Lab 1: Basics of Image Processing

Brian Hosler & Sarah Peachey

January 19, 2018

Abstract

In order to detect when images have been digitally manipulated, one must first have an understanding of how images can be maniputed. In this lab concepts such as contrast enhancement, high boost filtering, and edge detection were examined.

Contents

1 Contrast Enhancement

This section involved contrast enhancement, and a comparison of various different techniques.

1.1 Gamma Correction

First, a function, `Gcorrection.m`, was made, to do contrast enhancement through gamma correction. Given a grayscale image, and a value for gamma, the function rescales every pixel to a value between 0 and 1. Each pixel value is then raised to the power of gamma before being cast back to an intiger between 0 and 255. The code is shown below.

```
function [ img_out ] = Gcorrection(img_in, gama)
%Does gamma correction using the equation:
%    new=255*(old/255)^gamma
    img_out=uint8(255*(double(img_in)/255).^gama);
end
```

1.2 Effects of Gamma Correction

The Gcorrection function was then used with varying gamma arguments on the same photo to show the effects of gamma being larger than, smaller than, and equal to, unity. When gamma was set to 1, the mean squared error between the original and altered image was computed, to show that when gamma is equal to 1, every value is mapped to itself.

```
%read in the image
pout=imread('Assignment_1_Files/pout.tif');
figure
%Plot .4 enhanced image and histogram
subplot(2,3,1)
imshow(Gcorrection(pout,.4))
title('\gamma=0.4')
subplot(2,3,4)
imhist(Gcorrection(pout,.4))

%Plot unenhanced image and histogram
subplot(2,3,2)
imshow(Gcorrection(pout,1))
title(sprintf('\\gamma=1\nMSE from original: %d',immse(pout,
    Gcorrection(pout,1))))
subplot(2,3,5)
imhist(Gcorrection(pout,1))

%Plot 2.1 enhanced image and histogram
subplot(2,3,3)
imshow(Gcorrection(pout,2.1))
title('\gamma=2.1')
subplot(2,3,6)
imhist(Gcorrection(pout,2.1))
```

1.3 Histogram Equalization

Finally, Gamma correction was compared to histogram equalization using the photo MoonPhobos.tif. This photo has a bimodal distribution, with a concentration of pixel values near zero, and another near 255.

```
%Read in new photo
moonHobos=imread('Assignment_1_Files/MoonPhobos.tif');
figure
%Plot the enhanced image
subplot(1,2,1)
imshow(Gcorrection(moonHobos,.3))
title('\gamma=.3')
subplot(1,2,2)
imshow(histeq(moonHobos,256))
title('HistEQ')

figure
%Plot histograms of the enhanced image
subplot(1,2,1)
imhist(Gcorrection(moonHobos,.3))
title('\gamma=.3')
subplot(1,2,2)
imhist(histeq(moonHobos,256))
title('HistEQ')
```

The Gcorrection function is given an input γ less than 1, greater than 1 and equal to one. As seen in Figure 4, when the $\gamma = 0.4 < 1$ the picture becomes much lighter. Also, the image histogram is shifted to higher values and it's middle and high values are compressed. This results in the picture having higher grey values over a smaller range. When $\gamma = 1$ the original image is shown, therefore the image histogram can be used for reference. This is because any value raised to the first power is itself. When $\gamma = 2.1 < 1$ the image pixel values are lower but occupy a slightly larger range than the original image. The original image has most of it's pixel values concentraited in the center, around 127. Gamma correction does not take advantage of the full range of pixel values available in this instance, because it performs a unidirecitonal shift of the histogram, instead of spreading the concentration at the center.

