

Abstract

There are many methods of reducing both periodic and random noise within an image and different forms in which the image may be presented to best apply filters and enhancements. A noisy image containing both periodic and “salt and pepper” random noise has been provided along with a clean image containing no noise. Through the application of a select few transformations in sequence, an optimal approach is derived utilising a rectangular notch filter in the frequency domain followed by a median filter in the spatial domain achieving an MSE of 36.6824.

Introduction

This report documents a combination of multiple different image enhancement techniques in both the spatial and frequency domains for the purpose of enhancing the noisy image (table 1) with respects to its clean image counterpart. To compare the quality of different enhanced image results the MSE formula is used as a quantifiable measurement, and human visual interpretation is used for qualitative assessment. Both MATLAB and Python (OpenCV) were used to develop the various enhancement algorithms and both approaches are documented and compared. Both MATLAB and Python implementations make use of post-processing sharpening and contrast adjustment function, for which the specifics of these methods are documented in each section respectively.



Noisy Image	Clean Image
	

table 1

Methodology

Determining the Success of Image Enhancement - Mean Squared Error

Mean Squared Error is a formula (eq. 1) with which the difference between two images can be determined by calculating their pixel difference, with 0 being completely identical images.

$$\sigma^2 = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} |f(x, y) - f'(x, y)|^2 \quad (1)$$

This calculation iterates through both images, squaring the difference between each corresponding pixel and adding all output values to finally be divided by the number of pixels contained within either image.

The Mean Squared Error of the original clear image against the noisy image is 96.0576, so any MSE value lower than this can be considered an improvement.

Spatial and Frequency Domain

Images can be presented in the spatial and the frequency domain, the spatial domain (figure 1) represents images with their pixel values in a 2-dimensional array, whereas the frequency domain (figure 2) displays the range of frequencies of pixel intensity change within an image. Images are transformed between the spatial and frequency domains because the different domains allow for different enhancements.



figure 1

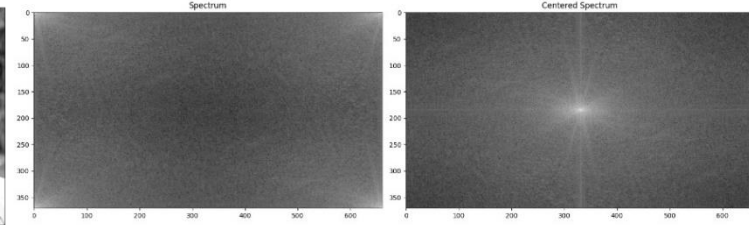


figure 2

It was decided to initially remove the random noise that is present in the image before attempting to remove the periodic noise. This would be necessary in order to produce clear 'periodic noise peaks' when it comes to enhancements in the frequency domain using low pass filters.

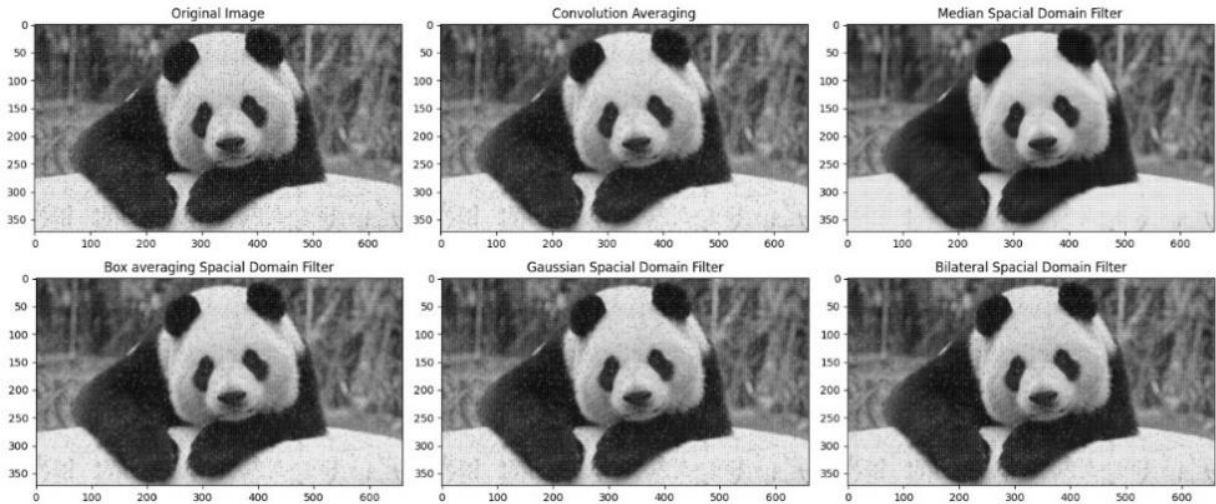


figure 3

The basic idea behind spatial filtering is to run a matrix mask of a certain size over the image, which will adjust the pixel values depending on the mask used. Since high pass filters are used for edge enhancement, whereas low pass filters are used for blurring, a range of low pass filters were tested to remove the random 'salt and pepper' noise.

Each filtered image in (figure 3) used a kernel size of 3x3, as having the kernel any larger would cause unnecessary blurring which only increased the MSE without providing any extra benefit.

(figure 3.2) is a custom averaging mask, which is implemented in python using OpenCV's `filter2D()`, which calculates the mean value of the pixels in the mask and replaces the centre pixel with that value. This filter was ineffective, as can be seen visually although the MSE is slightly reduced at 93.9403.

(figure 3.3) is a median filter, implemented using OpenCV's `medianBlur()` function, which works in a similar way to the previously described filter, however the centre pixel value is calculated using the median pixel value. As can be seen by the image, this filter is the most effective at removing the random noise. This makes sense as the 'salt and pepper' noise in the image is caused by an either extremely high or low anomalous pixel that would get disregarded by a median but would skew an average. Despite this filter subjectively looking the best, the MSE is again only slightly reduced at 93.6890.

(figure 3.5) uses the gaussian function (explained in the frequency domain section) to create a blurring mask, which is implemented using OpenCV's `GaussianBlur()` function, with a standard deviation in the x and y directions of 0. The MSE value returned however was worse than the previous filters described, at 94.7806. Also, the random noise appears to be barley changed in comparison to the original noisy image.

(figure 3.6) uses a bilateral filter, which takes a gaussian filter in both space (nearby pixels only are considered for blurring) and intensity (only pixels with similar intensities to the central pixel are considered for blurring) which in theory would preserve the edges as they generally have large variations in pixel intensities. Although the image is subjectively sharper than the other filters, the MSE value returned is the highest out of all the spatial filters, at 95.8828, whilst also having a similar if not worse amount of noise reduction to the gaussian filter.

From these results, it was concluded that the median filter is by far the best spatial filter to use for removing this type of 'salt and pepper' random noise. As such, this filtered image was used as the input for the frequency domain filtering.

To the Frequency Domain and Back Again - Discrete Fourier Transform

The discrete Fourier transform (DFT) converts images from the spatial domain (left image) to the frequency domain (middle image) in the time order of $O(N^2)$. A faster alternative to the DFT is the fast Fourier transform (FFT) which carries out the same transform but in the time order $O(N\log(N))$. The Fourier transform outputs the array elements in complex form, so to view the image `log(1 + abs(image))` must be done, then the above frequency images can be displayed.

Applying an FFTshift transform to the Fourier image will move the zero frequency elements to the centre with increasing frequency elements towards the edge. With this format, various filters can be applied to the image to remove periodic noise or enhance features with the image. After applying changes in the frequency domain, to see the resulting effect on the image it must first be transformed back into the spatial domain.

To convert the image back to the spatial domain, the inverse FFTshift (iFFTShift) and inverse FFT (iFFT) must be applied respectively to reset the image centre and transform the image back. Once complete, the image elements will still be in complex form, therefore the absolute value of each element must be taken and standardised between 0 and 255.

Filtering the Frequency Domain

If an image in the spatial domain exhibits periodic noise such as a regular dot pattern (figure 4), this noise will appear within the frequency domain of the image as bright peaks (figure 5).



figure 4

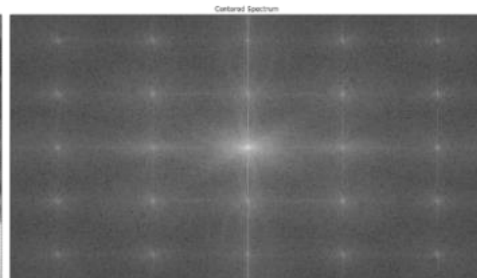


figure 5

When modifying the frequency domain of an image, there are three main approaches. Low pass filtering, where frequencies on the low end of the spectrum are maintained while removing or reducing the higher frequencies. High pass filtering is essentially the opposite of low pass; low frequencies are reduced, and high frequencies are kept. Finally, there is band pass filtering which is designed to remove the frequencies between an upper and a lower threshold leaving only the frequencies in the middle.

Low Pass filters

Gaussian filter

MATLAB approach:

A Gaussian filter is a low pass filter that follows a normal distribution. First, a mask was produced with the Gaussian pattern (figure 6) then the matrix containing the frequency domain was multiplied with the filter to produce a reduced spectrum (figure 7).

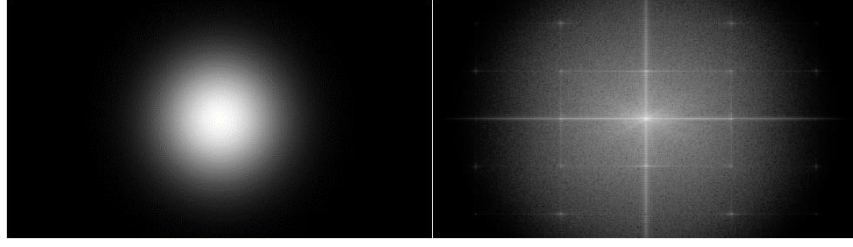


figure 6

figure 7

The approach taken in MATLAB involved steadily increasing the radius of the filter until reaching an optimum with the lowest MSE of 69.8508.

Python approach:

$$H(u, v) = e^{-D^2(u, v)/2D_0^2} \quad (2)$$

The approach taken in python can be seen in the `gaussianLP()` method, which is based on the Gaussian low pass formula (eq. 2).

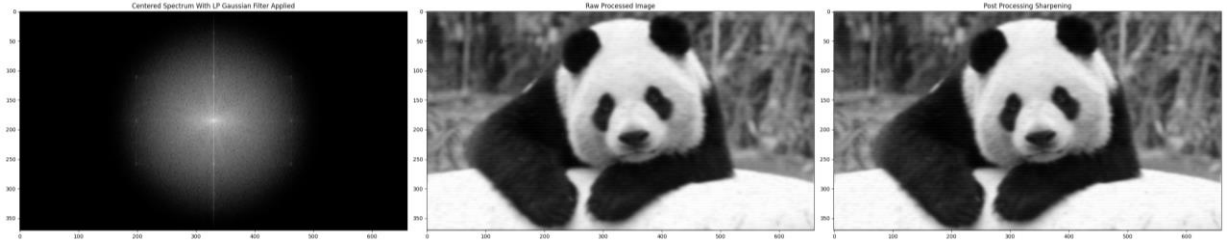


figure 8

Through trial and error, it was determined that the ideal value for D_0 is 40. The raw processed image (figure 8.2) gave an incredibly high MSE value of 454.2614, however after running the image through the `unsharp_mask()` function (figure 8.3) - which utilises a high pass gaussian - the MSE value reduced to 47.1768.

