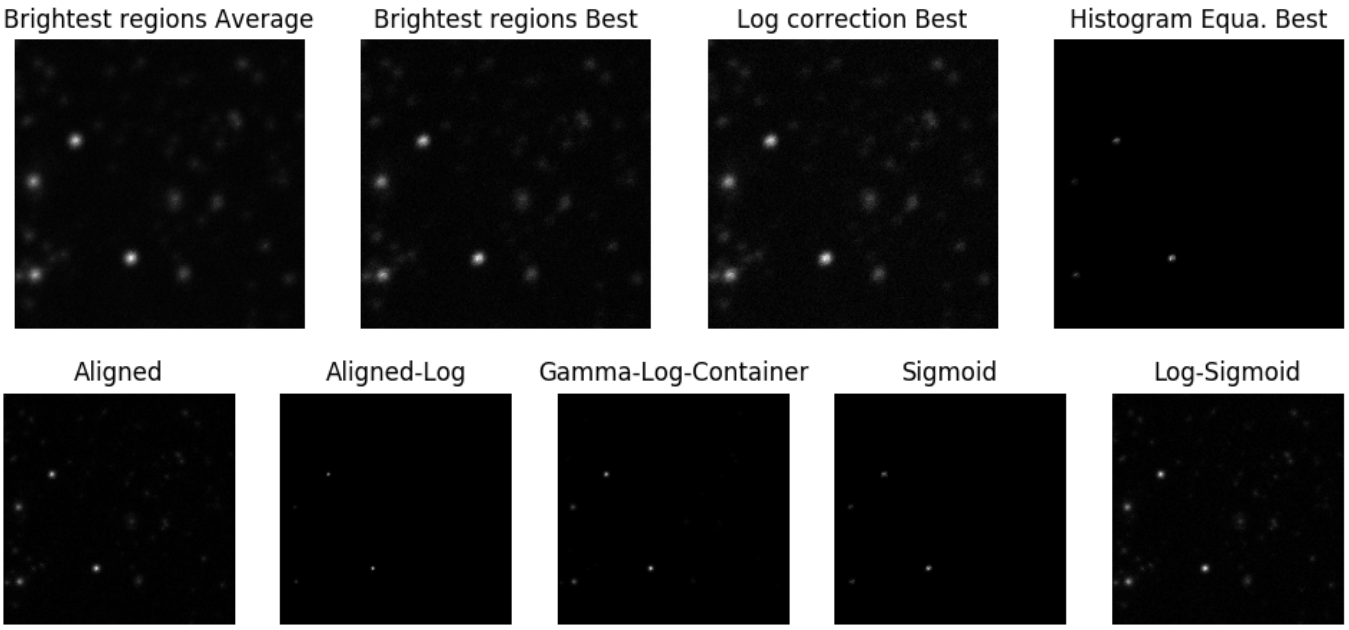


- **Extra transformations:**

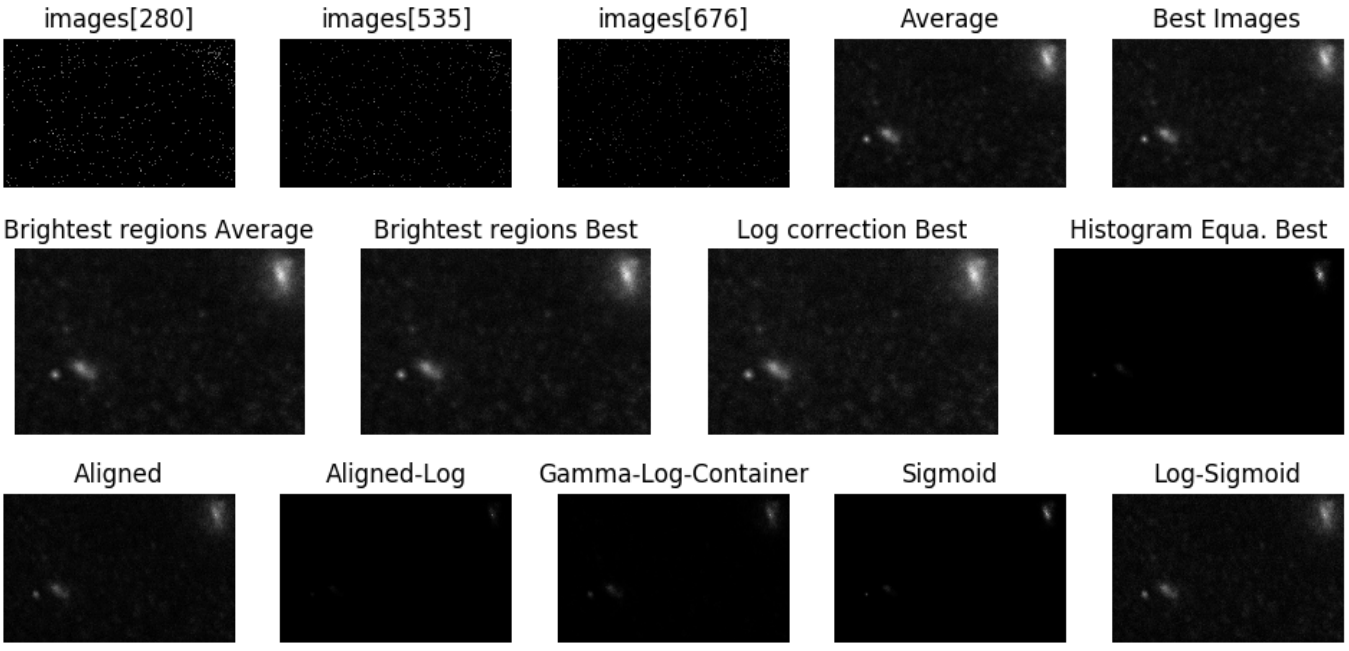
- **image_set_ag:**



- **image_set_gl**


Brightest regions Average
Brightest regions Best
Log correction Best
Histogram Equa. Best
Aligned
Aligned-Log
Gamma-Log-Container
Sigmoid
Log-Sigmoid

- **image_set_mc**


images[280]
images[535]
images[676]
Average
Best Images
Brightest regions Average
Brightest regions Best
Log correction Best
Histogram Equa. Best
Aligned
Aligned-Log
Gamma-Log-Container
Sigmoid
Log-Sigmoid

IV. Discussion of Results

As seen in the previous section, the alignment of images just by itself greatly improves the quality of the image which provides support to the idea of finding the best images by concentration of intensities in smaller regions as previously mentioned. In addition, it can be seen that the algorithm for alignment, although this algorithm characteristically makes small areas of images to disappear the impact, visually speaking seems to not prevent the alignment algorithm to produce results that are more sharp and clear than those seen in the average images.

In relation to the resulting images after alignment and transformations there is clearly much better results if a dataset is aligned and then the resulting image is intensity-corrected via the logarithmic correction. Histogram equalizations, gamma corrections, and sigmoid corrections all seem to produce images that are much less clear and with some areas as exceptions basically dark images. For the log correction k was set to three to add a little bit more than usual correction (for which we may set k=1), and for the alignment of images I used 400 images for the “gl” and “ag” datasets. For the “mc” dataset I decided to increase the number of images to include the entire dataset since the coordinate “safeguards” implemented in the alignment algorithm prevented many images to be used for alignment. As seen in the experimental results, we can observe clearly the details of the images in the “ag” and “gl” dataset, nevertheless for the “mc” the results are not necessarily improving if one considers that all the areas of the images should become clearer. On the other hand, the results do show an improved discretization of the three major/high intensity areas of such dataset.

