Project 1 Report

Part 1: Filtering

Gaussian Kernel

To define a Gaussian kernel of size $3\sigma \times 3\sigma$, we first uses NumPy's meshgrid function to creates the appropriately sized grid with coordinates (x, y), where $x, y \in [\lfloor -(3\sigma - 1)/2 \rfloor, \lfloor ((3\sigma - 1)/2 \rfloor)$. Next, the kernel itself is calculated using the Gaussian function. Lastly, the kernel is normalized by dividing it by the sum of its values.

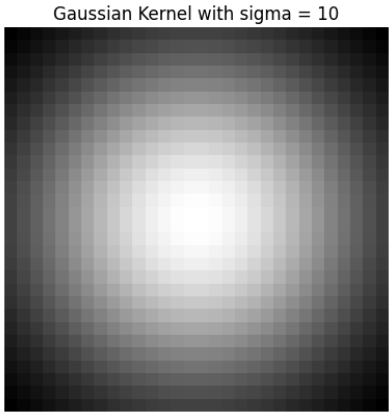


Figure 1: Visualization of a Gaussian kernel with standard deviation of 10.

The brightness of each cell in **Figure 1** represents the weight assigned to a specific point in the kernel. Brighter cells have greater weight than darker ones. We see that the kernel is brightest at its center and gradually darkens as the distance from the center increases, following a bell-shaped curve that reflects the Gaussian function. This results in a smooth transition between the bright and dark areas. Since this kernel has a standard deviation of 10, the majority of the weight is concentrated within a radius of 10 units around the center.

Convolution

We apply a filter to an input gray-scale image by performing convolution in the frequency domain using the Fast Fourier Transorm (FFT). Firstly, we calculate the dimensions of the input image and the filter. To handle the boundary cases on the edges of the image, the input image is padded with zeros. The amount of padding on each side is given by half the size of the filter (excluding the center). Next, we apply the FFT to both the padded input image and the filter, multiply the transformed images element-wise to obtain the convolution of the image with the filter. Then, transform the resulting image back to the spatial domain using the inverse FFT and remove the padding to obtain the filtered image.

Figure 2: Standard deviation of 3

Figure 3: Standard deviation of 5

Figure 4: Standard deviation of 10

Applying Gaussian filters of higher σ creates a more blurry image as they are a low-pass filter that attenuates

high-frequency components in an image (i.e. reducing the contrast between neighboring pixels). Examining the above figures, we see that the amount of blurring is clearly dictated by the standard deviation of the Gaussian function. A higher σ implies a wider Gaussian distribution and a more gradual decrease in the filter's response as the distance from the center increases. As a result, the filter has a greater smoothing effect on the image. This leads to a reduction in image sharpness and an increase in overall image blur. In contrast, a lower sigma value creates a narrower Gaussian distribution and a more rapid decrease in the filter's response, resulting in relatively less blurring and therefore more image detail.

Filters

Figure 5: h1 filter

Figure 6: h2 filter

Figure 7: h3 filter

The filter h1 appears to have the effect of sharpening and darkening the image by subtracting its neighboring pixels from each pixel in the input image. The filter matrix has a center value of 2, which places positive weight on the current pixel. The other values in the matrix (all $\frac{-1}{9}$) serve to reduce the overall brightness of the output image and invert its polarity, thereby enhancing contrast and distinct features. As a result, the filtered pixel value is higher if the neighbors are darker, and lower if they are brighter. All of this has the expected effect of creating a more pronounced edges and making the overall image darker.

The filter h2 appears to have the effect of sharpening the image along the horizontal direction since it is a 1×3 matrix. The central value (3) would positively weigh the current pixel while the adjacent values (-1) would negatively weigh the left and right pixels. As such, the output pixel would represent the difference in brightness between adjacent columns of pixels. This serves to highlight the horizontal edges in the image since if there is a significant difference in value between two neighboring pixels and the current pixel, the output pixel will have a high absolute value and vice versa.