Butterworth filter

MATLAB approach:

The Butterworth filter is similar to the Gaussian filter; however, rather than following a standard distribution, the Butterworth filter has a plateau and provides a smooth transition between 1 and 0 (see appendix A).

Again, a mask was created (figure 9) and the frequency domain matrix was multiplied by the mask values producing a reduced spectrum (figure 10).

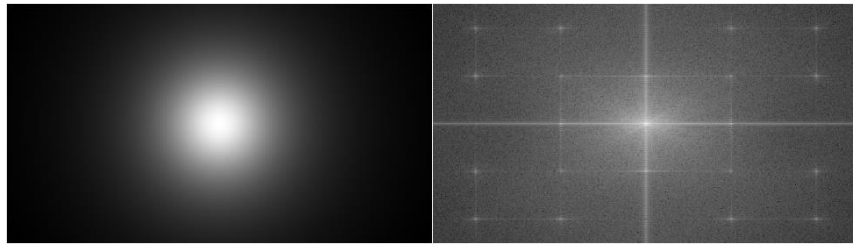


figure 9

figure 10

The same MATLAB approach used for producing an optimal Gaussian mask was taken for the Butterworth mask, resulting in an MSE of 74.0943.

Python approach:

$$H(u, v) = \frac{1}{1 + [D(u, v)/D_0]^{2n}} \quad (3)$$

The approach taken in python can be seen in the `butterworthLP()` method, which is based on the Butterworth low pass formula (eq. 3).

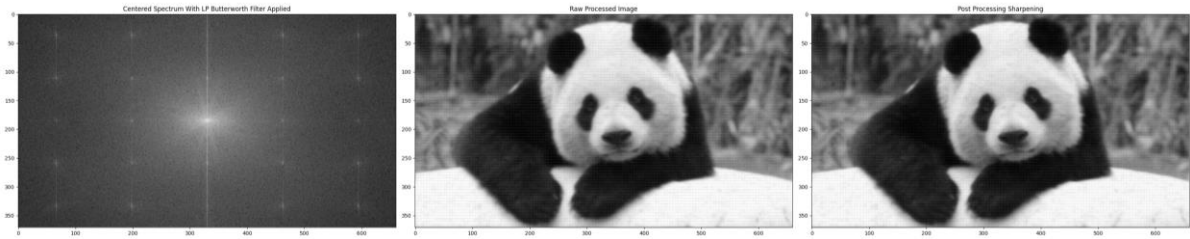


figure 11

Through trial and error, it was determined that the ideal value for D_0 was 55, and for n , 1. Again, the raw processed image (figure 11.2) had to be sharpened (figure 11.3) - using the same method previously described - returning an MSE of 67.0061

Ideal filter

MATLAB approach:

The Ideal filter is essentially just a circle of 1s surrounded by 0s. This creates a mask that will maintain the frequencies caught in the circle and sets all other frequencies to 0.

As with the other two filters, a mask was created (figure 12) and applied to the frequency spectrum (figure 13).

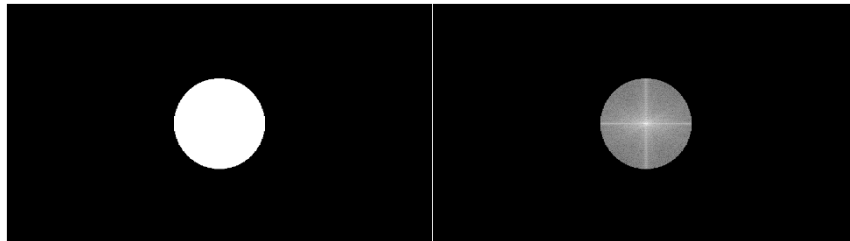


figure 12

figure 13

Again, the approach used with MATLAB was to increment the radius of the circle until an optimum MSE was found. The ideal filter produced an image with an MSE of 56.5724.

Python approach:

The python approach implemented the ideal low pass filter in the `idealFilterLP()` method with D_0 set at 133.

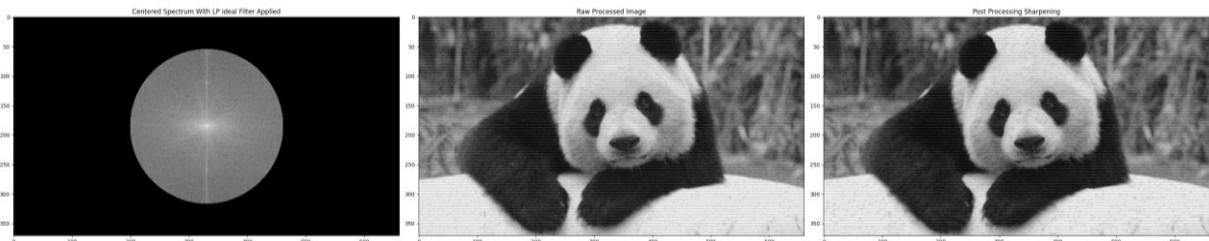


figure 14

The sharpened MSE value returned was 84.9374. Both visually and with regards to MSE, the ideal filter (figure 14) was the least effective frequency domain filter for the python implementation.

Notch filters

Simply applying a low pass filter to the image tends to be a relatively general approach to removing periodic noise. An alternative approach involves identifying and attempting to negate additional peaks in the frequency domain – this is known as notch filtering. (figure 15) shows the peaks identified for removal.

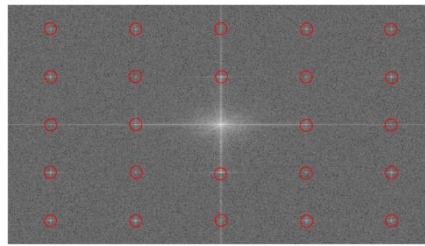


figure 15

After determining the coordinates of these additional peaks, some work can be put into reducing them.

Rectangular notch filter

The simplest way to remove these peaks is to simply set the area around them to 0. (figure 16) shows the modified frequency spectrum with the resulting output image shown in (figure 17). The MSE of the output image is 36.6824. This filter was implemented in MATLAB (see appendix A).

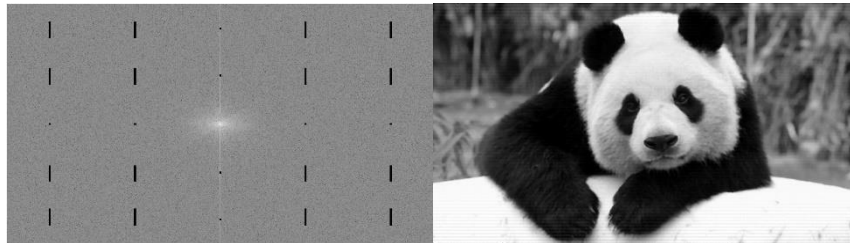


figure 16



figure 17

High pass notch filters

Notches can also be made from high pass filters of a different kind. A similar MATLAB approach to the one used in creating the low pass filters has been used to find the optimal radius for notches (see appendix B). The table below (table 2) shows the frequency spectrum, output image, and MSE values of three optimised notch filters.

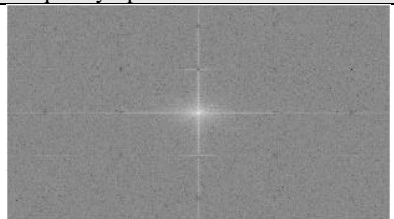

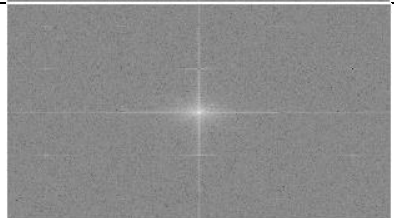



Filter Method	Frequency Spectrum	Output Image	MSE
Gaussian			37.1247
Butterworth			58.8636
Ideal			60.3517

table 2

Custom filters

Lines algorithm

In the python implementation a custom function, `removePeaksFromSpectrumLines()` was developed (see appendix F) that zeroed out single pixel horizontal lines at each noise peak level along the x axis.

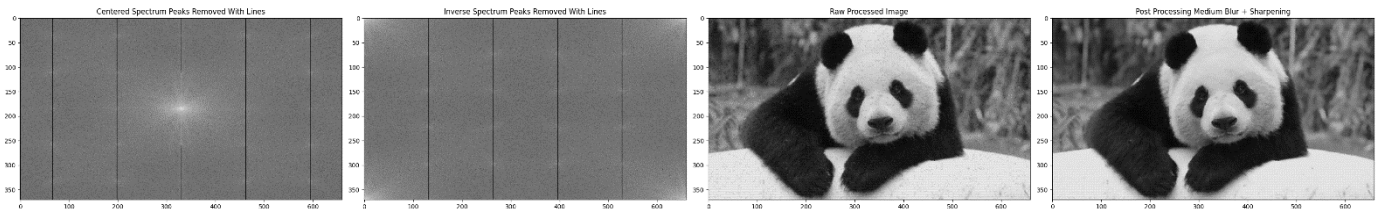


figure 18

Shown above are the centred (figure 18.1) and inverse (figure 18.2) spectrum image with this function applied, with the raw (figure 18.3) and post processed (figure 18.4) outputs. As can be seen in the raw output, this method introduced some random noise, which was removed with a post medium blur before the image went through the sharpening method. Despite the final output image looking very accurate to the original image visually, the MSE is only reduced to 74.2433.

