Efe Tarhan
22002840

EEE424 Digital Signal Processing Mini Project Report

1- Preliminary Work

a. 2D Complex Sinusoidals

The discrete 2D functions with the format $Re\left\{e^{\frac{2\pi}{N}(k_1 \cdot n_1 + k_2 \cdot n_2)}\right\}$ are inspected by following the instructions on the project guide. For the project N = 256 and the resultant signals will be printed for different values of k_1 and k_2.

For $(k_1, k_2) = 1,2$:

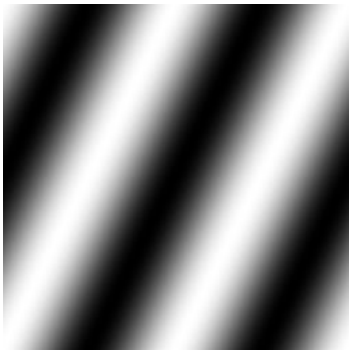

Fig. 1: The 2D image corresponding to $Re\left\{e^{\frac{2\pi}{256}(n_1 + 2 \cdot n_2)}\right\}$

For $(k_1, k_2) = 2,5$:

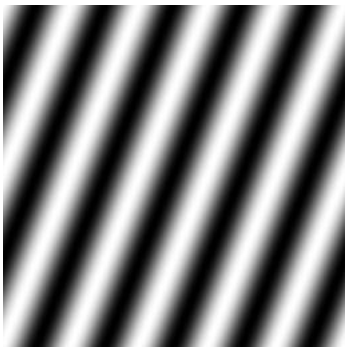

Fig. 2: The 2D image corresponding to $Re\left\{e^{\frac{2\pi}{256}(2 \cdot n_1 + 5 \cdot n_2)}\right\}$

For $(k_1, k_2) = 3,0$:

Fig. 3: The 2D image corresponding to $Re\left\{e^{\frac{2\pi}{256}\cdot 3\cdot n_1}\right\}$

For $(k_1, k_2) = 10,1$:

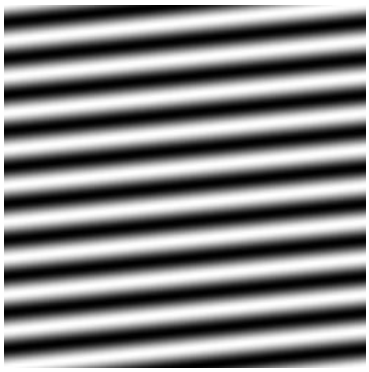

Fig. 4: The 2D image corresponding to $Re\left\{e^{\frac{2\pi}{256}\cdot (10\cdot n_1 + n_2)}\right\}$

For $(k_1, k_2) = 4,4$:

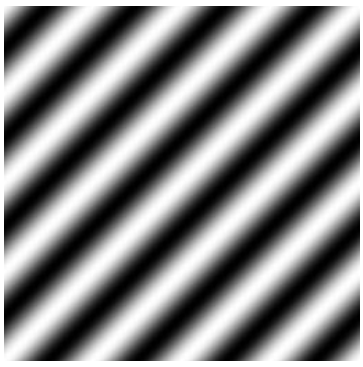

Fig. 5: The 2D image corresponding to $Re\left\{e^{\frac{2\pi}{256}\cdot (4\cdot n_1 + 4\cdot n_2)}\right\}$

These basis functions are real parts of the complex exponentials therefore they can be written as the following:

$$Re\left\{e^{\frac{2\pi}{N}(k_1 n_1 + k_2 n_2)}\right\} = \cos\left(\frac{2\pi}{N}(k_1 \cdot n_1 + k_2 \cdot n_2)\right)$$

The values of these cosines are in the range -1 to 1 therefore to view them in the greyscale format with 8-bit depth they must be rescaled and moved to the range of integers $i \in 0 \ldots . 255$.

Then the formula that has been used for scaling the values of the basis functions to the full dynamic 8-bit greyscale range if the matrix X represents the image with 'uint8' data format is the following:

$$X_{ij} = round\left(\frac{255}{2}\left(\cos\left(\frac{2\pi}{N}(k_1 \cdot i + k_2 \cdot j)\right) + \frac{255}{2}\right)\right)$$

this expression can be analyzed by checking for the relation between $k_1, i, k_2\ and\ j$.

For $\forall q \in \mathbb{Z}^+$:

$$X_{ij} = \begin{cases} 255, & if\ k_1 \cdot i + k_2 \cdot j = q \cdot \frac{N}{2} \\ 0, & x \geq 0 \end{cases}$$

So, the graphical interpretation of the 2D complex exponentials is that the from upper left corner the color starts with grey (#80) and to the lower left corner of the image the colors change sinusoidally with a linear shape that has a slope equal to $\frac{k_2}{k1}$. If we consider the mathematical expression for the number of lines, the following logic can be used:

Because of the periodic behavior of the sinusoidals there will be k_1 black regions at the vertical sideline of a given image and k_2 black regions at the horizontal sideline of a given image for the following 2D complex sinusoidal:

$$X[n_1, n_2] = e^{j(k_1 \cdot n_1 + k_2 \cdot n_2)}$$

Therefore, total number of black lines will be $k_1 + k_2$ in an image and the slope of these lines will be $\frac{k_2}{k1}$.

b. 2D DFT Implementation by using 1D FFTs.

