Project Report

On

PLOW FILTER

Submitted by:

Pranjal Somkuwar	180001036
Vinesh Katewa	180001061

Computer Science and Engineering

2nd year

Under the Guidance of

Dr. Kapil Ahuja

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Spring 2020

Contents:

Introduction:

Images play a large role in our day to day life, they have a wide range of applications from capturing events and trips to more serious applications including surveillance, security and medical fields. However, images that are captured using modern cameras or images from satellites are heavily corrupted with noise which makes it hard to use them for surveillance and security as the information cannot be easily interpreted and sometimes can also be wrongfully interpreted. While advances in optics and hardware technologies try to reduce these noises from captured images they usually have more cost to them therefore, software based denoising approaches are more popular as they are device independent and widely applicable. Image denoising is a process by which the resolution of an image is increased so as to create a more sharp and visually appealing image without the loss of finer details.

Objective:

- Implementing the wiener filter to remove gaussian noise from a noisy image.
- Using Gaussian Kernel to improve the quality of image.
- Complexity analysis of the algorithm.
- If possible improve the existing wiener filter

Algorithm Analysis:

Wiener Filter:

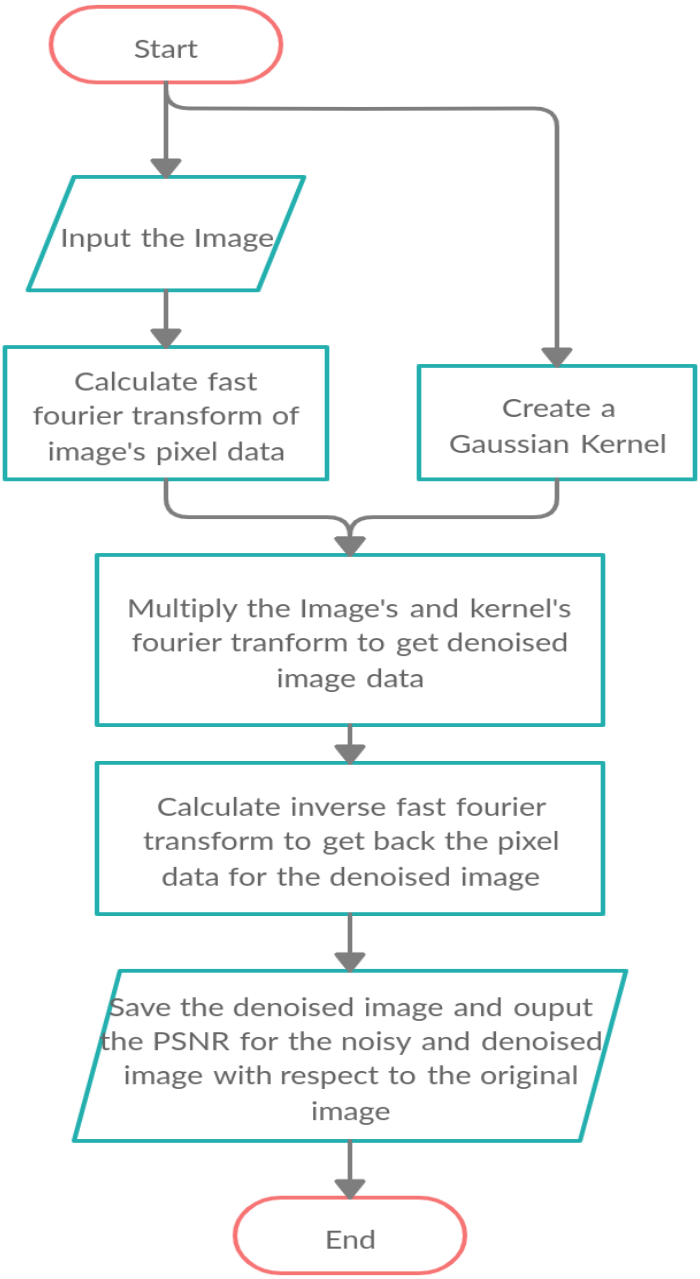
Wiener filter uses the gaussian kernel to average over the pixels of an image to remove the noise and smooth the image. Using this filter we take a small kernel preferrably 3x3 or 5x5 for averaging the value of pixels. The averaging is done by a gaussian kernel which gives more weights to the pixels at the centre and reduces the weights of the pixels as they go further away. The grayscale image is first copied over into an array then a fast fourier transform function is called over the entire array of the image. This function runs with computational complexity of O(nlogn). Same is done with our gaussian kernel. Now we multiply the image and the kernel to get a new image data, it is then passed through inverse fast fourier transform to get the image's pixel data.

