Daniel Yu-cua

CMSC 426 Project 1 Report

1. Introduction:

This project posted challenges in image processing and the mathematics involved in such processes. The hardest part of the project was definitively edge detection as it was both the most complicated and the most math intensive.

2. Filtering

2.1: Gaussian Kernel

Constructing the gaussian kernel was not particularly difficult an example from [1]. Essentially all we had to do was create a 3 * sigma grid then fill it with values resulting from the class given equation.

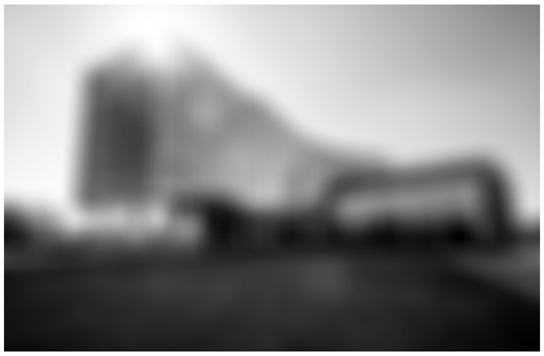
2.2: Convolution

Essentially this was multiplying the matrix we had generated from 2.1 on to the given image. Code was referenced from [2] to see how to efficiently convolve the arrays together.

By using both the Gaussian Kernel and the convolution function we are then able to apply the kernel onto any image we desire.

Gaussian kernel set to sigma 3

Gaussian kernel set to sigma 5


Gaussian kernel set to sigma 10

As we can see with each progressive kernel increase the image sucessfully gets blurrier as more and more values are being averaged together.

We can also use the convolution function to then apply the given filters from the project code.

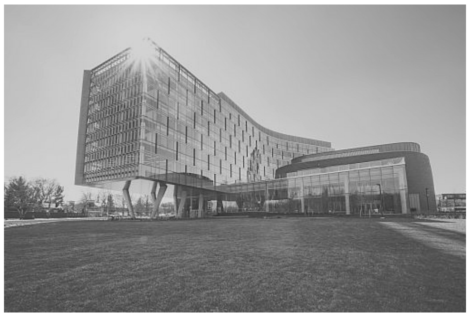


Figure 1.

Original

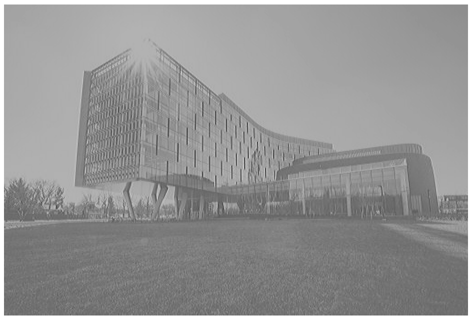


Figure 2.

Original

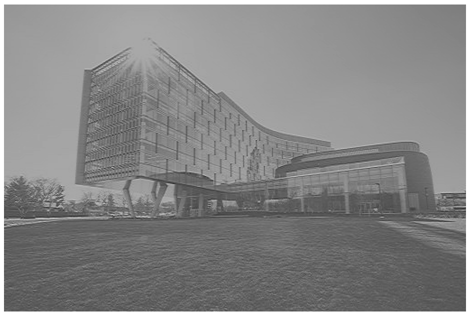


Figure 3.

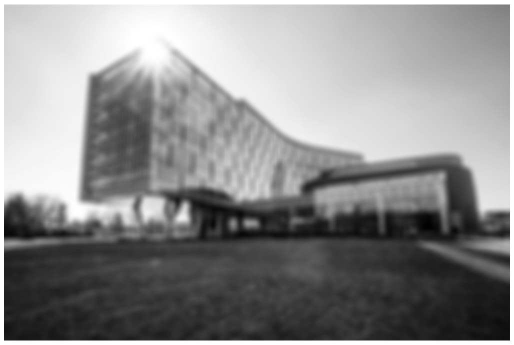
Original

As we can see with the given filters an ever-increasing fade effect is put onto the images. With each filter we can see that the image mimics that of the fade effect in photoshop or Instagram. It also seems that the intensity of the image is being averaged out while also being darkened.