Frequency components of a finite image with size M rows and N columns can be found by a similar method to 1D DFT which is the following:

$$X(k_1, k_2) = \sum_{n_1=0}^{M-1} \sum_{n_2=0}^{N-2} x[n_1, n_2] \cdot e^{-j(k_1 \cdot n_1 + k_2 n_2)}$$

Due to the separability of this equation, the DFT formula can be rewritten as the following:

$$X(k_1, k_2) = \sum_{n_1=0}^{M-1} e^{-jk_1 n_1} \cdot \sum_{n_2=0}^{N-1} x[n_1, n_2] \cdot e^{-jk_2 n_2}$$

Therefore, the 2D DFT can be completed by implementing 1D FFT's. In the first step the DFT's of each row has been found by implementing M many N-pt FFT operations by finding an intermediate result. This part can be written with the following equations:

$$Y[n_1, k_2] = \sum_{n_2=0}^{N-1} x[n_1, n_2] \cdot e^{-jk_2 n_2}$$

After obtaining this intermediate result, the algorithm can be proceeded by taking the FFT of the columns. The final operation will be completed by taking N many M-pt FFT's. This can be symbolically by the following equation:

$$X[k_1, k_2] = \sum_{n_1=0}^{M-1} Y[n_1, k_2] \cdot e^{-jk_1 n_1}$$

This method is called the 'row-column' FFT which is an efficient algorithm for calculating the 2D DFT's of 2D signals which are generally images. Therefore, the whole 2D DFT can be completed by implementing N many M-pt FFT's and M many N-pt FFT's. The algorithmic complexity of this operation can be found as the following by using the algorithmic complexity of an N pt FFT:

For N point FFT:

$$\mathcal{O}(NlogN)$$

Then for the 2D DFT with 1D FFT's:

$$M \cdot \mathcal{O}(NlogN) + N\mathcal{O}(MlogM) = \mathcal{O}(NMlogNM)$$

2- *Size Reduction*

Downsampling of an image is a similar implementation with downsampling a 1D signal. Downsampling a signal can be written mathematically in the following forms. Let y[n] be the downsampled version of x[n] by M. This means y[n] has elements where each its element is the qM th entry of x[n] where q and M are integers.

$$y[n] = x[Mn]$$

DTFT representation of this operation is the following:

$$Y(e^{j\omega}) = \frac{1}{M}\sum_{k=0}^{M-1} X\left(e^{\frac{j(\omega-2\pi k)}{M}}\right)$$

This operation corresponds to shifting the original frequency response of x[n] M-1 times with an amount of $\frac{2\pi k}{M}$ and expanding the frequency domain by M afterwards with an amplitude modification of 1/M.

Therefore, to avoid aliasing maximum frequency of the filter must satisfy the following equation where the investigated signal is real, i.e., has a conjugate symmetric frequency response therefore bandwith of the signal is $2 \cdot f_{max}$:

$$\omega_{max} = 2\pi\frac{f_{max}}{f_s}$$

$$2\pi\frac{f_{max}}{f_s} < \frac{\pi}{M}$$

$$f_s > 2Mf_{max}$$

Therefore, to avoid aliasing while implementing downsampling, the Nyquist Criterion is M times stricter. This is the reason for most digital images to be only able to satisfy the Nyquist Rate for the original first sampling, aliasing after downsampling is inevitable for some M. The project image has been downsampled 4 times by 4 and the effects of downsampling will be analyzed. Before starting the downsampling, the original image with full-grey range is used by stretching can be seen in Fig. 6

Fig.6: The original 4096x4096 image that has been stretched to full dynamic grey scale. [1]

a) Downsampling without Filtering

The first result is the 4 times downsampled version of the original image in Fig.6. The downsampled image can be seen in Fig.7. Mathematical representation of the operation done can be written as the following:

$$I_1[n_1, n_2] = I[4n_1, 4n_2]$$

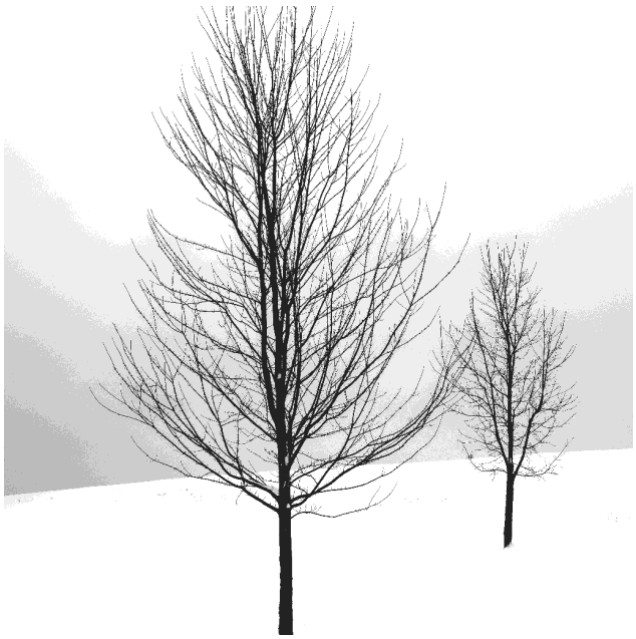

Fig.7: The 4 times downsampled image with size 1024x1024 again stretched to full dynamic grey scale.

