Exercise 3: Non Local Means

Non Local Means – Code Explanation:

"Non local means" is a method for denoising pictures. It separates every pixel of the noise picture by getting a squared patch around it. Then, by calculating the Euclidean distance between those patches (weights), we get a number that corresponds to how much those patches look like. To be able to get a patch for every pixel, we have to expand the image to every side. The new size depends on the size of patch.

$$w(i,j) = \frac{1}{Z(i)} e^{-\frac{||f(\mathcal{N}_i) - f(\mathcal{N}_j)||^2_{G(a)}}{\sigma^2}},$$

$$Z(i) = \sum_j e^{-\frac{||f(\mathcal{N}_i) - f(\mathcal{N}_j)||^2_{G(a)}}{\sigma^2}},$$

For example, if we have an image of 64x64 and patch size of 3x3, the new size will be calculated with this equation *"newSize = size + (patchSize-1)".*

Afterwards, we should add some values in those new lines and columns. Ways to do that is to mirror the values of the noise image or add the average value of each side.

Then we can get the patches for every pixel, calculate the weights and finally denoise the picture.

**Resized Images can be found on GitHub in folder "Figures".*

$\hat{f}(\mathbf{x}) = \sum_{\mathbf{y}\in\Omega} w(\mathbf{x},\mathbf{y}) f(\mathbf{y}), \quad \forall \mathbf{x} \in \Omega,$ $\hat{f}(\mathbf{x})$: Denoised Pixel x

$w(\mathbf{x},\mathbf{y})$: Weight between patch x and y $f(\mathbf{y})$: Noise Pixel y

a. *Description of the parallelism method:*

To begin with, firstly i created the sequential method, as I explained above, using *"C"*. I came to the realization that some parts of the code had a high complexity. To be precise, the part where the weights are being calculated, the complexity is *"O(n^4)"*.

For example, a picture with 64x64 pixels, took around 6 to 7 seconds for the sequential method "C". For bigger sizes it took even much more.

To reduce that time, i had to lower as much as I could the complexity of *"O(n^4)"*. Using *"CUDA"* I was able to make it equal to *"O(n^2)"*. By setting *"n"* blocks with *"n"* threads each block for the *"nonLocalMeans"* function, the function is being parallel called n^2 times.

Thus, instead of calculating the whole *"array of weights"* and afterwards creating the *"filtered – denoised image"*, I could calculate only the specific weight values that I needed to create each pixel in every call of the function. By doing this for n^2 times, *(Blocks=n, Threads=n)*, we have as a result the *"filtered - denoised image"*.

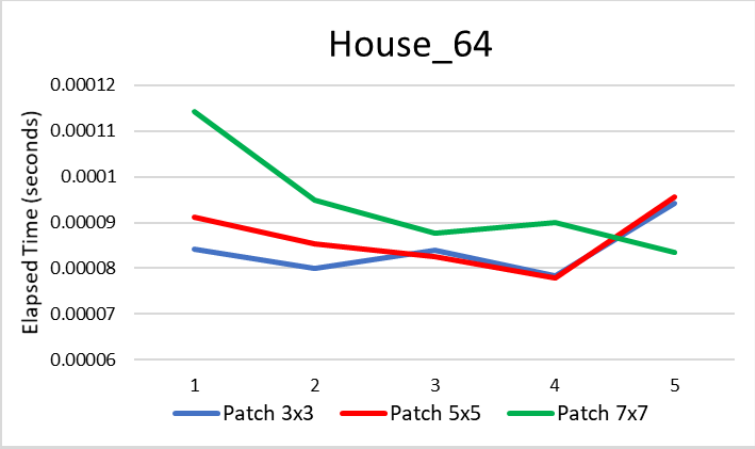
b. *Design and description of artificial entrance for accuracy check*

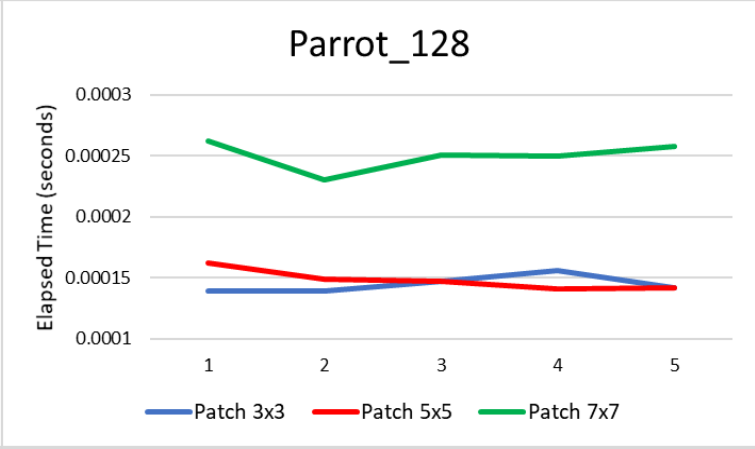
For pictures, i download *".jpg"* files and convert them to *".txt"* files with *"Matlab"*. Also, because the pixels were *"integers"*, i use a *"C"* code where i transform them into *"float"*.

