Applying Smoothing Filters to Remove Film Grain from Images

ECE 529 Course Project - Fall 2022
Semester

Prof. Jeffrey Rodriguez

University of Arizona College of Electrical & Computer Engineering

Nicolas Blanchard

December 6, 2022

TABLE OF CONTENTS

1.0 Introduction and Scope

Images and videos sourced from film present a unique challenge in the digital age. Film, being an analog format, is subject to visual imperfections that originate from a variety of sources. The most common include metallic silver particles from the photochemical coating that records light and dust particles from the developing environment. Both of these contaminate a film negative, leading to the interpositive and subsequent transfers to display speckles and grain.

Removing grain from video poses several benefits, most notably in video compression. Videos that are uniform (contain low grain) are more efficient to compress.

This project focuses on developing several competing smoothing filters to remove film grain from images. The custom smoothing filters include a local averaging filter, a local median filter, and a FIR low pass filter. Other considerations include adaptive filtering for edge-preservation and a logarithmic based filtering approach for multiplicative noise.

The custom smoothing filter implementations are qualitatively measured based on two factors: how much noise is removed from the image and how blurry the filtered image looks. They are also quantitatively measured by calculating the SNR of a filtered image whose noise level is known.

As a final comparison, each custom smoothing filter is pitted against Adobe Photoshop's noise reduction algorithm, a proven industry standard for removing noise from images (including film grain).

All of the code in the attached .zip file named 'MATLAB_code.zip' is handwritten by me. The most notable uses of outside libraries are calls to built-in MATLAB functions, namely *fft2() / ifft2()* to compute a two dimensional Fourier transform and inverse Fourier transform,

psnr() to compute the SNR and PSNR of filtered images, and *filter2()* / *conv2()* to convolve a filter with a 2D signal.

Significant hardship was encountered creating the ideal lowpass filter for this project. I went through many iterations of the ideal lowpass filter implementation, with only the final implementation functioning properly. I suspect that small nuances in the way I was handling FFTs and inverse FFTs caused the observed quirks in the output images, including circular patterns. I arrived at the working solution with the help of Jae S. Lim's *Two-Dimensional Signal Processing*, some helpful notes from Professor Rodriguez, and a lot of trial and error.

2.0 Common Smoothing Filters

Common smoothing filters utilized for noise reduction in images include a local averaging filter, a local median filter, and a low pass filter.

A local averaging filter replaces each pixel in an image with the average intensity of every pixel in the surrounding area. The amount of smoothing in the resulting image is controlled by the radius of the window that calculates each average. A large window for each average results in more smoothing in the final image.

A local median filter replaces each pixel in an image with the median intensity of every pixel in the surrounding area. Similar to the local averaging filter, the amount of smoothing in the resulting image is controlled by the radius of the window that calculates each median. A large window for each median results in more smoothing in the final image. Because a median is more resistant to outliers and doesn't create any new pixel values, median filters generally do a better job of preserving edges than averaging filters.

A low pass filter removes all of the high frequency components from an image's frequency domain. This is done by designing a filter with a known 'passband' centered in the low frequency region. Image noise is represented in the high frequencies, so removing these frequencies helps remove noise. An important note is that low pass filtering and local averaging are similar processes; the difference is that one is completed in the spatial domain and one is completed in the frequency domain. We can expect similar levels of smoothing from both.

Expansions on these filtering methods, including adaptive filtering for edge detection and considerations for multiplicative noise, are also incorporated.

3.0 Model for Film Grain

For general testing of each filter, images sourced from true film scans are used. However, to determine SNR (signal to noise ratio), images with precisely known noise are required.

The focus of this project is on filtering film grain out of digital signals, not developing a perfect model for film grain. Therefore, the model that I introduce here is utilized purely as a metric to qualitatively compare the effectiveness of the different methods I will introduce in upcoming sections. That being said, the process of filtering out noise can be greatly benefited from having a clear understanding of how that noise originates.

The model that I introduce here is an additive model for film grain noise. This means that the noise is adding intensity to the underlying noise-free pixels. However, in reality, film grain is an extremely complex form of noise that has varying origins and characteristics. Norkin & Birkbeck suggest that it is more accurately represented using a multiplicative model before it is processed (p.1). I use an additive model out of simplicity; additive noise is generally easier to filter out of images as well as easier to artificially create for testing purposes. Multiplicative

models for film grain are further discussed in *Section 5.4 - Logarithmic Approach for Multiplicative Noise*, where I explore a different method of filtering film grain under the assumption that the noise is multiplicative.

The additive model for film grain used for this project is implemented using Adobe Photoshop. Adobe Photoshop is a trusted industry standard for image processing, and their proprietary method for adding noise to an image is accepted in industry.

Noise is added to images using the following procedure:

1) Image is loaded into Photoshop workspace.
2) A new layer is added over the image. The fill is set to 50% gray and the blending is set to overlay.
3) Gray layer is converted to a 'Smart Object'.
4) Under the 'Filtering' tab, 'Add Noise' is selected.
5) The 'Add Noise' filter is applied to the gray layer with 20% intensity and a Gaussian pattern.
6) Under the 'Fintering' tab, 'Blur' is selected.
7) For the gray layer (which contains the noise) ONLY, a 1 pixel Gaussian blur is introduced. This blur mimics the softness of true film grain.

Figure 3.0.1 - Before film grain is added. Source: Ambulance *(2022).*

Figure 3.0.2 - After film grain is added.