Even from this point some non-severe distortions can be seen where the color changes are more rapid than other parts of the image. A comparison can be seen in Fig.8

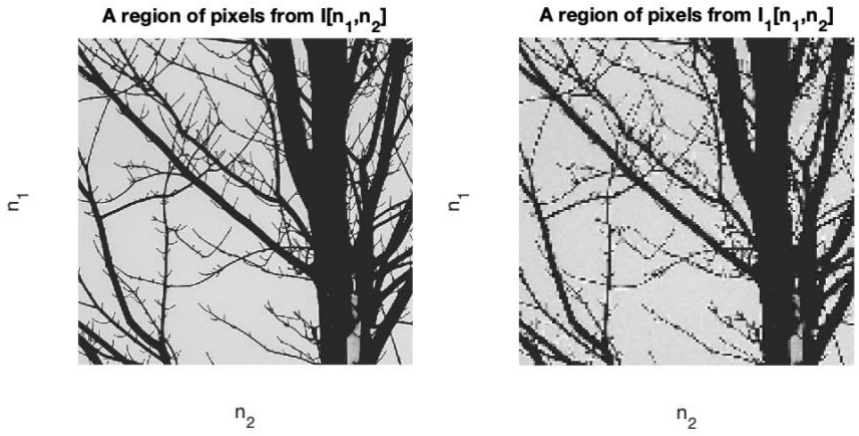

Fig.8: Comparison of the original image (left) and the 4 times downsampled image (right)

As it can be seen the high color changes appear around the branches of the tree are distorted because they are mostly determined by the high frequency components of the image and the aliasing has started with the tails or the high frequency region of the FFT of the image. The smoothness or the quality of details of the image has decreased around regions where large frequency differences appear for the first step of downsampling.

After the first downsampling, the second obtained image is again downsampled by 4 that gives a net downsampling of 16 from the original image. The new image can be seen in Fig.9

Fig.9: The 16 times downsampled image with size 256x256 again stretched to full dynamic grey scale.

This operation has been done by using the following mathematical operations.

$$I_2[n_1, n_2] = I_1[4n_1, 4n_2] = I[16n_1, 16n_2]$$

In this image the distortions caused by the aliasing can be easily detected without zooming in the image. Aliasing effect has shifted the tails of the frequency response of the image more to left and therefore the intersection is of the tails are increased. As a result, the aliasing effects can be detectable from less rapid color changes like the trunk of the tree or thicker branches. The aliasing distortions can be seen below in Fig.10:

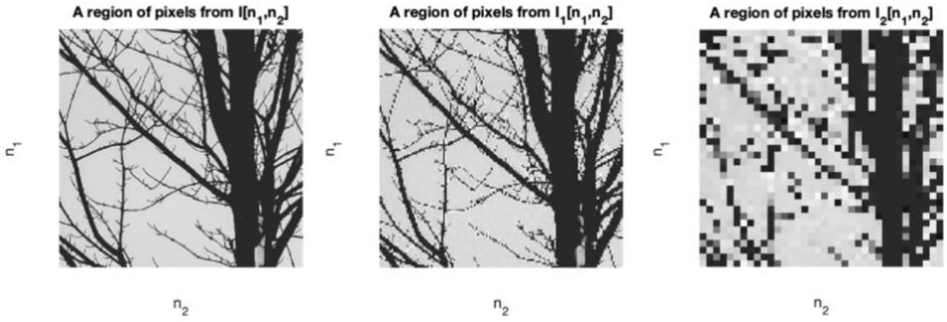

Fig.10: Comparison of the original image (left), 4 times downsampled image (middle) and 16 times downsampled image (right).

At this level, most of the details are disappeared from the image like thin or middle width branches or the grey tone of air has become a mosaic like structure because of the aliasing. As it can be seen downsampling by 4 significantly decreased quality of the image since the tails are now more than 4 times intersected.

As the last step the last obtained image has downsampled by 4 for 1 last time. This operation can be written mathematically as the following:

$$I_3[n_1, n_2] = I_2[4n_1, 4n_2] = I_1[16n_1, 16n_2] = I[64n_1, 64n_2]$$

The image obtained from the last downsampling can be examined in Fig.11.

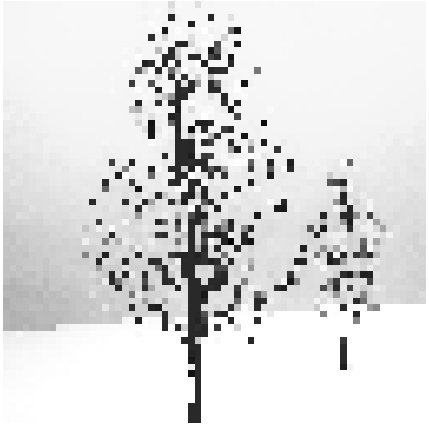

Fig.11: The 64 times downsampled image with size 64x64 again stretched to full dynamic grey scale.