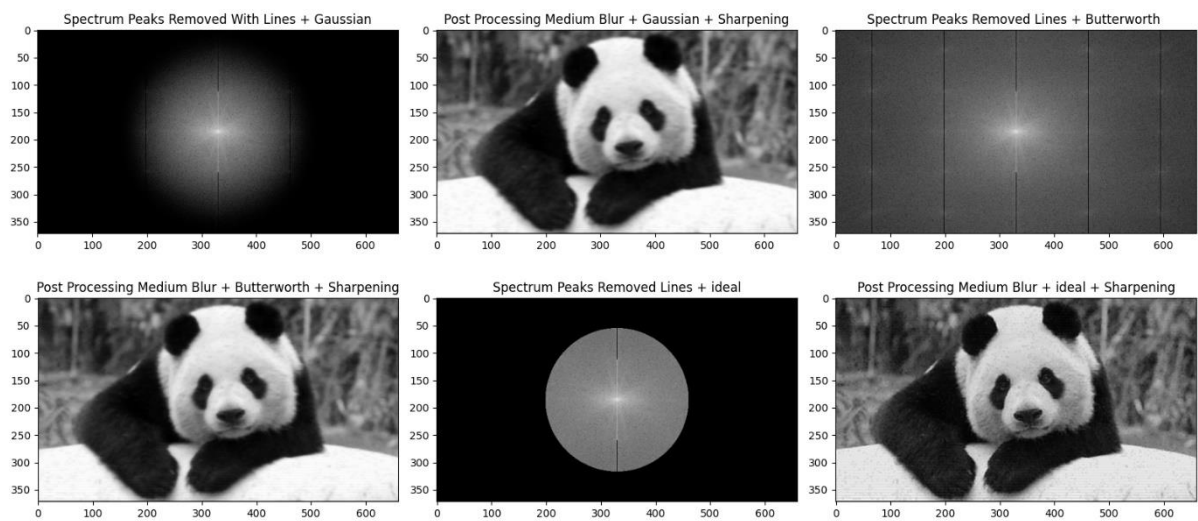


figure 19

After also applying the frequency mask filters (figure 19) to the custom spectrum image, the MSE results are much lower, however visually the images appear much blurrier. Gaussian MSE = 39.4573, Butterworth MSE = 42.1628, Ideal MSE = 73.5146.

Conclusion

In conclusion, the lowest MSE result of 36.6824 was achieved by first using the rectangular notch filter, followed by a median filter in the spatial domain (in MATLAB). Despite this image having the lowest MSE, some periodic noise can still be seen around the outside, which is not present in some images that gave a higher MSE. This suggests MSE possibly isn't a good indicator of image quality, and after reviewing academic literature [6] around the subject of quantifying image quality, it was concluded that qualitative human visual interpretation is still the best method for now.

Reflection

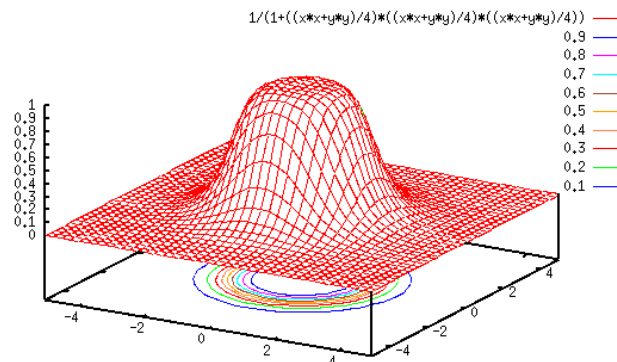
Another frequency domain filter we attempted to implement was the band pass filter, which uses a combination of high and low pass filtering at certain frequency values. However, all our implementations were unsuccessful as we weren't able to get accurate enough frequency data from the Fourier spectrum to produce any viable result. Furthermore, we could have used a combination of our lowest MSE MATLAB and Python results to possibly have gotten an even lower MSE image by copying the edge from the Python result to the MATLAB result (that had period noise around the edges).

References

1. <https://www.cs.uregina.ca/Links/class-info/425/Lab5/index.html>
2. http://fourier.eng.hmc.edu/e101/lectures/Fourier_Analysis/node10.html
3. <https://medium.com/@hicraigchen/digital-image-processing-using-fourier-transform-in-python-bcb49424fd82>
4. <https://stackoverflow.com/questions/4993082/how-can-i-sharpen-an-image-in-opencv>
5. <https://medium.com/@y1017c121y/python-computer-vision-tutorials-image-fourier-transform-part-3-e65d10be4492>
6. <https://www.sciencedirect.com/topics/computer-science/image-quality-assessment>

Appendix

A) 3D graph demonstrating a Butterworth low pass filter [2]



MATLAB Code:

B) rectangular_notch.m

```
clear all;
close all;
clc;

xSample = 1;
ySample = 1;

tallX = 25;
tallY = 2;
smallX = 2;
smallY = 2;

x1 = 28;    % top
x2 = 100;   % top middle
x3 = 250;   % bottom middle
x4 = 317;   % bottom
x5 = 185;   % middle

y1 = 66;    % left
y2 = 198;   % left middle
y3 = 462;   % right middle
y4 = 594;   % right
y5 = 330;   % middle

% Read noisy image
I = imread('PandaNoise.bmp');

% Perform a fast fourier transform on the image
unshifted = fft2(I);
% Shift the fourier output
shifted = fftshift(unshifted);
```



```

masked = shifted;
% Notch surrounding peaks
masked(x1:x1+tallX,y1:y1+tallY)=0;
masked(x2:x2+tallX,y1:y1+tallY)=0;
masked(x3:x3+tallX,y1:y1+tallY)=0;
masked(x4:x4+tallX,y1:y1+tallY)=0;
masked(x1:x1+tallX,y2:y2+tallY)=0;
masked(x2:x2+tallX,y2:y2+tallY)=0;
masked(x3:x3+tallX,y2:y2+tallY)=0;
masked(x4:x4+tallX,y2:y2+tallY)=0;
masked(x1:x1+tallX,y3:y3+tallY)=0;
masked(x2:x2+tallX,y3:y3+tallY)=0;
masked(x3:x3+tallX,y3:y3+tallY)=0;
masked(x4:x4+tallX,y3:y3+tallY)=0;
masked(x1:x1+tallX,y4:y4+tallY)=0;
masked(x2:x2+tallX,y4:y4+tallY)=0;
masked(x3:x3+tallX,y4:y4+tallY)=0;
masked(x4:x4+tallX,y4:y4+tallY)=0;

% Notch horizontal centre peaks
masked(x5:x5+smallX,y1:y1+smallY)=0;
masked(x5:x5+smallX,y2:y2+smallY)=0;
masked(x5:x5+smallX,y3:y3+smallY)=0;
masked(x5:x5+smallX,y4:y4+smallY)=0;

% Notch vertical centre peaks
masked(x1+10:x1+10+smallX,y5:y5+smallY)=0;
masked(x2+10:x2+10+smallX,y5:y5+smallY)=0;
masked(x3+10:x3+10+smallX,y5:y5+smallY)=0;
masked(x4+15:x4+15+smallX,y5:y5+smallY)=0;

figure
imshow(log(1+abs(masked)),[])

% Unshift the fourier
reshifted = ifftshift(masked);

% Revert fourier
inverted = uint8(abs(ifft2(reshifted)));
final = medfilt2(inverted);
final = imadjust(final);
figure
imshow(final)
title('Final')
imwrite(final, 'rectangular notch output.png')
ref = imread('PandaOriginal.bmp');

err = immse(I, ref);
fprintf('\n The mean-squared error between noisy and ref is %.4f\n', err);
err = immse(final, ref);
fprintf('\n The mean-squared error between final and ref is %.4f\n', err);

figure
imshowpair(final, ref, 'montage')
title('Final vs Ref')

```

C) butterworth_notch.m

```
clear all;
close all;
clc;

x1 = 0;
x2 = 264;
x3 = 528;
x4 = 792;
x5 = 1056;

y1 = 0;
y2 = 148;
y3 = 297;
y4 = 445;
y5 = 594;

pandanoise=imread('PandaNoise.bmp');
ref = imread('PandaOriginal.bmp');
%imshow(pandanoise)

%Determine good padding for Fourier transform
PQ = paddedsize(size(pandanoise));

% Calculate the discrete Fourier transform of the image
F=fft2(double(pandanoise),PQ(1),PQ(2));

% Make an attempt to find the best radius of the notches
mseMin = 100000;
mseCurrent = 99999;
currVal = 0.00;
while(mseCurrent <= mseMin)
    currVal = currVal + 0.1;
    mseMin = mseCurrent;
    %Create Notch filters corresponding to extra peaks in the Fourier transform
    % H2 = notch('btw', PQ(1), PQ(2), currVal, x2, y1);
    % H3 = notch('btw', PQ(1), PQ(2), currVal, x3, y1);
    % H4 = notch('btw', PQ(1), PQ(2), currVal, x4, y1);
    % H5 = notch('btw', PQ(1), PQ(2), currVal, x5, y1);
    %
    % H6 = notch('btw', PQ(1), PQ(2), currVal, x1, y2);
    % H7 = notch('btw', PQ(1), PQ(2), currVal, x2, y2);
    % H8 = notch('btw', PQ(1), PQ(2), currVal, x3, y2);
    % H9 = notch('btw', PQ(1), PQ(2), currVal, x4, y2);
    % H10 = notch('btw', PQ(1), PQ(2), currVal, x5, y2);
    %
    % H11 = notch('btw', PQ(1), PQ(2), currVal, x1, y3);
    % H12 = notch('btw', PQ(1), PQ(2), currVal, x2, y3);
    % H13 = notch('btw', PQ(1), PQ(2), currVal, x3, y3);
    % H14 = notch('btw', PQ(1), PQ(2), currVal, x4, y3);
    % H15 = notch('btw', PQ(1), PQ(2), currVal, x5, y3);
    %
    % H16 = notch('btw', PQ(1), PQ(2), currVal, x1, y4);
    % H17 = notch('btw', PQ(1), PQ(2), currVal, x2, y4);
    % H18 = notch('btw', PQ(1), PQ(2), currVal, x3, y4);
    % H19 = notch('btw', PQ(1), PQ(2), currVal, x4, y4);
```

```

% H20 = notch('btw', PQ(1), PQ(2), currVal, x5, y4);
%
% H21 = notch('btw', PQ(1), PQ(2), currVal, x1, y5);
% H22 = notch('btw', PQ(1), PQ(2), currVal, x2, y5);
% H23 = notch('btw', PQ(1), PQ(2), currVal, x3, y5);
% H24 = notch('btw', PQ(1), PQ(2), currVal, x4, y5);
% H25 = notch('btw', PQ(1), PQ(2), currVal, x5, y5);
% H2 = notch('gaussian', PQ(1), PQ(2), currVal, x2, y1);
% H3 = notch('gaussian', PQ(1), PQ(2), currVal, x3, y1);
% H4 = notch('gaussian', PQ(1), PQ(2), currVal, x4, y1);
% H5 = notch('gaussian', PQ(1), PQ(2), currVal, x5, y1);
%
% H6 = notch('gaussian', PQ(1), PQ(2), currVal, x1, y2);
% H7 = notch('gaussian', PQ(1), PQ(2), currVal, x2, y2);
% H8 = notch('gaussian', PQ(1), PQ(2), currVal, x3, y2);
% H9 = notch('gaussian', PQ(1), PQ(2), currVal, x4, y2);
% H10 = notch('gaussian', PQ(1), PQ(2), currVal, x5, y2);
%
% H11 = notch('gaussian', PQ(1), PQ(2), currVal, x1, y3);
% H12 = notch('gaussian', PQ(1), PQ(2), currVal, x2, y3);
% H13 = notch('gaussian', PQ(1), PQ(2), currVal, x3, y3);
% H14 = notch('gaussian', PQ(1), PQ(2), currVal, x4, y3);
% H15 = notch('gaussian', PQ(1), PQ(2), currVal, x5, y3);
%
% H16 = notch('gaussian', PQ(1), PQ(2), currVal, x1, y4);
% H17 = notch('gaussian', PQ(1), PQ(2), currVal, x2, y4);
% H18 = notch('gaussian', PQ(1), PQ(2), currVal, x3, y4);
% H19 = notch('gaussian', PQ(1), PQ(2), currVal, x4, y4);
% H20 = notch('gaussian', PQ(1), PQ(2), currVal, x5, y4);
%
% H21 = notch('gaussian', PQ(1), PQ(2), currVal, x1, y5);
% H22 = notch('gaussian', PQ(1), PQ(2), currVal, x2, y5);
% H23 = notch('gaussian', PQ(1), PQ(2), currVal, x3, y5);
% H24 = notch('gaussian', PQ(1), PQ(2), currVal, x4, y5);
% H25 = notch('gaussian', PQ(1), PQ(2), currVal, x5, y5);

H2 = notch('ideal', PQ(1), PQ(2), currVal, x2, y1);
H3 = notch('ideal', PQ(1), PQ(2), currVal, x3, y1);
H4 = notch('ideal', PQ(1), PQ(2), currVal, x4, y1);
H5 = notch('ideal', PQ(1), PQ(2), currVal, x5, y1);

H6 = notch('ideal', PQ(1), PQ(2), currVal, x1, y2);
H7 = notch('ideal', PQ(1), PQ(2), currVal, x2, y2);
H8 = notch('ideal', PQ(1), PQ(2), currVal, x3, y2);
H9 = notch('ideal', PQ(1), PQ(2), currVal, x4, y2);
H10 = notch('ideal', PQ(1), PQ(2), currVal, x5, y2);

H11 = notch('ideal', PQ(1), PQ(2), currVal, x1, y3);
H12 = notch('ideal', PQ(1), PQ(2), currVal, x2, y3);
H13 = notch('ideal', PQ(1), PQ(2), currVal, x3, y3);
H14 = notch('ideal', PQ(1), PQ(2), currVal, x4, y3);
H15 = notch('ideal', PQ(1), PQ(2), currVal, x5, y3);

H16 = notch('ideal', PQ(1), PQ(2), currVal, x1, y4);
H17 = notch('ideal', PQ(1), PQ(2), currVal, x2, y4);
H18 = notch('ideal', PQ(1), PQ(2), currVal, x3, y4);