Similarly, the filter h3 has the effect of sharpening the image along the vertical direction. As with h2, The central value (3) would positively weigh the current pixel while the values (-1) would negatively weigh the top and bottom pixels. This produces a filtered pixel value that indicates the difference in brightness between adjacent rows of pixels. This serves to highlight the vertical edges in the image since if there is a significant difference in value between two neighboring pixels and the current pixel, the output pixel will have a high absolute value and vice versa.

Part 2: Edge Detection

Noise reduction with Gaussian filtering

The implementation of the Canny edge detection algorithm beings with smoothing the image with a Gaussian filter. This simply involved convolving the input image with the Gaussian filter. Testing the algorithm with various σ values (1, 2, and 3) resulted in the following filtered images. As we've discussed previously, filtering with larger σ values result in more smoothed/blurry images.

Figure 8: sigma = 1

Figure 9: sigma = 2

Figure 10: sigma = 3

Finding image gradients

The next step involves finding the image gradients using the Sobel filter and computing the gradient direction as well as magnitude. Figures in this section will correspond to the algorithm employing parameters $\sigma = 1$, t_low= 0.4, and t_high= 0.7. Out of the various edge detection results, the edges obtained using these parameters performed best. To compute the each derivative, we convolve the filtered image with the corresponding Sobel filters.

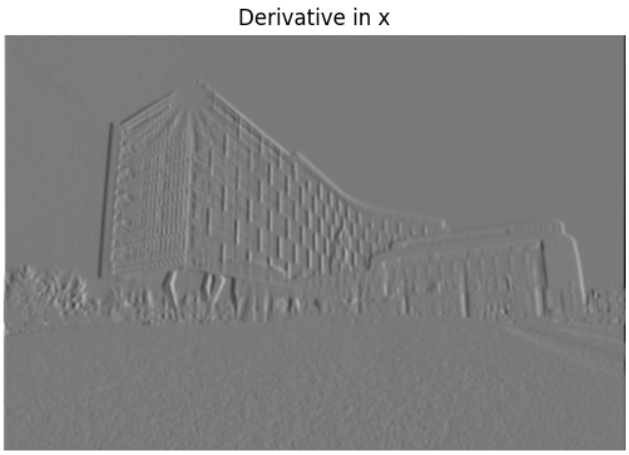


Figure 11: Visualization of the derivative $D_x(x, y)$

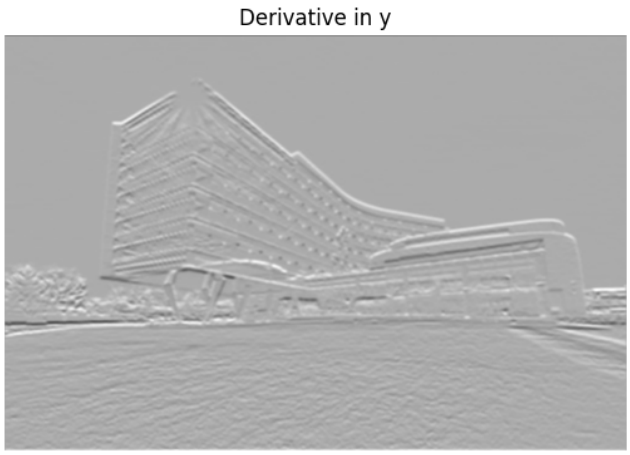


Figure 12: Visualization of the derivative $D_y(x, y)$

In **Figure 11** and **Figure 12**, we see how applying the filters output maps of gradients in the x and y direction. The brighter pixels indicate areas of the image where the gradient is higher in that direction.
We get the gradient magnitude by employing the Pythagorean theorem and the derivatives calculated previously.

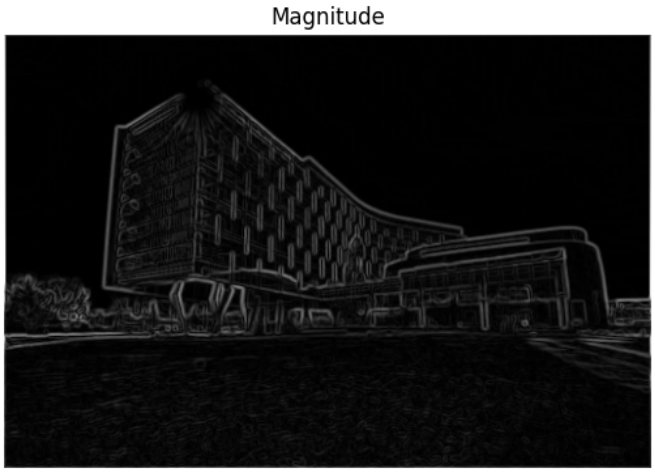


Figure 13: Visualization of the gradient magnitude.