At this last stage of downsampling the details of the image are completely lost, and distortions became severe. Although still the content of the image is understandable, other features of the image like medium and small details have been lost completely. The aliasing affected a lot of the frequency domain spectrum of the image. The comparison between the original and 3 downsampled images can be seen below in Fig.12.

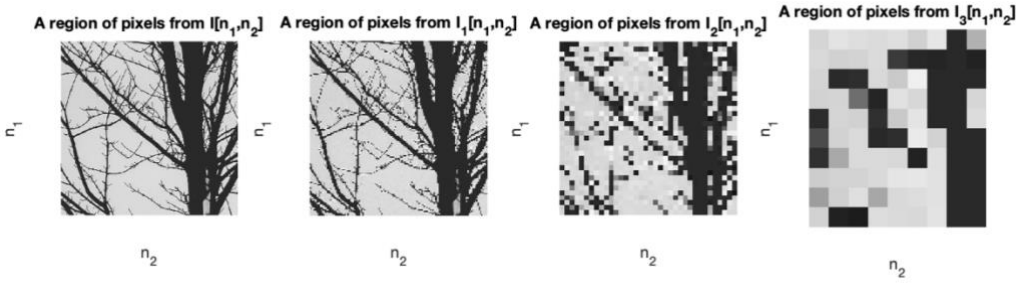

Fig.12: Comparison of the original image, 4 times downsampled image, 16 times downsampled image and 64 times downsampled image (from left to right).

As it can be seen the details of the image are completely lost due to the rate of downsampling. These problems can be avoided to a level by implementing a proper operation that prevent the signal from the effects of downsampling. The implementation that will be done is a low pass filtering operation that removes the tails of the frequency.

b) Downsampling with Using Antialiasing Filter

Frequency response of a 1D low pass filter which exhibits antialiasing behavior by cutting the tails of the frequency spectrum can be written as the following:

$$\omega_{LPF} = f(x) = \begin{cases} 1, & -\frac{\pi}{M} < x < \frac{\pi}{M} \\ 0, & else \end{cases}$$

Where M is the rate of downsampling operation. There are different types of low pass filters that can be selected for this operation. A square separable low pass filter has been used for the purpose of this part of the assignment by taking only 1/16 low frequency coefficients of the frequency spectrum of the image. The generic LPF used for this part of the project with the size 4096x4096 can be examined from Fig.13.

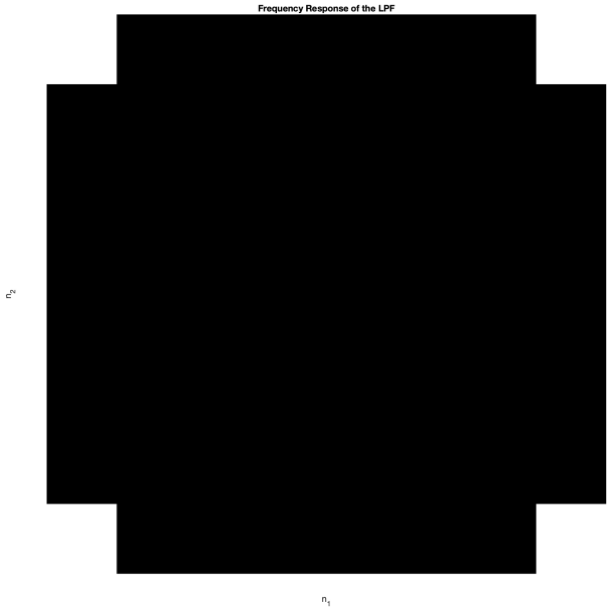

Fig.13: The designed LPF that only takes the 1/16 of the DFT coefficients of the image.

It also must be noted that the frequency response of the LPF filter must be symmetric since a real filter response must be obtained to be able to obtain a real output. The outputs of the filtered images will be displayed and discussed below.

For the first image the original 4096 x 4096 image has been passed through the low pass filter and then downsampled by 4. The result of this operation can be seen in Fig.14

Fig.14: The filtered and 4 times downsampled image with size 1024 x 1024 again stretched to full dynamic grey scale.

The image has seemed to be like the original image with some minor differences. Firstly, the background or the general contrast of the colors of the image has decreased where white colors shifted to black values and black values has shifted to white since the high frequency components are eliminated and a coarser image has been obtained. Besides from these detailed differences between the original, only downsampled and filtered-downsampled images can be examined from Figures 15, 16 and 17.