```

```

H19 = notch('ideal', PQ(1), PQ(2), currVal, x4, y4);
H20 = notch('ideal', PQ(1), PQ(2), currVal, x5, y4);

H21 = notch('ideal', PQ(1), PQ(2), currVal, x1, y5);
H22 = notch('ideal', PQ(1), PQ(2), currVal, x2, y5);
H23 = notch('ideal', PQ(1), PQ(2), currVal, x3, y5);
H24 = notch('ideal', PQ(1), PQ(2), currVal, x4, y5);
H25 = notch('ideal', PQ(1), PQ(2), currVal, x5, y5);
% Apply the notch filters to the Fourier spectrum of the image
FS_pandanoise =
F.*H2.*H3.*H4.*H5.*H6.*H7.*H8.*H9.*H10.*H11.*H12.*H13.*H14.*H15.*H16.*H17.*H18.*H19.*H20
.*H21.*H22.*H23.*H24.*H25;
% convert the result to the spacial domain.
F_pandanoise=real(iff2(FS_pandanoise));

% Crop the image to undo padding
F_pandanoise=F_pandanoise(1:size(pandanoise,1), 1:size(pandanoise,2));
mseCurrent = immse(uint8(medfilt2(F_pandanoise)), ref);
fprintf('\nCurrVal: %0.2f, Ideal MSE: %0.4f', currVal, mseCurrent);
end

bestVal = currVal - 0.1; % Undo last increment to revert to best
fprintf('\n\nMSE Current: %0.4f\n', mseCurrent);
fprintf('MSE Min: %0.4f\n', mseMin);
fprintf('Best Val: %0.2f\n', bestVal);

%Create Notch filters corresponding to extra peaks in the Fourier transform
% H2 = notch('btw', PQ(1), PQ(2), bestVal, x2, y1);
% H3 = notch('btw', PQ(1), PQ(2), bestVal, x3, y1);
% H4 = notch('btw', PQ(1), PQ(2), bestVal, x4, y1);
% H5 = notch('btw', PQ(1), PQ(2), bestVal, x5, y1);
%
% H6 = notch('btw', PQ(1), PQ(2), bestVal, x1, y2);
% H7 = notch('btw', PQ(1), PQ(2), bestVal, x2, y2);
% H8 = notch('btw', PQ(1), PQ(2), bestVal, x3, y2);
% H9 = notch('btw', PQ(1), PQ(2), bestVal, x4, y2);
% H10 = notch('btw', PQ(1), PQ(2), bestVal, x5, y2);
%
% H11 = notch('btw', PQ(1), PQ(2), bestVal, x1, y3);
% H12 = notch('btw', PQ(1), PQ(2), bestVal, x2, y3);
% H13 = notch('btw', PQ(1), PQ(2), bestVal, x3, y3);
% H14 = notch('btw', PQ(1), PQ(2), bestVal, x4, y3);
% H15 = notch('btw', PQ(1), PQ(2), bestVal, x5, y3);
%
% H16 = notch('btw', PQ(1), PQ(2), bestVal, x1, y4);
% H17 = notch('btw', PQ(1), PQ(2), bestVal, x2, y4);
% H18 = notch('btw', PQ(1), PQ(2), bestVal, x3, y4);
% H19 = notch('btw', PQ(1), PQ(2), bestVal, x4, y4);
% H20 = notch('btw', PQ(1), PQ(2), bestVal, x5, y4);
%
% H21 = notch('btw', PQ(1), PQ(2), bestVal, x1, y5);
% H22 = notch('btw', PQ(1), PQ(2), bestVal, x2, y5);
% H23 = notch('btw', PQ(1), PQ(2), bestVal, x3, y5);
% H24 = notch('btw', PQ(1), PQ(2), bestVal, x4, y5);
% H25 = notch('btw', PQ(1), PQ(2), bestVal, x5, y5);

```

```

% H2 = notch('gaussian', PQ(1), PQ(2), bestVal, x2, y1);
% H3 = notch('gaussian', PQ(1), PQ(2), bestVal, x3, y1);
% H4 = notch('gaussian', PQ(1), PQ(2), bestVal, x4, y1);
% H5 = notch('gaussian', PQ(1), PQ(2), bestVal, x5, y1);
%
% H6 = notch('gaussian', PQ(1), PQ(2), bestVal, x1, y2);
% H7 = notch('gaussian', PQ(1), PQ(2), bestVal, x2, y2);
% H8 = notch('gaussian', PQ(1), PQ(2), bestVal, x3, y2);
% H9 = notch('gaussian', PQ(1), PQ(2), bestVal, x4, y2);
% H10 = notch('gaussian', PQ(1), PQ(2), bestVal, x5, y2);
%
% H11 = notch('gaussian', PQ(1), PQ(2), bestVal, x1, y3);
% H12 = notch('gaussian', PQ(1), PQ(2), bestVal, x2, y3);
% H13 = notch('gaussian', PQ(1), PQ(2), bestVal, x3, y3);
% H14 = notch('gaussian', PQ(1), PQ(2), bestVal, x4, y3);
% H15 = notch('gaussian', PQ(1), PQ(2), bestVal, x5, y3);
%
% H16 = notch('gaussian', PQ(1), PQ(2), bestVal, x1, y4);
% H17 = notch('gaussian', PQ(1), PQ(2), bestVal, x2, y4);
% H18 = notch('gaussian', PQ(1), PQ(2), bestVal, x3, y4);
% H19 = notch('gaussian', PQ(1), PQ(2), bestVal, x4, y4);
% H20 = notch('gaussian', PQ(1), PQ(2), bestVal, x5, y4);
%
% H21 = notch('gaussian', PQ(1), PQ(2), bestVal, x1, y5);
% H22 = notch('gaussian', PQ(1), PQ(2), bestVal, x2, y5);
% H23 = notch('gaussian', PQ(1), PQ(2), bestVal, x3, y5);
% H24 = notch('gaussian', PQ(1), PQ(2), bestVal, x4, y5);
% H25 = notch('gaussian', PQ(1), PQ(2), bestVal, x5, y5);

H2 = notch('ideal', PQ(1), PQ(2), bestVal, x2, y1);
H3 = notch('ideal', PQ(1), PQ(2), bestVal, x3, y1);
H4 = notch('ideal', PQ(1), PQ(2), bestVal, x4, y1);
H5 = notch('ideal', PQ(1), PQ(2), bestVal, x5, y1);

H6 = notch('ideal', PQ(1), PQ(2), bestVal, x1, y2);
H7 = notch('ideal', PQ(1), PQ(2), bestVal, x2, y2);
H8 = notch('ideal', PQ(1), PQ(2), bestVal, x3, y2);
H9 = notch('ideal', PQ(1), PQ(2), bestVal, x4, y2);
H10 = notch('ideal', PQ(1), PQ(2), bestVal, x5, y2);

H11 = notch('ideal', PQ(1), PQ(2), bestVal, x1, y3);
H12 = notch('ideal', PQ(1), PQ(2), bestVal, x2, y3);
H13 = notch('ideal', PQ(1), PQ(2), bestVal, x3, y3);
H14 = notch('ideal', PQ(1), PQ(2), bestVal, x4, y3);
H15 = notch('ideal', PQ(1), PQ(2), bestVal, x5, y3);

H16 = notch('ideal', PQ(1), PQ(2), bestVal, x1, y4);
H17 = notch('ideal', PQ(1), PQ(2), bestVal, x2, y4);
H18 = notch('ideal', PQ(1), PQ(2), bestVal, x3, y4);
H19 = notch('ideal', PQ(1), PQ(2), bestVal, x4, y4);
H20 = notch('ideal', PQ(1), PQ(2), bestVal, x5, y4);

H21 = notch('ideal', PQ(1), PQ(2), bestVal, x1, y5);
H22 = notch('ideal', PQ(1), PQ(2), bestVal, x2, y5);
H23 = notch('ideal', PQ(1), PQ(2), bestVal, x3, y5);
H24 = notch('ideal', PQ(1), PQ(2), bestVal, x4, y5);