Pseudo Code:

- Img = Input image with gaussian noise
- Gaussian_kernel = Kernel with gaussian distribution, default size as 3x3
- For each pixel data in image:
    - Calculate fast fourier transform
- Result = Matrix multiply Img and Gaussian_kernel
- For each data in Result:
    - Calculate inverse fast fourier transform
- Return the Result of the inverse fast fourier transform

Algorithm Design:

Wiener Filter:



Complexity Analysis:

First we have to input the image, Image is essentially an array of data where each value represents the intensity of a particular pixel. For example, a value of [0, 0, 255] in a 3 dimensional array will represent the intensity of red and green light of the pixel as zero and 255 (maximum value) of the blue light i.e. [0, 0, 255] represents a pixel of blue color. But this algorithm only works on black and white i.e. grayscale images. Grayscale images, unlike rgb, only contain data in 2 dimensions. Where each value represents the intensity of white light i.e. 0 means black and 255 means absolute white. So, while inputting an image we create a matrix with a number of entries same as that of pixels in the image. Hence, complexity of this step is O(n) where n is the number of pixels.

We also need a noisy image to do the filtering on, we can create our own noisy image by writing a small function for that. We create a normal gaussian distribution with size same as that of our image by using a predefined function in numpy library i.e. np.random.normal(). It takes parameters as mean, width of the distribution and size of the image. We then simply add both these arrays and check the values, if any of them is greater than 255 we set them to 255 and any less than 0 is set to 0 because pixel intensities lie from 0 to 255. Also, the mean of the distribution is set to 0, because by increasing this value we are increasing the intensity of pixels and thus making the image brighter. This process iterates over the entire array of data, making the complexity of this process O(n).

In the second step, we compute a gaussian kernel matrix of size 3x3 or 5x5 and we also calculate the fast fourier transform of our image's pixel data. The former can be given a complexity of O(1) but the latter is computationally heavy and the most efficient algorithm has a complexity of O(nlogn).

We do the same with the gaussian kernel and increase its size to be the same as that of our input image. This process is done in such a way that it does not create new data. Since we are increasing the size to be the same as our image this process will have a complexity of O(n).

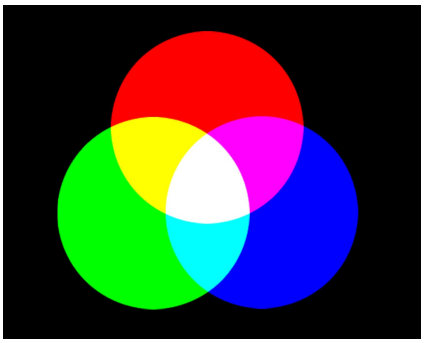
Now, we are multiplying the image and the gaussian kernel to get our filtered image. This will be a matrix multiplication which has a complexity of O(n^3).

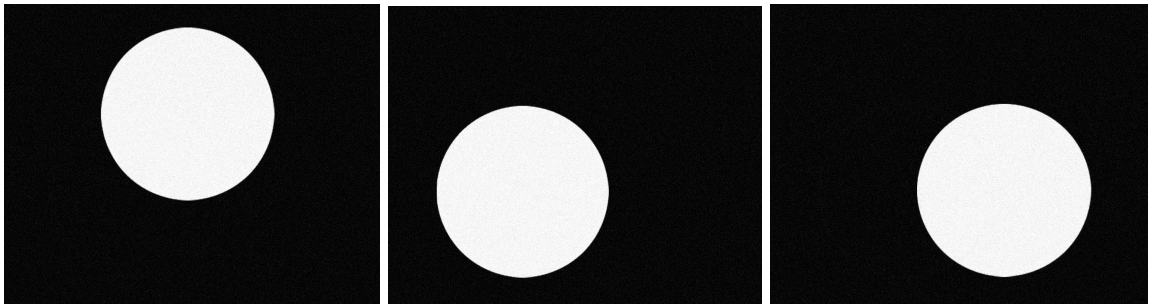
Finally, we calculate the inverse fast fourier transform of the result of matrix multiplication. This is the pixel data of the filtered image. We then save this image in png/jpeg format for further reference and also calculate the PSNR(peak signal-to-noise ratio) of the noisy and filtered image with respect to the original image. For calculating psnr value we simply go over each pixel value once and hence, this step has a complexity of O(n).

