ID:80585887
CS 5363
Dr. Olac Fuentes

I. Problem Description

For Lab 1 our task was to implement a series of visual transformations for images in Python. For example, we needed to implement an upside-down transformation, a mirror transformation, convert-to-grayscale transformation, etc. In addition, we needed to implement an algorithm to search for the area with the brightest regions for each of the three colored channels (red, blue, green) as well as the brightest region in the grayscale image and the least bright region in the grayscale image.

II. Algorithms Implemented

Problem 1

For Image 1

Was provided as part of the starter code

For Image 2

Basically we return an array for which every entry is a channel of the image

```
def get_channels(image):
    return [image[:,:,0], image[:,:,1], image[:,:,2]]
```

For Image 3

For this image we need to compute an image that shows the intensity of every channel at every pixel so we implement:

```
def color_index(image,index):
    if index == 0:
        return 2*image[:,:,0] - (image[:,:,1]+image[:,:,2])
    elif index == 1:
        return 2*image[:,:,1] - (image[:,:,2]+image[:,:,0])
    else:
        return 2*image[:,:,2] - (image[:,:,0]+image[:,:,1])
```

For Image 4

For this image we need to compute the grayscale version of an image which results from performing a weighted combination of the channels of the image, so the code is:

```
def gray_level(image):
    return 0.299*image[:,:,0]+0.587*image[:,:,1]+0.114*image[:,:,2]
```

For Image 5

For this image we need to compute the grayscale version of an image which results from performing a weighted combination of the channels of the image, and then we compute the inverse by subtracting the value of every entry of the image to 1. So, the code is:

```
def negative_gray_level(image):
    return np.absolute(1-(0.299*image[:,:,0]+0.587*image[:,:,1]+0.114*image[:,:,2]))
```

For Image 6

For this image all we need to do is to inverse the columns of the image, hence the code is:

```
def mirror(image):
    return image[:,::-1]
```

For Image 7

For this image all we need to do is to inverse the rows of the image, hence the code is:

```
def upside_down(image):
    return image[::-1]
```

For Image 8

For this image we compute the vertical edges of an image by subtracting the original image minus a modified version of the image which is the same image but moved one column to the left. To keep consistency the modified image gets the last column set to zeros so that the subtraction process yields the last row not as zeros.

```
def vert_edges(gray_image):
    right = np.copy(gray_image)
    right[:,:-1] = right[:,1:]
    right[:,-1] = 0
    return (gray_image-right)
```

For Image 9

Similarly to problem 8, the horizontal edges are obtained by subtracting the rows.So, in this case a modified image which is just the original image but shifted on row up will get subtracted to the original image, this yields the following code

```
def hor_edges(gray_image):
    right = np.copy(gray_image
    right[:-1] = right[1:]
    right[-1,:] = 0
    return (gray_image-right)
```

For Image 10

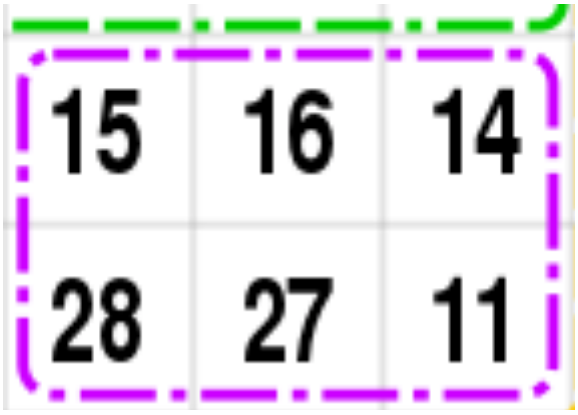
For this image the procedure involves taking the results of calling the hor_edges function and the vert_edges function. Both functions are combined by squaring the entries of every picture, adding after the individual squaring and then obtaining the square root of such image. This yields the following code:

```
np.sqrt(hor_edges(negative_gray_level(image))**2+vert_edges(negative_gray_level(image))**2)
```

