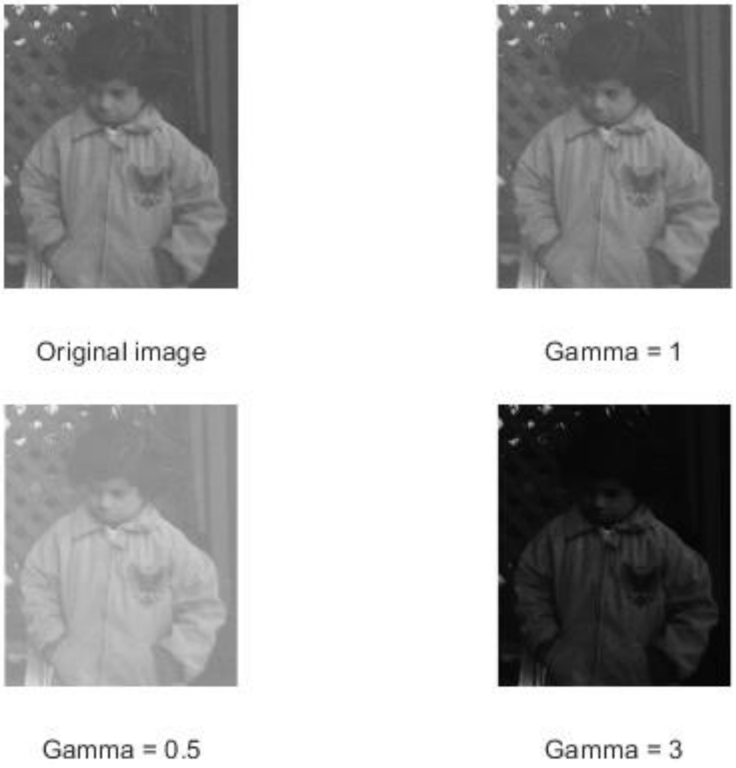
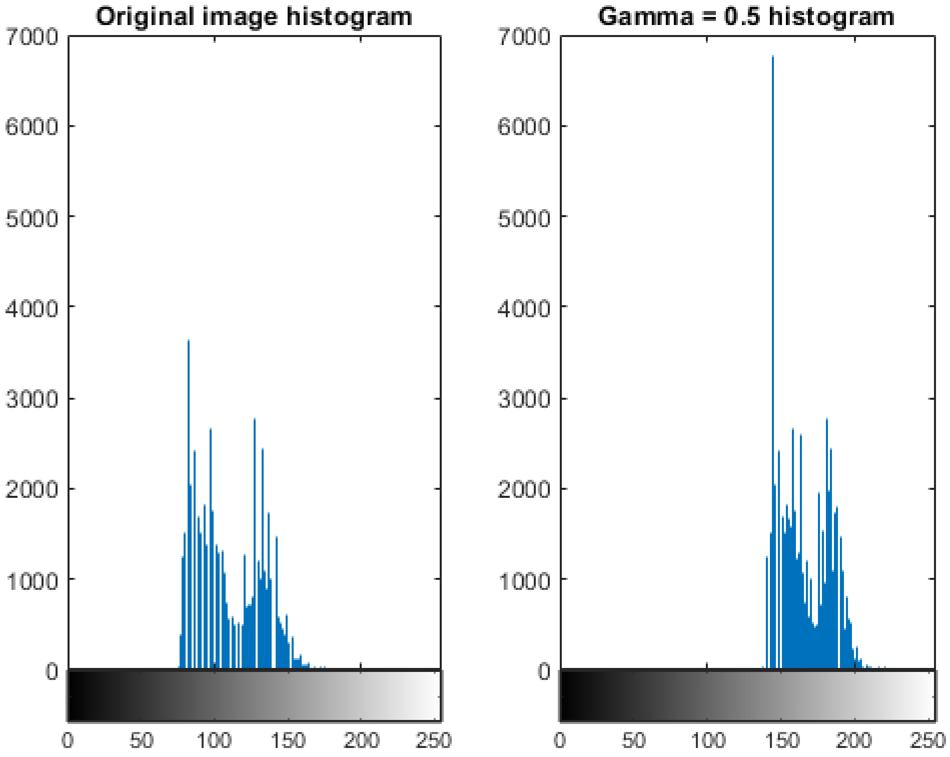
Kleanthis E Karavangelas, Basem Saleh

Part 1

```
x = imread('pout.tif');

% for gamma = 1

gc1 = GammaCorrection('pout.tif', 1);

% for gamma = 0.5;

gc2 = GammaCorrection('pout.tif', 0.5);

% for gamma = 3;

gc3 = GammaCorrection('pout.tif', 3);

figure(1);

subplot(2,2,1), imshow('pout.tif'), xlabel('Original image');
subplot(2,2,2), imshow(gc1), xlabel('Gamma = 1');
subplot(2,2,3), imshow(gc2), xlabel('Gamma = 0.5');
subplot(2,2,4), imshow(gc3), xlabel('Gamma = 3');

figure(2);
%sgtitle('Histogram comparison');
subplot(1,2,1), imhist(x); title('Original image histogram'); ylim([0 7000]);
subplot(1,2,2), imhist(gc2); title('Gamma = 0.5 histogram'); ylim([0 7000]);
```

