CSE576_HW1_1826630_Yao-Chung Liang

Task 1:

Discussion:

In the algorithm of my convolution, firstly, I create a buffer which you can choose zero padding or reflected padding with the flags, but right now reflected padding will crash. Secondly, I store the image data into my buffer. In the end, doing convolution on buffer and store the RGB value to pass to original image.

Time complexity is dominated by
O(imageHeight*imageWidth*kernelHeight*kernelWidth).

Memory complexty is dominated by O(imageHeight*imageWidth*3)

Task 2:

Fig. 1 Gaussian blur of Seattle with sigma (standard deviation) to be 4.0.

Discussion:

I calculate the 2D gaussian distribution with respect to get a kernel and using the kernel to do the convolution

Time complexity is dominated by
O(imageHeight*imageWidth*kernelHeight*kernelWidth).

Memory complexty is dominated by O(2*imageHeight*imageWidth*3)

(all these are dominated by convolution part)

2 is because I created a buffer with my original image memory, 3 is because there are rgb values and 9 is the kernel size.

Task3:

Fig. 2 Separable gaussian blur of Seattle.jpg with sigma (standard deviation) to be 4.0.

Discussion:

In this part, I make 1D gaussian kernel separately and convolute it twice.

Time complexity is dominated by

O(imageHeight*imageWidth*kernelHeight*kernelWidth).

Memory complexty is dominated by O(3*imageHeight*imageWidth*3)

(all these are dominated by convolution part)

3 is because I created buffer twice for each direction and the original image memory.

Another 3 is because there are rgb values and 9 is the kernel size.

Task 4a:

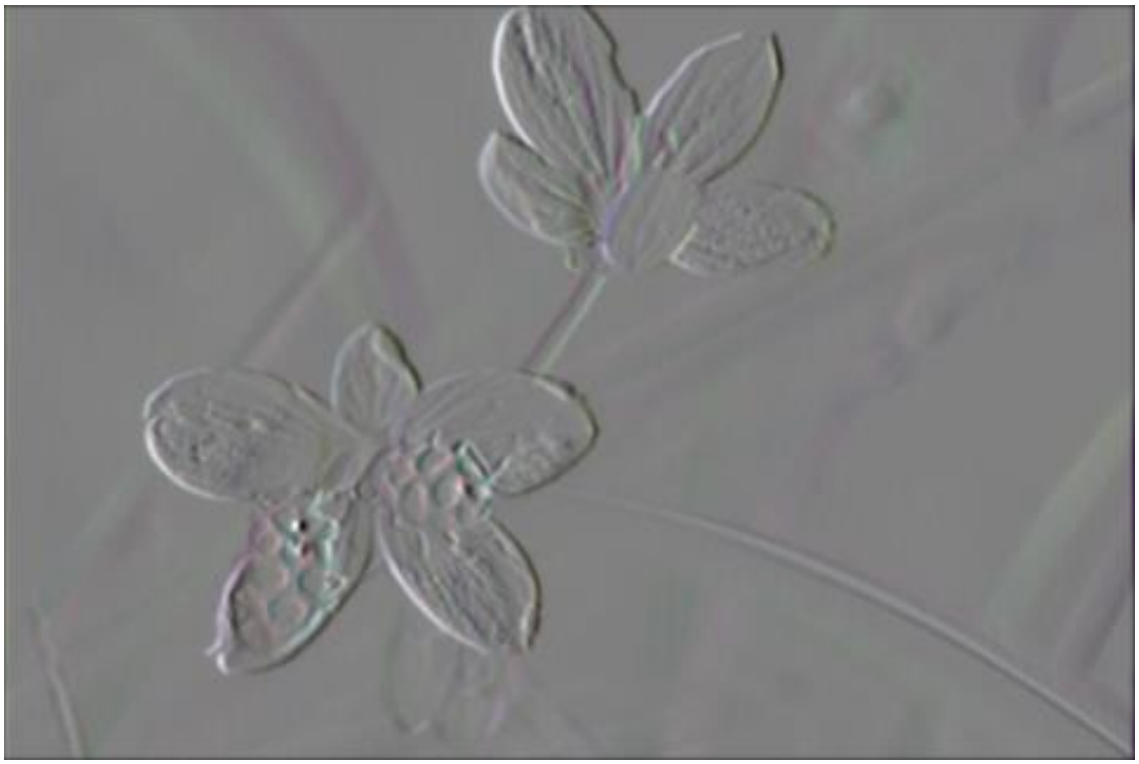

Fig 3. X derivative of Ladybug.jpg

Discussion:

First, I checked the standard deviation to be positive and make a 3by3 kernel {0,0,0,-1,0,1,0,0,0}. Actually I think I can only just use one dimensional kernel.

