CS3IA16 Coursework Assignment 1 - Image Enhancement

Abstract

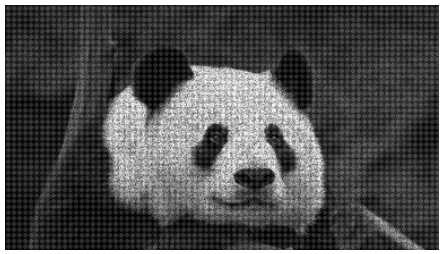
This report will detail the implementation, methodology and results of implementing image enhancement in the spatial and frequency domains. This will include implementing a Fast Fourier Transform (FFT) and highlighting the results in the frequency domain with noise. Following this, frequency domain filters will then be implemented in order to remove the periodic noise across the image through analysis of the magnitude spectrum. Various filters will then be applied to the image in order to aim at reducing the random noise across the image, such as Gaussian filtering, Median filtering and Average filtering. In order to judge the efficacy of the image enhancement techniques applied, the average mean square error of the original image and the enhanced image will be calculated and their results analysed.

Introduction

The images that will be used in this project in order to conduct image enhancement can be seen in the two figures below:

Figure 1. Original Panda

Figure 2. Noisy Panda



The image shown above in figure 1 details the original image of a Panda. This same image has then become distorted as can be highlighted through the image displayed in figure 2. In order to remove the noise from the image and conduct image enhancement techniques, MatLab's Image Processing Toolbox will be used. MatLab provides a comprehensive set of reference-standard algorithms for image enhancement and analysis[1].

Methodology

In order to begin removing the periodic noise from the image, a Fast Fourier Transform (FFT) function must be performed on the image in order to translate it into its relative frequency domain. This can be achieved through using the MatLab function fft2(). FFT is an efficient implementation of a Discrete Fourier Transform, which is a specific form of Fourier analysis to convert one function in the spatial domain into another in the frequency domain. Applying filters to images in the frequency domain is computationally faster to do so than in the image domain, and therefore has significant benefits[2].

The image must first be read in and then the fft2() function applied to this image. The function decomposes the image into its real and imaginary components, which is a representation of the image in its relative frequency domain. The number of frequencies in the frequency domain is equal to the number of pixels in the spatial domain. This is given

[1] https://uk.mathworks.com/products/image.html

[2] https://software.intel.com/en-us/articles/implementation-of-fast-fourier-transform-for-image-processing-in-directx-10

by the following equation as seen in figure 3. The inverse of this equation, which will revert the image back to the spatial domain, is given by the equation seen in figure 4.

Figure 3.

$$F(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)e^{-j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

Figure 4.

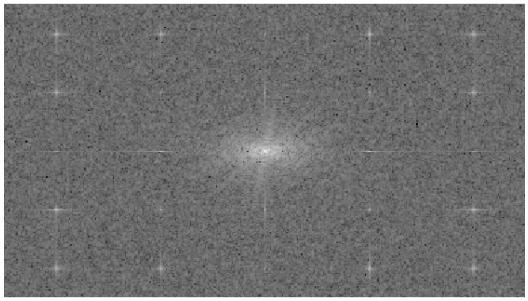
$$f(m,n) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(x,y)e^{j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

Having performed a FFT on the image, another function fftshift()[3], is required in order to shift the zero frequency component to the centre of the magnitude spectrum. This works by rearranging the Fourier transform passed as an argument and shifts the zero-frequency component to the centre of the array given. A logarithmic function is then used on this array in order to be able to display the magnitude spectrum. This is given by the following code:

```
img = imread('PandaNoise.bmp');
spec_orig = fft2(double(img));
spec_img = fftshift(spec_orig);
figure;subplot(2,1,1);
axes('Units', 'normalized', 'Position', [0 0 1 1]);
magnitude_spectrum_img = log(1 + spec_img);
imshow(magnitude_spectrum__img,[]);
```

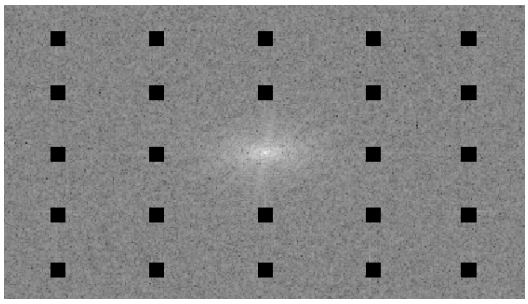
This provides a visualisation of the magnitude spectrum seen in figure 5 produced by a FFT of the noisy image from figure 2.