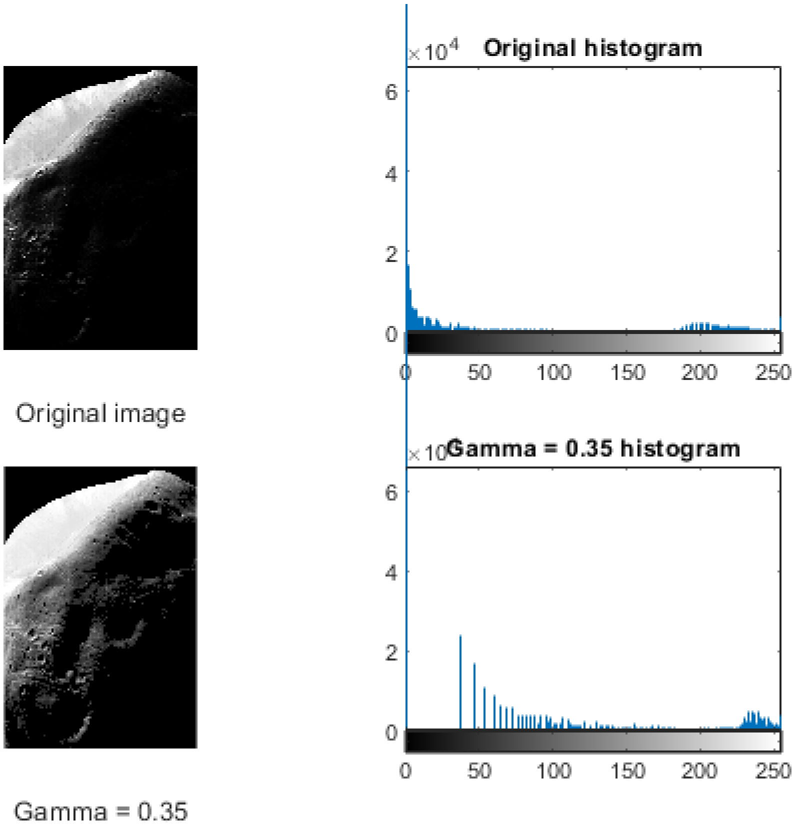

Original image
Gamma = 1
Gamma = 0.5
Gamma = 3


Original image histogram
Gamma = 0.5 histogram
7000
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250

For gamma > 1, the gamma correction expression makes every pixel value get closer to 0, therefore making the image darker. For gamma < 1, the expression makes every pixel value get closer to 255, giving the image a lighter, or whiter color. This can be also derived from the image's histogram (for gamma=0.5, pixel values shift to the right). For gamma = 1, the pixel values do not change, so we recover the original image.

Moon Phobos

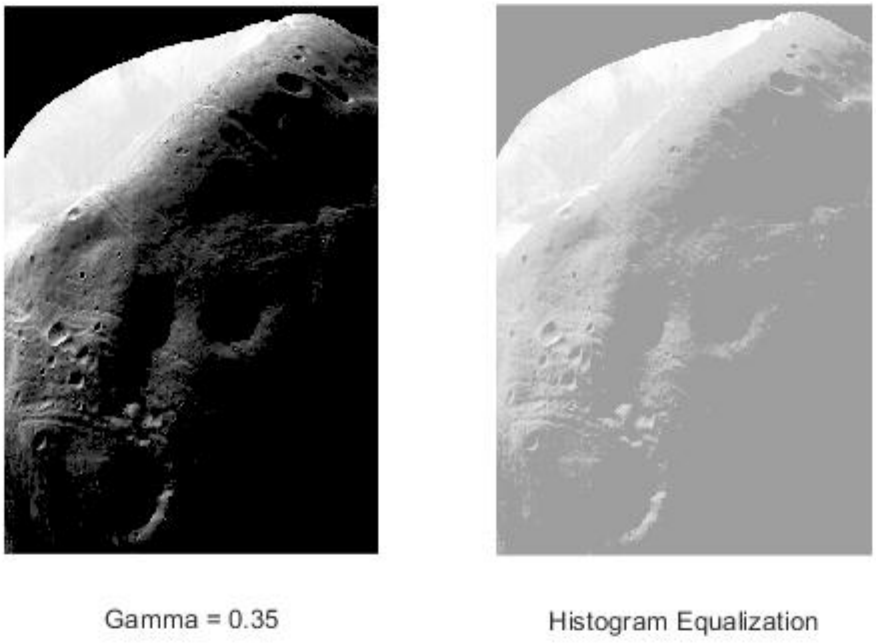
```
x = imread('MoonPhobos.tif');

gc = GammaCorrection('MoonPhobos.tif',0.35);

figure(3);
subplot(2,2,1), imshow('MoonPhobos.tif'), xlabel('Original image');
subplot(2,2,2), imhist(x), title('Original histogram');
subplot(2,2,3), imshow(gc), xlabel('Gamma = 0.35');
subplot(2,2,4), imhist(gc), title('Gamma = 0.35 histogram');
```