Time complexity is dominated by O(imageHeight*imageWidth*9).

Memory complexty is dominated by O(2*imageHeight*imageWidth*3)

(all these are dominated by convolution part)

2 is because I created a buffer, 3 is because there are rgb values and 9 is the kernel size.

Task 4b:

Fig 4. Y derivative of Ladybug.jpg

Discussion:

First, I checked the standard deviation to be positive and make a 3by3 kernel {0,-1,0,0,0,0,0,1,0}. Actually I think I can only just use one dimensional kernel.

Time complexity is dominated by O(imageHeight*imageWidth*9).

Memory complexity is dominated by O(2*imageHeight*imageWidth*3)

(all these are dominated by convolution part)

2 is because I created a buffer, 3 is because there are rgb values and 9 is the kernel size.

Task 4c:

Fig 5. Second derivative of Ladybug.jpg with sigma = 1.0

Discussion:

First, I checked the standard deviation to be positive and make a 3by3 kernel {0,1,0,1,-4,1,0,1,0 }.

Time complexity is dominated by O(imageHeight*imageWidth*9).

Memory complexity is dominated by O(2*imageHeight*imageWidth*3)

(all these are dominated by convolution part)

2 is because I created a buffer, 3 is because there are rgb values and 9 is the kernel size.

Task 5:

Fig 6. Sharpen "Yosemite.png" with a sigma of 1.0 and alpha of 5.0

Discussion:

Here, I create a buffer and copy the image into it and make a second derivative on the buffer. And then using the original image rgb the subtract the Gaussian-smoothed second derivative of the buffer (at the same time subtracted by 128 and times the alpha which is a kind of weight).

Time complexity is dominated by O(imageHeight*imageWidth*9).

Memory complexity is dominated by O(2*imageHeight*imageWidth*3)

(all these are dominated by convolution part)

2 is because I created a buffer, 3 is because there are rgb values and 9 is the kernel size.

Task 6:

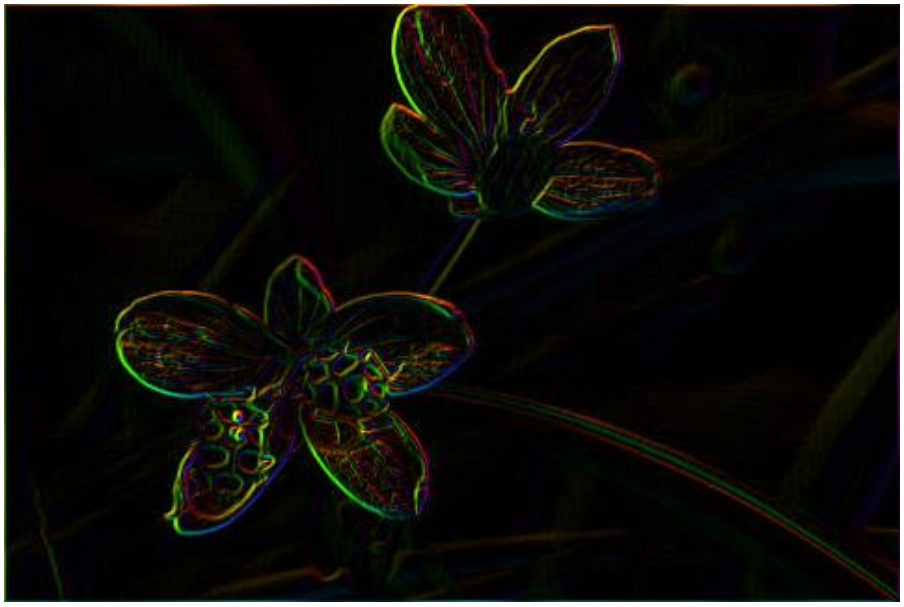

Fig 7. Sobel edge detection of “Ladybug.jpg”

Discussion:

At first, I create a buffer to store image data and and turn them all into gray scale. Then I did x derivative on original image and did y derivative on buffer. After that I divided the value I just got by 8 to avoid spurious edges. And then, calculate the magnitude and orientation of each pixel and assign them back to original image.