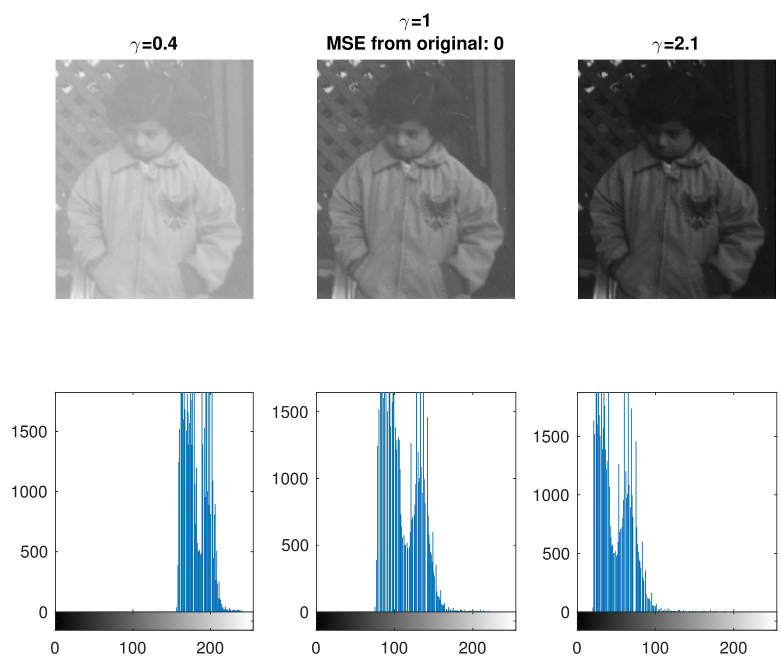

Figure 1: Varying gamma in Gcorrection function

The Gcorrection function was used to edit moonPhobos.tiff and the best percieved version of the picture was at $\gamma = 0.3$. For comparison the MATLAB built in function histeq was also used. The results of both can be see in 2. As seen in 3, the Gcorrection actually seemed to spread out the values of historgram better, but by making the pixel values lower, the image became much darker, so some of the details are lost. Whereas histeq does a better job of making sure the darker details and lighter details are maintained.

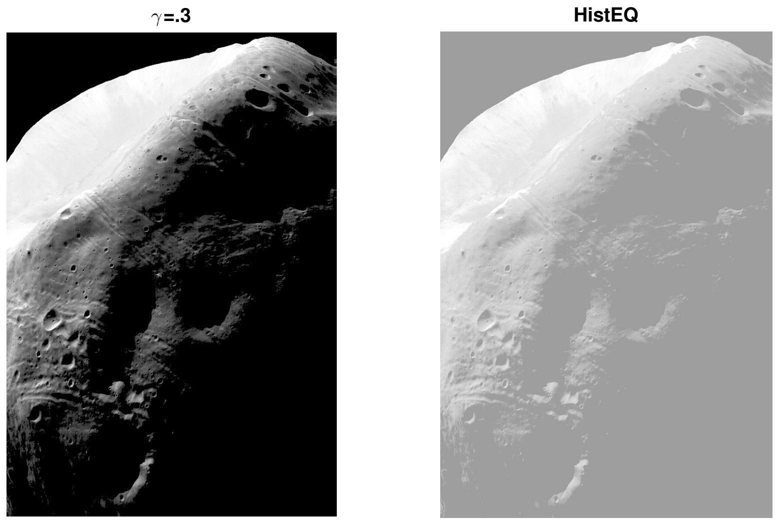

Figure 2: Gcorrection vs Histeq

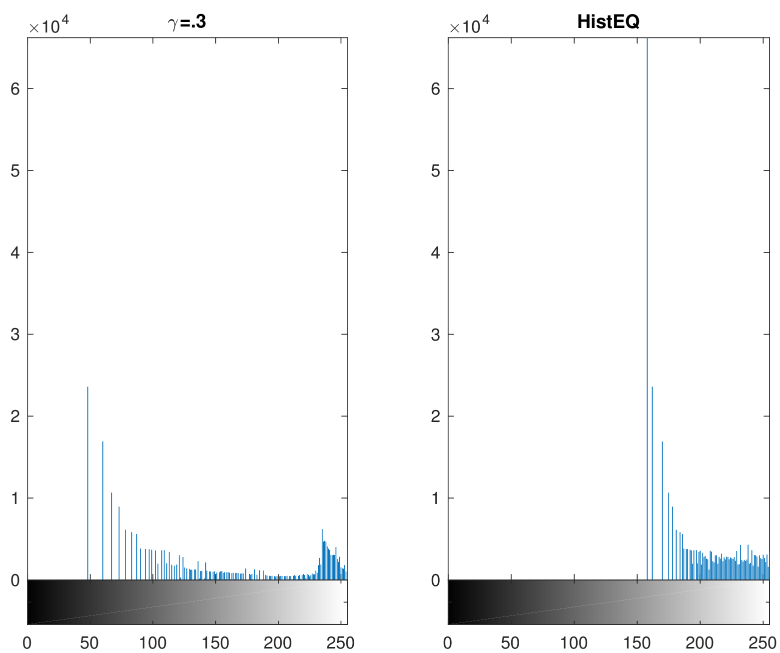

Figure 3: Histograms of Gcorrection vs Histeq

2 High Frequency Content

This section investigates the effect that high-frequency content has on an image.