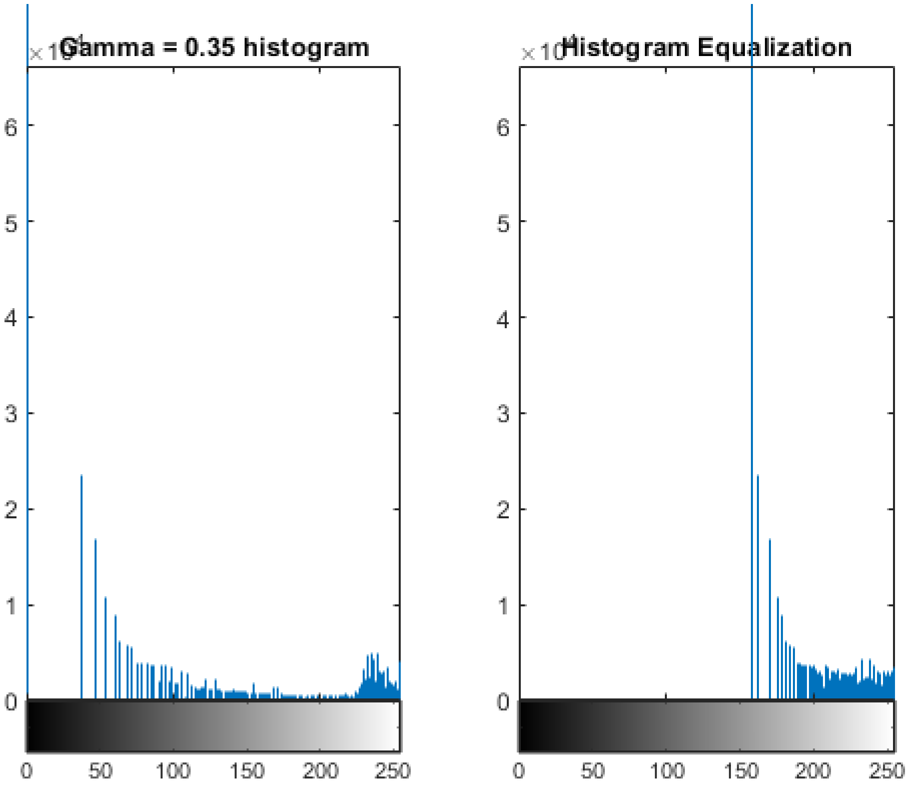


After trying out many gamma values, the one that seems to produce the best result to the eye is 0.35. At 0.35, you are able to see more information regarding the image, such as edges. At values above 0.35, the picture starts to get darker, making hard to notice any edges. At values below 0.35, the image starts to look like it has been processed a lot, and also looks kind of fake.

Histogram equalization

```
xHisEq = histeq(x,256);

figure(4);
subplot(1,2,1), imshow(gc), xlabel('Gamma = 0.35');
subplot(1,2,2), imshow(xHisEq), xlabel('Histogram Equalization');

figure(5);
subplot(1,2,1), imhist(gc), title('Gamma = 0.35 histogram');
subplot(1,2,2), imhist(xHisEq), title('Histogram Equalization');
```

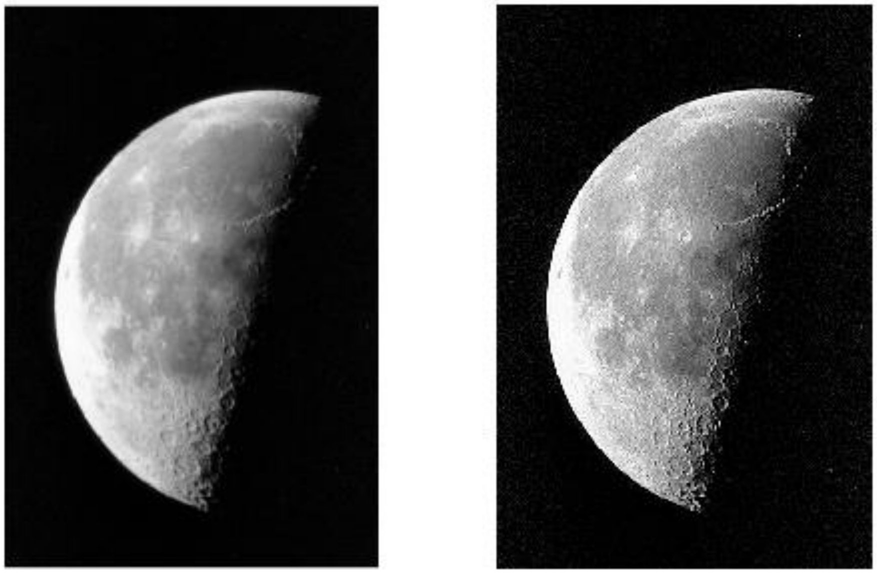




Visually, the picture looks better when performing gamma correction, compared to histogram equalization.