```

```

H25 = notch('ideal', PQ(1), PQ(2), bestVal, x5, y5);

% Apply the notch filters to the Fourier spectrum of the image
FS_pandanoise =
F.*H2.*H3.*H4.*H5.*H6.*H7.*H8.*H9.*H10.*H11.*H12.*H13.*H14.*H15.*H16.*H17.*H18.*H19.*H20
.*H21.*H22.*H23.*H24.*H25;

% convert the result to the spacial domain.
F_pandanoise=real(ifft2(FS_pandanoise));

% Crop the image to undo padding
F_pandanoise=F_pandanoise(1:size(pandanoise,1), 1:size(pandanoise,2));

%Display the blurred image
figure, imshow(F_pandanoise,[])

% Display the Fourier Spectrum
% Move the origin of the transform to the center of the frequency rectangle.
Fc=fftshift(F);
Fc=fftshift(FS_pandanoise);

% use abs to compute the magnitude and use log to brighten display
S1=log(1+abs(Fc));
S2=log(1+abs(Fc));
figure, imshow(S1,[])
figure, imshow(S2,[])

final = uint8(medfilt2(F_pandanoise));
final = imadjust(final);
figure, imshow(final,[])

imwrite(final, 'ideal notch output.png')

err = immse(pandanoise, ref);
fprintf('\n The mean-squared error between noisy and ref is %0.4f\n', err);
err = immse(final, ref);
fprintf('\n The mean-squared error between final and ref is %0.4f\n', err);

figure
imshowpair(final, ref, 'montage')
title('Final vs Ref')

```


D) gaussian_butterworth_ideal.m

```
clear all;
close all;
clc;

input = imread('PandaNoise.bmp');
pandaNoise = medfilt2(input);

ref = imread('PandaOriginal.bmp');
imshow(pandaNoise)

%Determine good padding for Fourier transform
PQ = paddedsize(size(pandaNoise));

% Calculate the discrete Fourier transform of the image just so I don't
% have to do it over and over in the loop
D0 = 0.01*PQ(1);
gaussian = lpfilter('gaussian', PQ(1), PQ(2), D0);
F=fft2(double(pandaNoise),size(gaussian,1),size(gaussian,2));

% find best Gaussian Width
mseMin = 100000;
mseCurrent = 99999;
currVal = 0.00;
while(mseCurrent < mseMin)
    currVal = currVal + 0.01;
    mseMin = mseCurrent;
    D0 = currVal*PQ(1);
    gaussian = lpfilter('gaussian', PQ(1), PQ(2), D0);
    % Apply the filter to the Fourier spectrum of the image
    gaussianSpectrum_panda = gaussian.*F;
    % convert the result to the spacial domain.
    gaussianFiltered_panda=real(iff2(gaussianSpectrum_panda));
    % Crop the image to undo padding
    gaussOutImage=gaussianFiltered_panda(1:size(pandaNoise,1), 1:size(pandaNoise,2));
    mseCurrent = immse(uint8(gaussOutImage), ref);
    fprintf('\nCurrVal: %0.2f, Gaussian MSE: %0.4f', currVal, mseCurrent);
end
currVal = currVal - 0.01; % Undo last increment to revert to best
fprintf('\n\nMSE Current: %0.4f\n', mseCurrent);
fprintf('MSE Min: %0.4f\n', mseMin);
fprintf('Best Val: %0.2f\n', currVal);
gaussianBestVal = currVal;

% best butterworth
mseMin = 100000;
mseCurrent = 99999;
currVal = 0.00;
while(mseCurrent < mseMin)
    currVal = currVal + 0.01;
    mseMin = mseCurrent;
    D1 = currVal*PQ(1);
    butterworth = lpfilter('btw', PQ(1), PQ(2), D1);
    % Apply the filter to the Fourier spectrum of the image
    butterworthSpectrum_panda = butterworth.*F;
```

```

% convert the result to the spacial domain.
butterworthFiltered_panda=real(fft2(butterworthSpectrum_panda));

% Crop the image to undo padding
butterOutImage=butterworthFiltered_panda(1:size(pandaNoise,1),
1:size(pandaNoise,2));

mseCurrent = immse(uint8(butterOutImage), ref);
fprintf('\nCurrVal: %0.2f, Butterworth MSE: %0.4f', currVal, mseCurrent);
end
currVal = currVal - 0.01; % Undo last increment to revert to best
fprintf('\n\nMSE Current: %0.4f\n', mseCurrent);
fprintf('MSE Min: %0.4f\n', mseMin);
fprintf('Best Val: %0.2f\n', currVal);
butterworthBestVal = currVal;

% best ideal
mseMin = 100000;
mseCurrent = 99999;
currVal = 0.00;
while(mseCurrent < mseMin)
    currVal = currVal + 0.01;
    mseMin = mseCurrent;
    D2 = currVal*PQ(1);
    ideal = lpfilter('ideal', PQ(1), PQ(2), D2);

% Apply the filter to the Fourier spectrum of the image
idealSpectrum_panda = ideal.*F;

% convert the result to the spacial domain.
idealFiltered_panda=real(fft2(idealSpectrum_panda));

% Crop the image to undo padding
idealOutImage=idealFiltered_panda(1:size(pandaNoise,1), 1:size(pandaNoise,2));

mseCurrent = immse(uint8(idealOutImage), ref);
fprintf('\nCurrVal: %0.2f, Ideal MSE: %0.4f', currVal, mseCurrent);
end
currVal = currVal - 0.01; % Undo last increment to revert to best
fprintf('\n\nMSE Current: %0.4f\n', mseCurrent);
fprintf('MSE Min: %0.4f\n', mseMin);
fprintf('Best Val: %0.2f\n', currVal);
idealBestVal = currVal;

%Create a Gaussian Lowpass filter with optimal width of the Fourier transform
D0 = gaussianBestVal*PQ(1);
gaussian = lpfilter('gaussian', PQ(1), PQ(2), D0);
figure, imshow(fftshift(gaussian),[]), title('Gaussian Mask')

%Create a Butterworth Lowpass filter 5% the width of the Fourier transform
D1 = butterworthBestVal*PQ(1);
butterworth = lpfilter('btw', PQ(1), PQ(2), D1);
figure, imshow(fftshift(butterworth),[]), title('Butterworth Mask')

%Create an ideal Lowpass filter 10% the width of the Fourier transform
D2 = idealBestVal*PQ(1);

```

```

ideal = lpfilter('ideal', PQ(1), PQ(2), D2);
figure, imshow(fftshift(ideal),[]), title('Ideal Mask')

% Apply the filters to the Fourier spectrum of the image
gaussianSpectrum_panda = gaussian.*F;
butterworthSpectrum_panda = butterworth.*F;
idealSpectrum_panda = ideal.*F;

% convert the result to the spacial domain.
gaussianFiltered_panda=real(ifft2(gaussianSpectrum_panda));
butterworthFiltered_panda=real(ifft2(butterworthSpectrum_panda));
idealFiltered_panda=real(ifft2(idealSpectrum_panda));

% Crop the image to undo padding
gaussianFiltered_panda=gaussianFiltered_panda(1:size(pandaNoise,1),
1:size(pandaNoise,2));
butterworthFiltered_panda=butterworthFiltered_panda(1:size(pandaNoise,1),
1:size(pandaNoise,2));
idealFiltered_panda=idealFiltered_panda(1:size(pandaNoise,1), 1:size(pandaNoise,2));

%Display the blurred image
figure, imshow(gaussianFiltered_panda, []), title('Gaussian output')
figure, imshow(butterworthFiltered_panda, []), title('Butterworth output')
figure, imshow(idealFiltered_panda, []), title('Ideal output')

% Display the Fourier Spectrum
% Move the origin of the transform to the center of the frequency rectangle.
Fc=fftshift(F);
Fcfg=fftshift(gaussianSpectrum_panda);
Fcfb=fftshift(butterworthSpectrum_panda);
Fcfi=fftshift(idealSpectrum_panda);

% use abs to compute the magnitude and use log to brighten display
S1=log(1+abs(Fc));
S2=log(1+abs(Fcfg));
S3=log(1+abs(Fcfb));
S4=log(1+abs(Fcfi));
figure, imshow(S1,[]), title('Original spectrum')
figure, imshow(S2,[]), title('Gaussian spectrum')
figure, imshow(S3,[]), title('Butterworth spectrum')
figure, imshow(S4,[]), title('Ideal spectrum')

```

E) MATLAB scripts provided by[1] implementing notch() and lpfilter() along with other helper functions.

lpfilter.m

```
function H = lpfilter(type, M, N, D0, n)
%LPFILTER Computes frequency domain lowpass filters
% H = LPFILTER(TYPE, M, N, D0, n) creates the transfer function of
% a lowpass filter, H, of the specified TYPE and size (M-by-N). To
% view the filter as an image or mesh plot, it should be centered
% using H = fftshift(H).
%
% Valid values for TYPE, D0, and n are:
%
% 'ideal'    Ideal lowpass filter with cutoff frequency D0. n need
%             not be supplied. D0 must be positive
%
% 'btw'      Butterworth lowpass filter of order n, and cutoff D0.
%             The default value for n is 1.0. D0 must be positive.
%
% 'gaussian' Gaussian lowpass filter with cutoff (standard deviation)
%             D0. n need not be supplied. D0 must be positive.

% Use function dftuv to set up the meshgrid arrays needed for
% computing the required distances.
[U, V] = dftuv(M, N);

% Compute the distances D(U, V).
D = sqrt(U.^2 + V.^2);

% Begin filter computations.
switch type
case 'ideal'
    H = double(D <= D0);
case 'btw'
    if nargin == 4
        n = 1;
    end
    H = 1./(1 + (D./D0).^(2*n));
case 'gaussian'
    H = exp(-(D.^2)./(2*(D0^2)));
otherwise
    error('Unknown filter type.')
end
```

notch.m

```
function H = notch(type, M, N, D0, x, y, n)
%notch Computes frequency domain notch filters
% H = NOTCH(TYPE, M, N, D0, x, y, n) creates the transfer function of
% a notch filter, H, of the specified TYPE and size (M-by-N). centered at
% Column X, Row Y in an unshifted Fourier spectrum.
% Valid values for TYPE, D0, and n are:
%
% 'ideal'    Ideal highpass filter with cutoff frequency D0. n
%             need not be supplied. D0 must be positive
%
% 'btw'      Butterworth highpass filter of order n, and cutoff D0.
```

```

%           The default value for n is 1.0. D0 must be positive.
%
%   'gaussian' Gaussian highpass filter with cutoff (standard deviation)
%           D0. n need not be supplied. D0 must be positive.
%
% The transfer function Hhp of a highpass filter is 1 - Hlp,
% where Hlp is the transfer function of the corresponding lowpass
% filter. Thus, we can use function lpfilter to generate highpass
% filters.

if nargin == 6
    n = 1; % Default value of n.
end

% Generate highpass filter.
Hlp = lpfilter(type, M, N, D0, n);
H = 1 - Hlp;
H = circshift(H, [y-1 x-1]);

```

dftuv.m

```

function [U, V] = dftuv(M, N)
%DFTUV Computes meshgrid frequency matrices.
% [U, V] = DFTUV(M, N) computes meshgrid frequency matrices U and
% V. U and V are useful for computing frequency-domain filter
% functions that can be used with DFTFILT. U and V are both M-by-N.

% Set up range of variables.
u = 0:(M-1);
v = 0:(N-1);

% Compute the indices for use in meshgrid
idx = find(u > M/2);
u(idx) = u(idx) - M;
idy = find(v > N/2);
v(idy) = v(idy) - N;

% Compute the meshgrid arrays
[V, U] = meshgrid(v, u);

```

paddedsizes.m

```

function PQ = paddedsizes(AB, CD, PARAM)
%PADDEDSIZE Computes padded sizes useful for FFT-based filtering.
% PQ = PADDEDSIZE(AB), where AB is a two-element size vector,
% computes the two-element size vector PQ = 2*AB.
%
% PQ = PADDEDSIZE(AB, 'PWR2') computes the vector PQ such that
% PQ(1) = PQ(2) = 2^nextpow2(2*m), where m is MAX(AB).
%
% PQ = PADDEDSIZE(AB, CD), where AB and CD are two-element size
% vectors, computes the two-element size vector PQ. The elements
% of PQ are the smallest even integers greater than or equal to
% AB + CD - 1.