4.0 Frequency Domain Approach - FIR Ideal Low Pass Filter, Window Method

A frequency domain approach is implemented to apply a lowpass filter to the image. The exact implementation is provided in the file 'custom_lowpass.m'. This is an ideal lowpass filter, which, as described by Gonzalez & Woods, allows certain frequencies within a range to pass and completely attenuates frequencies outside of that range. The pass region is radially symmetric about the origin (Gonzalez and Woods, p. 292). An FIR filter is preferred for this application because it is guaranteed to be stable and will not reshape the phase content of the image.

4.1 First Attempt - Frequency Sampling

My first attempt at crafting a lowpass filter relied on the frequency sampling method. First, a matrix of samples of the desired frequency response is created. For an ideal lowpass filter, this is a radially symmetric region that allows low frequencies to pass.

```
% Making our desired frequency response for an IDLP:
a = 0:(M-1);                                % Create a vector from 0 to M-1.
position_x = find(a>M/2);
a(position_x) = a(position_x) - M;          % Shift that vector so that it is centered at the origin.
b = 0:(N-1);                                % Create a vector from 0 to N-1.
position_y = find(b>N/2);
b(position_y) = b(position_y) - N;          % Shift that vector so that it is centered at the origin.
[B, A] = meshgrid(b, a);                    % Create a mesh grid using our vectors. This essentially creates matrices A and B
                                            % which are composed of repeating vectors a and b.
distance = sqrt(A.^2 + B.^2);               % To eventually make our mask, we will need to know the distance of each point from
                                            % the origin. This line calculates distance.
Hd = double(distance <= cutoff);            % Now, anywhere with a distance less than our cutoff gets kept. This creates a disk of radius 'cutoff'.
```

Then, the ideal impulse response is obtained by taking the inverse Fourier transform of the frequency response data.

```
% Obtaining hd(n1, n2), the desired impulse response:
hd = ifft2(Hd);                             % Inverse Fourier Transform our frequency response
```

Next, the actual impulse response of our filter is obtained by multiplying the ideal impulse response by a window function. For this application, I chose a separable 2D Hamming window.

```
% Creating our window
w = hamming(M) * hamming(N)';               % 2D Hamming window

% Obtaining h(n1, n2), our actual impulse response:
h = hd .* w;
```

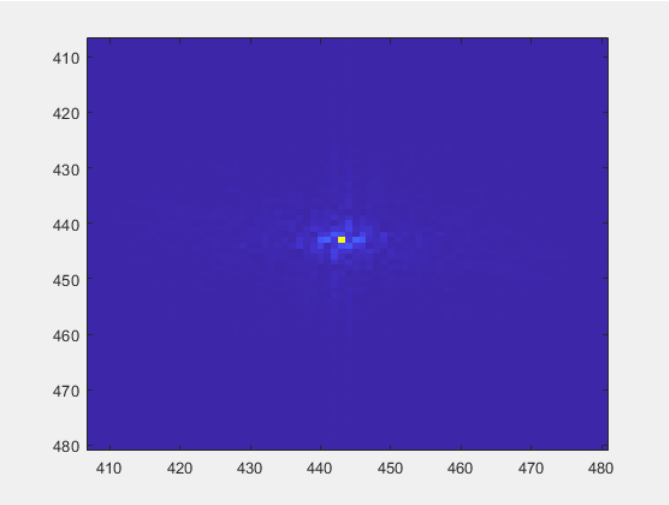

Figure 4.1.1 - Frequency domain representation of a test image sourced from the 2006 movie 300.

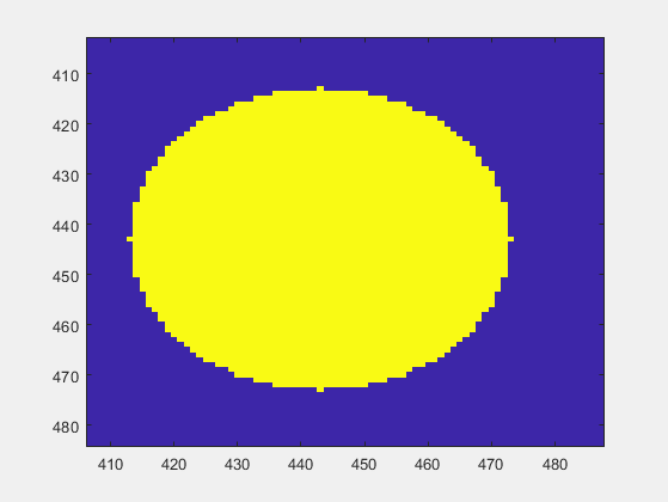

Figure 4.1.2 - Frequency response of the lowpass filter implemented using frequency sampling. This represents the 'pass' region of the filter.

Finally, the impulse response of the filter is convolved with the image to yield the filtered result.

```
% Convolve h(n1, n2) with our signal:
final_image = conv2(image, h);
```

However, the result is less than satisfactory. The filtered image displayed an obvious circular pattern introduced by the low pass filter.

Figure 4.1.3 - Input image to the filter.

Figure 4.1.4 - Output (filtered) image.