Part 2

```
figure(6);

hbf = HighBoostFiltering('moon.tif',5);

subplot(1,2,1), imshow('moon.tif'), xlabel('Original photo');
subplot(1,2,2), imshow(hbf), xlabel('High-boost filter applied');
```

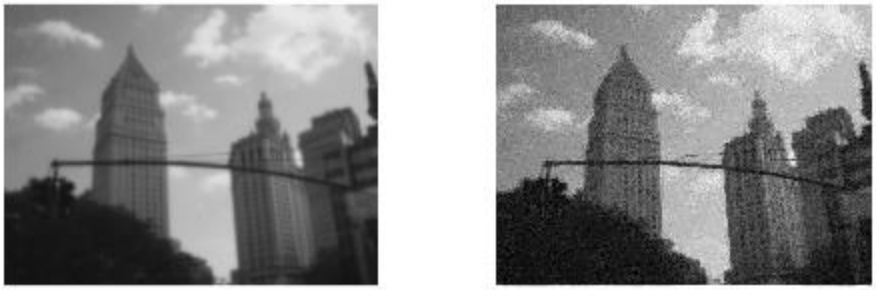


Original photo

High-boost filter applied

After trying out many different values, the group settled with a = 5, a value which produced an image with sharper edges, and almost no grain. Values below 5 did not reveal much information, while values above 5 resulted in a lot of grain.

```
hbf = HighBoostFiltering('outoffocus.tif',30);

figure(7);
subplot(1,2,1), imshow('outoffocus.tif'), xlabel('Out of focus');
subplot(1,2,2), imshow(hbf), xlabel('High-boost filter applied');
```



Out of focus

High-boost filter applied

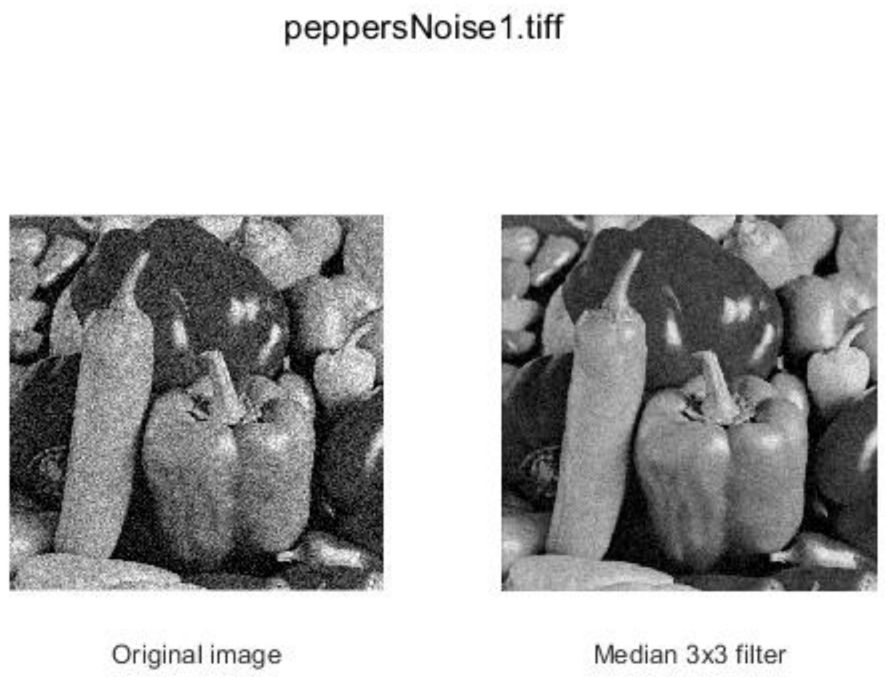
The picture looks like it is in focus, but the more you increase a, the more grain you notice. The image can be recovered and seem to be in focus, but grain or pepper noise will exist.

The downsides or unintended artifacts when sharpening an image is that the more you sharpen it, the more grain you see on the picture.