Time complexity is dominated by O(imageHeight*imageWidth*9).

Memory complexity is dominated by O(2*imageHeight*imageWidth*3)

2 is because I created a buffer, 3 is because there are rgb values and 9 is the kernel size.

Task 7:

Fig 8. Rotate the image "Yosemite.png" by 20 degrees

Discussion:

In bilinear Interpolation function:

At first, I check the pixel position to be located in the image display range or the RGB at the position will be set to be 0. And I check if the pixel position x y are integer or not, if it is, then upper bound of x will be set to be x+1 and upper bound y will be set to be y+1. Or they will just be ceil(x) and cei(y). for the lower bound, both x and y are floor(x) and floor(y).

After I got the reference x and y' upper bound and lower bound, I can compute the four reference points and doing the bilinear interpolation.

After we also consider the RotateImage function,

Time complexity = O(imageHeight*imageWidth*3)

Memory complexity = O(2*imageHeight*imageWidth*3)

2 is because I created a buffer, 3 is because there are rgb values and the 3 in time complexity is RGB.

Task 8:

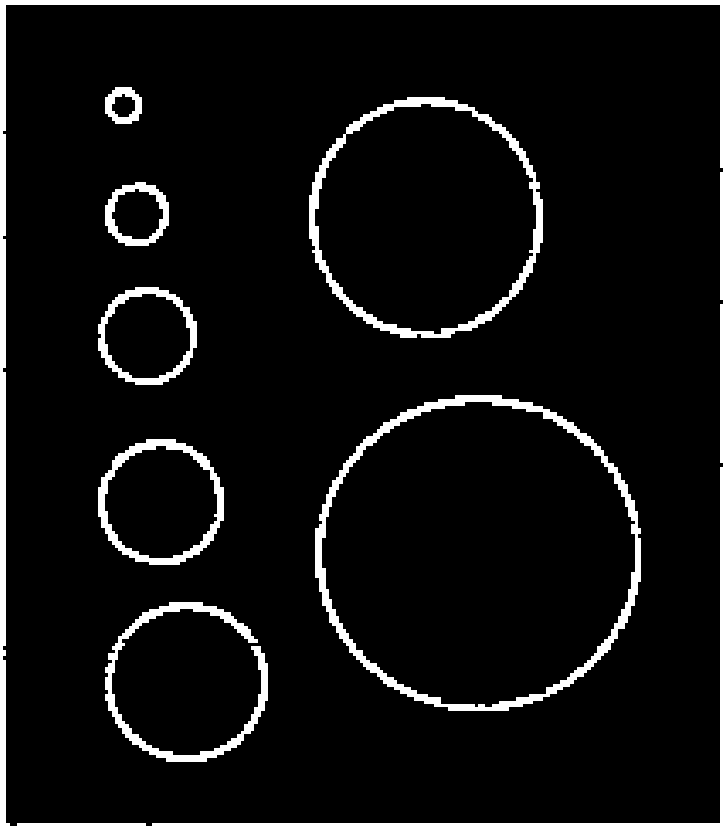

Fig 9. peak responses in "Circle.png" with threshold = 40.0

Discussion:

In the beginning, I use the Sobel operator to get the magnitude and orientation and use them to compute the upper bound, lower bound points and did the interpolation. After that, I get peak response and compare them with two points' peak responses which are also in the line perpendicular to the edge and compare with a threshold. If it satisfied all these constraints, then it will be regarded as a peak.

Task 9a:

Fig 10. Random seeds clustering on "Flower.png" with num_clusters = 4.