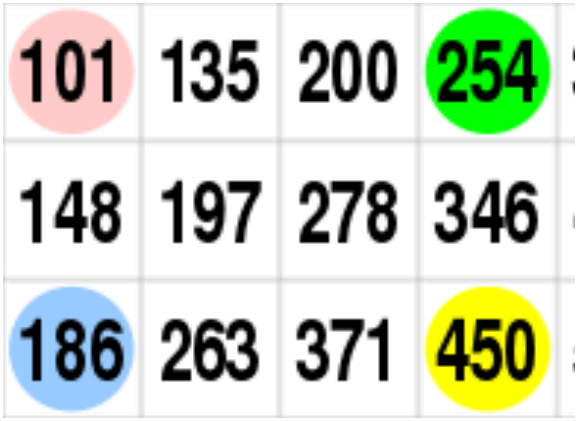
Problem 2

For this problem the Summed Area Table algorithm was used. This algorithm allows through the use of geometric properties to obtain in constant time the summation of an area and prevents the repetition of computation. The algorithm first computes a table, or an integral image which basically adds the entries of the array in one direction, for example through every row. And then it adds the entries that were previously integrated horizontally. This yields an image which at the bottom right contains the addition of all the pixels, at the top left there is the entry of what was in the area originally before the integral computation. And, finally, in the other two extremes we find the addition of just the entries that are either horizontal and vertical. In order to find the summation of an area all that is needed is that the area for which the

summation will be computed is extended one row and column to the left. For example, the following area:



For which the computation can be done as follows:



$$15 + 16 + 14 + 28 + 27 + 11 =$$
$$101 + 450 - 254 - 186 = 111$$

***Example taken from: https://en.wikipedia.org/wiki/Summed-area_table**

III. Experimental Results

*In general the left images are the resulting images after transformations applied and the right are originals.

Problem1:

- **Display Image**

Original image

- **Image Channels**

Red

Green

Blue

- **Indexed Channels**

Red

Green

Blue

- **Grayscale Image**

Gray-level image

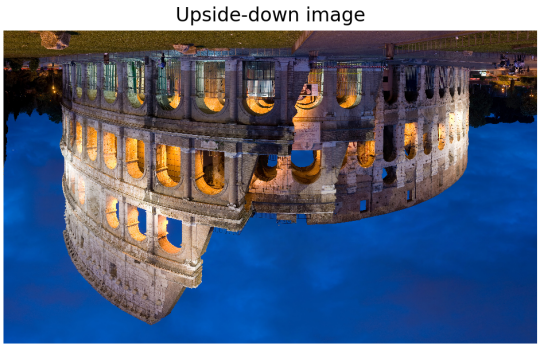
- **Negative grayscale Image**

Negative gray-level image

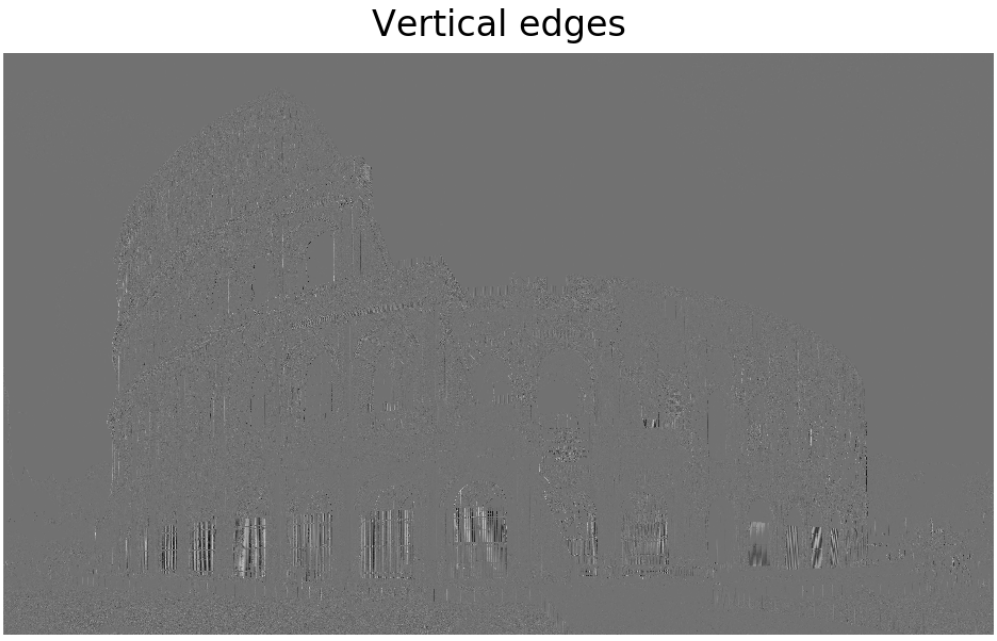
- **Mirrored Image**

Mirrored image

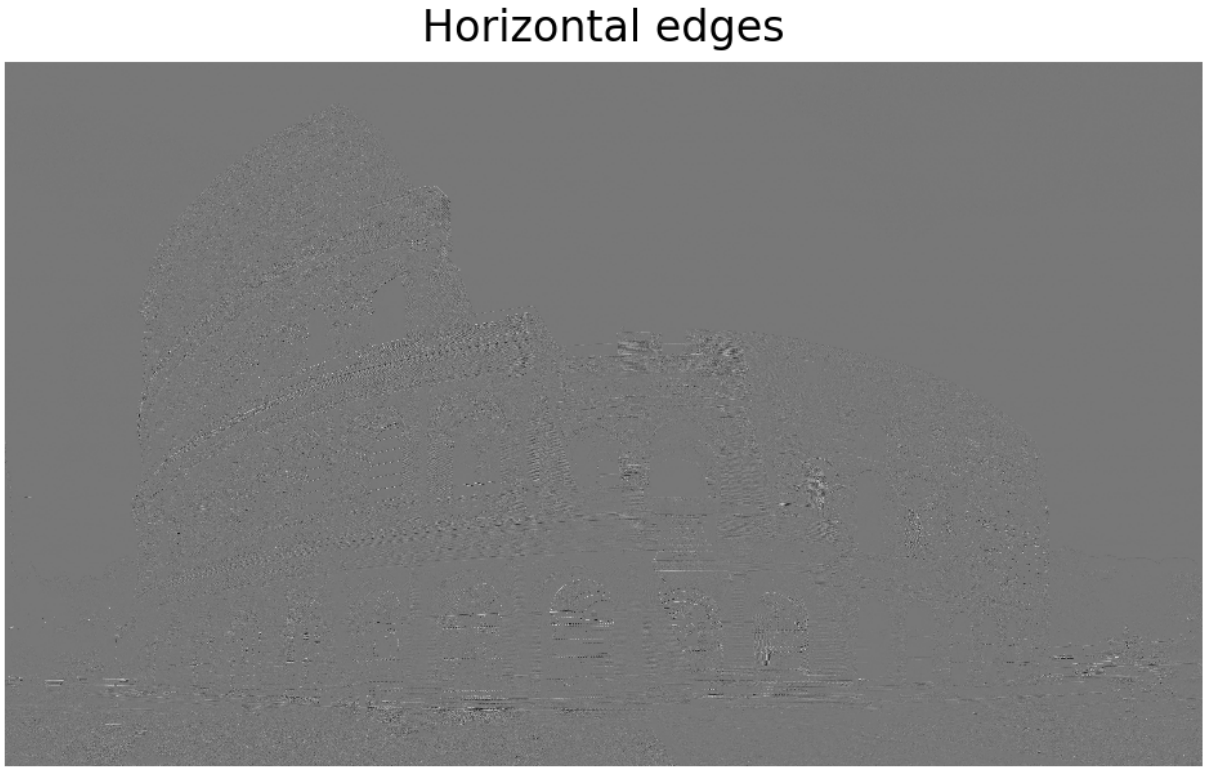
- **Upside Down Image**



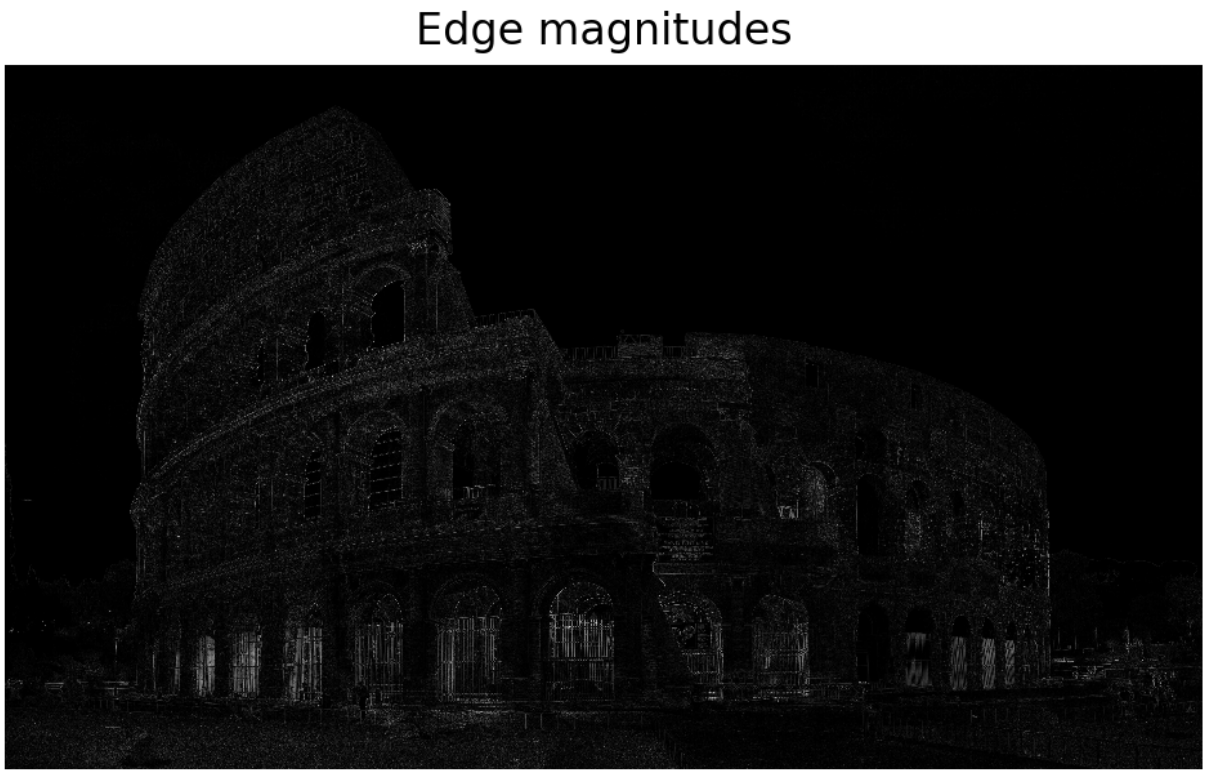
- **Vertical Edges Image**



- **Horizontal Edges Image**

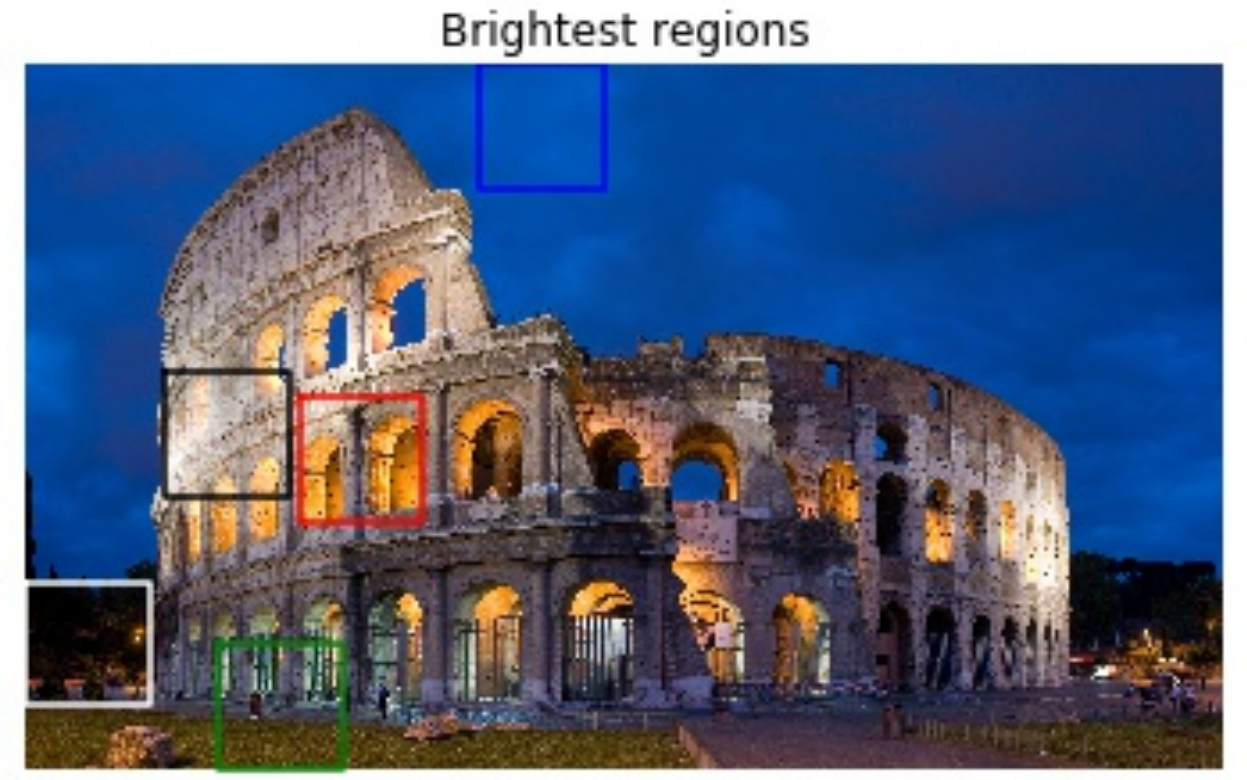


- **Magnitude Edges Image**



- **Subsampled image**

Problem 2 and 2.1:



IV. Discussion of Results

The images seem all to be “in order” and seem to be what the assignment required us to do. In addition I saw that Image channels were different. For example, the sky in the red channel looks very black, but in the blue channel looks very bright, which makes sense since the sky mostly contains high intensity blues. In relation to the grayscale image I thin that image looks amazing, very interesting. For the image about the negative grayscale image I notice a stark contrast with respect to the original grayscale image at, for example, the balconies of the colosseum, which in the original grayscale image look bright but in the negative image look dark.

With respect to the rest of the images I think the effects are pretty to observe, specially the image about the green indexed image since in general there is an absence of green in this image. In addition, I think the images that are just created from the channels of the original image reveal quite a lot of information, for example we can see an absence of red in the sky, or in the blue one which mainly shows there is blue everywhere. This analysis helps to see that the image that highlights the most bright areas of every channel and in general the most bright and least bright areas are indeed those highlighted in the resulting, brightest-areas image.

About the brightest area it is my belief that we actually have some other spots that might qualify as the most bright of every channel and in general most or least bright-intense areas of the image, obviously those we found are product of the implementation where the simple “>” vs. “=>” could make all the difference. In relation to time taken by the algorithms, all the algorithms take under the 2 second mark and that’s very efficient considering that the original image is 4827 x 2833 in size.