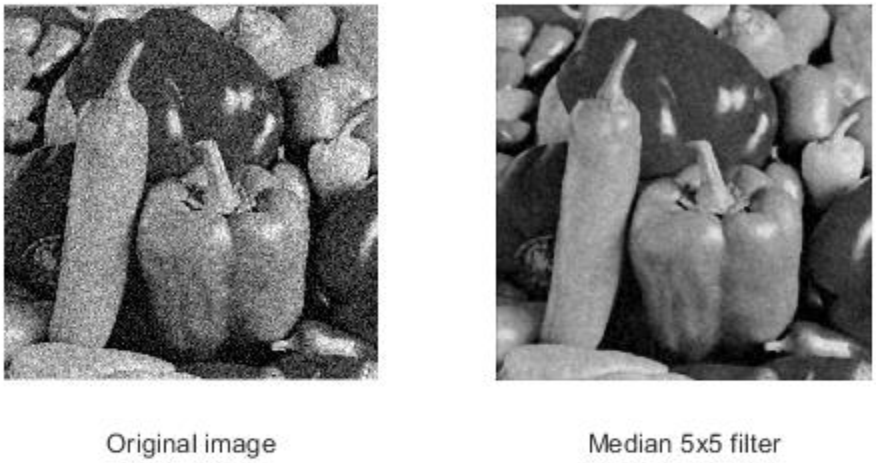
Part 3

```
p1 = imread('peppersNoise1.tiff');

% 3x3 median filter
med3 = medfilt2(p1, [3 3]);

% 5x5 median filter
med5 = medfilt2(p1, [5 5]);

% 3x3 average filter
avg3 = ones(3)/9;
p1avg3 = filter2(avg3,p1);

% 5x5 average filter
avg5 = ones(5)/25;
p1avg5 = filter2(avg5,p1);

figure(8);
sgtitle('peppersNoise1.tiff');
```

```
subplot(1,2,1), imshow('peppersNoise1.tiff'), xlabel('Original image');
subplot(1,2,2), imshow(med3), xlabel('Median 3x3 filter');

figure(9);
sgtitle('peppersNoise1.tiff');
subplot(1,2,1), imshow('peppersNoise1.tiff'), xlabel('Original image');
subplot(1,2,2), imshow(med5), xlabel('Median 5x5 filter');

figure(10);
sgtitle('peppersNoise1.tiff');
subplot(1,2,1), imshow('peppersNoise1.tiff'), xlabel('Original image');
subplot(1,2,2), imshow(uint8(p1avg3)), xlabel('Average 3x3 filter');

figure(11);
sgtitle('peppersNoise1.tiff');
subplot(1,2,1), imshow('peppersNoise1.tiff'), xlabel('Original image');
subplot(1,2,2), imshow(uint8(p1avg5)), xlabel('Average 5x5 filter');
```



