Brandon Johnson
Computational Photography

Assignment 3

PART 1

Considering that image gradients are directional alterations in intensity within an image, in order to differentiate an image in the X direction, you will need to subtract each pixel in the x direction from the previous pixel. The code “image_copy[r][c] = abs(int(image_copy[r][c + 1]) - int(image_copy[r][c]))” accomplishes this goal because changes in the “c” or column, moves in a lateral fashion thus calculating the change in the x direction.

To differentiate in the y direction, we need to subtract the rows of the image. To accomplish this, the following code is used: image_copy[r][c] = abs(int(image_copy[r+1][c]) - int(image_copy[r][c])). The absolute value function was utilized

Compute gradient:
For the first part of this function, we have to develop our kernel. This is done with an initial nested for loop. The iteration conditions used is:

```
for r in range (1, rows-1)
        for c in range (1, cols-1)
```

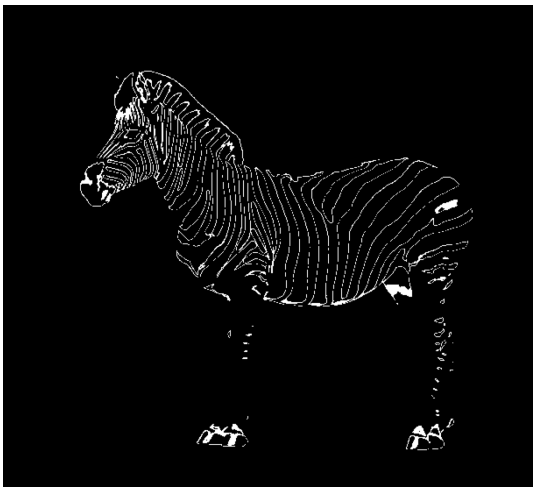
The reason we start from 1 and then iterate until rows/cols -1 is because for a image we cannot get the first outermost rows and columns. That is why we use these conditions. Then we have to create a variable for the sum. After that we create another nested for loop to navigate through the kernel. It is important to note that we must create it outside of the second nested for loop. Then for the second nested for loop we will have it go through the range (-1,2). The center point of the kernel is (1,1) and (-1,2) will navigate through the kernel. Then within the second nested for loop, we must have the sum calculate the sum of the surrounding n number of the kernel that is multiple by the corresponding kernel index.

PART 2

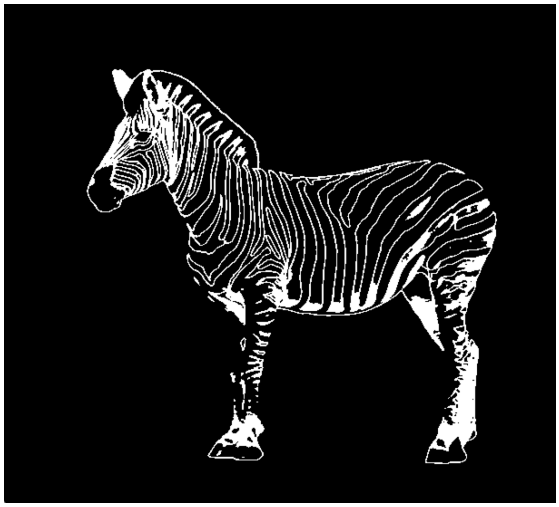
To produce an edge detected output image similar to an image produced using the Canny function, I first created a function called edge that called the “compute_gradient(image, kernel)” and the “convert_to_bw(image)” functions.

The compute_gradient function was utilized to create a blur. The blur level is used as a base to create the solid border when called on by the convert_to_bw function for the edge detection function. All the pixels that form the blur effect are within a similar range of intensity values. By playing with the threshold to convert these values to white and everything else to black, an edge is created.

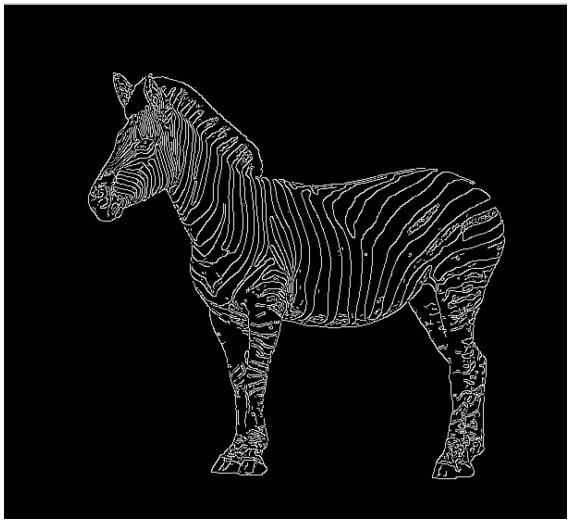
I altered the convert_to_bw function to create a range. Instead of changing the values of the pixels that were under a particular number, I wanted to isolate the gray pixels with a particular range to accentuate it and create an edge.



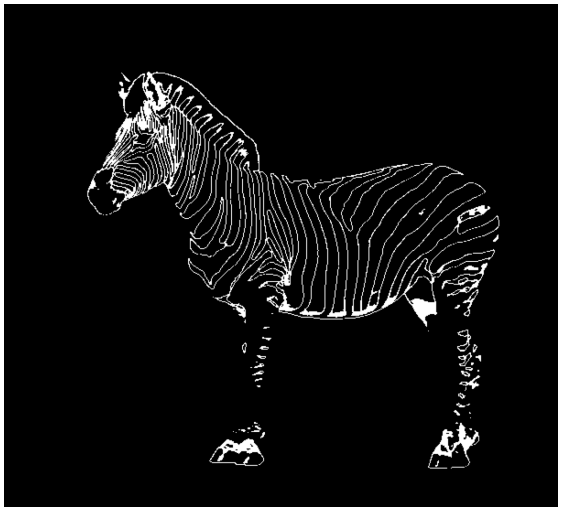
This is the image produced with a threshold of $50 < x < 100$. The edges generated are skinny and scattered



This is the image produced with a threshold of $80 < x < 200$. The edges generated are too bold.



This is the image produced using the built in Canny function. As depicted, the edges are evenly distributed



This is the final image produced with a threshold of $70 < x < 150$. The edges are somewhat evenly distributed, similar to that of Canny. There are some blotches in the image which can be attributed to the pixel intensity range.