4.2 Second Attempt - Using Impulse Response of an ILPF

My next attempt at crafting a lowpass filter bypassed the frequency sampling method and went straight to the impulse response of an ideal lowpass filter. The impulse response of an ideal lowpass filter in two dimensions is as follows:

$$h(n_1, n_2) = \frac{\omega_c}{2\pi\sqrt{n_1^2 + n_2^2}} \cdot J_1\left(\omega_c \cdot \sqrt{n_1^2 + n_2^2}\right) \qquad (7.31)$$ *(Lim, p.356)*

This ideal impulse response is implemented in MATLAB code.

```
% Determining digital filter
a = (M - 1) / 2;                                    % Half of filter length
b = (N - 1) / 2;                                    % Half of filter height
[n1, n2] = meshgrid(-a:a, -b:b);                    % Matrices of filter length and height
n = sqrt(n1.^2 + n2.^2);                            % Distance from origin of every point in the matrix
hd = (cutoff ./ (2 * pi * n)) .* besselj(1, cutoff*n);   % Impulse response of an ideal lowpass filter for n not equal to 0
hd(a+1, b+1) = (cutoff^2)/(4*pi);                   % Impulse response of an ideal lowpass filter at n equal to 0
```

Then, the ideal impulse response is multiplied by a 2D separable Hamming window.

```
% Creating our window w(n1, n2) = w1(n1) * w1(n2):
w = hamming(M) * hamming(N)';                       % 2D Hamming window

% Obtaining h(n1, n2) = hd(n1, n2) * w(n1, n2), our actual impulse response:
h = hd .* w;
```

Finally, the impulse response of the filter is convolved with the image to yield the filtered result.

```
% Convolve h(n1, n2) with our signal:
final_image = filter2(image, h);
```

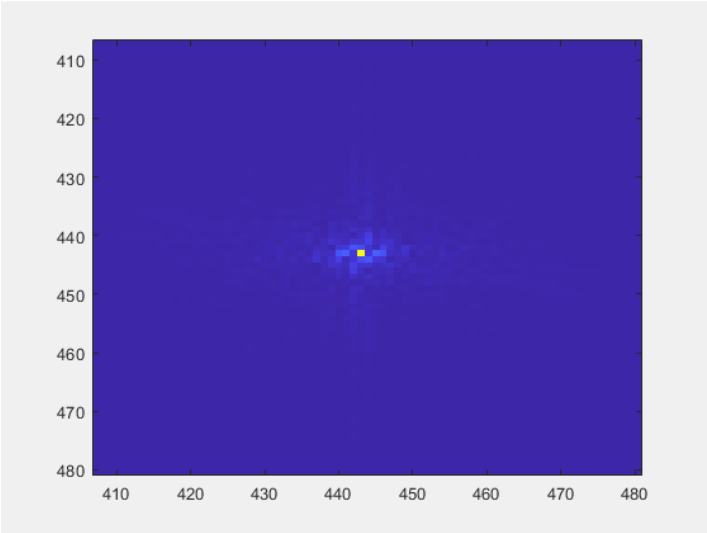

Figure 4.2.1 - Frequency domain representation of a test image sourced from the 2006 movie 300.

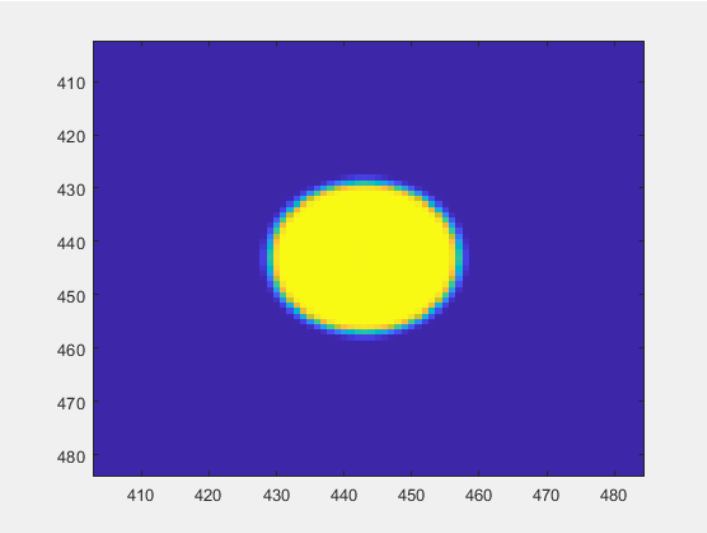

Figure 4.2.2 - Frequency response of the lowpass filter implemented using the ideal lowpass filter impulse response. This represents the 'pass' region of the filter.

Although the frequency response of this filter (shown in *Figure 4.2.2*) looks extremely similar to the frequency response of the previous filter (shown in *Figure 4.1.2*), the result does not suffer from the layer of circular artifacts.

Figure 4.2.3 - Input image to the filter.

Figure 4.2.4 - Output (filtered) image: $\omega_c = 0.2$ *rad.*

For the remainder of this paper, any discussion of a lowpass filter refers to this filter, created using the known impulse response of an ideal lowpass filter in two dimensions. The results shown in *Section 7.0* demonstrate the effectiveness of this implementation.

5.0 Spatial Domain Approach

Multiple spatial domain approaches are taken to develop smoothing filters. These are discussed in sections *5.1 - 5.4*

5.1 Local Averaging [custom_average.m]

One of the most simple smoothing techniques is an averaging filter. We begin by finding the size of the image:

```
% Size of image:
[M, N] = size(image);
```

Then, we enter into the main portion of the algorithm. We have nested for loops that iterate over the entire image, pixel by pixel:

```
% Begin moving window
for current_y = window_radius+1:M-window_radius
    for current_x = window_radius+1:N-window_radius
```

Inside the nested for loops, we have more nested for loops that calculate the average pixel value of the surrounding area.

```
for i = window_radius*-1:window_radius
    for j = window_radius*-1:window_radius
        temp_sum = temp_sum + uint32(image(current_y + i, current_x + j));
    end
end
```

Finally, we replace the pixel value of interest with the calculated average of the surrounding area:

```
% Update the pixel value in the center with the median value within
% the frame
 final_image(current_y, current_x) = temp_sum / ((2*window_radius + 1)^2);

 % Reset temp_sum for the next loop
 temp_sum = 0;
```

5.2 Local Median [custom_median.m]

A similar smoothing filter is a local median filter. It uses the same nested for loop structure as discussed in section *5.1*. The implementation only differs in the calculation performed within each window radius:

```
for i = 1:window_radius*2 + 1
    for j = 1:window_radius*2 + 1
        loop_variable = loop_variable + 1;
        temp_array(loop_variable) = window(i, j);
    end
end


% Update the pixel value in the center with the median value within
% the frame
sorted_array = sort(temp_array);
final_image(current_y, current_x) = median(sorted_array);
loop_variable = 0;
```

Here, instead of calculating the average within the window, we are finding the median value within the window. Each pixel is updated to the median value of the window around it.