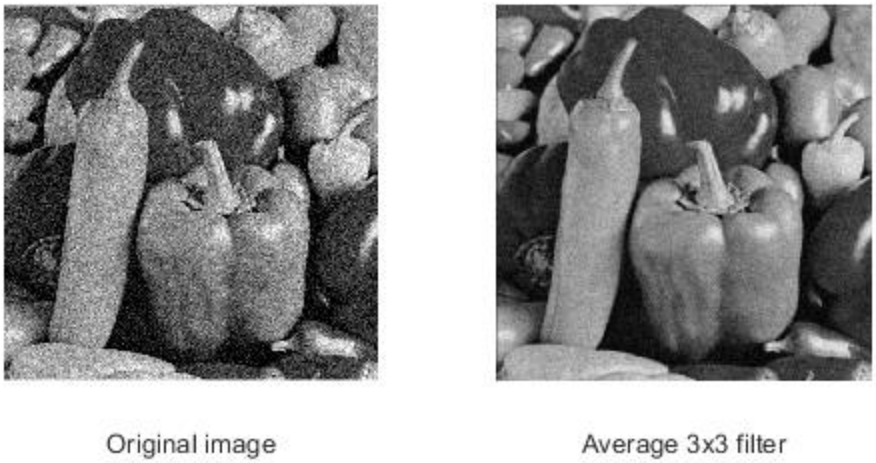
peppersNoise1.tiff



Original image

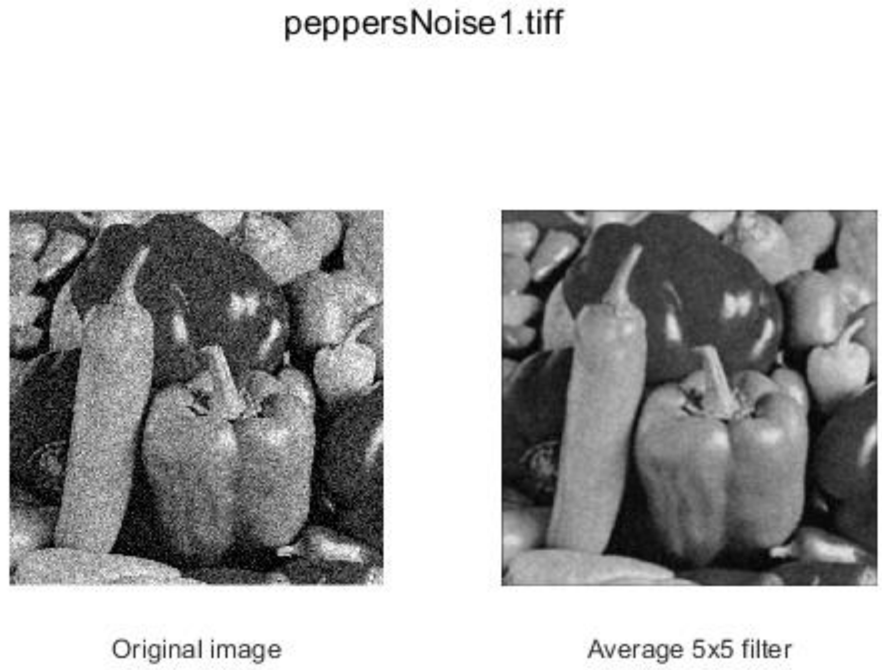
Median 5x5 filter

peppersNoise1.tiff



Original image

Average 3x3 filter



peppersNoise1.tiff

Original image

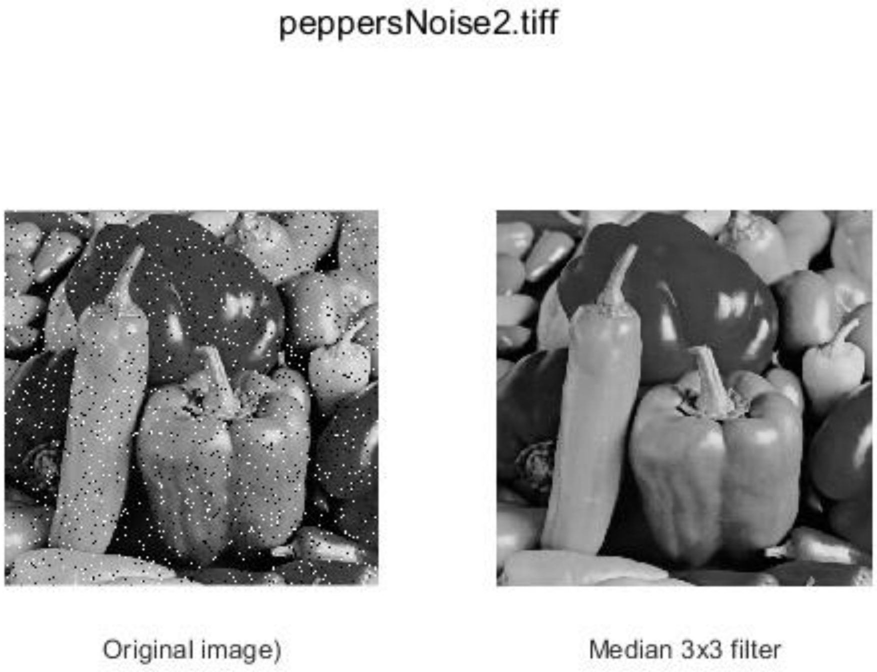
Average 5x5 filter

For the first pepper image, even though the two filters used are different, they both seem to work similarly, at least to the eye. Both the median and the average filter in this case do a decent job of removing grain or snow from the original image. With regards to the size of the filter, the 5x5 filters remove more snow, but blur the picture more compared to the 3x3 filters.

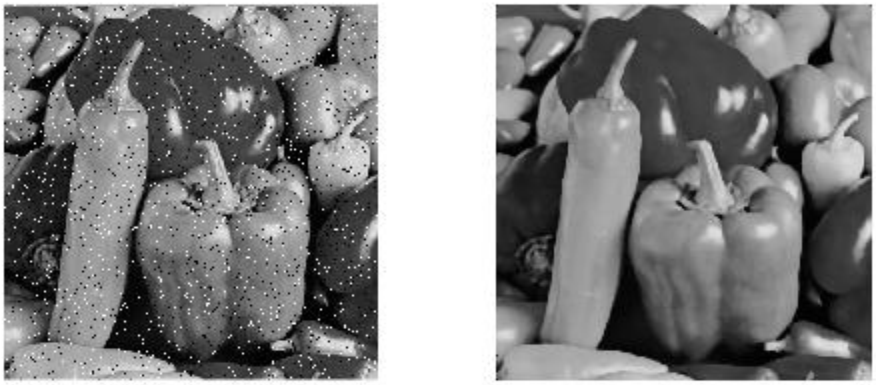
pepper 2

```
p2 = imread('peppersNoise2.tiff');

% 3x3 median filter
med3pep2 = medfilt2(p2, [3 3]);

% 5x5 median filter
med5pep2 = medfilt2(p2, [5 5]);

% 3x3 average filter
avg3 = ones(3)/9;
p2avg3 = filter2(avg3,p2);

% 5x5 average filter
avg5 = ones(5)/25;
p2avg5 = filter2(avg5,p2);

figure(12);
sgtitle('peppersNoise2.tiff');
```

```
subplot(1,2,1), imshow('peppersNoise2.tiff'), xlabel('Original image)');
subplot(1,2,2), imshow(med3pep2), xlabel('Median 3x3 filter');

figure(13);
sgtitle('peppersNoise2.tiff');
subplot(1,2,1), imshow('peppersNoise2.tiff'), xlabel('Original image');
subplot(1,2,2), imshow(med5pep2), xlabel('Median 5x5 filter');

figure(14);
sgtitle('peppersNoise2.tiff');
subplot(1,2,1), imshow('peppersNoise2.tiff'), xlabel('Original image');
subplot(1,2,2), imshow(uint8(p2avg3)), xlabel('Average 3x3 filter');

figure(15);
sgtitle('peppersNoise2.tiff');
subplot(1,2,1), imshow('peppersNoise2.tiff'), xlabel('Original image');
subplot(1,2,2), imshow(uint8(p2avg5)), xlabel('Average 5x5 filter');
```