3. Edge Detection

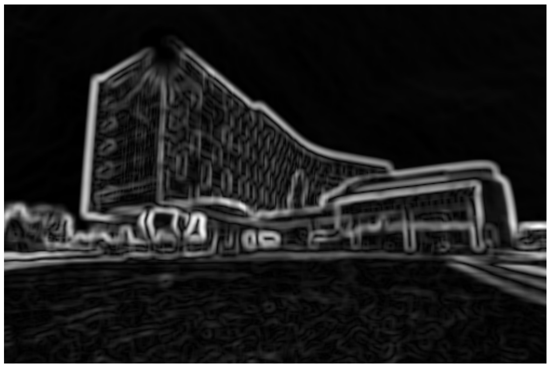
3.1: Noise reduction with Gaussian filtering

All we need to do is simply apply the gaussian filter that we coded in 2.1 to the image of the iribe to get a smoothed-out picture.


Gaussian kernel set to sigma 3

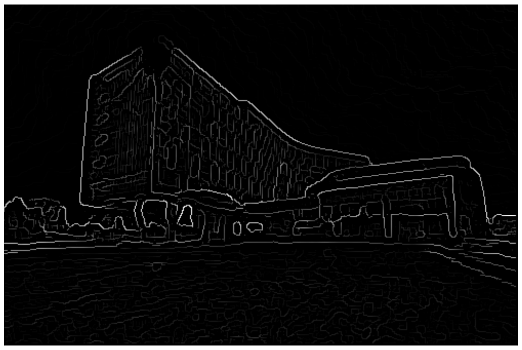
3.2: Image Gradients

We then compute the image gradients buy applying the given filters onto the image. We then take the two new images generated by the filter applications get the hypotenuse between the two matrices to get G then divide by G.max() and multiply by the max intensity value of (255) to get a rough edge of the image.



3.3: Edge thining

We can then get theta by taking the arctan of the filtered images. To thin the edges, we apply the formula described to us in the project to reject or accept pixels given theta.

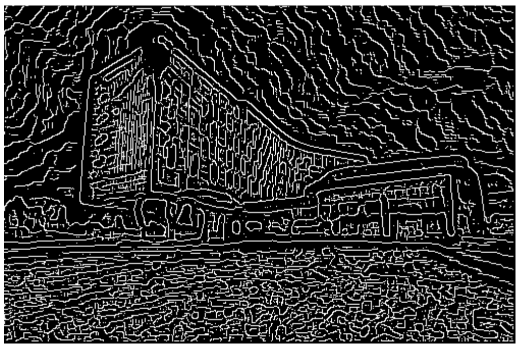


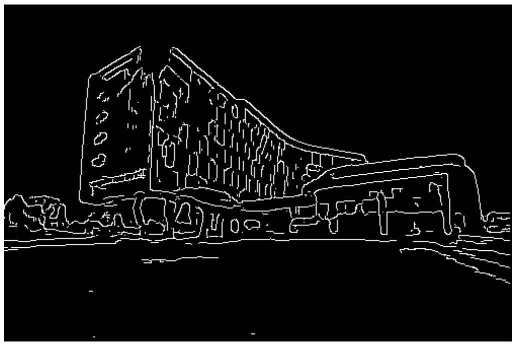
3.4: Hystersis Thresholding

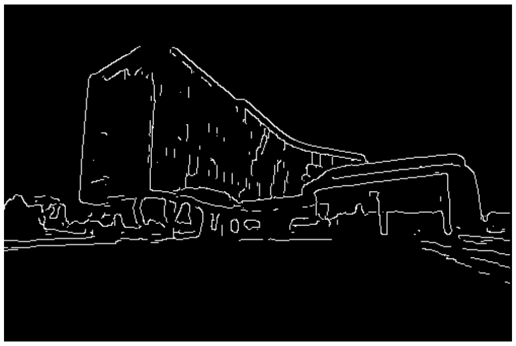
We now introduce a minimum and a maximum threshold for the image. Any pixel that is above the maximum threshold is automatically taken to the max value of 255 while any pixel that is below the minimum value is then tossed and set to the min value of 0. Any pixel that falls between the two is evaluated to check whether it is part of a path that contains a pixel that is of the max value.