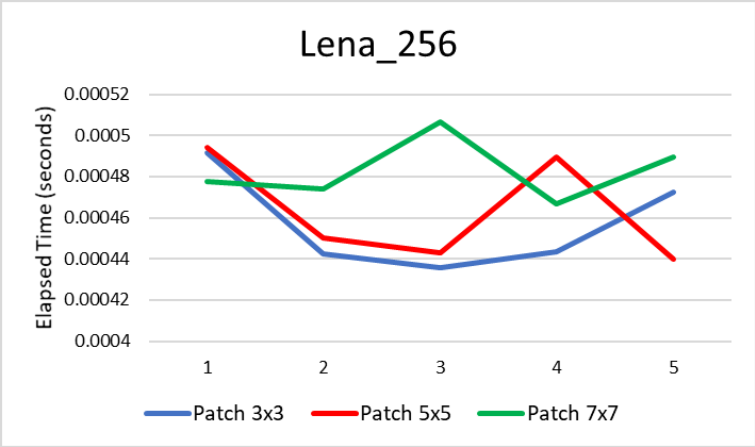
Then to check if my code was working correctly, i save the *"filtered image"* into a text file and use a *"Matlab"* script for visualization. I also save the "resized image" to check how the image look like after being expanded.

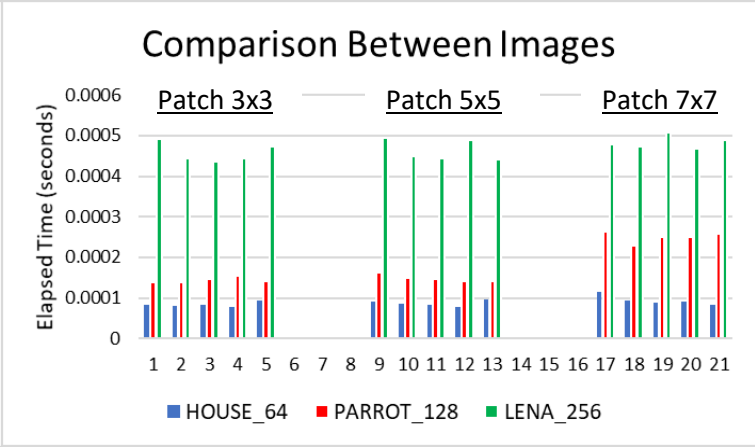
**"C" codes and "Matlab" scripts can be found on GitHub in Folder "Extra".*

c. *Comments - Elapsed Time:*









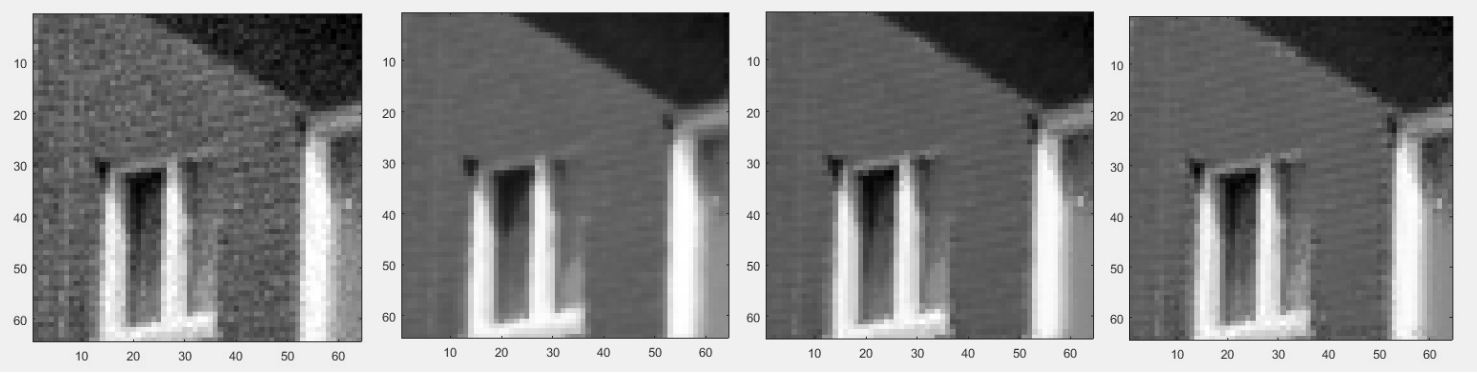
**Elapsed Time: Resizing Process + Calculating Weights + Denoising Image.*
**For compiling/running I used Google Colab. Instructions for it can be found on GitHub Link.*

To make sure that the elapsed time that I was getting as output, was correct, I use the "sleep()" function from "unistd.h" library.