5.3 Adaptive Filtering for Edge Preservation [custom_adaptive.m]

As you will see in section *7.0 - Comparison of Results*, the local averaging and local median smoothing filters make the image blurry. Edges become less defined after the filter has been applied. In order to preserve these edges, we implement a form of edge detection that utilizes the Prewitt Kernel. The Prewitt Kernel changes size based on the selected window radius, but shown below are the Prewitt Kernels in the x and y directions for a window radius of 5 pixels:

```
prewitt_x =

    -1     0     0     0     0     0     0     0     0     0     1
    -1     0     0     0     0     0     0     0     0     0     1
    -1     0     0     0     0     0     0     0     0     0     1
    -1     0     0     0     0     0     0     0     0     0     1
    -1     0     0     0     0     0     0     0     0     0     1
    -1     0     0     0     0     0     0     0     0     0     1
    -1     0     0     0     0     0     0     0     0     0     1
    -1     0     0     0     0     0     0     0     0     0     1
    -1     0     0     0     0     0     0     0     0     0     1
    -1     0     0     0     0     0     0     0     0     0     1
    -1     0     0     0     0     0     0     0     0     0     1
```

Figure 5.3.1 - Prewitt Kernel for the x direction.

```
prewitt_y =

    -1    -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0
     1     1     1     1     1     1     1     1     1     1     1
```

Figure 5.3.2 - Prewitt Kernel for the y direction.

By multiplying these kernels element-wise with each selected window, we are able to determine if an edge is present within the window. If the sum of each kernel multiplied by the window is large, then it indicates a large difference in pixel intensities between the top and bottom of the image (in the case of the y-direction Prewitt Kernel) or between the left and right of the image (in the case of the x-direction Prewitt Kernel). This means that there is an edge present.

```
% If there is not a large difference in pixel values between the two
% sides or top and bottom, then we are not on an edge:
if ( abs(sum(prewitt_x .* window, 'all')) < alpha && abs(sum(prewitt_y .* window, 'all')) < alpha )
```

If an edge is detected, no smoothing is performed and the original pixel values are retained.

```
        % Update the pixel value in the center with the median value within
        % the frame
        sorted_array = sort(temp_array);
        final_image(current_y, current_x) = median(sorted_array);
        loop_variable = 0;
    end
    % Otherwise, if there is a large difference in pixel values between
    % the two sides or top and bottom, then we are on an edge and no
    % smoothing should occur:
end
```

For my project, edge detection is implemented on a local median filter in the file ‘custom_adaptive.m’.

5.4 Logarithmic Approach for Multiplicative Noise [custom_adaptive_multiplicative.m]

As discussed in section *3.0 - Model for Film Grain*, some sources (including Norkin & Birbeck, p.1) model unprocessed film grain as multiplicative noise rather than additive noise. Multiplicative noise is more difficult to remove with simple smoothing filters than additive noise because it is signal dependent. One way of transforming multiplicative noise into additive noise is by taking the logarithm of each pixel value (Bioucas-Dias & Figueiredo, 3).

```
% Take the natural log of our image:
% (this effectively transforms the multiplicative noise into additive
% noise):
image = log(double(image+1));
```

Then, after the filtering is finished, the final image must be exponentiated to reverse the effects of the initial logarithm:

```
% Now, we must transform our log image back into its original form:
final_image = uint8(exp(final_image));
```

The logarithmic approach to applicative noise is used in conjunction with an adaptive median filter in my project.

6.0 Adobe Photoshop Approach

To determine how effective my custom developed filtering methods are, a control method is required to compare them against. I have selected Adobe Photoshop's Reduce Noise Filtering functionality as this control method. All of the previously discussed filters that I have implemented in MATLAB will be compared to the Adobe Photoshop filter to see how well they perform relative to an industry standard.