```

```

%
%   PQ = PADDEDSIZE(AB, CD, 'PWR2') computes the vector PQ such that
%   PQ(1) = PQ(2) = 2^nextpow2(2*m), where m is MAX([AB CD]).

if nargin == 1
    PQ = 2*AB;
elseif nargin == 2 & ~ischar(CD)
    PQ = AB + CD - 1;
    PQ = 2 * ceil(PQ / 2);
elseif nargin == 2
    m = max(AB); % Maximum dimension.

    % Find power-of-2 at least twice m.
    P = 2^nextpow2(2*m);
    PQ = [P, P];
elseif nargin == 3
    m = max([AB CD]); %Maximum dimension.
    P = 2^nextpow2(2*m);
    PQ = [P, P];
else
    error('Wrong number of inputs.')
end

```


Python code:

F) Python implementation

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from math import sqrt, exp
import warnings

def distance(point1, point2):
    return sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)

# source: https://medium.com/@hicraigchen/digital-image-processing-using-
# fourier-transform-in-python-bcb49424fd82
def gaussianLP(D0, imgShape):
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2, cols/2)
    for x in range(cols):
        for y in range(rows):
            base[y, x] = exp(((distance((y, x), center) ** 2) / (2 * (D0 ** 2))))
    return base

# source: https://medium.com/@hicraigchen/digital-image-processing-using-
# fourier-transform-in-python-bcb49424fd82
def gaussianHP(D0, imgShape):
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2, cols/2)
    for x in range(cols):
        for y in range(rows):
            base[y, x] = 1 - exp(((distance((y, x), center) ** 2) / (2 * (D0 **
2)))))
    return base

# source: https://medium.com/@hicraigchen/digital-image-processing-using-
# fourier-transform-in-python-bcb49424fd82
def butterworthLP(D0, imgShape, n):
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2, cols/2)
    for x in range(cols):
        for y in range(rows):
            base[y, x] = 1 / (1 + (distance((y, x), center) / D0) ** (2 * n))
    return base

# source: https://medium.com/@hicraigchen/digital-image-processing-using-
# fourier-transform-in-python-bcb49424fd82
def idealFilterLP(D0, imgShape):
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2, cols/2)
    for x in range(cols):
        for y in range(rows):
            if distance((y, x), center) < D0:
                base[y, x] = 1
    return base

# source: https://stackoverflow.com/questions/4993082/how-can-i-sharpen-an-
# image-in-opencv
def unsharp_mask(image, kernel_size=(3, 3), sigma=1.0, amount=1.0, threshold=0):
    """Return a sharpened version of the image, using an unsharp mask."""
    blurred = cv2.GaussianBlur(image, kernel_size, sigma)
    sharpened = float(amount + 1) * image - float(amount) * blurred
    sharpened = np.maximum(sharpened, np.zeros(sharpened.shape))
    sharpened = np.minimum(sharpened, 255 * np.ones(sharpened.shape))
    sharpened = sharpened.round().astype(np.uint8)
```

```

if threshold > 0:
    low_contrast_mask = np.absolute(image - blurred) < threshold
    np.copyto(sharpened, image, where=low_contrast_mask)
return sharpened

def removePeaksFromSpectrumLines(kx, ky, a):
    #b = np.full_like(a, np.mean(a)) # create lines with pixel value of average
    #spectrum value
    b=np.zeros_like(a) # create lines with pixel value of zero (ideal but only
    #slightly)
    for i in range(660): # image width hard coded
        for j in range(371): # image height hard coded
            if not ((66-kx<i<66+kx) or (198-kx<i<198+kx) or ((330-kx<i<330+kx)
and (j<185.5-ky or j>185.5+ky)) or (462-kx<i<462+kx) or (594-kx<i<594+kx))):
                b[j,i]=a[j,i] # if pixel is not along any noise lines, then copy
                #pixel from original spectrum to zero'd array

    return b # return modified spectrum image

def removePeaksFromSpectrumBoxes(spectrum):
    highest = 0
    location = []
    peaks = [[35, 110, 260, 335], [66, 198, 462, 594]]

    SQ_Size = 10
    LN_Len = 60

    img_FR = spectrum.copy()

    # Find Centre
    for x in range(img_FR.shape[1]):
        for y in range(img_FR.shape[0]):
            if np.abs(img_FR[y][x]) > highest:
                highest = np.abs(img_FR[y][x])
                location = [y, x]

    centreY = location[0]
    centreX = location[1]

    peaks[0].append(centreY)
    peaks[1].append(centreX)

    for y in peaks[0]:
        for x in peaks[1]:
            if (x == centreX) and (y == centreY):
                continue
            else:
                for y1 in range(y - SQ_Size, y + SQ_Size):
                    for x1 in range(x - SQ_Size, x + SQ_Size):
                        img_FR[y1][x1] = 0
                for x2 in range(x - LN_Len, x + LN_Len):
                    img_FR[y][x2] = 0
                for y2 in range(y - int(LN_Len // 1.9), y + int(LN_Len // 1.9)):
                    img_FR[y2][x] = 0

    return img_FR

# source: https://medium.com/@y1017c12ly/python-computer-vision-tutorials-image-
#fourier-transform-part-3-e65d10be4492
def convertFromFreqToGreyscale(filtered_img):
    filtered_img = np.abs(filtered_img)
    filtered_img -= filtered_img.min()
    filtered_img = filtered_img * 255 / filtered_img.max()
    filtered_img = filtered_img.astype(np.uint8)

    return filtered_img

```

```

def meanSquareError(imageA, imageB):
    with warnings.catch_warnings():
        warnings.simplefilter('ignore') # suppress warning caused by imageA[y,
x] - imageB[y, x]
        mseSumSq = 0
        for x in range(imageA.shape[1]):
            for y in range(imageA.shape[0]):
                mseSumSq += np.square(np.abs(imageA[y, x] - imageB[y, x]))

    mse = mseSumSq / (imageA.size)

    return mse

# IMAGE INPUT

img = cv2.imread("./PandaNoise.bmp", 0) # input noisy image
img_clear = cv2.imread("./PandaOriginal.bmp", 0) # input original clean image
print("Original image vs Noisy image mse: ", meanSquareError(img_clear, img))
print("")

# INITIAL SPACIAL DOMAIN FILTERS FOR RANDOM NOISE REMOVAL
print("INITIAL SPACIAL DOMAIN FILTERS FOR RANDOM NOISE REMOVAL")

kernel_size = 3
kernel_convolution = np.ones((kernel_size, kernel_size), np.float32)/9

img_spacial_convolution = cv2.filter2D(img, -1, kernel_convolution) #
convolution averaging
print("Spacial convolution averaging mse: ", meanSquareError(img_clear,
img_spacial_convolution))

img_spacial_median = cv2.medianBlur(img, kernel_size) # median blur filter
print("Spacial median blur filter mse: ", meanSquareError(img_clear,
img_spacial_median))

img_spacial_box = cv2.blur(img, (kernel_size, kernel_size)) # low pass filter
print("Spacial 3x3 low pass filter mse: ", meanSquareError(img_clear,
img_spacial_box))

img_spacial_gaussian = cv2.GaussianBlur(img, (kernel_size, kernel_size), 0) #
gaussian filter
print("Spacial gaussian filter mse: ", meanSquareError(img_clear,
img_spacial_gaussian))

img_spacial_bilateral = cv2.bilateralFilter(img, kernel_size, 255, 255) #
bilateral filter
print("Spacial bilateral filter mse: ", meanSquareError(img_clear,
img_spacial_bilateral))

print("")

# FFT FOR FREQUENCY DOMAIN FILTERS FOR PERIODIC NOISE REDUCTION

img_fft_spectrum = np.fft.fft2(img_spacial_median) # generate spectrum from
median filtered image
img_fft_spectrum_centered = np.fft.fftshift(img_fft_spectrum)

# GAUSSIAN FILTER IN FREQUENCY DOMAIN
print("GAUSSIAN FILTER IN FREQUENCY DOMAIN")

# LOW PASS

img_freq_lp_gaussian_centered = img_fft_spectrum_centered * gaussianLP(40,
img.shape) # apply gaussian low pass to centered spectrum with d 40 (ideal)
img_freq_lp_gaussian_inverse = np.fft.ifftshift(img_freq_lp_gaussian_centered) #
inverse shift filtered spectrum

```

```

img_freq_lp_gaussian_processed = np.fft.ifft2(img_freq_lp_gaussian_inverse) #
raw output of freq domain gaussian filter
print("Raw output of freq domain low pass gaussian filter mse: ",
meanSquareError(img_clear, img_freq_lp_gaussian_processed))

img_freq_lp_gaussian_processed_sharpened =
unsharp_mask(convertFromFreqToGreyscale(img_freq_lp_gaussian_processed)) #
sharpened output of freq domain gaussian filter
print("Sharpened output of freq domain low pass gaussian filter mse: ",
meanSquareError(img_clear, img_freq_lp_gaussian_processed_sharpened))

# HIGH PASS

img_freq_hp_gaussian_centered = img_fft_spectrum_centered * gaussianHP(40,
img.shape) # apply gaussian high pass to centered spectrum with d 40 (ideal)
img_freq_hp_gaussian_inverse = np.fft.ifftshift(img_freq_hp_gaussian_centered) #
inverse shift filtered spectrum

img_freq_hp_gaussian_processed = np.fft.ifft2(img_freq_hp_gaussian_inverse) #
raw output of freq domain gaussian filter
print("Raw output of freq domain high pass gaussian filter mse: ",
meanSquareError(img_clear, img_freq_hp_gaussian_processed))

img_freq_hp_gaussian_processed_sharpened =
unsharp_mask(convertFromFreqToGreyscale(img_freq_hp_gaussian_processed)) #
sharpened output of freq domain gaussian filter
print("Sharpened output of freq domain high pass gaussian filter mse: ",
meanSquareError(img_clear, img_freq_hp_gaussian_processed_sharpened))

print("")