With respect to dataset “mc” I theorize the resulting images lacked the quality as those resulting from datasets “ag” and “gl” because the images are not that great, maybe the dataset does not contain good images and the alignment fails miserably. It is important to mention other sizes of windows, as well as other combinations of transformations were tried but the experiments were just too many to show here.

V. Appendix

```
# Simple program to read images dataset

import numpy as np
import matplotlib.pyplot as plt
import utils
from scipy.stats import rankdata
import cv2 as cv

data_dir = './' # Use your own path here
data_files =  ['image_set_ag.npz','image_set_gl.npz','image_set_mc.npz']

plt.close('all')

def show_image(image,title='',save_im=True,filename=None):
    # Display image in new window
    fig, ax = plt.subplots()
    ax.imshow(image,cmap='gray')
    ax.axis('off')
    ax.set_title(title)
    if save_im:
        if filename==None:
            filename=title
        fig.savefig(filename+'.png',bbox_inches='tight', pad_inches=0.1,dpi=200)
    return fig, ax

def compute_integral(img):
    integral = np.zeros((img.shape[0]+1,img.shape[1]+1))
    if len(img.shape) == 2:
        integral[1:,1:] = img
    else:
        integral[1:,1:] = np.mean(img,axis=2)
    integral = np.cumsum(np.cumsum(integral,axis=0),axis=1)
    return integral

def compute_brightest_area(img,l,w):
    integral = compute_integral(img)
    brightest = np.zeros((integral.shape[0]-l,integral.shape[1]-w))
    brightest = integral[l:,w:] + integral[:integral.shape[0]-l,:integral.shape[1]-w] -
integral[l:,:integral.shape[1]-w] - integral[:integral.shape[0]-l,w:]
    y,x = np.unravel_index(np.argmax(brightest, axis=None), brightest.shape)
    x = [x,x,x+w,x+w,x]
```

```
    y = [y,y+l,y+l,y,y]
    return x,y

def select_best(dataset):
    elements = {}
    fractions = []
    for img in range(len(dataset)):
        x,y = compute_brightest_area(dataset[img],80,80)
        c,r = x[0],y[0]
        sub_image_sum = np.sum(dataset[img,r:r+80,c:c+80])
        x,y = compute_brightest_area(dataset[img,r:r+80,c:c+80],40,40)
        c,r = x[0],y[0]
        smaller_sum = np.sum(dataset[img,r:r+40,c:c+40])
        x,y = compute_brightest_area(dataset[img,r:r+40,c:c+40],20,20)
        c,r = x[0],y[0]
        focus = np.sum(dataset[img,r:r+20,c:c+20])
        result = focus/smaller_sum/sub_image_sum
        elements[result] = img
        fractions.append(result)
    return fractions,elements

def equalize_historgram(img):
    im_h = rankdata(img.reshape(-1),method='max').reshape(img.shape)
    im_h = im_h/np.amax(im_h)
    return im_h

def gamma_correction(img,gamma):
    return img**gamma

def log_correction(img):
    return np.log(img+1)

def sigmoid(img):
    return 1/(1 + np.exp(-img))

def align(images, coordinates_average,size=100):
    print(images.shape)
    container = np.zeros((images.shape[1],images.shape[2]))
    x_avg,y_avg = coordinates_average
    print(coordinates_average)
    count=0
    for i in range(len(images)):
        x,y = compute_brightest_area(images[i],size,size)
        diff_x, diff_y = x_avg[0]-x[0],y_avg[0]-y[0]
        if diff_x == 0 :
            diff_x -= 1
```

```
        if diff_y == 0:
            diff_y -= 1
        if abs(diff_x) > 35 or abs(diff_y) > 35:
            continue
        if diff_x < 0 and diff_y < 0:
            container[:diff_y,:diff_x] += images[i,abs(diff_y):,abs(diff_x):]
            count+=1
        elif diff_x > 0 and diff_y > 0:
            container[diff_y:,diff_x:] += images[i,:-diff_y,:-diff_x]
            count+=1
        elif diff_x < 0 and diff_y > 0:
            container[diff_y:,:diff_x] += images[i,:-diff_y,abs(diff_x):]
            count+=1
        else: #diff_x >0 and diff_y < 0
            container[:diff_y,diff_x:] += images[i,abs(diff_y):,:-diff_x]
            count+=1
    print(f'Aligned {count} Images')
    plt.show()
    return container/np.amax(container)

def get_collection(dataset):
    frac,elements = select_best(dataset)
    frac.sort(reverse=True)
