CENG466-Take Home Exam 2

1st Seda Civelek
METU Computer Engineering
No: 2237147
seda.civelek@metu.edu.tr

2nd Kağan Erdoğan
METU Computer Engineering
No: 2098986
e209898@metu.edu.tr

***Abstract*—This document is a report for Ceng466-Fundamentals of Image Processing course's Take Home Exam 2. This homework is given to learn doing noise reduction using frequency domain, edge detection using Fourier domain and compression using wavelet decomposition. The answers of each question can be found in subsections respectively. In order to implement algorithms, python is used. To run the code for part1,part2 and part3 write following commands:**

- **python3 the2_part1.py**
- **python3 the2_part2.py**
- **python3 the2_part3.py inputimagepath**

***Index Terms*—noise reduction, edge detection, image compression**

I. NOISE REDUCTION

In the first part, it is asked to design a filter to reduce noises in the images while preserving edges and boundaries as much as possible. Since we should use frequency domain filtering, we should examine the images in frequency domain. All the images have different type of noises, so we have designed the filters according the specific image. We have chosen the programming language as python, since we are more familiar. We have used numpy for array operations, matplotlib for reading and saving images and scipy for fft2, fftshift, ifft2 and ifftshift functions.

For A1.png, after we have read the image, we have used the fft2 to transform image to frequency space and fftshift to centralize the image. The noise was the vertical line passing from the center of image.

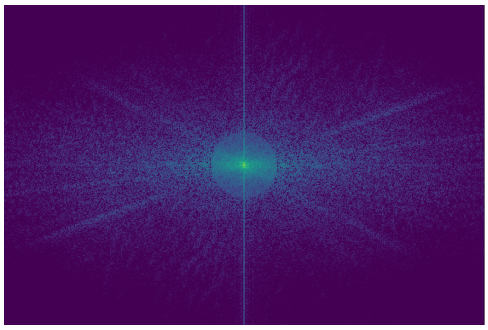

Fig. 1. Centralized image of A1.png in frequency domain

We have found coordinates of this line and assign zero to this values in the array using for loop and if statement. After applying filter we have used ifftshift and ifft2 functions to obtain filtered image. Here is the result :

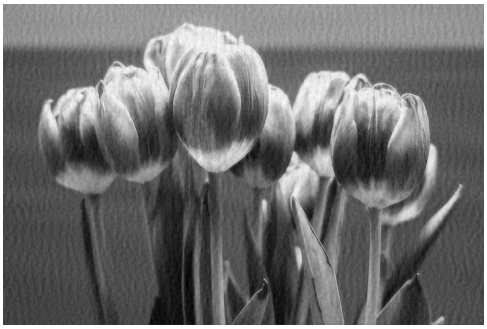

Fig. 2. Denoised image of A1.png

There were no big difference between the original image and the denoised image.

For A2.png, again we have used fft2 and fftshift function to examine noises in frequency domain. This time noises were very different.

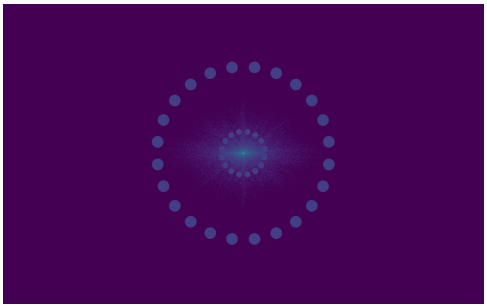

Fig. 3. Centralized image of A2.png in frequency domain

There were two concentric circles. Firstly, we have applied a band pass filter with random radius. We have defined the band pass filter as array with full of ones and we have assign zero to some values with respect to radius of filter. The image was denoised. But we thought that we can do it better. So we have defined two band pass filter for the outer and inner circle. We have found the coordinates of noises and we have defined the radius of filters according to these coordinates. After constructing filter, we have multiplied the image with that filter and applied ifft2, ifftshift functions respectively. Here is the result :

Fig. 4. Denoised image of A2.png

The result was satisfying for us, because most of the noises were reduced.

For A3.png, the situation was different from the previous images. Because third image were colored. To examine the image, firstly we have split the image in R,G,B values and for all the images we have applied fft2 and fftshift functions separately. Here are the results for r,g,b parts of centralized image of A3.png in frequency domain :