# BUTTERWORTH FILTER IN FREQUENCY DOMAIN
print("BUTTERWORTH FILTER IN FREQUENCY DOMAIN")

img_freq_butterworth_centered = img_fft_spectrum_centered * butterworthLP(55,
img.shape, 1) # apply butterworth low pass to centered spectrum with d 55
(ideal)
img_freq_butterworth_inverse = np.fft.ifftshift(img_freq_butterworth_centered) #
inverse shift filtered spectrum

img_freq_butterworth_processed = np.fft.ifft2(img_freq_butterworth_inverse) #
raw output of freq domain butterworth filter
print("Raw output of freq domain low pass ideal filter mse: ",
meanSquareError(img_clear, img_freq_butterworth_processed))

sharpened_image =
cv2.medianBlur(convertFromFreqToGreyscale(img_freq_butterworth_processed), 3)

img_freq_butterworth_processed_sharpened = unsharp_mask(sharpened_image) #
sharpened output of freq domain butterworth filter
print("Sharpened output of freq domain low pass ideal filter mse: ",
meanSquareError(img_clear, img_freq_butterworth_processed_sharpened))

print("")

# IDEAL FILTER IN FREQUENCY DOMAIN
print("IDEAL FILTER IN FREQUENCY DOMAIN")

img_freq_ideal_centered = img_fft_spectrum_centered * idealFilterLP(131,
img.shape) # apply ideal low pass to centered spectrum with d 131 (ideal)
img_freq_ideal_inverse = np.fft.ifftshift(img_freq_ideal_centered) # inverse
shift filtered spectrum

img_freq_ideal_processed = np.fft.ifft2(img_freq_ideal_inverse) # raw output of
freq domain ideal filter
print("Raw output of freq domain low pass ideal filter mse: ",

```

```

meanSquareError(img_clear, img_freq_ideal_processed))

sharpened_image =
cv2.medianBlur(convertFromFreqToGreyscale(img_freq_ideal_processed), 3)

img_freq_ideal_processed_sharpened = unsharp_mask(sharpened_image) # sharpened
output of freq domain ideal filter
print("Sharpened output of freq domain low pass ideal filter mse: ",
meanSquareError(img_clear, img_freq_ideal_processed_sharpened))

print("")

# REMOVAL OF PEAKS THROUGH LINES
print("REMOVAL OF PEAKS THROUGH LINES")

img_freq_peaks_removed_lines = removePeaksFromSpectrumLines(1, 73,
img_fft_spectrum_centered)
img_freq_peaks_removed_lines_inverse =
np.fft.ifftshift(img_freq_peaks_removed_lines) # inverse shift filtered spectrum

img_freq_peaks_removed_lines_processed =
np.fft.ifft2(img_freq_peaks_removed_lines_inverse) # raw output of freq domain
line removal gaussian filter
print("Raw output of freq domain peak line removal mse: ",
meanSquareError(img_clear, img_freq_peaks_removed_lines_processed))

img_freq_peaks_removed_lines_processed_blur =
cv2.medianBlur(convertFromFreqToGreyscale(img_freq_peaks_removed_lines_processed)
, 3) # spacial medium blur before sharpening to remove introduced random noise
img_freq_peaks_removed_lines_processed_sharp =
unsharp_mask(img_freq_peaks_removed_lines_processed_blur) # sharpened output of
freq domain line removal gaussian filter
print("Sharpened output of freq domain peak line removal mse: ",
meanSquareError(img_clear, img_freq_peaks_removed_lines_processed_sharp))

print("")

# REMOVAL OF PEAKS THROUGH LINES WITH GAUSSIAN
print("REMOVAL OF PEAKS THROUGH LINES WITH GAUSSIAN")

img_freq_peaks_removed_lines_gaussian = removePeaksFromSpectrumLines(1, 73,
img_fft_spectrum_centered) * gaussianLP(40, img.shape) # apply gaussian freq
filter to lines peak removed spectrum
img_freq_peaks_removed_lines_gaussian_inverse =
np.fft.ifftshift(img_freq_peaks_removed_lines_gaussian) # inverse shift filtered
spectrum

img_freq_peaks_removed_lines_gaussian_processed =
np.fft.ifft2(img_freq_peaks_removed_lines_gaussian_inverse) # raw output of freq
domain line removal gaussian filter
print("Raw output of freq domain peak line removal with gaussian mse: ",
meanSquareError(img_clear, img_freq_peaks_removed_lines_gaussian_processed))

img_freq_peaks_removed_lines_gaussian_processed_blur =
cv2.medianBlur(convertFromFreqToGreyscale(img_freq_peaks_removed_lines_gaussian_p
rocessed), 3) # spacial medium blur before sharpening to remove introduced
random noise
img_freq_peaks_removed_lines_gaussian_processed_sharp =
unsharp_mask(img_freq_peaks_removed_lines_gaussian_processed_blur, (7, 7)) #
sharpened output of freq domain line removal gaussian filter, mask of 7x7 ideal
print("Sharpened output of freq domain peak line removal with gaussian mse: ",
meanSquareError(img_clear,
img_freq_peaks_removed_lines_gaussian_processed_sharp))

print("")

# REMOVAL OF PEAKS THROUGH LINES WITH BUTTERWORTH

```

```

print("REMOVAL OF PEAKS THROUGH LINES WITH BUTTERWORTH")

img_freq_peaks_removed_lines_butterworth = removePeaksFromSpectrumLines(1, 73,
img_fft_spectrum_centered) * butterworthLP(55, img.shape, 1) # apply butterworth
freq filter to lines peak removed spectrum
img_freq_peaks_removed_lines_butterworth_inverse =
np.fft.ifftshift(img_freq_peaks_removed_lines_butterworth) # inverse shift
filtered spectrum

img_freq_peaks_removed_lines_butterworth_processed =
np.fft.ifft2(img_freq_peaks_removed_lines_butterworth_inverse) # raw output of
freq domain butterworth filter
print("Raw output of freq domain peak line removal with butterworth mse: ",
meanSquareError(img_clear, img_freq_peaks_removed_lines_butterworth_processed))

img_freq_peaks_removed_lines_butterworth_processed_blur =
cv2.medianBlur(convertFromFreqToGreyscale(img_freq_peaks_removed_lines_butterworth
h_processed), 3) # spacial medium blur before sharpening to remove introduced
random noise
img_freq_peaks_removed_lines_butterworth_processed_sharp =
unsharp_mask(img_freq_peaks_removed_lines_butterworth_processed_blur, (7, 7)) #
sharpened output of freq domain butterworth filter, mask of 7x7 ideal
print("Sharpened output of freq domain peak line removal with butterworth mse: ",
meanSquareError(img_clear,
img_freq_peaks_removed_lines_butterworth_processed_sharp))

print("")

# REMOVAL OF PEAKS THROUGH LINES WITH IDEAL
print("REMOVAL OF PEAKS THROUGH LINES WITH IDEAL")

img_freq_peaks_removed_lines_ideal = removePeaksFromSpectrumLines(1, 73,
img_fft_spectrum_centered) * idealFilterLP(131, img.shape) # apply ideal freq
filter to lines peak removed spectrum
img_freq_peaks_removed_lines_ideal_inverse =
np.fft.ifftshift(img_freq_peaks_removed_lines_ideal) # inverse shift filtered
spectrum

img_freq_peaks_removed_lines_ideal_processed =
np.fft.ifft2(img_freq_peaks_removed_lines_ideal_inverse) # raw output of freq
domain ideal filter
print("Raw output of freq domain peak line removal with ideal mse: ",
meanSquareError(img_clear, img_freq_peaks_removed_lines_ideal_processed))

img_freq_peaks_removed_lines_ideal_processed_blur =
cv2.medianBlur(convertFromFreqToGreyscale(img_freq_peaks_removed_lines_ideal_proc
essed), 3) # spacial medium blur before sharpening to remove introduced random
noise
img_freq_peaks_removed_lines_ideal_processed_sharp =
unsharp_mask(img_freq_peaks_removed_lines_ideal_processed_blur, (7, 7)) #
sharpened output of freq domain ideal filter, mask of 7x7 ideal
print("Sharpened output of freq domain peak line removal with ideal mse: ",
meanSquareError(img_clear, img_freq_peaks_removed_lines_ideal_processed_sharp))

print("")

# REMOVAL OF PEAKS THROUGH BOXES
print("REMOVAL OF PEAKS THROUGH BOXES")

img_freq_peaks_removed_boxes =
removePeaksFromSpectrumBoxes(img_fft_spectrum_centered)
img_freq_peaks_removed_boxes_inverse =
np.fft.ifftshift(img_freq_peaks_removed_boxes) # inverse shift filtered spectrum

img_freq_peaks_removed_boxes_processed =
np.fft.ifft2(img_freq_peaks_removed_boxes_inverse) # raw output of freq domain
line removal gaussian filter

```



```

print("Raw output of freq domain peak line removal mse: ",
meanSquaredError(img_clear, img_freq_peaks_removed_boxes_processed))

img_freq_peaks_removed_boxes_processed_blur =
cv2.medianBlur(convertFromFreqToGreyscale(img_freq_peaks_removed_boxes_processed)
, 3) # spacial medium blur before sharpening to remove introduced random noise
img_freq_peaks_removed_boxes_processed_sharp =
unsharp_mask(img_freq_peaks_removed_boxes_processed_blur) # sharpened output of
freq domain line removal gaussian filter
print("Sharpened output of freq domain peak line removal mse: ",
meanSquaredError(img_clear, img_freq_peaks_removed_boxes_processed_sharp))

print("")

# REMOVAL OF PEAKS THROUGH BOXES WITH GAUSSIAN
print("REMOVAL OF PEAKS THROUGH BOXES WITH GAUSSIAN")

img_freq_peaks_removed_boxes_gaussian =
removePeaksFromSpectrumBoxes(img_fft_spectrum_centered) * gaussianLP(40,
img.shape) # apply gaussian freq filter to boxes peak removed spectrum
img_freq_peaks_removed_boxes_gaussian_inverse =
np.fft.ifftshift(img_freq_peaks_removed_boxes_gaussian) # inverse shift filtered
spectrum

img_freq_peaks_removed_boxes_gaussian_processed =
np.fft.ifft2(img_freq_peaks_removed_boxes_gaussian_inverse) # raw output of freq
domain line removal gaussian filter
print("Raw output of freq domain peak line removal with gaussian mse: ",
meanSquaredError(img_clear, img_freq_peaks_removed_boxes_gaussian_processed))

img_freq_peaks_removed_boxes_gaussian_processed_blur =
cv2.medianBlur(convertFromFreqToGreyscale(img_freq_peaks_removed_boxes_gaussian_p
rocessed), 3) # spacial medium blur before sharpening to remove introduced
random noise
img_freq_peaks_removed_boxes_gaussian_processed_sharp =
unsharp_mask(img_freq_peaks_removed_boxes_gaussian_processed_blur, (7, 7)) #
sharpened output of freq domain line removal gaussian filter, mask of 7x7 ideal
print("Sharpened output of freq domain peak line removal with gaussian mse: ",
meanSquaredError(img_clear,
img_freq_peaks_removed_boxes_gaussian_processed_sharp))

print("")