2.1 High Boot Filtering

High Boost filtering is a technique that increases high-frequencies. This is achieved by extracting the ”sharpness” using a laplacian filter, and adding back to the original image a magnified version of that sharpness. The code to do so is shown below.

```
function [ img_out ] = HBfilt(img_in, alph)
%High boost filtering using a laplaccian filter
    img_out=img_in+uint8(alph*conv2(double(img_in),[0 -.25 0;
        -.25 1 -.25; 0 -.25 0],'same'));
end
```

2.2 Effect of High Boost Filtering

An image of the moon is filtered using the HBfilt function to better see the surface texture.

```
%Read in and filter the moon image
moon=imread('Assignment_1_Files/moon.tiff');
figure
imshow(HBfilt(moon,2.4))
title('\alpha=2.4')
```

2.3 High Boost Filtering to Sharpen an Image

Finally, an out-of-focus image is sharpened using High Boost filtering. Various alpha values were used to find the sharpest image.

```
%Read in a blurry image and high-boost filter it
oof=imread('Assignment_1_Files/outoffocus.tif');
figure
imshow(HBfilt(oof,4))
title('\alpha=4')
%High frequency noise added with increasing alpha(7)
```

During trial and error testing for the α value on the moon.tiff file an optimal value of 2.4 was selected. Values from zero to three with itations of 0.1 were scanned through. An optimal value was hard to select due to the small difference between successive images.

The concept of ”in-focus” does not seem to be well described since we were never the ”in-focus” image to compare to. We don’t beleive it is possible to recover the original image because blurring is a nonlinear operation so the original pixel values can only be approximated. We also found that when α was increase above 7 high frequency noise became apparent.

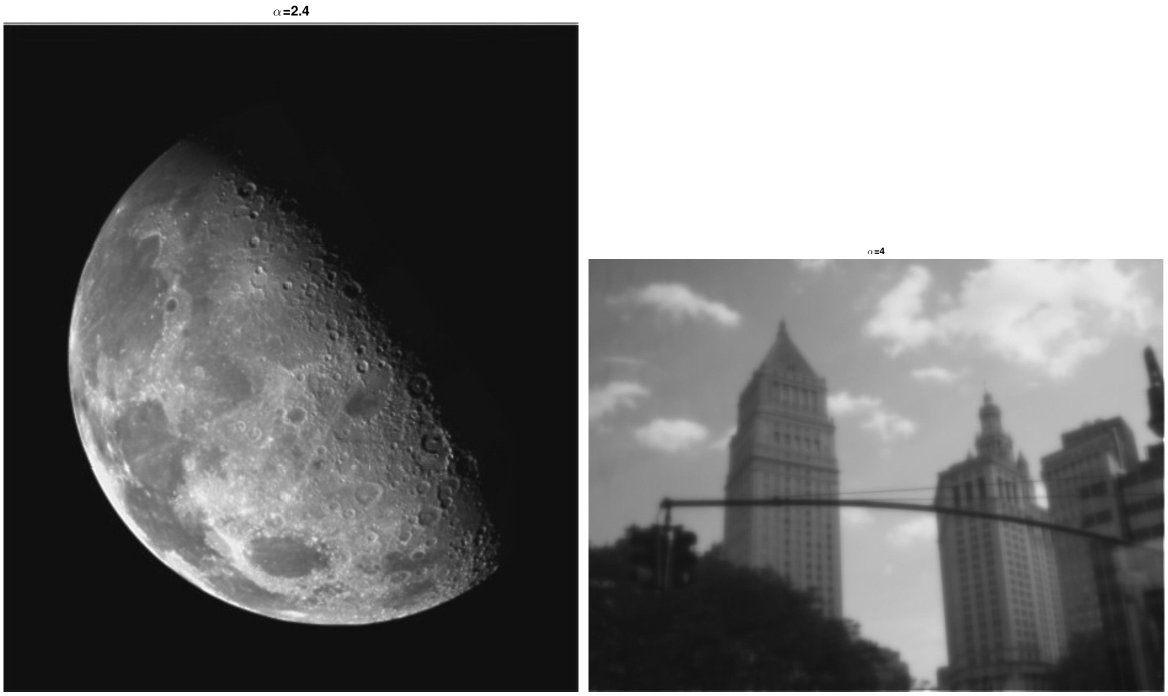

Figure 4: Sharpened image using high boost filter

3 Filtering Noise

Different types of noise respond differently to different de-noising fiters. Two images were created by adding two different typs of noise to the same initial image.