The function that calculates the psnr value is fairly simple. First it calculates the standard deviation of the image in question with respect to the original image. The standard deviation can have a large number for even a small amount of noise if the resolution of image is high, to counter this we take the logarithm of division of maximum pixels and standard deviation of pixels. And returns the value multiplied by 20. This process again has a complexity of O(n) where n is the number of pixels.

Optimization and Modification:

Wiener filter was originally designed to work only on images that had data in 2 dimensional format i.e. grayscale images. We further modified the algorithm to work for colored images as well. To do this we simply took the information of rgb pixels and split the entire rgb image into 3 grayscale images where each image's data represents the red, green and blue pixel data of the original data. After that the wiener filter is applied on all of them separately. Finally, we combine these 3 2-dimensional arrays into a single 3-dimensional array to create a rgb image that has been filtered by the wiener filter algorithm.



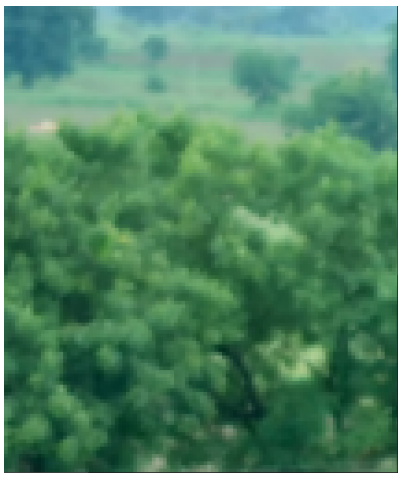
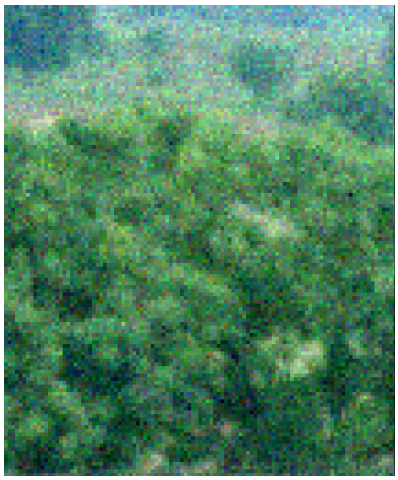
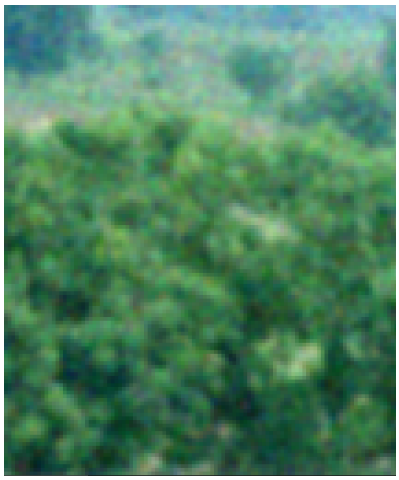


Red pixels Green pixels Blue pixels

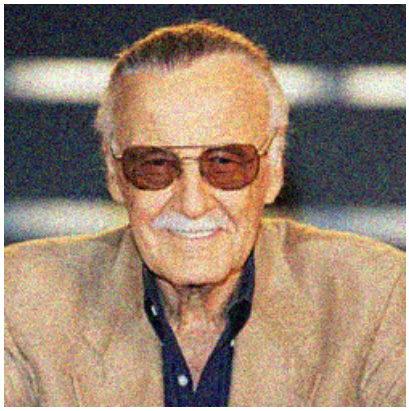
Implementation and Results:

We implemented this filter on a number of images both rgb and grayscale and gained noticeable results on the image. The noisy images were being smoothened by the filter and though, not as good as the original the filtered images were rather appealing in comparison to the noisy image.

Tree:

Stanley:


Butterfly:



City:



Apples:



PSNR:

Image name	Noisy PSNR value	Filtered PSNR value
Tree	22.93700682686653	4.692300727310059
Stanley	22.19094498607261	5.9645877799132005
Butterfly	22.279066876046393	6.320805524839112
City	22.315478697283307	5.0725833563616005
Apples	24.153139815389125	1.9753776829916525

References:

1. P. Chatterjee and P. Milanfar, "Patch-Based Near-Optimal Image Denoising," in IEEE Transactions on Image Processing, vol. 21, no. 4, pp. 1635-1649, April 2012.
2. “Wiener Filter” in Wikipedia n.d.

Code:

https://github.com/Vinesh0299/wiener-filter-colorized