We see in **Figure 13** that the highest values of the gradient magnitude are indicated by brighter pixels. As such, these pixels in the original image are more likely to be part of an edge.
The gradient direction θ is acquired by solving $\arctan(\frac{G_y}{G_x})$.

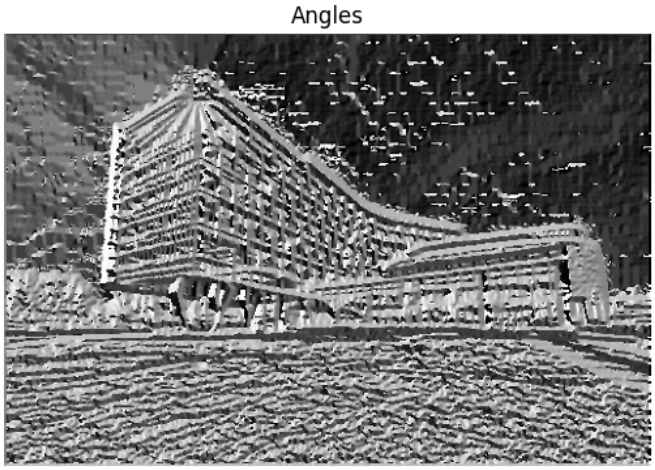


Figure 14: Visualization of the angle of the gradient.

Figure 14 demonstrates the direction in which the intensity of the image is changing most rapidly. For example, if the gradient angle at a pixel is near 0 degree, it is more likely to be vertical edge.
We quantize values of θ to one of four directions: 0, 45, 90, or 135. To do so, I iterate over each element in θ and calculate whether it belongs in certain ranges.

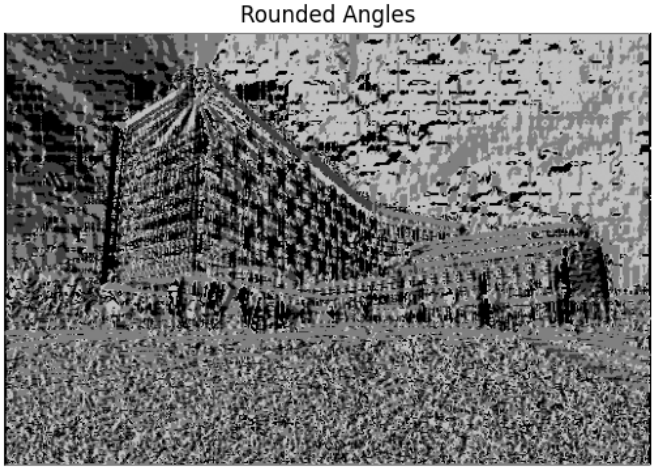


Figure 15: Visualization of the rounded angle of the gradient.

The resulting gradient direction θ' will a finite set of discrete angles, which simplifies the process of edge thinning or non-maximum suppression. This process also helps to reduce noise, as it will suppress smaller variation in the angles.

Edge thinning / Non-maximum suppression

Before applying hysteresis thresholding, we perform non-maximum suppression to thin the edges. To achieve this, I iterate over each element in the gradient magnitude and check whether it is larger than the adjacent elements in the corresponding angle. If not, that edge will be thinned. Again, figures in this section will correspond to the algorithm employing parameters $\sigma = 1$, t_low= 0.4, and t_high= 0.7.

