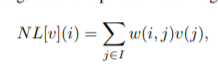
6)

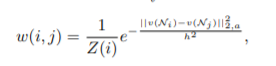
Non Local Means

Non local means denoising uses the self-similarity commonly found in natural images to denoise them. If we can find the similar pixel block to the one we are trying to denoise within the same image, we can just average them, and we would reduce the gaussian noise. There are two main steps to this type of image denoising: finding the similar blocks of pixels and determining how to combine them. An easy approach is to use mean squared error to determine the similarity of the blocks of pixels. Once we have found our blocks of pixels, we could do a weighted average of the average of each block, based on how similar the pixel block is to the one we are trying to denoise. There are many ways to do these steps, but this is the basic idea for non-local means denoising.

I used the prebuilt sci-kit image library to implement the NLM denoising. Their NLM denoising algorithm is as follows:



Where ‘I’ is the area in which we want to search for, ‘i’ is the pixel we are denoising, and ‘j’ is the pixel currently being used to denoise. The function ‘w(i,j)’ computes the weights for the values ‘v(j)’ of the noisy image.



Here, Ni and Nj are the neighborhoods around pixels i and j respectively. We compute their Euclidian distance, and divide by ‘h^2’, which is a parameter that controls the rate of decay.



Z(i) is a normalizing factor of the above form, such that all weights sum to 1.

For the experimentation part of the project, I compared the performance of a NLM denoising scheme for different types of images. I found that generally, NLM denoising works best for large images with bland color schemes. This was most clear with the Mario images I put through the algorithm. When I used a large Mario image, I could add enough gaussian noise so that the image was barely recognizable, and then I could use NLM denoising to make it fairly clear. When I cropped Mario to about one tenth of his original size, the performance of the NLM denoising suffered tremendously. The MSE of the small image compared to its base was about 750% larger than that of the large image compared to its base. This could also be caused by the different color scheme in the cropped image, if Mario’s face is more complex than the rest of the image.

Most natural images have self-similarity, which is the basis for non-local means denoising, so I decided to compare the performance of the NLM denoiser on a natural photograph and an unnatural photoshopped image. The photoshopped image is a somewhat famous image that was photoshopped so that no items in the picture can be distinguished, but it looks like a normal photograph at first glance. As expected, the MSE for the photograph was half of the MSE of the unnatural image. Although the images were of similar size, the color spectrum in the real photograph was beneficial for the NLM denoising algorithm. As I increased the noise so that both images were unrecognizable, however, they performed similarly.

I found that there are many ways of improving the performance of the NLM algorithm in python with little effort, but they increase the time needed to produce the final image. For example, cascading NLM filters always reduced the MSE of the final image, but it is time consuming and has diminishing improvements with each cascade. Also, we could allow for more dissimilar patches to be averaged into each pixel by increasing our h value. This usually decreased the MSE, but blurred the images. We could also increase our search area, but this would greatly increase the computation time as our computation has complexity (image.size)\*(patch\_size^(image dimensions))\*(patch\_distance^(image dimensions)). This was one of the areas where I saw that a lower MSE doesn’t always mean a better image visually, as the blurring from these last two methods actually decreased the clarity of the image. I also noticed this with the large Mario image, which had the best visual performance by far, but not necessarily the best MSE performance when compared to the smaller images.