# REMOVAL OF PEAKS THROUGH BOXES WITH BUTTERWORTH
print("REMOVAL OF PEAKS THROUGH BOXES WITH BUTTERWORTH")

img_freq_peaks_removed_boxes_butterworth =
removePeaksFromSpectrumBoxes(img_fft_spectrum_centered) * butterworthLP(55,
img.shape, 1) # apply butterworth freq filter to boxes peak removed spectrum
img_freq_peaks_removed_boxes_butterworth_inverse =
np.fft.ifftshift(img_freq_peaks_removed_boxes_butterworth) # inverse shift
filtered spectrum

img_freq_peaks_removed_boxes_butterworth_processed =
np.fft.ifft2(img_freq_peaks_removed_boxes_butterworth_inverse) # raw output of
freq domain butterworth filter
print("Raw output of freq domain peak line removal with butterworth mse: ",
meanSquaredError(img_clear, img_freq_peaks_removed_boxes_butterworth_processed))

img_freq_peaks_removed_boxes_butterworth_processed_blur =
cv2.medianBlur(convertFromFreqToGreyscale(img_freq_peaks_removed_boxes_butterwort
h_processed), 3) # spacial medium blur before sharpening to remove introduced
random noise
img_freq_peaks_removed_boxes_butterworth_processed_sharp =
unsharp_mask(img_freq_peaks_removed_boxes_butterworth_processed_blur, (7, 7)) #
sharpened output of freq domain butterworth filter, mask of 7x7 ideal
print("Sharpened output of freq domain peak line removal with butterworth mse: ",

```

```

meanSquareError(img_clear,
img_freq_peaks_removed_boxes_butterworth_processed_sharp))

print("")

# REMOVAL OF PEAKS THROUGH BOXES WITH IDEAL
print("REMOVAL OF PEAKS THROUGH BOXES WITH IDEAL")

img_freq_peaks_removed_boxes_ideal =
removePeaksFromSpectrumBoxes(img_fft_spectrum_centered) * idealFilterLP(131,
img.shape) # apply ideal freq filter to boxes peak removed spectrum
img_freq_peaks_removed_boxes_ideal_inverse =
np.fft.ifftshift(img_freq_peaks_removed_boxes_ideal) # inverse shift filtered
spectrum

img_freq_peaks_removed_boxes_ideal_processed =
np.fft.ifft2(img_freq_peaks_removed_boxes_ideal_inverse) # raw output of freq
domain ideal filter
print("Raw output of freq domain peak line removal with ideal mse: ",
meanSquareError(img_clear, img_freq_peaks_removed_boxes_ideal_processed))

img_freq_peaks_removed_boxes_ideal_processed_blur =
cv2.medianBlur(convertFromFreqToGreyscale(img_freq_peaks_removed_boxes_ideal_proc
essed), 3) # spacial medium blur before sharpening to remove introduced random
noise
img_freq_peaks_removed_boxes_ideal_processed_sharp =
unsharp_mask(img_freq_peaks_removed_boxes_ideal_processed_blur, (7, 7)) #
sharpened output of freq domain ideal filter, mask of 7x7 ideal
print("Sharpened output of freq domain peak line removal with ideal mse: ",
meanSquareError(img_clear, img_freq_peaks_removed_boxes_ideal_processed_sharp))
# IMAGE PLOTS

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False) # initial spacial
domain filters comparison

plt.subplot(161), plt.imshow(img, "gray"), plt.title("Original Image")
plt.subplot(162), plt.imshow(img_spacial_convolution, "gray"),
plt.title("Convolution Averaging")
plt.subplot(163), plt.imshow(img_spacial_median, "gray"), plt.title("Median
Spacial Domain Filter")
plt.subplot(164), plt.imshow(img_spacial_box, "gray"), plt.title("Box averaging
Spacial Domain Filter")
plt.subplot(165), plt.imshow(img_spacial_gaussian, "gray"), plt.title("Gaussian
Spacial Domain Filter")
plt.subplot(166), plt.imshow(img_spacial_bilateral, "gray"), plt.title("Bilateral
Spacial Domain Filter")

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False) # fft spectrum no
filters

plt.subplot(141), plt.imshow(img_spacial_median, "gray"), plt.title("Median
Spacial Domain Filter")
plt.subplot(142), plt.imshow(np.log(1 + np.abs(img_fft_spectrum)), "gray"),
plt.title("Spectrum")
plt.subplot(143), plt.imshow(np.log(1 + np.abs(img_fft_spectrum_centered)),
"gray"), plt.title("Centered Spectrum")

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False) # gaussian low pass
frequency domain filters

plt.subplot(131), plt.imshow(np.log(1 + np.abs(img_freq_lp_gaussian_centered)),
"gray"), plt.title("Centered Spectrum With LP Gaussian Filter Applied")
plt.subplot(132), plt.imshow(np.abs(img_freq_lp_gaussian_processed), "gray"),
plt.title("Raw Processed Image")
plt.subplot(133), plt.imshow(img_freq_lp_gaussian_processed_sharpened, "gray"),
plt.title("Post Processing Sharpening")

```

```

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False) # gaussian high
pass frequency domain filters

plt.subplot(141), plt.imshow(np.log(1 + np.abs(img_freq_hp_gaussian_centered)),
"gray"), plt.title("Centered Spectrum With HP Gaussian Filter Applied")
plt.subplot(142), plt.imshow(np.log(1 + np.abs(img_freq_hp_gaussian_inverse)),
"gray"), plt.title("Inverse Spectrum With HP Gaussian Filter Applied")
plt.subplot(143), plt.imshow(np.abs(img_freq_hp_gaussian_processed), "gray"),
plt.title("Raw Processed Image")
plt.subplot(144), plt.imshow(img_freq_hp_gaussian_processed_sharpened, "gray"),
plt.title("Post Processing Sharpening")

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False) # butterworth low
pass frequency domain filters

plt.subplot(131), plt.imshow(np.log(1 + np.abs(img_freq_butterworth_centered)),
"gray"), plt.title("Centered Spectrum With LP Butterworth Filter Applied")
plt.subplot(132), plt.imshow(np.abs(img_freq_butterworth_processed), "gray"),
plt.title("Raw Processed Image")
plt.subplot(133), plt.imshow(img_freq_butterworth_processed_sharpened, "gray"),
plt.title("Post Processing Sharpening")

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False) # ideal low pass
frequency domain filters

plt.subplot(131), plt.imshow(np.log(1 + np.abs(img_freq_ideal_centered)),
"gray"), plt.title("Centered Spectrum With LP ideal Filter Applied")
plt.subplot(132), plt.imshow(np.abs(img_freq_ideal_processed), "gray"),
plt.title("Raw Processed Image")
plt.subplot(133), plt.imshow(img_freq_ideal_processed_sharpened, "gray"),
plt.title("Post Processing Sharpening")

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False) # frequency domain
peaks removed with lines

plt.subplot(141), plt.imshow(np.log(1 + np.abs(img_freq_peaks_removed_lines)),
"gray"), plt.title("Centered Spectrum Peaks Removed With Lines")
plt.subplot(142), plt.imshow(np.log(1 +
np.abs(img_freq_peaks_removed_lines_inverse)), "gray"), plt.title("Inverse
Spectrum Peaks Removed With Lines")
plt.subplot(143), plt.imshow(np.abs(img_freq_peaks_removed_lines_processed),
"gray"), plt.title("Raw Processed Image")
plt.subplot(144), plt.imshow(img_freq_peaks_removed_lines_processed_sharp,
"gray"), plt.title("Post Processing Medium Blur + Sharpening")

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False) # frequency domain
peaks removed with lines + filters

plt.subplot(161), plt.imshow(np.log(1 +
np.abs(img_freq_peaks_removed_lines_gaussian)), "gray"), plt.title("Spectrum
Peaks Removed With Lines + Gaussian")
plt.subplot(162),
plt.imshow(img_freq_peaks_removed_lines_gaussian_processed_sharp, "gray"),
plt.title("Post Processing Medium Blur + Gaussian + Sharpening")
plt.subplot(163), plt.imshow(np.log(1 +
np.abs(img_freq_peaks_removed_lines_butterworth)), "gray"), plt.title("Spectrum
Peaks Removed Lines + Butterworth ")
plt.subplot(164),
plt.imshow(img_freq_peaks_removed_lines_butterworth_processed_sharp, "gray"),
plt.title("Post Processing Medium Blur + Butterworth + Sharpening")
plt.subplot(165), plt.imshow(np.log(1 +
np.abs(img_freq_peaks_removed_lines_ideal)), "gray"), plt.title("Spectrum Peaks
Removed Lines + ideal")
plt.subplot(166), plt.imshow(img_freq_peaks_removed_lines_ideal_processed_sharp,
"gray"), plt.title("Post Processing Medium Blur + ideal + Sharpening")

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False) # frequency domain

```

```

peaks removed with boxes

plt.subplot(141), plt.imshow(np.log(1 + np.abs(img_freq_peaks_removed_boxes)),
"gray"), plt.title("Centered Spectrum Peaks Removed With boxes")
plt.subplot(142), plt.imshow(np.log(1 +
np.abs(img_freq_peaks_removed_boxes_inverse)), "gray"), plt.title("Inverse
Spectrum Peaks Removed With boxes")
plt.subplot(143), plt.imshow(np.abs(img_freq_peaks_removed_boxes_processed),
"gray"), plt.title("Raw Processed Image")
plt.subplot(144), plt.imshow(img_freq_peaks_removed_boxes_processed_sharp,
"gray"), plt.title("Post Processing Medium Blur + Sharpening")

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False) # frequency domain
peaks removed with boxes + filters

plt.subplot(161), plt.imshow(np.log(1 +
np.abs(img_freq_peaks_removed_boxes_gaussian)), "gray"), plt.title("Spectrum
Peaks Removed With boxes + Gaussian")
plt.subplot(162),
plt.imshow(img_freq_peaks_removed_boxes_gaussian_processed_sharp, "gray"),
plt.title("Post Processing Medium Blur + Gaussian + Sharpening")
plt.subplot(163), plt.imshow(np.log(1 +
np.abs(img_freq_peaks_removed_boxes_butterworth)), "gray"), plt.title("Spectrum
Peaks Removed boxes + Butterworth")
plt.subplot(164),
plt.imshow(img_freq_peaks_removed_boxes_butterworth_processed_sharp, "gray"),
plt.title("Post Processing Medium Blur + Butterworth + Sharpening")
plt.subplot(165), plt.imshow(np.log(1 +
np.abs(img_freq_peaks_removed_boxes_ideal)), "gray"), plt.title("Spectrum Peaks
Removed boxes + ideal")
plt.subplot(166), plt.imshow(img_freq_peaks_removed_boxes_ideal_processed_sharp,
"gray"), plt.title("Post Processing Medium Blur + ideal + Sharpening")

plt.show()

```