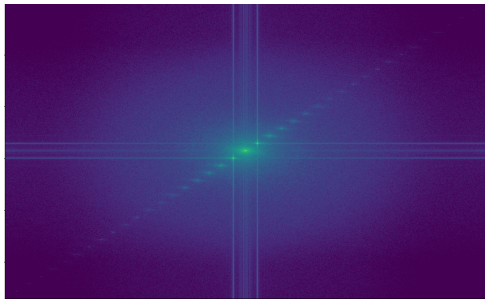

Fig. 5. Centralized R image of A3.png in frequency domain

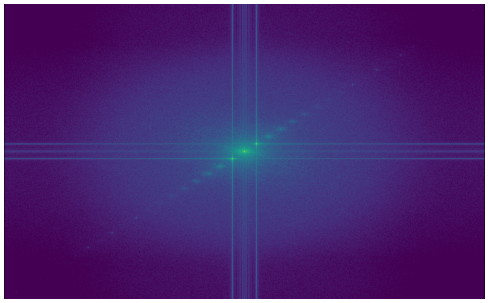

Fig. 6. Centralized G image of A3.png in frequency domain

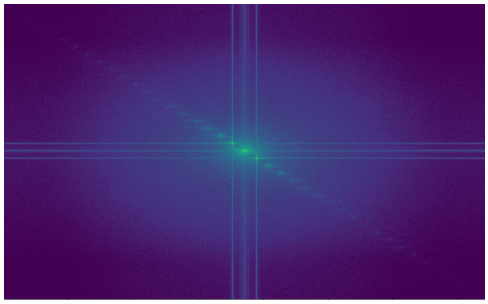

Fig. 7. Centralized B image of A3.png in frequency domain

There were horizontal and vertical lines in the images. So we have detected the coordinates of this lines and applied our filter separately. We have assigned zero values to this coordinates in the arrays using for loop and if statement. Then we have applied ifft2 and ifftshift functions to all images and we have merged the images. After normalizing array, we have obtained the following image :

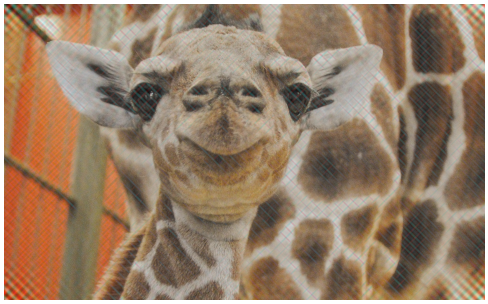

Fig. 8. Denoised image of A3.png

The result was better but it was more decoloured than the original image.

In this part, we have learned that images have different noises and we should design our filters according the type of noise. Also, we have learned to transform images to frequency domain, how we can apply filter in frequency domain and to obtain denoised image from the filtered image in frequency space.

II. EDGE DETECTION

As we seen in lectures, we decided to implement a high pass filter to detect edges. High pass filters pass the values that are high frequencies, while filtering and eliminating low frequencies. Edges have high frequencies so if we apply high pass filter to images, we can detect edges. We firstly converted rgb images to gray scale images so that we can easily compute filters correctly. While implementing the filter, we observed that the radius value that is used in high pass filter is important for better result. Also we learnt that selecting the parameters and values is design issue. So we decided to determine values using trial and error procedure.
For 'B1.jpg' image, we determined radius value as 110 since

there are too many details in the image. You can see the related output image below.

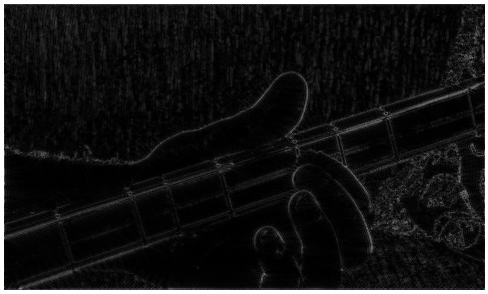

Fig. 9. High pass filtered image of B1.jpg with radius value 110

For 'B2.jpg' image, we determined radius value as 100 since there are too many details in the image. You can see the related output image below.

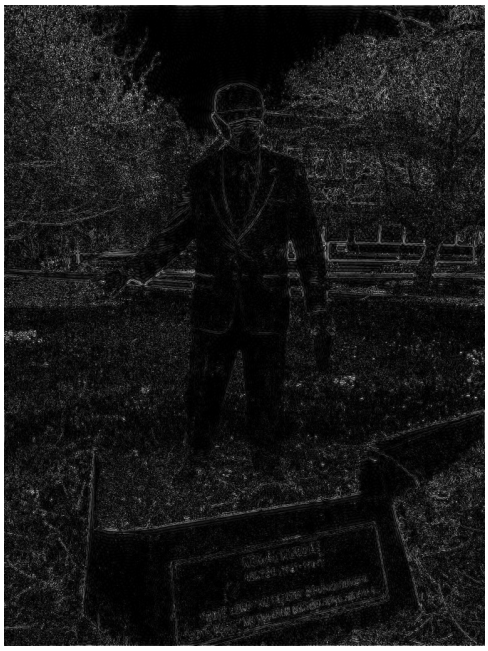

Fig. 10. High pass filtered image of B2.jpg with radius value 100

For 'B3.jpg' image, we determined radius value as 50 since there is not much detail in the image. You can see the related output image below.