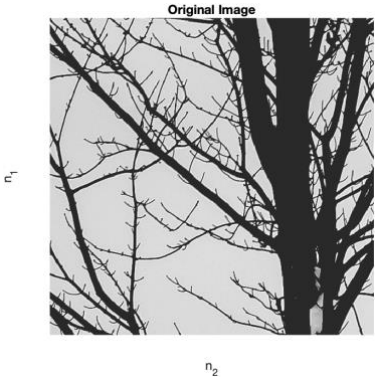

Fig.15: A segment from the original 4096x4096 image

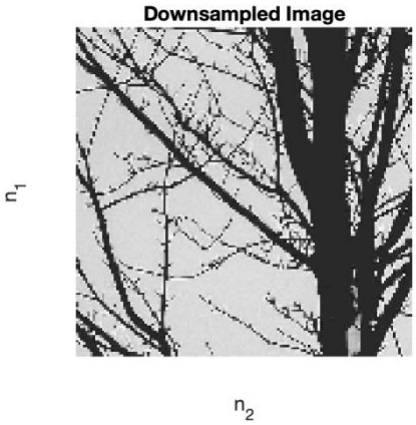

Fig.16: A segment from the downsampled image by 4 times to size 1024x1024.

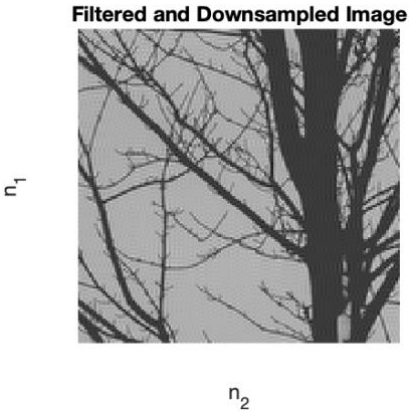

Fig.17: A segment from the filtered and downsampled image by 4 times to size 1024x1024.

As it can be seen from the images the filtered image is nearly the same with the original image besides the change of colors that has been explained above, some patterns of periodic extensions of the sharp boundaries can also be detectable from the image. This is because that the low pass filter cancels the high frequency components that allows rapid color shifts of the image. The periodic extensions can be seen in Fig.18.

Fig.18: A segment from the filtered image that shows the periodic extensions at sharp boundaries.

The obtained image in Fig.14 is again passed through a low pass filter that allows the pass of 1/16 of the DTFT coefficients and again downsampled by 4. The obtained image can be seen in Fig.19

Fig.19: The filtered and 16 times downsampled image with size 256 x 256 again stretched to full dynamic grey scale.

Like in the previous case the image has lost DFT coefficients and lost some of its quality because of the loss of this. But compared to the nonfiltered and downsampled image the boundaries and quality of the image protected better, and details has managed to exist. While the effect of aliasing has caused severe distortions on the result nonfiltered downsampling, the version of the image that has been downsampled after passing through the antialiasing filter protected from harsh distortions.

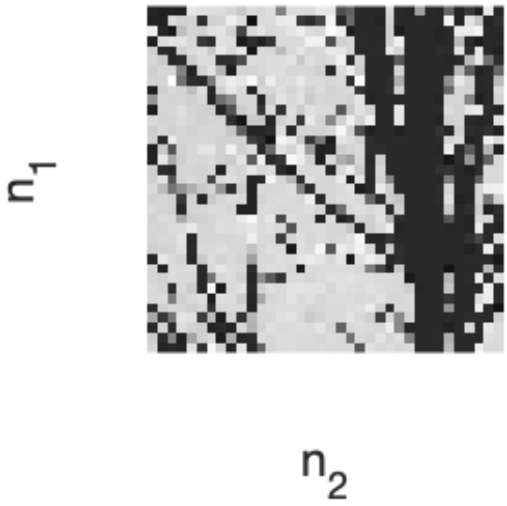

Fig.21: A segment from the downsampled image by 16 times to size 256 x 256.

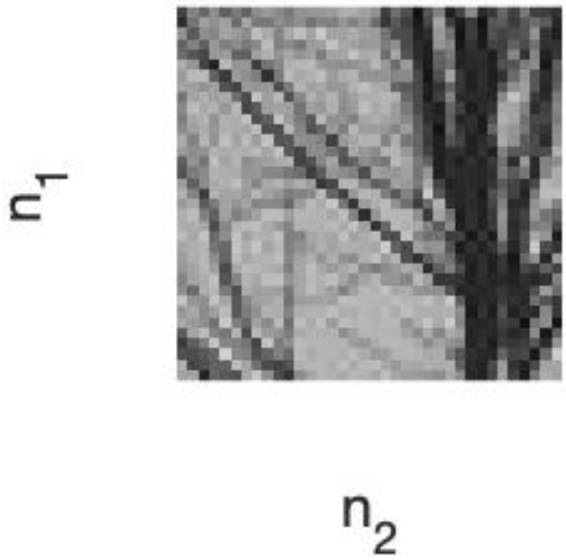

Fig.20: A segment from the filtered and downsampled image by 16 times to size 1024x1024.

The periodic extensions of the boundaries and sharp edges are again repeated and can be examined from the image. Periodic extensions and repetitions of the high frequency changes in the image is more distinguishable due to amount of filtering and downsampling. This can be seen from the Fig.22.

Fig.22: A segment from the filtered image after the filtering and downsampling by 4 that shows the periodic extensions at sharp boundaries.

Lastly the obtained image has been passed through the low pass filter one last time and downsampled by 4 to obtain the 64x64 image. The obtained image can be seen in Fig.23

Fig.23: The filtered and 64 times downsampled image with size 64 x 64 again stretched to full dynamic grey scale.