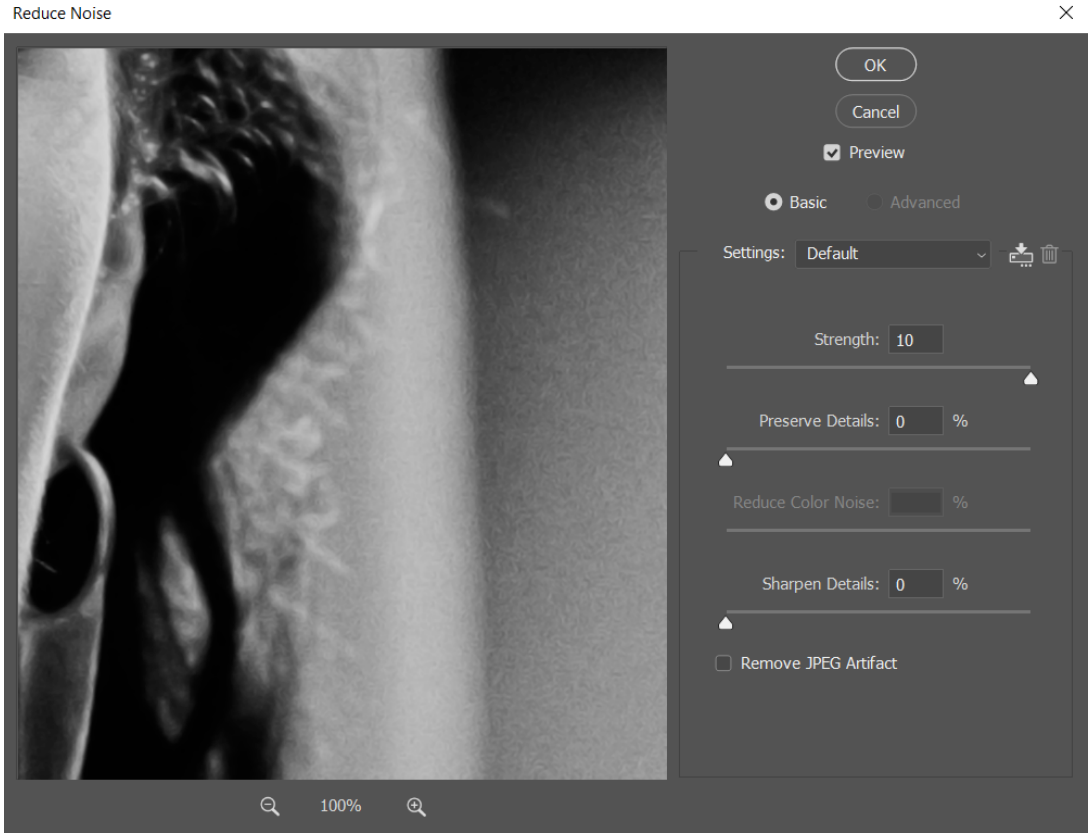

Figure 6.0.1 - Settings used to obtain filtered images through Adobe Photoshop.

To obtain control results from Photoshop, the images were subjected to the following procedure:

1) Image is loaded into Photoshop workspace.
2) Image is converted to a 'Smart Object'.
3) Under the 'Filtering' tab, 'Reduce Noise' is selected.
4) The 'Reduce Noise' filter is applied with *Strength* at 10, *Preserving Details* at 0, *Sharpen Details* at 0, and *Remove JPG Artifact* unchecked.

7.0 Comparison of Results

7.1 Image Sourced from Film Scan (Natural Grain)

The following image, sourced from a high-resolution uncompressed capture of the 2006 movie *300*, is used as the input image for the following filter tests. This image was captured on 35mm film and features pronounced film grain.

Figure 7.1.1 - Cropped section of a screenshot from a high quality source of 300.

Important Note: The full extent of the grain reduction displayed in the following images cannot be totally appreciated without zooming in or using a large screen. The full images are included in the .zip file. For my analysis in *Section 8.0 - Conclusion*, I closely inspected the uncompressed full size images.

Lowest Smoothing Level

Local Average Filter
[3 px window radius]

Local Median Filter
[3 px window radius]

Adaptive Local Median Filter
[3 px window radius; α = 250]

Adaptive Local Median Filter with Logarithm
[3 px window radius; α = 15]

Lowpass Filter
[cutoff = 0.8 rad]

Photoshop Filter
[strength = 10]

Table 7.1.1 - Low Smoothing With a 3px Window, Adaptive Filtering Alphas of 250 and 15, a Lowpass Cutoff of 0.8 rad, and Photoshop Strength of 10.

Medium Smoothing Level

Local Average Filter
[5 px window radius]

Local Median Filter
[5 px window radius]

Adaptive Local Median Filter
[5 px window radius; α = 500]

Adaptive Local Median Filter with Logarithm
[5 px window radius ; α = 16]

Lowpass Filter
[cutoff = 0.5 rad]

Photoshop Filter
[strength = 10]

Table 7.1.2 - Medium Smoothing With a 5px Window, Adaptive Filtering Alphas of 500 and 16, a Lowpass Cutoff of 0.5 rad, and Photoshop Strength of 10.

Highest Smoothing Level

Local Average Filter
[7 px window radius]

Local Median Filter
[7 px window radius]

Adaptive Local Median Filter
[7 px window radius; α = 900]

Adaptive Local Median Filter with Logarithm
[7 px window radius; α = 20]

Lowpass Filter
[cutoff = 0.2 rad]

Photoshop Filter
[strength = 10]

Table 7.1.3 - High Smoothing With a 7px Window, Adaptive Filtering Alphas of 900 and 20, a Lowpass Cutoff of 0.2 rad, , and Photoshop Strength of 10.

7.2 Image With Known Noise (Artificial Grain)

In order to accurately determine the SNR of each output image, we need to know what is noise and what is part of the original image. Therefore, a grainy image is created from a noise free image using the process described in section *3.0 - Model for Film Grain*. The noise free image and artificially grainy image are shown below.

Figure 7.2.1 - Grain free image sourced from the 2022 movie Ambulance*. Used as reference to determine what noise remains in images post filtering.*

Figure 7.2.2 - Grainy image that is fed into each filter as an input.