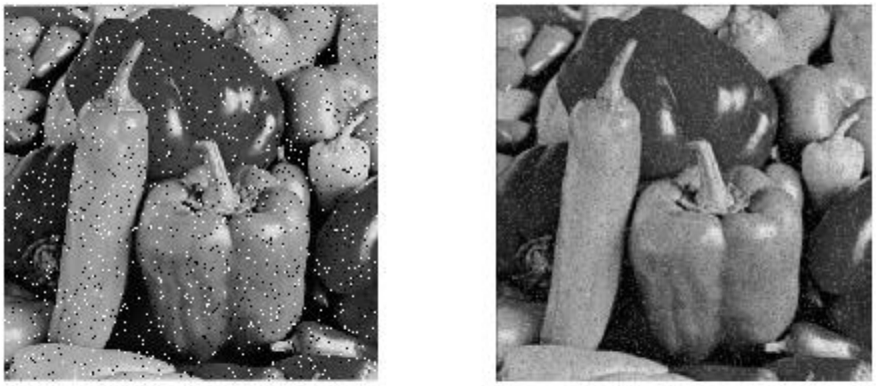
peppersNoise2.tiff



Original image

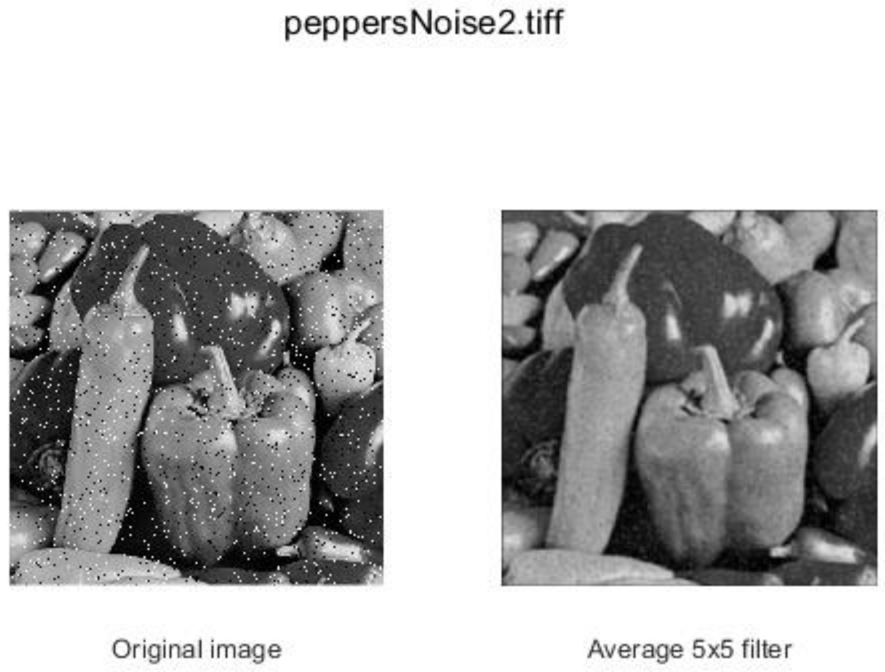
Median 5x5 filter

peppersNoise2.tiff



Original image

Average 3x3 filter



peppersNoise2.tiff

Original image

Average 5x5 filter

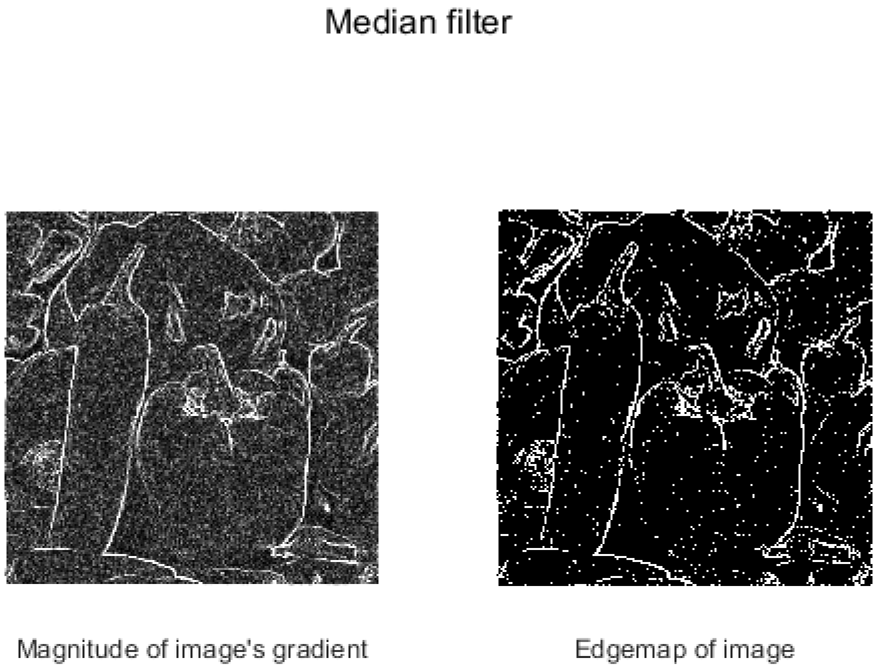
When talking about the second pepper image, there are more differences regarding each filter used. In this case, the median filter performed better in removing the pepper-salt noise (or random white and black dots in the picture) compared to the average filter. Once again, increasing the size of the filter resulted in a blurrier image.