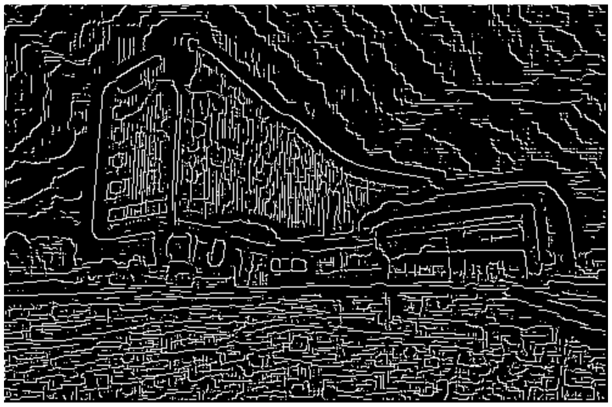
The way this is calculated is by applying the ndimage label function onto our image. This gives us an np.array with each feature or path in our image labeled from 1 to n. We can then convert this into a dictionary with the path labels as keys (1 -n) and the values being the coordinates of the paths themselves. This can be converted into a dictionary where the values of the keys can either be true or false depending on whether a path contains a pixel with a maximum value.

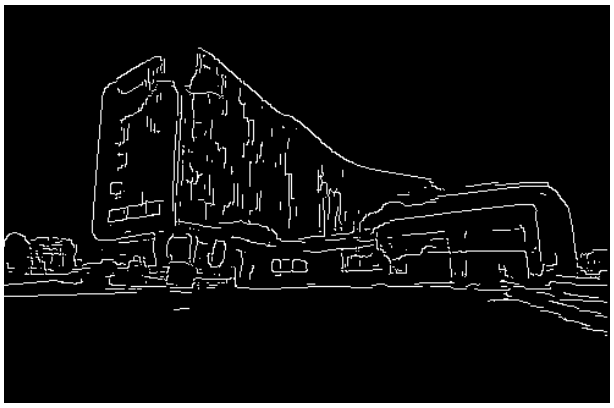
By playing around with the min and max thresholds we can find a proper balance to not introducing too much noise and without missing out on important information.