Figure 5.



Through analysis of Fourier spectra, it is known that the peaks in the frequency domain indicate global periodic structures in the spatial domains. Therefore it possible to create a notch-pass filter relative to the speaks shown in the spectra in order to remove some of the periodic noise from the image. The peaks appear at relatively frequent intervals across the entire image. Having implemented this by altering the array constructed, an image of the notch-pass filter onto of the previously generated spectra can be seen in figure 6.

Figure 6.



[3] https://uk.mathworks.com/help/matlab/ref/fftshift.html

Notch filters are the most useful of the selective filters. A notch filter in this case rejects frequencies in a predefined neighbourhood given by the black rectangles in figure 6. These frequencies will be rejected and therefore this aims to remove some of the periodic noise from the image. Having now altered the magnitude spectrum by applying a notch filter, the inverse FFT must be used in order to convert the image back from the frequency domain into the spatial domain. This is conducted via the MatLab function ifft2(). This function returns the 2D discrete inverse Fourier transform of a matrix using the FFT algorithm[4].

The result of this image enhancement technique can be highlighted by the comparison of the original image (left) and the enhancement image (right) in figure 7.

Figure 7.

Whilst this initial change certainly helps to improve the quality of the image, additional filters can also be implemented in order to further improve the image quality. The MatLab function fspecial() creates a 2D filter of the specified type. The first of these used on the image will be a Gaussian smoothing operator. This is a 2D convolution operator that is used to effectively blur images, removing both some details and also noise[5]. This works by outputting a weighted average of each pixel's neighbourhood, with the average being more weighted towards the value of the central pixels. This operates by convolving across the entire image in order to create a new, filtered image. This filter operator is given by the following code:

```
G = fspecial('gaussian');
gauss = imfilter(new_img, G, 'replicate');
imshow(gauss,[]);
```

Figure 8 below details the result of applying Gaussian smoothing to the previously enhanced image. This improves the results further by removing more noise from the image.

Figure 8.

[4] https://uk.mathworks.com/help/matlab/ref/ifft2.html

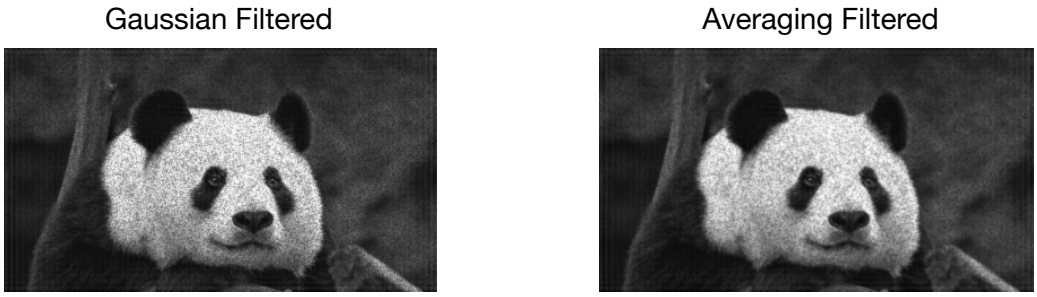
[5] https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

Implementing a median filter onto the Gaussian Filtered image from figure 9 did not help to improve the quality and instead increased the mean square error between the enhanced image and the original, however by implementing an averaging filter instead the image quality was further improved in the spatial domain. An averaging filter is a simple method of smoothing images, reducing the intensity variation between pixels. This works by convolving across the image, calculating the mean value of all of the pixels within a given window and replacing each pixel value with an average value of its own neighbours. This eliminates some noise, since it decreases the intensity of pixel values which are vastly different from their neighbours[6]. This filter when applied is given by the following code:

```
avg = fspecial('average');
avg = imfilter(gauss, H, 'replicate');
imshow(avg, []);
```

Having applied the averaging filter, the final result of the image can be seen in figure 9.