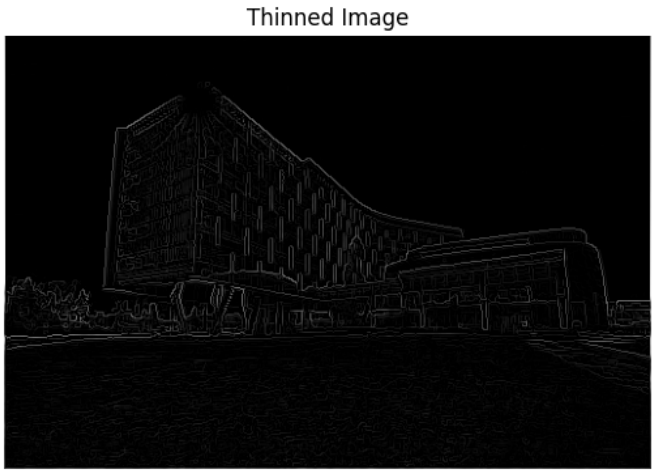


Figure 16: Visualization of the thinned edges.

Clearly, the edges in **Figure 16** are less thick compared to the gradient magnitude and preserved the strongest edges while suppressing the weaker one.

Hysteresis Thresholding

Last step for the algorithm is to filter out noise via hysteresis thresholding. To do so, I applied a double threshold technique to the gradient magnitude using a lower and upper threshold. Potential edges (i.e. pixels where magnitude $>$ upper threshold (a.k.a strong edge) or upper threshold $>$ magnitudes $>$ lower threshold (a.k.a weak edge)) are labeled such that each connected component has a unique integer value. Then, for each pixel in the labeled, I check whether the corresponding magnitude value is a strong edge. If so, it is definitely an edge. Otherwise, I look at whether there is a single element in the same dimension horizontally, vertically, or diagonally is a strong edge. If so, I convert all of those elements to be an edge in gradient magnitude. This will ultimately convert an entire component with a strong edge to be edges.

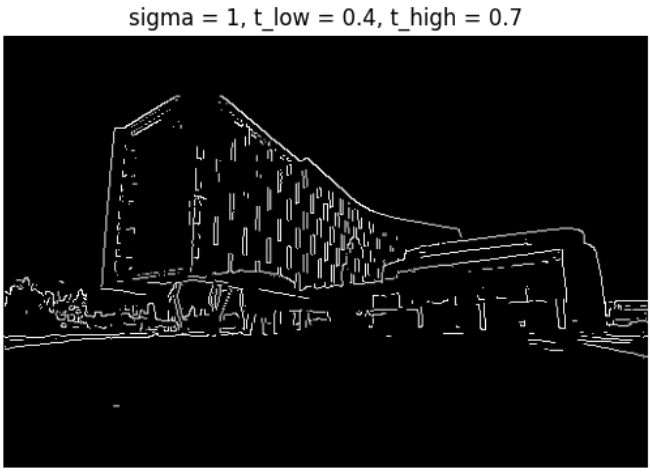
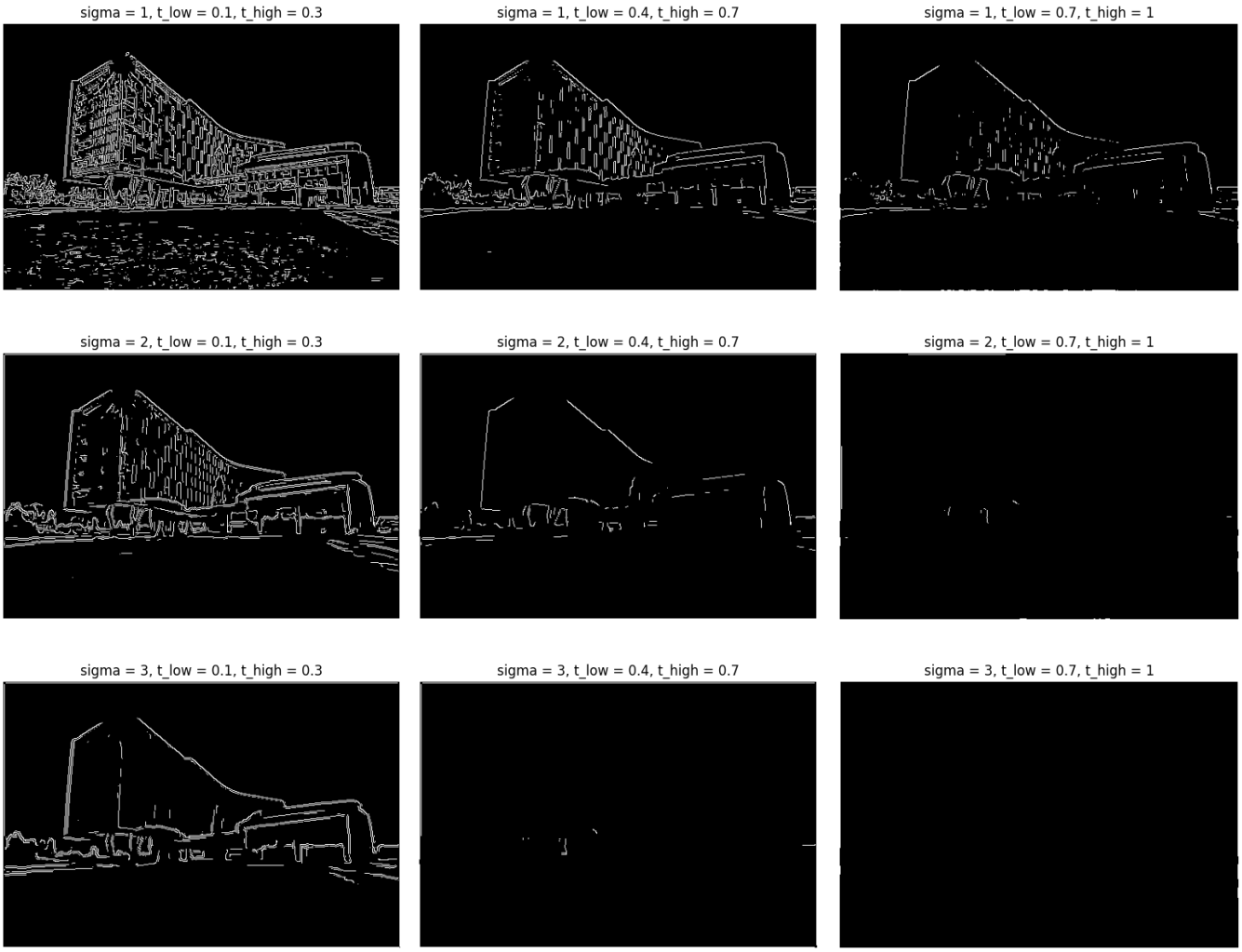