    indeces = np.array([elements[value] for value in frac[0:size]])
    collection = dataset[indeces]
    return  collection


for d, data_file in enumerate(data_files):
    for size in [400]:#,400,600,1000]:
        if data_file == 'image_set_mc.npz': size+=600
        dataset = np.load(data_dir+data_file)['images']
        collection = get_collection(dataset)
        rand_choice = np.sort(np.random.randint(0,dataset.shape[0],3))
        image_list = [dataset[i] for i in rand_choice]
        titles = ['images[{}]'.format(i) for i in rand_choice]
        fig_title= 'Random images from '+data_file
        average = np.mean(dataset,axis=0)
        image_list.append(average)
        titles.append("Average")
        best = np.mean(collection,axis=0)
        image_list.append(best)
        titles.append("Best Images")
        utils.show_images(image_list, titles, fig_title) # Show some images

##################################################################
```

```
    image_list = []
    titles = []

    x_avg,y_avg = compute_brightest_area(average,10,10)
    log = log_correction(best)
    gamma = gamma_correction(log,4)
    container = (align(collection,(x_avg,y_avg),10))
    sig = sigmoid(container)

    titles.append('Brightest regions Average')
    titles.append('Brightest regions Best')
    titles.append('Log correction Best')
    titles.append('Histogram Equa. Best')
    image_list.append(average)
    image_list.append(best)
    image_list.append(log)
    image_list.append(gamma)
    fig_title = f'GroundTruths_{data_file}'
    utils.show_images(image_list, titles, fig_title)

#####################################################################
    image_list = []
    titles = []

    titles.append('Aligned')
    titles.append('Aligned-Log')
    titles.append('Gamma-Log-Container')
    titles.append('Sigmoid')
    titles.append('Log-Sigmoid')
    image_list.append(container)
    image_list.append(log_correction(container))
    image_list.append(gamma_correction(log_correction(container),2))
    image_list.append(gamma)
    image_list.append(log_correction(sig))
    fig_title = f'Transformations_{data_file}'
    utils.show_images(image_list, titles, fig_title)
    ########################Extra
Images##########################################
    show_image(container,f'Aligned_{data_file}')
    show_image(log_correction(container), f'Aligned-Log_{data_file}')
    show_image(best, f'Best_{data_file}')
    plt.show()
```

VI. Conclusion

Overall I believe this lab had impressive results, not only because alignment seems to be a ver useful tool for many other dataset but also because I got to observe how transformations can make a huge difference on how an images is observe and how visible the details of such image are. Overall I think in this lab we got to see the importance of defining ad-hoc techniques to “solve” issues with images. In this case I used the fractional intensity sum of the pixels with respect to sizes of regions and with that I basically developed a technique for this problem, as mentioned before other problems could have other solutions, maybe in terms of distributions, etc.

There is something to be said about the alignment algorithm with respect to the artifacts that it produces on the resulting images. For example, it can be seen that squares of high intensity (white-looking squares) appear as a result of the alignment procedure/algorithm, so I think this bring an interesting point about what kind of effects are produced in order to clarify the rest of the images. I think it must be kept in mind that when addressing a problem such that presented in this lab, and even when performing corrections, etc. The image is being changed in subtle ways, most likely, but still is being changed, and sometimes such changes might be visible, and that is ok as long as the end product is something that aids to understand the data better.

Statement of Academic Honesty:

“I certify that this project is entirely my own work. I
wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.”