Figure 9.



This filtering technique further improves the quality of the image in the spatial domain. As a comparison between the original images and the final product of image enhancement, figure 10 details the final result highlighting a significant improvement when considering the image quality of the enhanced image compared to the noisy image.

Figure 10.

Results & Discussion

In order to gather metrics regarding the quality of the image enhancement techniques used at each stage of image enhancement, the Average Mean Square Error (AMSE) will be calculated. The AMSE is the cumulative squared error between two images. This is given be the following mathematical formula in figure 11.

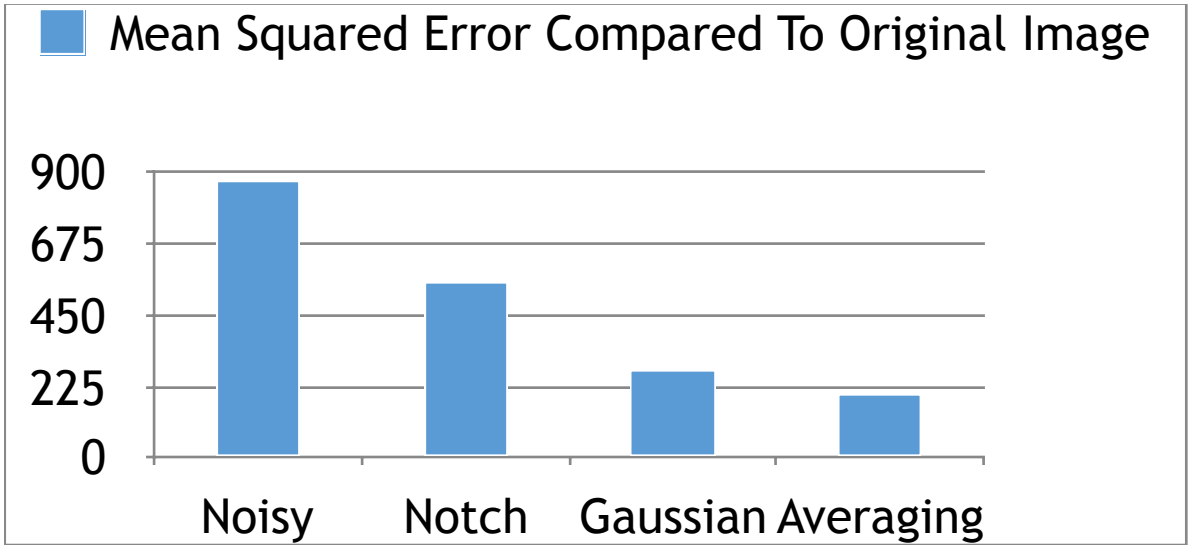
[6] https://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm

Figure 11.

$$MSE = \frac{1}{MN} \sum_{y=1}^{M} \sum_{x=1}^{N} [I(x,y) - I'(x,y)]^2$$

When implementing this in MatLab, this is provided by the function immse() which calculates the mean squared error between two arrays[7]. The AMSE of the noisy image when compared to the original image gives a value of 875.2103. Through applying the initial notch filter, this reduces the AMSE significantly to 557.9883. By applying the Gaussian filter, this further reduces the AMSE down to 279.7951. A median filter was considered instead of using a Gaussian filter, however using a median filter at the stage yielded a AMSE of 355.920, which was much less effective than using a Gaussian filter. After applying an averaging filter to this as well, the AMSE was reduced further down to 199.5355. The results of the image enhancement process can be visualised in the graph given in figure 12.

Figure 12.



Other image processing techniques were also considered such as altering the contrast of the image and using the imsharpen() function[8], however this not help to reduce the overall AMSE of the image. Overall it appears that the most efficient method to enhance the quality of this image and remove the noise found would be to implement the notch filter, followed by the gaussian filter and then finally an averaging filter.