Discussion :

First, I create a buffer a store which pixel belongs which class. And a cluster 2D array to store mean RGB for each classes. And I use rand()%256 to generate random number of RGB. In the while loop, I calculate the loss from each pixel's RGB with respect to each class RGB. If it is the smallest loss, then this pixel belongs to this class. After I go through all pixels, I calculate the total loss by summing up loss from each cluster. The termination point is that absolute value of previous total cost subtracted by current total cost which is not greater than number of clusters multiplied by epsilon. Or the other termination point is when the total iteration is greater than max iteration.

Task 9b:

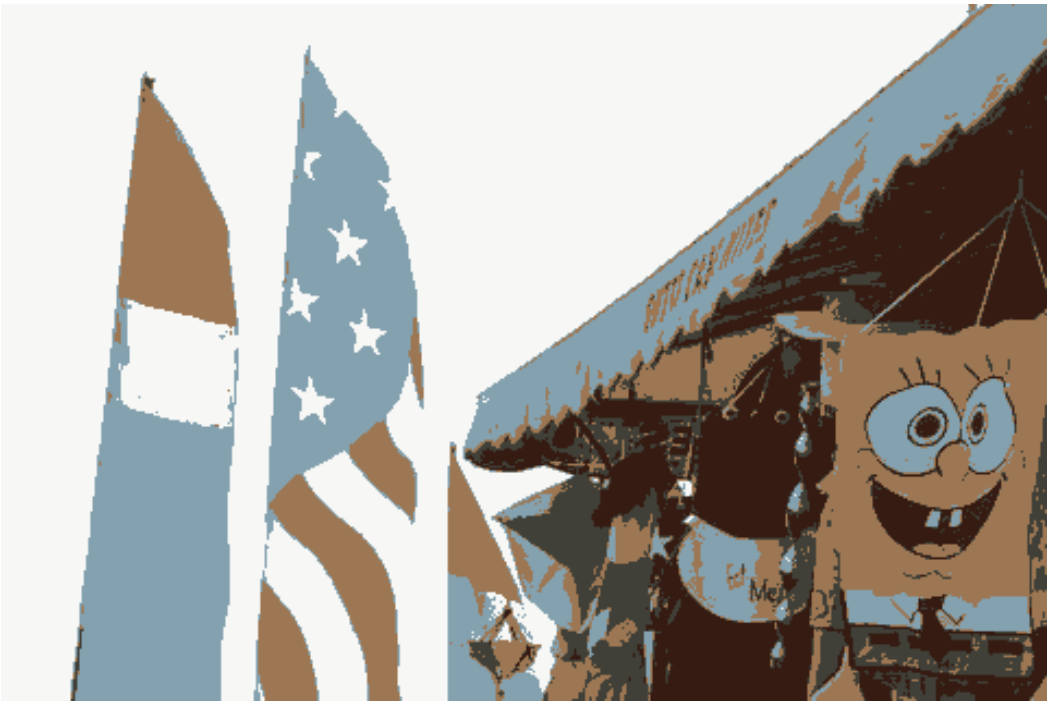

Fig 11. Pixel seeds clustering on "Flags.png" with num_clusters = 5

Discussion:

First, I create a buffer a store which pixel belongs which class. And a cluster 2D array to store mean RGB for each classes. And I use rand()%(imageHeight*imageWidth) to get random pixel's RGB value and assign into the cluster's RGB. In the while loop, I calculate the loss from each pixel's RGB with respect to each class RGB. If it is the smallest loss, then this pixel belongs to this class. After I go through all pixels, I calculate the total loss by summing up loss from each cluster. The termination point is that absolute value of previous total cost subtracted by current total cost which is not greater than number of clusters multiplied by epsilon. Or the other termination point is when the total iteration is greater than max iteration.

Extra Credits:

1.

Histogram K-clustering:

Fig 12. Grayscale histogram clustering on "Flags.png" with num_clusters = 5.

Discussion:

I computed the grayscale histogram and tried the uniform distributed grayscale to be the initiation of cluster center point value instead of top k amount of grayscale. And I think the result is still fine. EX: if number of cluster is 5, then I will select the grayscale 0, 63,127,191,255 to be my first cluster center.

2.

Median Image:

It still crashes till now.

3.

Reflected padding:

It still crashes till now.