3.1 Average and Median Filtering

The noisy images were filtered each with a 3x3 and a 5x5 median filter. The images were also filtered with a 3x3 and a 5x5 averaging filter.

```
%Read in two noidy images
pep1=imread('Assignment_1_Files/peppersNoise1.tiff');
pep2=imread('Assignment_1_Files/peppersNoise2.tiff');
figure

%Denoise images with a 3x3 median filter
subplot(4,2,1)
imshow(medfilt2(pep1,[3,3]))
title(sprintf('peppersNoise1\nMedian 3x3'))
subplot(4,2,2)
imshow(medfilt2(pep2,[3,3]))
title(sprintf('peppersNoise2\nMedian 3x3'))

%Denoise images with a 5x5 median filter
subplot(4,2,3)
imshow(medfilt2(pep1,[5,5]))
title('Median 5x5')
subplot(4,2,4)
imshow(medfilt2(pep2,[5,5]))
title('Median 5x5')

%Denoise images with a 3x3 averaging filter
```

```
subplot(4,2,5)
imshow(uint8(filter2(ones(3,3)/9,pep1)))
title('Averaging 3x3')
subplot(4,2,6)
imshow(uint8(filter2(ones(3,3)/9,pep2)))
title('Averaging 3x3')

%Denoise images with a 5x5 averaging filter
subplot(4,2,7)
imshow(uint8(filter2(ones(5,5)/25,pep1)))
title('Averaging 5x5')
subplot(4,2,8)
imshow(uint8(filter2(ones(5,5)/25,pep2)))
title('Averaging 5x5')
```

3.2 Edge Detection and Noise

In this section, the two denoising filters are compared for their edge preserving properties.

```
%Save the average and median filtered images
pep1avg=uint8(filter2(ones(3,3)/9,pep1));
pep1med=medfilt2(pep1,[3,3]);
figure
th=60000;
subplot(1,2,1)
sx=filter2([-1,0,1;-2,0,2;-1,0,1],pep1avg).^2;%Xgradient
sy=filter2([-1,0,1;-2,0,2;-1,0,1].',pep1avg).^2;%Ygradient
imshow((sx+sy)>th)%Magnitude squared
subplot(1,2,2)
sx=filter2([-1,0,1;-2,0,2;-1,0,1],pep1med).^2;%Xgradient
sy=filter2([-1,0,1;-2,0,2;-1,0,1].',pep1med).^2;%Ygradient
imshow((sx+sy)>th)%Magnitude squared
```

As seen in Figure 6 the 3 x 3 filters result in sharper looking images as compared to the 5 x 5. The 5 x 5 median filter begins to make the image look blurry; the 3 x 3 median filter is suffient. A median filter almost perfectly eliminates salt-n-peppa' noise, whereas the averaging filter is susceptible to outliers such as the salt-n-peppa' "flakes". The averaging filter performs better than median filter for eliminating Gaussian white noise. The noise has zero mean so it averages to the correct value.

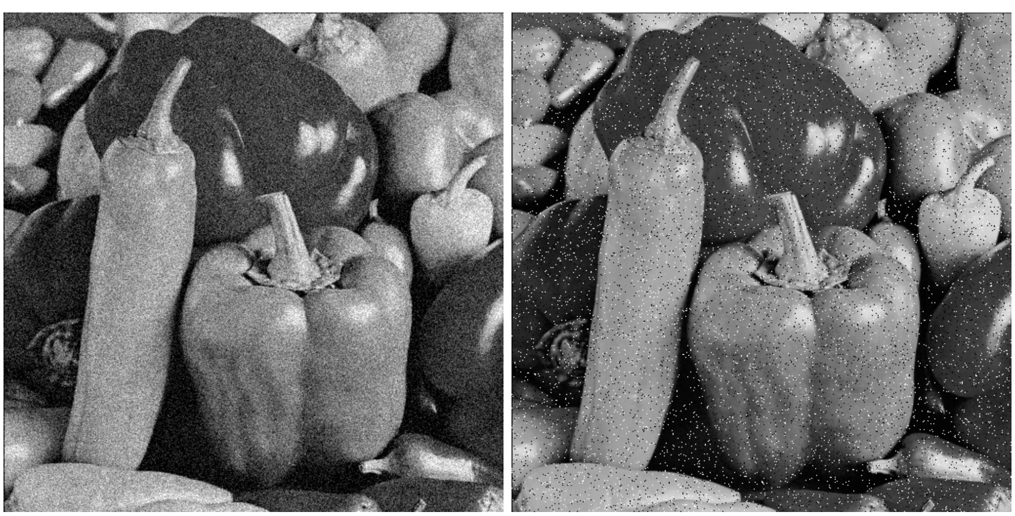

Figure 5: Pepper image with white noise (left) and salt-n-peppa' noise (right)

Figure 6: Median and averaging filter with two types of noise

The median filter preserves more edges although it maintains more noise than averaging filter. Comparing to the original image Figure 5, the noise appears to be incomplete edges caused by the median filtering of the Gaussian white noise.

Figure 7: Edge detection after median (right) and average (left) filter