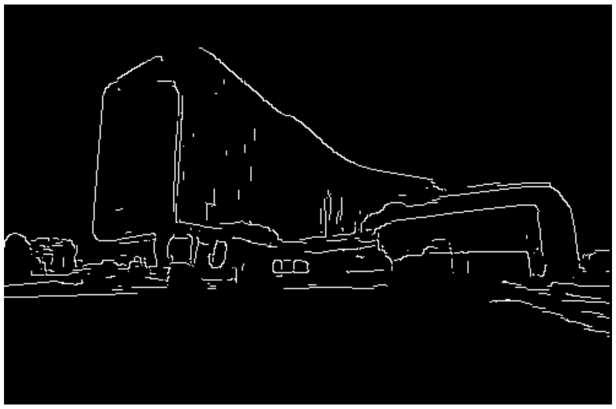

Edge detection Sig: 3 Thresh: 0.5 – 1.0


Edge detection Sig: 3 Thresh: 10 – 50

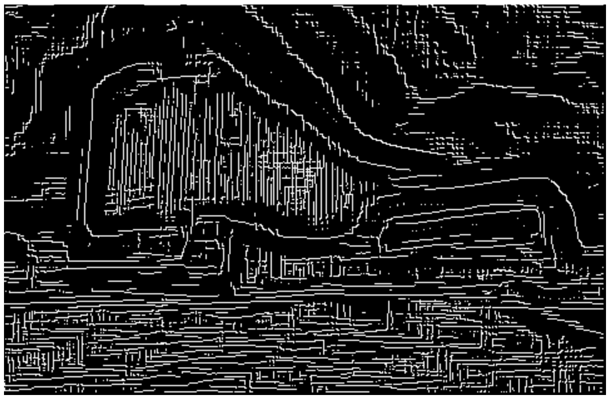

Edge detection Sig:3 Thresh: 5 – 80

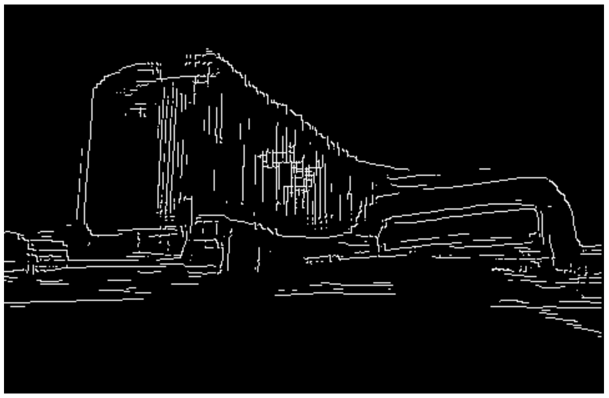

Edge detection Sig: 5 Thresh: 0.5 – 1.0

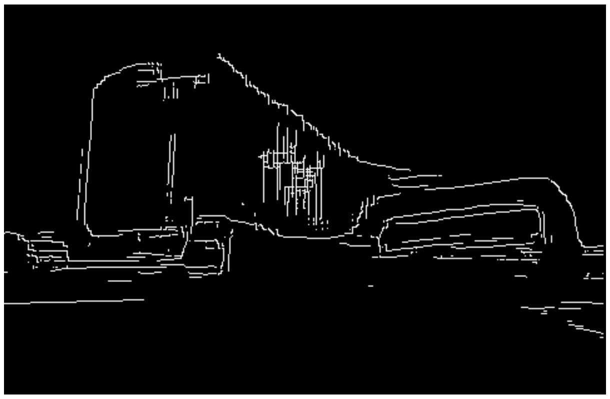

Edge detection Sig: 5 Thresh: 10 - 50


Edge detection Sig: 5 Thresh: 5 – 80

As sig increases, we get a blurrier image and as such we get more sporadic edges that do not fit and curve as well as with a sigma of a lower value.


Edge detection Sig: 10 Thresh: 0.5 – 1.0


Edge detection Sig: 10 Thresh: 10 – 50


Edge detection Sig: 10 Thresh: 5 - 80

In the most extreme case of sigma being 10 we see that almost no curved lines exist, and every path seems to have devolved to create some sort of abstract expressionist art.

4. Hybrid Imaging

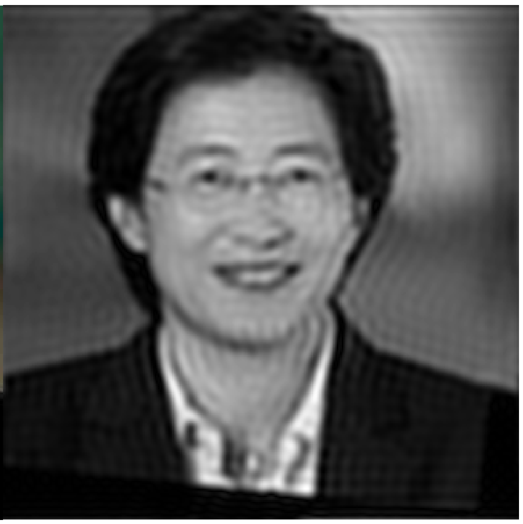
4.1: Filter creation [4]