The MATLAB *psnr()* function is used to calculate both peak signal to noise ratio and overall signal to noise ratio. The *psnr()* function uses the following formulas for SNR and PSNR:

$$\text{SNR} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[f(x,y)-\hat{f}(x,y)]^2}$$

Figure 7.2.3 - Definition of SNR from Gonzalez & Woods, p. 376.

$$PSNR = 10\log_{10}(peakval^2/MSE)$$

$$\text{MSE} = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[f(x,y)-\hat{f}(x,y)]^2$$

Figure 7.2.4 - Definition of PSNR from MATLAB documentation and definition of MSE from Gonzalez & Woods, p. 376.

Shown below are the PSNR and SNR of each filter method when applied to the grainy image shown in *Figure 7.2.2*. The parameters (window size, alpha, and cutoff frequency) are the same as described in section *7.1*.

Lowest Smoothing

Method	No Filtering	Local Average Filter	Local Median Filter	Adaptive Local Median Filter	Adaptive Local Median Filter for Multiplicative Noise	Lowpass Filter	Photoshop Filter (strength = 10)
SNR (dB)	20.9587	23.3311	23.4227	23.5583	23.2743	21.7752	24.1287
PSNR (dB)	32.1122	34.5422	34.6351	34.7632	34.3794	32.9601	35.3394

Table 7.2.1 - SNR and PSNR for a 3px window radius, α = 250 and α = 15 (logarithmic), cutoff frequency =0.8 rad..

Medium Smoothing

Method	No Filtering	Local Average Filter	Local Median Filter	Adaptive Local Median Filter	Adaptive Local Median Filter for Multiplicative Noise	Lowpass Filter	Photoshop Filter (strength = 10)
SNR (dB)	20.9587	22.0602	22.5604	23.1067	22.7383	21.6541	24.1287
PSNR (dB)	32.1122	33.2939	33.7924	34.3263	33.8592	32.8643	35.3394

Table 7.2.2 - SNR and PSNR for a 5px window radius, α = 500 and α = 16 (logarithmic), and cutoff frequency = 0.5 rad.

Highest Smoothing

Method	No Filtering	Local Average Filter	Local Median Filter	Adaptive Local Median Filter	Adaptive Local Median Filter for Multiplicative Noise	Lowpass Filter	Photoshop Filter (strength = 10)
SNR (dB)	20.9587	20.9944	21.9003	22.6468	22.5607	19.0981	24.1287
PSNR (dB)	32.1122	32.2487	33.1497	33.8781	33.6873	30.3648	35.3394

Table 7.2.3 - SNR and PSNR for a 7px window radius, α = 900 and α = 20 (logarithmic), and cutoff frequency = 0.2 rad.

Based on the data shown in *Tables 7.2.1 - 7.2.3*, it is clear that out of the custom filters, the highest SNR and PSNR is achieved using an adaptive local median filter with a window size of 3 pixels and an alpha of 250.

Examining *Table 7.2.3*, we see that at large window radii, the performance of the adaptive local median filter and the adaptive local median filter for multiplicative noise are comparable. The local median filter is the next highest, and the local average filter performs the worst.

The lowpass filter is an interesting case - the SNR initially increases a lot with cutoff frequency (between 0.2 rad and 0.5 rad) but subsequently increases only a small amount (between 0.5 rad and 0.8 rad). This is further explored in *Section 7.3*.

7.3 Performance Analysis With Comprehensive Parameter Variation

Varying individual parameters gives additional insight into the performance of each filter. The plots below show how changing window size, alpha, and cutoff frequency affects the SNR of the resulting image. Once again, the grainy image shown in *Figure 7.2.2* is used as an input.

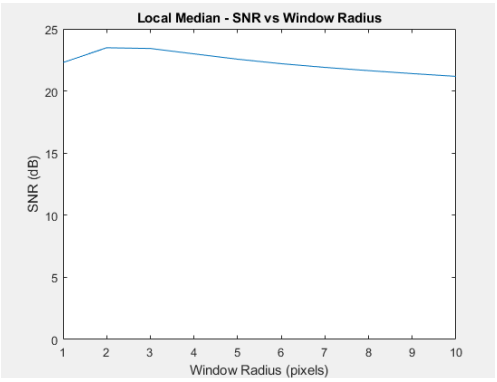

Figure 7.3.1 - SNR vs Window Radius for the Local Median Filter.

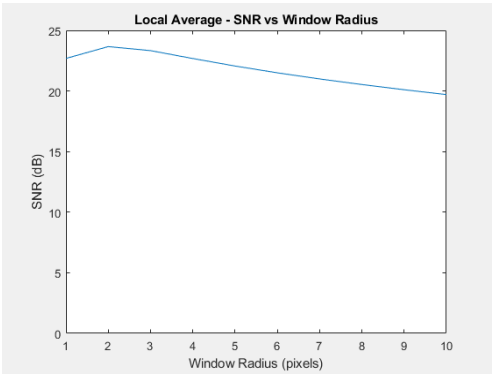

Figure 7.3.2 - SNR vs Window Radius for the Local Average Filter.

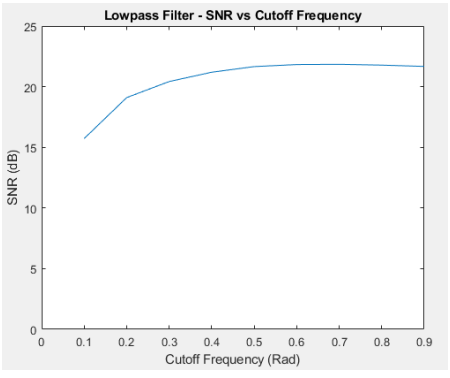

Figure 7.3.3 - SNR vs Cutoff Frequency for the Lowpass Filter. At $\omega_c = 0$ *rad, SNR = -∞.*

Figures 7.3.1 - 7.3.2 demonstrate that for local median and local averaging filters, SNR initially increases with window radius. However, at a window radius of 2 pixels, SNR begins to steadily decrease. To maximize SNR, it is best to choose a small window size (2 pixel radius).