First of all, we can see that the elapsed times have been extremely reduced in comparison with the sequential code in C. Also, I have to mention that the sequential method with CUDA, was also much faster than the sequential with C.
The reason why is that being parallel helps to reduce the complexity of the code, but the elapsed time is also being affected by the runtime type. GPUs runs a lot faster than CPUs. Furthermore, according to the graphs above, it is noticeable that the patch size affects the elapsed time but not as much as the picture size does. For different size of patches, the times differ by little. On the other hand, when the size of pictures doubles, the time approximately doubles too. *("Comparison Between Images")*
Those times were been calculated by declaring the "Gauss Filter Array" as shared. By being declared as global, it did not make that much different but that is because of its size.

d. *Comments - Denoising:*

1. *House 64x64:*



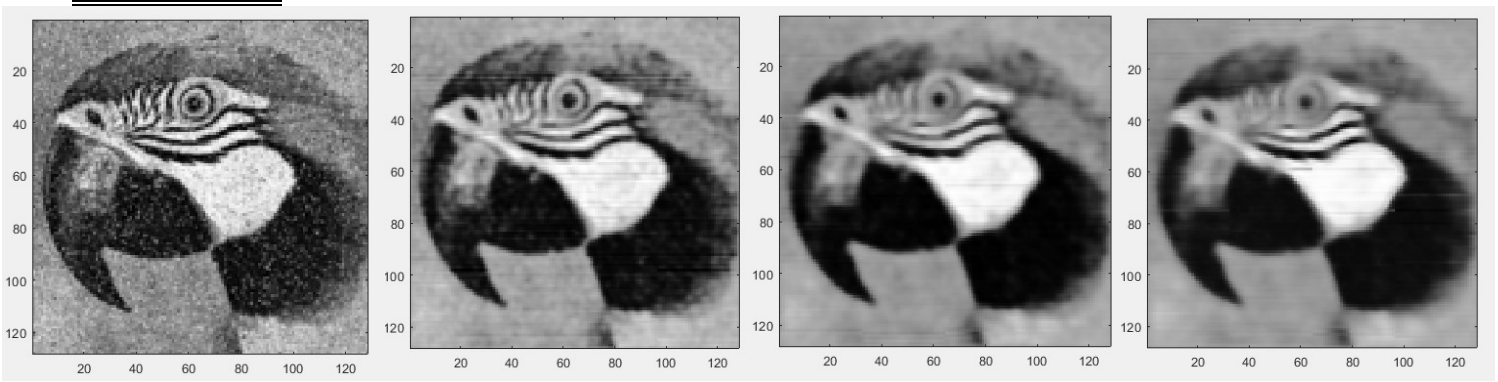
d) Original Image

c) Denoised Image *(Patch Size 3x3)*

b) Denoised Image *(Patch Size 5x5)*

a) Denoised Image *(Patch Size 7x7)*

2. *Parrot 128x128:*



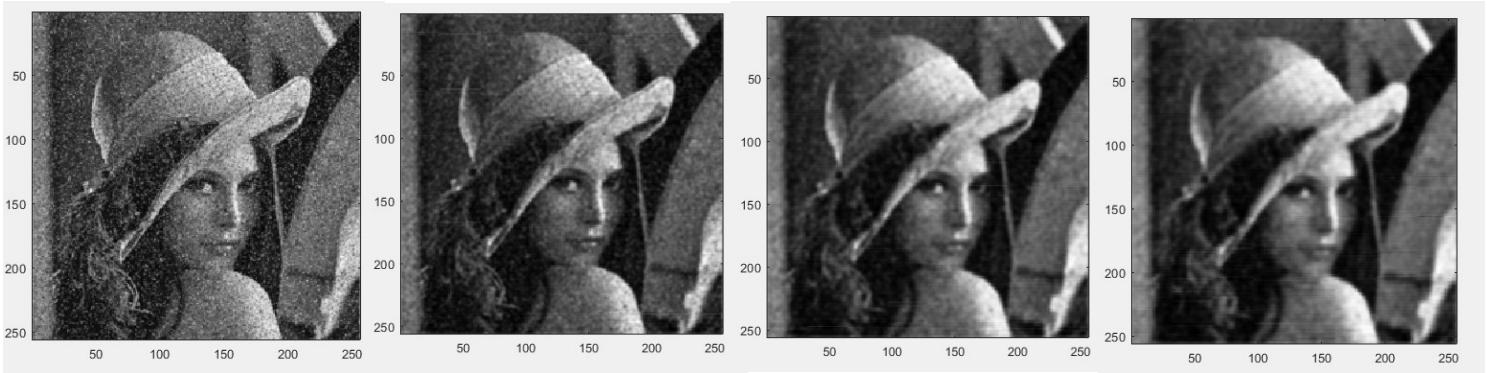
a) Original Image

b) Denoised Image *(Patch Size 3x3)*

c) Denoised Image *(Patch Size 5x5)*

d) Denoised Image *(Patch Size 7x7)*

3. *Lena 256x256*



a) Original Image

c) Denoised Image *(Patch Size 3x3)*

b) Denoised Image *(Patch Size 5x5)*

a) Denoised Image *(Patch Size 7x7)*

*As I said before, "Resized Images" can be found on GitHub in folder "Figures".

By comparing the figures of each picture for different sizes of patch size, I came to conclusion that the smaller the picture is, the denoising will be better for a smaller patch too. The opposite happens as the picture gets bigger.
To be specific, the "House" with a patch 3x3, the "Parrot" with a patch 5x5 and the "Lena" with a patch 7x7, have better denoising effect than the rest patch sizes.

GitHub Link: https://github.com/mariostavr/Non-Local-Means