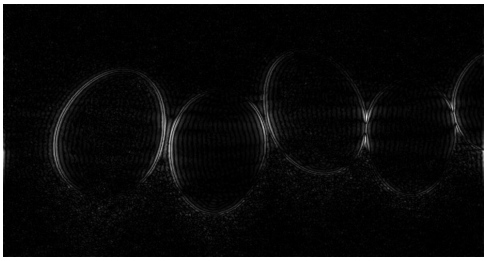

Fig. 11. High pass filtered image of B3.jpg with radius value 50

III. IMAGE COMPRESSION

In the third question, it is asked to design an algorithm based on wavelet decomposition for image compression. So, we decided to implement an algorithm that can use different wavelet families such as 'db1','haar', etc. Also, we implemented an algorithm that the epsilon value for threshold can be changed. We divided our implementation into two parts, namely compress and decompress.

A. Compress

In the compress function, firstly, we get coefficients list of image based on different wavelet families using pywt.wavedec2 function. This function is used for 2-D wavelet decomposition. Then, we get the coefficient array and coefficient slices by using pywt.coeffs_to_array function. Coefficient slices are the list of slices of corresponding coefficient. Then we take the coefficient array and sort it to define a threshold and take the values above the threshold. Compress function returns the decreased sized array using threshold and the coefficient slices to use it on decompression.

B. Decompress

Decompress function take three argument. namely coefficient array, coefficient slices list and wavelet family. Firstly, it reconvert the coefficient array that applied threshold and came from compress function and coefficient slices list to coefficients list using pywt.array_to_convert. Then, we reconstructed the compressed image using pywt.waverec2 function.

C. Experiment on threshold values and wavelet family

While implementing the algorithm, we notice that different threshold values have different impact on the size and reconstructed image. Also, different wavelet families have different impact on the size and reconstructed image too. So, we decided to give different values to threshold while keeping the wavelet family same and see the impacts of this change. With using 0.005 threshold value and 'db1' wavelet family, result is shown below.

Fig. 12. Reconstructed image with 0.005 threshold and 'db1' wavelet

MSE and Compression Ratio using 0.005 threshold value and db1 wavelet family
MSE: 629.79
Compression ratio: 3.70
As you can see from the output values and reconstructed image, compression ratio is high but, the reconstructed image is quite bad due to loss of information.
With using 0.01 threshold value and 'db1' wavelet family, result is shown below.

Fig. 13. Reconstructed image with 0.01 threshold and 'db1' wavelet

MSE and Compression Ratio using 0.01 threshold value and db1 wavelet family
MSE: 527.51
Compression ratio: 2.26
Compared to previous figure, the loss of information is lower and MSE value is smaller, but compression ratio is also lower too.
With using 0.02 threshold value and 'db1' wavelet family, result is shown below.

Fig. 14. Reconstructed image with 0.02 threshold and 'db1' wavelet

MSE and Compression Ratio using 0.02 threshold value and db1 wavelet family
MSE: 415.44
Compression ratio: 1.39
Compared to previous figure, the loss of information is lower and MSE value is smaller, but compression ratio is also lower too. We can conclude that, if threshold value is sufficiently large, we can compress image with lower loss of information.
Secondly, we kept threshold value same and changed wavelet family to see the impacts on the image.
With using 0.02 threshold value and 'haar' wavelet family, result is shown below.

Fig. 15. Reconstructed image with 0.02 threshold and 'haar' wavelet

MSE and Compression Ratio using 0.02 threshold value and haar wavelet family
MSE: 415.44
Compression ratio: 1.39
Using 'haar' instead of 'db1' doesn't have a significant effect on the outcome.
With using 0.02 threshold value and 'bior2.4' wavelet family,

result is shown below.

Fig. 16. Reconstructed image with 0.02 threshold and 'bior2.4' wavelet

MSE and Compression Ratio using 0.02 threshold value and bior2.4 wavelet family
MSE: 380.67
Compression ratio: 0.28
Using 'bior2.4' has the lower MSE, that means loss of information is lower than the methods that are mentioned before, but it doesn't compress the image as expected. So, we conclude that using 0.02 threshold and 'db1' wavelet is the best way to implement our algorithm. You can see other images reconstructed with best threshold value and wavelet family sets.

Fig. 17. Reconstructed image with 0.005 threshold and 'db1' wavelet

MSE and Compression Ratio using 0.005 threshold value and db1 wavelet family
MSE: 84.98
Compression ratio: 1.65

Fig. 18. Reconstructed image with 0.01 threshold and 'haar' wavelet

MSE and Compression Ratio using 0.01 threshold value and haar wavelet family
MSE: 573.46
Compression ratio: 2.65