Figure 7.3.3 shows that for the lowpass filter, peak SNR is achieved at a cutoff frequency of 0.7 rad. This explains why we saw the SNR of the lowpass filtered image initially increased a lot but then slowed down in *Tables 7.2.1 - 7.2.3.*

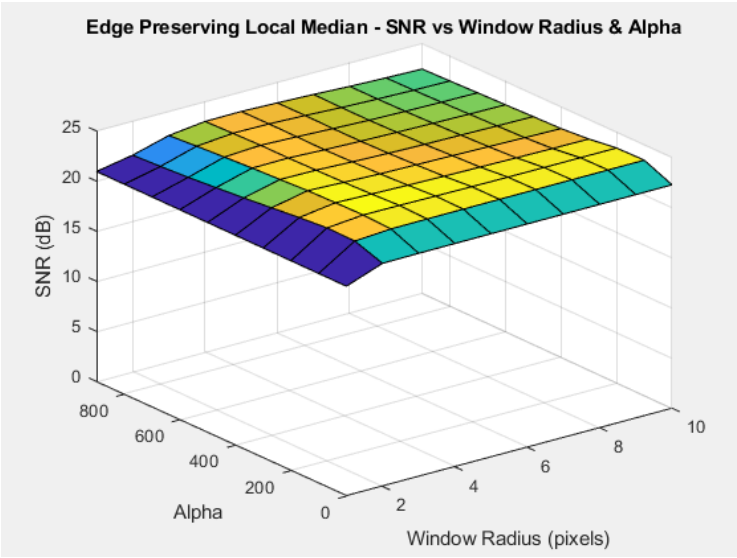

Figure 7.3.4 - SNR vs Window Radius and Alpha for the Edge Preserving Local Median Filter.

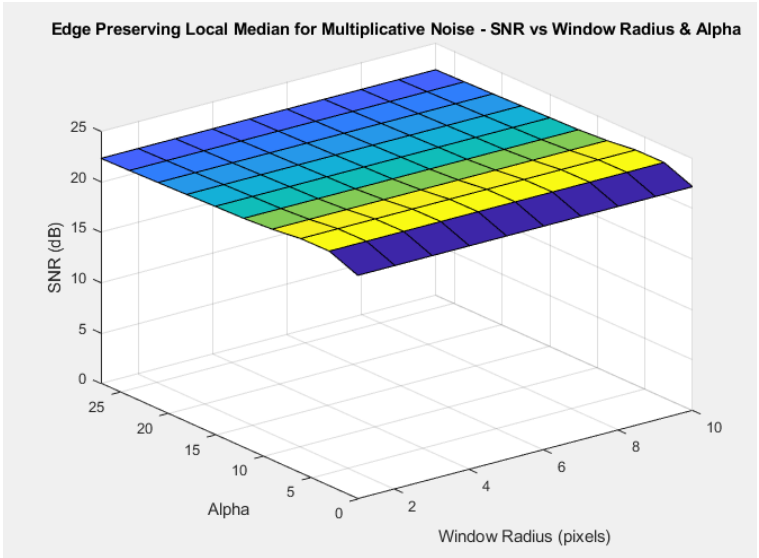

Figure 7.3.5 - SNR vs Window Radius and Alpha for the Edge Preserving Local Median Filter for Multiplicative Noise.

Figures 7.3.4-7.3.5 demonstrate the effects of window radius and alpha on the two edge-preserving local median filters. The result is similar for both; at extremely low alpha values (when little smoothing is happening), SNR increases slightly with window size. However, at higher alpha values (when the image is being smoothed and only edges are being preserved), SNR significantly decreases with window size. Therefore, in any useful application where we need smoothing to take place, we should minimize the window size to increase SNR. This

explains why the higher SNR values are obtained for the smallest window size in *Tables 7.2.1 - 7.2.3*.

8.0 Discussion & Conclusion

8.1 Visual Comparison

As briefly noted in *Section 7.0*, the full effectiveness of each filter is difficult to appreciate looking at the small and compressed images present in this PDF. If the reader wishes to compare the results for themselves, then I suggest utilizing the full size images located in the '300 results' directory of 'MATLAB_code.zip'. Similarities and differences between the filters are most obvious under the highest smoothing parameters, but the patterns exist at all levels of smoothing.

Looking at *Table 7.1.3* with the results of each filter performing the highest level of smoothing, we see that the local average filter creates the most blurry image. The lowpass filter yields similar levels of blurriness, which is expected - however, it also introduces a severe 'ringing' effect. This 'ringing' effect is discussed in more detail in *Section 9.0 - Further Research*. Both of these filters destroy the fine detail and edges within the image, but remove virtually all of the grain.

The local median filter is comparable to the local average filter and lowpass filter in terms of overall grain reduction. However, the result is significantly less blurry and retains far more detail. The definition of hairs, the iris of the eye, and the earring reveal a marked improvement of detail retention when compared to the local averaging filter.

Furthermore, the adaptive median filter also shares the same level of grain reduction, but retains even more fine detail. Zooming in on edges reveals the algorithm at work - sharp edges

and areas of high contrast are un-smoothed. Edge preservation allows for facial hairs to be retained, specular highlights on the hair to shine through, definition around the lips and eyes to remain sharp, and reflections on the earring to remain unaltered.