At this final version of the image, there are obvious quality differences compared to Fig.11. The usage of LPF perfectly cancelled the aliasing effects and distortions. As a result, rather than obtaining an image with sharp and inhomogeneous pixel changes that cause heavy distortions, we obtained a more flu but understandable and better image in terms of details.

The comparison between the only downsampled and filtered-downsampled images can be seen in Fig.24 and Fig.25.

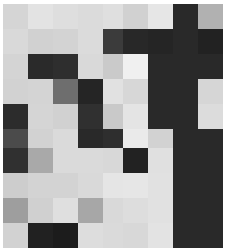

Fig.24: A segment from the downsampled image by 16 times to size 256 x 256.

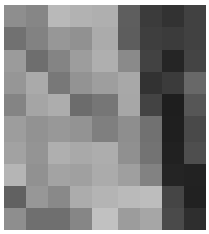

Fig.25: A segment from the filtered and downsampled image by 64 times to size 64 x 64.

The periodic extensions of the boundaries and sharp edges are again repeated and can be examined from the image. Periodic extensions and repetitions of the high frequency changes in the image is dominant due to amount of filtering and downsampling. This can be seen from the Fig.26.

Fig.26: A segment from the filtered image after the filtering and downsampling by 16 that shows the periodic extensions at sharp boundaries.

3- Approximations

In this part DFT coefficients will be selected based on their magnitudes rather than their frequencies. For this purpose, magnitudes of the DFT coefficients of the images have been saved for analysis.

a) 20 Largest DFT Coefficients:

DFT coefficients that has the largest 20 magnitudes are printed and plotted in the Table.1.

TABLE I
20 LARGEST DFT COEFFICIENTS BY THEIR MAGNITUDES

Rank	$\lvert X(k_1, k_2)\rvert$	k_1	k_2
1	$3.7017 \cdot 10^9$	1	1
2	$0.2437 \cdot 10^9$	2	1
3	$0.2437 \cdot 10^9$	4096	1
4	$0.1720 \cdot 10^9$	1	2
5	$0.1720 \cdot 10^9$	1	4096
6	$0.1312 \cdot 10^9$	1	3
7	$0.1312 \cdot 10^9$	1	4095
8	$0.1179 \cdot 10^9$	3	1
9	$0.1179 \cdot 10^9$	4095	1
10	$0.1176 \cdot 10^9$	1	4
11	$0.1176 \cdot 10^9$	1	4094
12	$0.0718 \cdot 10^9$	1	6
13	$0.0718 \cdot 10^9$	1	4092
14	$0.0592 \cdot 10^9$	4096	2
15	$0.0592 \cdot 10^9$	2	4096
16	$0.0481 \cdot 10^9$	1	7
17	$0.0481 \cdot 10^9$	1	4091
18	$0.0447 \cdot 10^9$	1	9
19	$0.0447 \cdot 10^9$	1	4089
20	$0.0381 \cdot 10^9$	2	3

b) DFT Approximations:

Images are constructed by selecting portions of DFT coefficients of the original image that satisfies a magnitude threshold. This operation has been done for ratios 1/16, 1/64, 1/256. First step for this application is to determine magnitude thresholds for selecting enough portion of DFT coefficients:

- For 1/4 portion of the DFT Coefficients $\Longrightarrow |X(k_1, k_2)| > 72{,}770$
- For 1/16 portion of the DFT Coefficients $\Longrightarrow |X(k_1, k_2)| > 245{,}000$
- For 1/64 portion of the DFT Coefficients $\Longrightarrow |X(k_1, k_2)| > 547{,}000$
- For 1/256 portion of the DFT Coefficients $\Longrightarrow |X(k_1, k_2)| > 1{,}070{,}000$

The obtained images can be seen in Fig. 27, 28, 29 and 30.

Fig.27: Image reconstructed by using 1/4 largest DFT coefficients by their magnitudes.

Fig.28: Image reconstructed by using 1/16 largest DFT coefficients by their magnitudes.

Fig.29: Image reconstructed by using 1/64 largest DFT coefficients by their magnitudes.

Fig.30: Image reconstructed by using 1/256 largest DFT coefficients by their magnitudes.

The images are rescaled to uint8 format with full greyscale range. As the number of DFT coefficients decreases through the samples the image qualities decrease. First loss of quality appears to be on the background color since the color shits to grey from white because of the loss of high frequency components. Also, if the images are examined in detail, the background becomes granular with noise like distortions. Second quality loss is the decrease of sharp edges that causes flue patterns when the number of DFT coefficients are decreased significantly. Reason of these distortions is that the since the number of eigen signals that reconstruct the image decreases some information are forgotten and therefore the operation becomes lossy.

4- References

[1] Trapani, Todd. Two Bare Trees. 2019. Pexels. https://www.pexels.com/photo/two-bare-trees-1903900/.