Figure 17: Visualization of the edges detected by the Canny edge detection algorithm.

In **Figure 17**, we notice how noise edges like that of grass are mostly removed while the edges of the building become more pronounced.

Testing canny edge detection

Experimenting with various parameter values, we notice the impact of increasing/decreasing σ, t_low, and t_high



By inspection, we see how increasing σ reduces the number of edges present in the final results. This is due to the fact that applying Gaussian filters with larger standard of deviation makes images more smoothed. As a result, the edges will be less distinct and harder to determine. We also notice that by increasing t_low and t_high, more edges are blacked out. This makes sense as higher threshold more rigorously suppress weaker edges. Independently, increasing t_low values means more gaps are allowed to form. This is because weaker edges that should be included are instead ruled out since they do not meet the lower threshold. On the other hand, increasing the t_high value itself will have the effect of removing connected edges all together. More edges will be unable to fulfil the criteria and weaker edges connected to them will also disappear.

Extra Credit: Hybrid Images

To create a hybrid image, I combined the low frequencies of one image with the high frequencies of another. I achieve this by first applying a low-pass filter to one of the images. In this case, I decided to use the Gaussian kernel as the low-pass filter since it effectively blurs the image. Second, I applied a high-pass filter to the other image. The high-filter is applied by subtracting the low-pass filtered version of the original image. This effectively removes the low-frequency components of the image, leaving only the high-frequency details. Finally, the low-pass filtered image and high-pass filtered image are then added together to form the hybrid image.

Figure 18: A hybrid image constructed from high-frequency components of a gray-scale image of grazing bisons and low-frequency components of a gray-scale image of a resting tiger.

In the above image, we see that it contains the low-frequency information of the tiger and the high-frequency information of the bisons.