V. Appendix

```
import numpy as np
import matplotlib.pyplot as plt
import os
import matplotlib.image as mpimg

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 show_images(images,titles=None,save_im=True,filename=None):
    if titles==None:
        titles = ['' for i in range(len(images))]
    # Display image in new window
    fig, ax = plt.subplots(1,len(images),figsize=(12, 4))
    for i in range(len(images)):
        ax[i].imshow(images[i],cmap='gray')
        ax[i].axis('off')
        ax[i].set_title(titles[i])
    if save_im:
        if filename==None:
            filename='show_images'
        fig.savefig(filename+'.jpg',bbox_inches='tight', pad_inches=0.1,dpi=200)
    return fig, ax

def color_index(image,index):
    if index == 0:
        return 2*image[:,:,0] - (image[:,:,1]+image[:,:,2])
    elif index == 1:
        return 2*image[:,:,1] - (image[:,:,2]+image[:,:,0])
    else:
        return 2*image[:,:,2] - (image[:,:,0]+image[:,:,1])

def get_channels(image):
    return [image[:,:,0], image[:,:,1], image[:,:,2]]
```

```
def subsample(image,r,c):
    return image[::2,::2]

def gray_level(image):
    return 0.299*image[:,:,0]+0.587*image[:,:,1]+0.114*image[:,:,2]

def negative_gray_level(image):
    return np.absolute(1-(0.299*image[:,:,0]+0.587*image[:,:,1]+0.114*image[:,:,2]))

def vert_edges(gray_image):
    right = np.copy(gray_image)
    right[:-1] = right[1:]
    right[-1,:] = 0
    return (gray_image-right)

def hor_edges(gray_image):
    right = np.copy(gray_image)
    right[:,:-1] = right[:,1:]
    right[:,-1] = 0
    return (gray_image-right)

def mirror(image):
    return image[:,::-1]

def upside_down(image):
    return image[::-1]

def brightest_region(image,reg_rows,reg_cols):
    if len(image.shape) > 2:
        sum_img = np.zeros((image.shape[0]+1,image.shape[1]+1,image.shape[2]))
        result = np.zeros((image.shape[0],image.shape[1],image.shape[2]))
    else:
        sum_img = np.zeros((image.shape[0]+1,image.shape[1]+1))
        result = np.zeros((image.shape[0],image.shape[1]))
    sum_img[1:,1:] = image
    sum_img = np.cumsum(np.cumsum(sum_img,axis=1),axis=0)
    result = sum_img[:image.shape[0]-reg_rows+1,:image.shape[1]-
reg_cols+1]+sum_img[reg_rows:,reg_cols:]- sum_img[reg_rows:,:image.shape[1]-
reg_cols+1] - sum_img[:image.shape[0]-reg_rows+1,reg_cols:]
    if len(result.shape) > 2: result = np.mean(result,axis=2)
    y,x = np.unravel_index(np.argmax(result, axis=None), result.shape)
    print(x,y)
    x = [x,x,x+reg_cols,x+reg_cols,x]
    y = [y,y+reg_rows,y+reg_rows,y,y]
    return x,y,result
```

```
if __name__ == "__main__":

    plt.close("all")
    image_dir = '/Users/oscargalindo/Desktop/Classes/CS 5363/Lab 1/images/'

    image_files = os.listdir(image_dir)

    for image_file in image_files[:]:
        if image_file == '.DS_Store': continue
        image = mpimg.imread(image_dir+image_file)
        print('Image shape',image.shape)
        if np.amax(image)>1:
            image = (image/255).astype(np.float32)
            print('Converted image to float')

        show_image(image,'Original image')

        show_image(subsample(image,2,2),'Subsampled image')

        show_images(get_channels(image),['Red','Green','Blue'],filename='channels')

        col_ind = [color_index(image,index) for index in range(3)]
        show_images(col_ind,['Red','Green','Blue'],filename='color indices')

        show_image(gray_level(image),'Gray-level image')

        show_image(negative_gray_level(image),'Negative gray-level image')

        show_image(mirror(image),'Mirrored image')

        show_image(upside_down(image),'Upside-down image')

        show_image(vert_edges(negative_gray_level(image)),'Vertical edges')

        show_image(hor_edges(negative_gray_level(image)),'Horizontal edges')


show_image(np.sqrt(hor_edges(negative_gray_level(image))**2+vert_edges(negative_gr
ay_level(image))**2),'Edge magnitudes')

        fig, ax = show_image(image,'Brightest regions',save_im=False)

        reg_rows, reg_cols = 500,500
        x,y,s = brightest_region(image,reg_rows, reg_cols)
        ax.plot(x,y,color='k')
```

```
x,y,s = brightest_region(-image,reg_rows, reg_cols)
ax.plot(x,y,color='w')

c = 'rgb'
for i in range(3):
    x,y,s = brightest_region(color_index(image,i),reg_rows, reg_cols)
    ax.plot(x,y,color=c[i])

fig.savefig('brightest_regions.jpg',bbox_inches='tight', pad_inches=0.1)
```

VI. Conclusion

This lab was entertaining to do, I had the opportunity to learn about the edge magnitude and how with few steps find the edges of an image. I also learned about the Summed-Area table algorithm, which this was the first time I actually implemented for a Computer Vision purpose in mind. It was nice to observe that the images do contain a lot of hidden information in the color channels, as I previously discussed. In addition, I would like to mention the discussion we held during class to "solve" this lab were very helpful to understand other concepts, like filters. In general, I liked this lab and I think it taught me some things that could be helpful for either research purposes or job purposes.

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."