Conclusion

Overall the image enhancement techniques utilised were highly successful and by using a combination of image enhancement techniques in both the frequency and the spatial domain, the AMSE of the image was reduced by almost 80%. This could potentially have been further improved upon by altering the notch filter designed in figure 6. If circles had been used rather than rectangles to reject some frequencies, this would have given a smoother result to the final image and may have helped to further reduce the noise from the image. The most effective image enhancement technique used was the notch (band-stop) filter, which reduced the AMSE by a value of 318.

[7] https://uk.mathworks.com/help/images/ref/immse.html

[8] https://uk.mathworks.com/help/images/ref/imsharpen.html

Appendix

References

1 - https://uk.mathworks.com/products/image.html

2 - https://software.intel.com/en-us/articles/implementation-of-fast-fourier-transform-for-image-processing-in-directx-10

3 - https://uk.mathworks.com/help/matlab/ref/fftshift.html

4 - https://uk.mathworks.com/help/matlab/ref/ifft2.html

5 - https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

6 - https://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm

7 - https://uk.mathworks.com/help/images/ref/immse.html

8 - https://uk.mathworks.com/help/images/ref/imsharpen.html

Code

ImageEnhancement.m

```
img = imread('PandaNoise.bmp');
orig_img = imread('PandaOriginal.bmp');
avg_img = imread('AvgImage.bmp');

genh_avg = rgb2gray(avg_img);
spec_orig = fft2(double(img));
spec_img = fftshift(spec_orig);

%loops to alter magnitude spectrum to stop frequencies relative to peaks
for j = 35:45
   for n = 190:200
        spec_img(n,j) = 0;
   end
    for  n = 150:160
        spec_img(n,j) = 0;
   end
    for n = 105:115
        spec_img(n,j) = 0;
    end
    for n = 60:70
        spec_img(n,j) = 0;
    end
    for n = 20:30
        spec_img(n,j) = 0;
    end
end

for j = 108:118
    for n = 190:200
        spec_img(n,j) = 0;
    end
    for  n = 150:160
        spec_img(n,j) = 0;
   end
    for n = 105:115
        spec_img(n,j) = 0;
    end
    for n = 60:70
        spec_img(n,j) = 0;
```

```
    end
    for n = 20:30
        spec_img(n,j) = 0;
    end
end


for j = 188:198
   for n = 190:200
        spec_img(n,j) = 0;
   end
   for  n = 150:160
        spec_img(n,j) = 0;
   end

    for n = 60:70
        spec_img(n,j) = 0;
    end
    for n = 20:30
        spec_img(n,j) = 0;
    end
end

for j = 268:278
   for n = 190:200
        spec_img(n,j) = 0;
   end
   for  n = 150:160
        spec_img(n,j) = 0;
   end
    for n = 105:115
        spec_img(n,j) = 0;
    end
    for n = 60:70
        spec_img(n,j) = 0;
    end
    for n = 20:30
        spec_img(n,j) = 0;
    end
end

for j = 338:348
   for n = 190:200
        spec_img(n,j) = 0;
   end
   for  n = 150:160
        spec_img(n,j) = 0;
   end
    for n = 105:115
        spec_img(n,j) = 0;
    end
    for n = 60:70
        spec_img(n,j) = 0;
    end
    for n = 20:30
        spec_img(n,j) = 0;
    end
end

%Getting Back the Image for Magnitude Spectrum
%figure;subplot(2,1,1);
%axes('Units', 'normalized', 'Position', [0 0 1 1]);
%spec_img = log(1 + spec_img);
```

```
%imshow(spec_img,[]);
%delete background from figure
axes('Units', 'normalized', 'Position', [0 0 1 1]);
%inverse FFT
new_img = real(ifft2(ifftshift(spec_img)));

%Gaussian filter
H = fspecial('gaussian');
gauss = imfilter(new_img, H, 'replicate');
%imshow(gauss,[]);


%Averaging filter
avg = fspecial('average');
avg = imfilter(gauss, H, 'replicate');

imshow(avg, []);
%MEAN SQUARE ERROR CALCULATION
err = immse(genh_avg, orig_img);
fprintf('\n The MSE is %0.4f\n', err);
%L = size(genh_img);
%fprintf('\n The size is %0.4f\n', L);
```