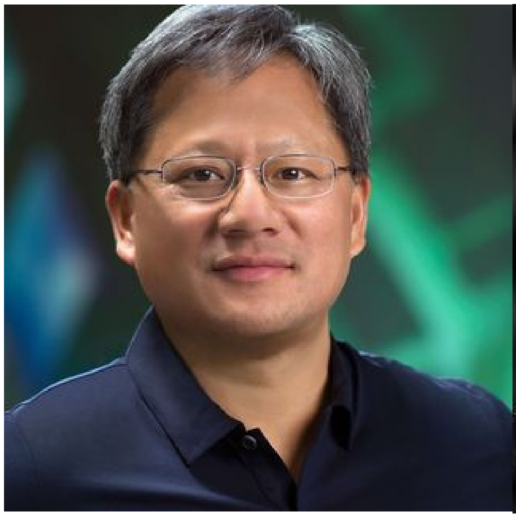
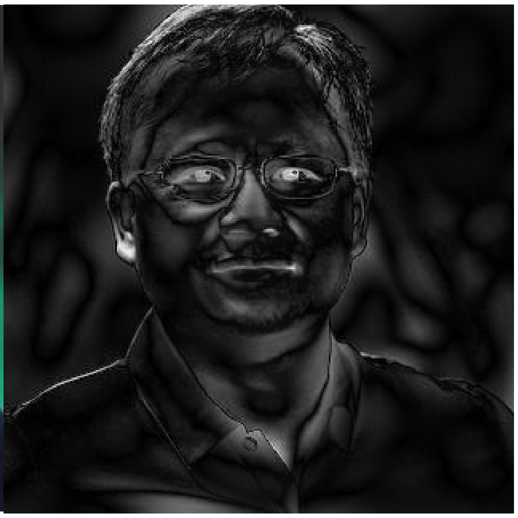
There are two steps to creating the hybrid image shown below. We need to create the filters used by high pass and low pass filters used by fft domains. The low pass filter being an array of zeros with ones amassing in a circle at the center. The high pass filter being the exact inverse with ones filling most of the array and zeros forming a circle in the center.

4.2: Fourier domains

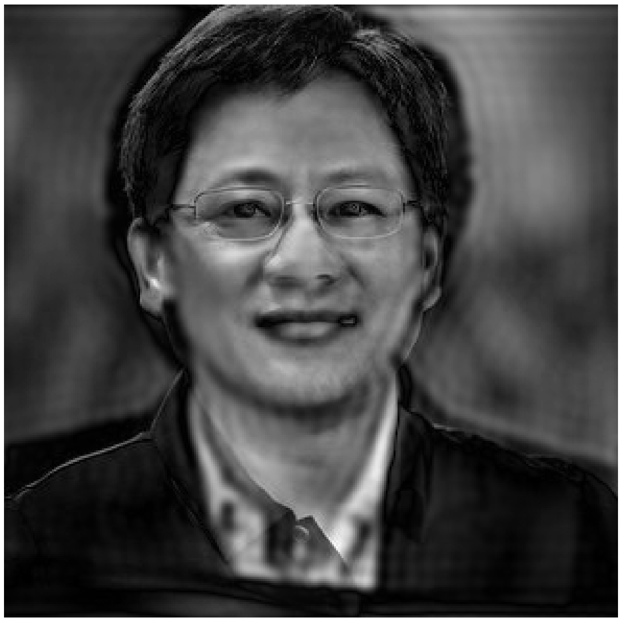
We can then take those filters and them apply them to the code provided in lecture 8. As we can see below, we have the CEO of AMD Lisa Su and the CEO of NVIDIA Jensen Huang respectively. By taking the low frequencies of the image of Lisa Su and the high frequency image of Jensen Huang we can simply add the two images together to achieve our hybrid image.

AMD CEO: Lisa Su low frequency



NVIDIA CEO: Jensen Huang high frequency



HYBRID IMAGE

Bibliography

[1] https://stackoverflow.com/questions/29731726/how-to-calculate-a-gaussian-kernel-matrix-efficiently-in-numpy.

[2] https://www.reddit.com/r/learnpython/comments/79l86u/convolution_of_matrices_in_numpy/

[3] https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

[4] https://hicraigchen.medium.com/digital-image-processing-using-fourier-transform-in-python-bcb49424fd82