5- Appendix

```
%% Read in a 24-bit grayscale image
project_img = imread("ProjectPictureMod.tif");
% No stretching is required


%% (k1,k2) = (1,2)
% The minimum is -1 and the maximum is 1. Therefore the mapping is round(255/2*x + 255/2)
x1(1:256,1:256) = 0;
for k = (1:256)
   for l = (1:256)
```

```
        x1(k,l) = round(255/2*real(exp(2*pi*(k+2*l)*1i/256))+255/2);
    end
end

x1 = uint8(x1);
imwrite(x1,'1-1.tif')

%% (k1,k2) = (2,5)
% The minimum is -1 and the maximum is 1. Therefore the mapping is round(255/2*x + 255/2)
x2(1:256,1:256) = 0;
for k = (1:256)
    for l = (1:256)
        x2(k,l) = round(255/2*real(exp(2*pi*(2*k+5*l)*1i/256))+255/2);
    end
end

x2 = uint8(x2);
imwrite(x2,'1-2.tif')

%% (k1,k2) = (3,0)
% The minimum is -1 and the maximum is 1. Therefore the mapping is round(255/2*x + 255/2)
x3(1:256,1:256) = 0;
for k = (1:256)
    for l = (1:256)
        x3(k,l) = round(255/2*real(exp(2*pi*(3*k+0*l)*1i/256))+255/2);
    end
end

x3 = uint8(x3);
imwrite(x3,'1-3.tif')
%% (k1,k2) = (10,1)
% The minimum is -1 and the maximum is 1. Therefore the mapping is round(255/2*x + 255/2)
x4(1:256,1:256) = 0;
for k = (1:256)
    for l = (1:256)
        x4(k,l) = round(255/2*real(exp(2*pi*(10*k+1*l)*1i/256))+255/2);
```

```
    end
end

x4 = uint8(x4);
imwrite(x4,'1-4.tif')
%% (k1,k2) = (4,4)
% The minimum is -1 and the maximum is 1. Therefore the mapping is round(255/2*x + 255/2)
x5(1:256,1:256) = 0;
for k = (1:256)
    for l = (1:256)
        x5(k,l) = round(255/2*real(exp(2*pi*(4*k+4*l)*1i/256))+255/2);
    end
end

x5 = uint8(x5);
imwrite(x5,'1-5.tif')

%% Downsampling images

I = project_img;

I1((1:1024),(1:1024)) = 0;
for k = (1:1024)
    for l = (1:1024)

        I1(k,l) = I(4*k-3,4*l-3);
    end
end

I1 = uint8(I1);
imwrite(I1,"2-a-1.tif")

%%

I2((1:256),(1:256)) = 0;
```

```
for k = (1:256)
    for l = (1:256)

        I2(k,l) = I1(4*k-3,4*l-3);
    end
end

I2 = uint8(I2);
imwrite(I2,"2-a-2.tif")

%%

I3((1:64),(1:64)) = 0;

for k = (1:64)
    for l = (1:64)

        I3(k,l) = I2(4*k-3,4*l-3);
    end
end

I3 = uint8(I3);
imwrite(I3,"2-a-3.tif")

%%
% Compute the 2D FFT of the image
F = fft2(double(project_img));

% Create a low-pass filter mask
mask = zeros(4096, 4096);
mask(1:512, 1:512) = 1;
mask(512*7+2:4096, 1:512) = 1;
mask(1:512, 512*7+2:4096) = 1;
mask(512*7+2:4096, 512*7+2:4096) = 1;

% Apply the filter mask to the 2D FFT
```

```
F_filtered = F .* mask;

% Compute the inverse 2D FFT of the filtered coefficients
invimg = ifft2(F_filtered);
maximg = max(max(invimg));
minimg = min(min(invimg));
invimg_(1:4096,1:4096) = 0;
for k = (1:4096)
   for l = (1:4096)
      invimg_(k,l) = round(255/(maximg-minimg)*invimg(k,l)+255*minimg/(minimg-maximg));
   end
end

reconimg = uint8(invimg_);
% Display the original and filtered images
figure;

I = reconimg;
clear I1
I1((1:1024),(1:1024)) = 0;
for k = (1:1024)
   for l = (1:1024)

      I1(k,l) = I(4*k-3,4*l-3);
   end
end

I1 = uint8(I1);
imwrite(I1,"2-b-iv-1.tif")

%%
F = fft2(double(I1));

% Create a low-pass filter mask
mask = zeros(1024, 1024);
mask(1:128, 1:128) = 1;
```

```
mask(1024-126:1024, 1:128) = 1;
mask(1:128, 1024-126:1024) = 1;
mask(1024-126:1024, 1024-126:1024) = 1;

% Apply the filter mask to the 2D FFT
F_filtered = F .* mask;

% Compute the inverse 2D FFT of the filtered coefficients
invimg = ifft2(F_filtered);
maximg = max(max(invimg));
minimg = min(min(invimg));
clear invimg_
invimg_(1:1024,1:1024) = 0;
for k = (1:1024)
   for l = (1:1024)
      invimg_(k,l) = round(255/(maximg-minimg)*invimg(k,l)+255*minimg/(minimg-maximg));
   end
end

reconimg = uint8(invimg_);
% Display the original and filtered images
figure;