The adaptive median filter for multiplicative noise yields extremely similar results to the regular adaptive median filter. Once again, the overall level of grain reduction is the same, but it appears that edges are preserved slightly more consistently. This is seen in the specular highlights on the eye under *Table 7.1.3*.

The photoshop filter achieves a level of grain reduction that is on par with the other filters on the lowest smoothing level (*Table 7.1.1*). The results are visually similar to the adaptive median filter for multiplicative noise. Specular highlights in the eyes, definition around the lips, and reflections on the earrings are all preserved. However, the photoshop filter preserves more fine detail in the hair (especially in the reflections) than any of the custom MATLAB filters.

8.2 Conclusion

Considering the results of the visual comparison as well as the SNR and PSNR values recorded in *Tables 7.2.1 - 7.2.3*, we conclude that the adaptive median filter performed the best out of the custom implementations. It consistently had the highest SNR and PSNR and it effectively preserved edges within the test image while reducing grain by just as much as every other filter.

The adaptive median filter for multiplicative noise also performed extremely well. Although the SNR and PSNR were slightly lower than for the normal adaptive median filter, there is evidence that it was more effective at preserving the fine edges within the test image.

The lowpass filter and local average filter did successfully remove grain, but also removed all of the fine detail within the image, rendering the images useless. The local median filter also successfully removed grain, and it preserved more detail than the lowpass filter and local average filter, but did not preserve detail as well as the adaptive filters.

None of the custom implementations were quite as effective as the photoshop filter. As shown in *Tables 7.2.1 - 7.2.3*, the photoshop noise reduction had a higher SNR and PSNR than all of the custom implementations. That being said, the SNR was only 0.6 dB above the highest SNR attained by the adaptive median filter, so it was a close competition.

Overall, although none of the filters quite matched the effectiveness of Photoshop's noise reduction service, they were all effective in reducing grain in film images. The adaptive median filters were even competitive with the Photoshop filter when it came to preserving edges and fine detail.

9.0 Further Research

The next step in continuing my work is to improve upon the adaptive filtering methods. Although the Prewitt Kernel is reliable in detecting sudden transitions in an image, it does not always detect fine details like hairs and subtle specular highlights. A more advanced algorithm for detecting and preserving edges, such as the gradient magnitude approach suggested by Mittal et al in *An Efficient Edge Detection Approach to Provide Better Edge Connectivity for Image Analysis*, would improve the visual quality of the filtered images.

Another area that can be expanded upon is the frequency domain filtering. The implemented ideal lowpass filter leaves 'ringing' on the filtered image. This is caused by the spatial domain representation of the filter; in the spatial domain, the filter resembles a *sinc* function. The sidelobes of the *sinc* function cause a 'ringing' pattern in the filtered image (Gonzalez & Woods, p. 295). A Gaussian low pass filter or Butterworth low pass filter would not suffer from this 'ringing' effect in the spatial domain.

In addition, implementing a circularly symmetric window in the 2D FIR filter design process is a direction for further research. This window would treat all frequencies of the same magnitude the same, regardless of phase.

A final area for improvement is to implement more comprehensive qualitative analysis of the results. It is critical, especially in the movie industry where the test images were sourced from, that a filtered image retains enough detail to not be noticeably degraded. Therefore, expanded qualitative analysis is called for to more accurately classify the performance of each filter. A preliminary thought is to conduct a brief study where participants are asked to blindly rank each filtered image on several factors, including prevalence of grain and blurriness.

10.0 Reproducibility

To reproduce the results of my work, please see the submitted .zip file named 'MATLAB_code.zip'. MATLAB scripts are included in the 'MATLAB' directory. To run each filter, see the 'main.m' MATLAB file.

Within 'main.m', a user can control the various parameters used in the filters. These parameters are set in the 'Filter Parameters and Choosing Source Image' section at the top of the file. This section is pictured below:

```
%% Filter Parameters and Choosing Source Image
clear;
clc;
image = imread('../images/300_cropped_bw.png');
image = im2gray(image);

% Window size for filters:
window_size = 7;

% Sensitivity for Prewitt Kernel based edge detection:
alpha = 900;
multiplicative_alpha = 20;

% Cutoff frequency for lowpass filter (rad):
cutoff_freq = 0.4;
```

Figure 10.0.1 - Filter parameter control in 'main.m'.

Further down in the file, the 'Displaying results' section deals with the results of the image filtering. Edit this section to display the results in different ways.

In the 'images' directory, a collection of test images are included. Use any of these images for demonstration purposes or use your own.

11.0 References & Acknowledgements

J. M. Bioucas-Dias and M. A. T. Figueiredo, "Multiplicative Noise Removal Using Variable Splitting and Constrained Optimization," in IEEE Transactions on Image Processing, vol. 19, no. 7, pp. 1720-1730, July 2010, doi: 10.1109/TIP.2010.2045029.

A. Norkin and N. Birkbeck, "Film Grain Synthesis for AV1 Video Codec," 2018 Data Compression Conference, 2018, pp. 3-12, doi: 10.1109/DCC.2018.00008.

Gonzalez, Rafael C. and Woods, Richard E. *Digital Image Processing*. 3rd Edition, Pearson Prentice Hall, Upper Saddle River, 2008.

Lim, Jae S. *Two-Dimensional Signal Processing*. Prentice Hall, 1990.

M. Mittal et al., "An Efficient Edge Detection Approach to Provide Better Edge Connectivity for Image Analysis," in IEEE Access, vol. 7, pp. 33240-33255, 2019, doi: 10.1109/ACCESS.2019.2902579.

Thank you to **Professor Jeffrey Rodriguez** for assistance troubleshooting the lowpass filter!