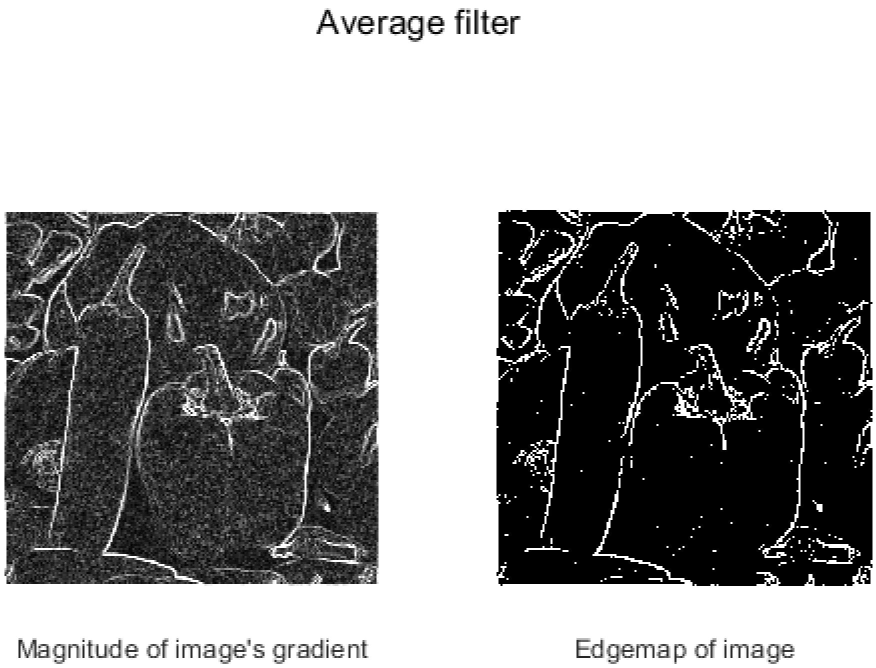
```
% median filter

% Sobel filter for row direction
SobelX = [-1 0 1; -2 0 2; -1 0 1];

% column direction
SobelY = SobelX.';

% gradient for each direction
Gx = filter2(SobelX, med3);
Gy = filter2(SobelY, med3);

% gradient magnitude
gradMag = (Gx.^2 + Gy.^2).^.5;

% threshold value
thres = 128;
% edgemap
edgemap = gradMag > thres;
```

```
figure(16);
sgtitle('Median filter');
subplot(1,2,1), imshow(uint8(gradMag)), xlabel('Magnitude of image''s gradient');
subplot(1,2,2), imshow(edgemap), xlabel('Edgemap of image');
```

average filter

```
Gx = filter2(SobelX, p1avg3);
Gy = filter2(SobelY, p1avg3);

gradMag = (Gx.^2 + Gy.^2).^.5;

thres = 128;
edgemap = gradMag > thres;

figure(17);
sgtitle('Average filter');
subplot(1,2,1), imshow(uint8(gradMag)), xlabel('Magnitude of image''s gradient');
subplot(1,2,2), imshow(edgemap), xlabel('Edgemap of image');
```





Average filter

Magnitude of image's gradient

Edgemap of image

For the same threshold value, it seems that the average filter produces a cleaner edge map, with less pepper noise. It should be mentioned that the median filter preserves more properties, but the extra grain preserved is most likely not useful information.

Gamma Correction function

```
function gc = GammaCorrection(Img, Gamma)

    % first, we read the image passed into the function
    x = imread(Img);

    % we then convert the data from unit8 to double, so we can apply the
    % mapping
    x = double(x);

    % here we modify each pixel value, with the following expression
    gc = 255*(x/255).^Gamma;

    % convert back to unit8 in order to be able to show the image
    gc = uint8(gc);

end
```

High-Boost Filtering function

```
function hbf = HighBoostFiltering(Img,a)

    % first we read the image
    orig = imread(Img);

    % then convert it from uint8 to double so we can modify it
    orig = double(orig);

    % create the Laplacian filter
    Lf = [0 -0.25 0;
        -0.25 1 -0.25;
        0 -0.25 0];

    % apply the filter to the image to get g(x,y)
    g = filter2(Lf,orig);

    % add back to original image to obtain sharpened version
    hbf = orig + a.*g;

    % convert back to uint8 to be able to show the image
    hbf = uint8(hbf);

end
```

Published with MATLAB® R2019b