subplot(1, 2, 1); imshow(project_img); title('Original Image');
subplot(1, 2, 2); imshow(reconimg); title('Filtered Image');
clear I2
I2((1:256),(1:256)) = 0;

for k = (1:256)
   for l = (1:256)

      I2(k,l) = reconimg(4*k-3,4*l-3);
   end
end

I2 = uint8(I2);
imwrite(I2,"2-b-iv-2.tif")
```

```
%%
F = fft2(double(I2));

% Create a low-pass filter mask
mask = zeros(256, 256);
mask(1:32, 1:32) = 1;
mask(256-30:256, 1:32) = 1;
mask(1:32, 256-30:256) = 1;
mask(256-30:256, 256-30:256) = 1;

% Apply the filter mask to the 2D FFT
F_filtered = F .* mask;

% Compute the inverse 2D FFT of the filtered coefficients
invimg = ifft2(F_filtered);
maximg = max(max(invimg));
minimg = min(min(invimg));
clear invimg_
invimg_(1:256,1:256) = 0;
for k = (1:256)
    for l = (1:256)
        invimg_(k,l) = round(255/(maximg-minimg)*invimg(k,l)+255*minimg/(minimg-maximg));
    end
end

reconimg = uint8(invimg_);
clear I3
I3((1:64),(1:64)) = 0;

for k = (1:64)
    for l = (1:64)
        I3(k,l) = reconimg(4*k-3,4*l-3);
    end
end
```

```
I3 = uint8(I3);
imwrite(I3,"2-b-iv-3.tif")

%% Approximations
project_img = imread("ProjectPictureMod.tif");
fftimage = fft2(project_img);
imabs = abs(fftimage);
clear a
a(1:20) = 0;
b(1:2,1:20) = 0;
% a contains the maximum values in each column and b1 contains its
% location in the column(the row)

%a contains the maximum value and b2 is the location of the column.
for k = 1:20
   [a1,b1] = max(imabs);
   [a2,b2] = max(max(imabs));
   a(k) = a2;
   b(1,k) = b1(b2);
   b(2,k) = b2;
   imabs(b1(b2),b2) = min(min(imabs));
end

%%
%{
1/4 of the DFT coefficients corresponds to 4194304 (>72770)
1/16 of the DFT coefficients corresponds to 1048576 (>245000)
1/64 of the DFT coefficients corresponds to 262144 (>547000)
1/256 of the DFT coefficients corresponds to 65536 (>1070000)
%}
I1 = zeros(4096,4096);
for k = (1:4096)
   for l = (1:4096)
      if abs(fftimage(k,l)) > 72770
         I1(k,l) = fftimage(k,l);
      end
```

```
    end
end
I1imgfft = ifft2(I1);
I1_img = zeros(4096,4096);
maximg = max(max(I1imgfft));
minimg = min(min(I1imgfft));
for k = (1:4096)
    for l = (1:4096)
        I1_img(k,l) = round(255/(maximg-minimg)*I1imgfft(k,l)+255*minimg/(minimg-maximg));
    end
end
I1_img = uint8(I1_img);
imwrite(I1_img,"3-b-iii-1.tif")

%%

I2 = zeros(4096,4096);
for k = (1:4096)
    for l = (1:4096)
        if abs(fftimage(k,l)) > 245000
            I2(k,l) = fftimage(k,l);
        end
    end
end
I2imgfft = ifft2(I2);
I2_img = zeros(4096,4096);
maximg = max(max(I2imgfft));
minimg = min(min(I2imgfft));
for k = (1:4096)
    for l = (1:4096)
        I2_img(k,l) = round(255/(maximg-minimg)*I2imgfft(k,l)+255*minimg/(minimg-maximg));
    end
end
I2_img = uint8(I2_img);
imwrite(I2_img,"3-b-iii-2.tif")
```

```
%%

I3 = zeros(4096,4096);
for k = (1:4096)
   for l = (1:4096)
      if abs(fftimage(k,l)) > 547000
         I3(k,l) = fftimage(k,l);
      end
   end
end
I3imgfft = ifft2(I3);
I3_img = zeros(4096,4096);
maximg = max(max(I3imgfft));
minimg = min(min(I3imgfft));
for k = (1:4096)
   for l = (1:4096)
      I3_img(k,l) = round(255/(maximg-minimg)*I3imgfft(k,l)+255*minimg/(minimg-maximg));
   end
end
I3_img = uint8(I3_img);
imwrite(I3_img,"3-b-iii-3.tif")

%%

I4 = zeros(4096,4096);
for k = (1:4096)
   for l = (1:4096)
      if abs(fftimage(k,l)) > 1070000
         I4(k,l) = fftimage(k,l);
      end
   end
end
I4imgfft = ifft2(I4);
I4_img = zeros(4096,4096);
maximg = max(max(I4imgfft));
minimg = min(min(I4imgfft));
```

```
for k = (1:4096)
    for l = (1:4096)
        I4_img(k,l) = round(255/(maximg-minimg)*I4imgfft(k,l)+255*minimg/(minimg-maximg));
    end
end
I4_img = uint8(I4_img);
imwrite(I4_img,"3-b-iii-4.tif")
```