EEE321 Signals and Systems

Lab 4

Sait Sarper Özaslan

22002861

Part 1

Nothing needs to be provided for this part.

Part 2

Here is the derivation of convolution in 2D DS LSI System:

Let's essentially replicate the 1D case for this

we can write $x[m,n]$ as a product of impulse train where,

$$x[m,n] = \sum_{k=-\infty}^{\infty} \sum_{\ell=-\infty}^{\infty} x[k,\ell]\ \delta[m-k, n-\ell] \qquad (1)$$

In which impulse $\delta[m-k, n-\ell]$ is defined as

$$= \begin{cases} 1 & , m=k, n=\ell \\ 0 & \text{Otherwise (a.w.)} \end{cases}$$

To find or rather define $h[m,n]$, we can say that it is the response of DS LSI systems to the impulse response

$$\delta[m,n] \longrightarrow \boxed{\begin{matrix}DS\\LSI\end{matrix}} \longrightarrow h[m,n]$$

As this system is Linear and space invariant we can write it as:

$$a\,\delta[m-\gamma, n-\gamma_0] \longrightarrow \boxed{\begin{matrix}DS\\LSI\end{matrix}} \longrightarrow a\,h[m-x_0, n-y_0]$$

In such case we can say that if input is $x[m,n]$, as in (1) to an DS LSI system the output $y[m,n]$ will be equal to:

$$y[m,n] = \sum_{k=-\infty}^{\infty} \sum_{\ell=-\infty}^{\infty} x[k,\ell]\, h[m-k, n-\ell]$$

This is the convolution of $y[m,n] = x[m,n] * h[m,n]$

we can switch up the order from commutative by of convolution

$$y[m,n] = \sum_{k=-\infty}^{\infty} \sum_{\ell=-\infty}^{\infty} x[m-k, n-\ell]\, h[k,\ell]$$

Figure 1: Derivation of 2D Convolution

Part 3:

Here is the relation between y[m,n]'s matrix sizes My, Ny in terms of Mx, Nx and Mh, Nh which are respectively x[m,n] and h[m,n]:

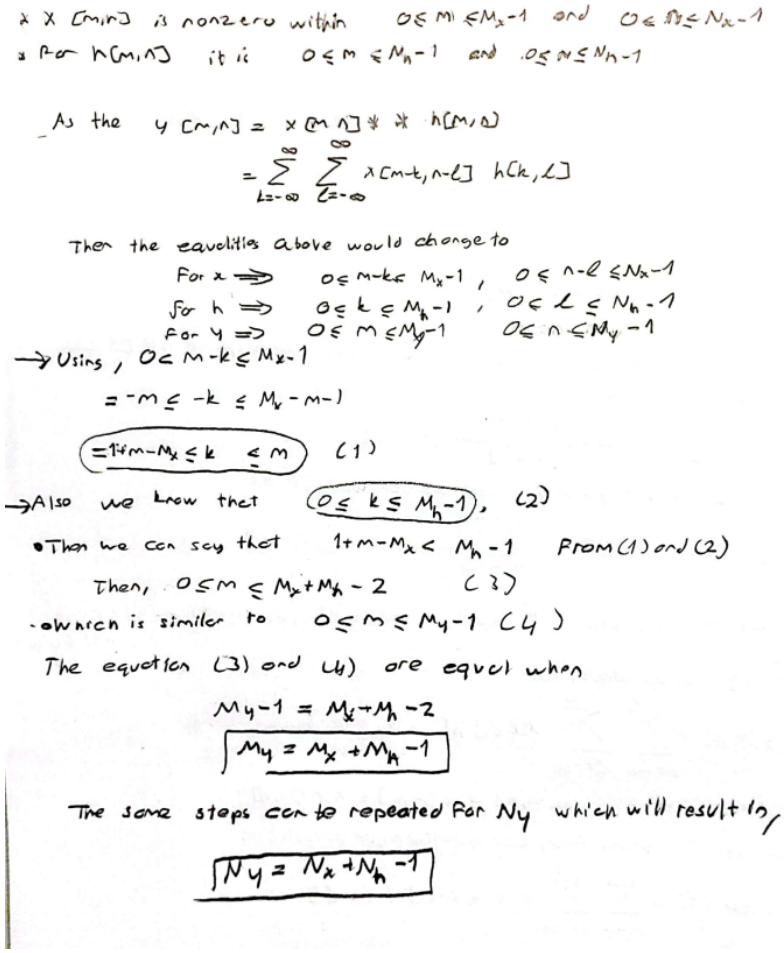

Figure 2: Relation of Ny and My with Other Terms

Here is the code for test of part 3:

```
x = [1 0 2; -1 3 1;-2 4 0];

h= [1 -1 ;0 2];

[Mx, Nx]= size(x);

[Mh, Nh]=size(h);

y = zeros(Mh-1+Mx,Nh-1+Nx);

for k=0:Mh-1

        for l=0:Nh-1
```

```
            y(k+1:k+Mx,l+1:l+Nx)=y(k+1:k+Mx,l+1:l+Nx)+h(k+1,l+1)*x;
        end
end
```

Here is the result according to specified values:

```
x =

     1     0     2
    -1     3     1
    -2     4     0

>> h

h =

     1    -1
     0     2

>> y

y =

     1    -1     2    -2
    -1     6    -2     3
    -2     4     2     2
     0    -4     8     0
```

Figure 3: Result of convolution of x[m,n] and h [m,n]

Here is the function itself for reference:

```
function y = DSLSI2D(h,x)
[Mx, Nx]= size(x);
[Mh, Nh]=size(h);
y = zeros(Mh-1+Mx,Nh-1+Nx);
for k=0:Mh-1
        for l=0:Nh-1
```

```
            y(k+1:k+Mx,l+1:l+Nx)=y(k+1:k+Mx,l+1:l+Nx)+h(k+1,l+1)*x;
        end
end
```

In this part, we essentially used the relation we found at the beginning of part 3 which can be seen in the figure.

Part 4 – Image Denoising

Important part!! For this part to work more efficiently I changed “DisplaymyImage” function’s line 3 to “Figure (1)” from “Figure”.

As a general sinc function has most of its values mostly condensed around point 0 we can say that it is suitable choice as a low pass filter. The constant “B”s effect is important as it works as the determination for bandwidth of the lowpass filter. This can be seen in the figure below.

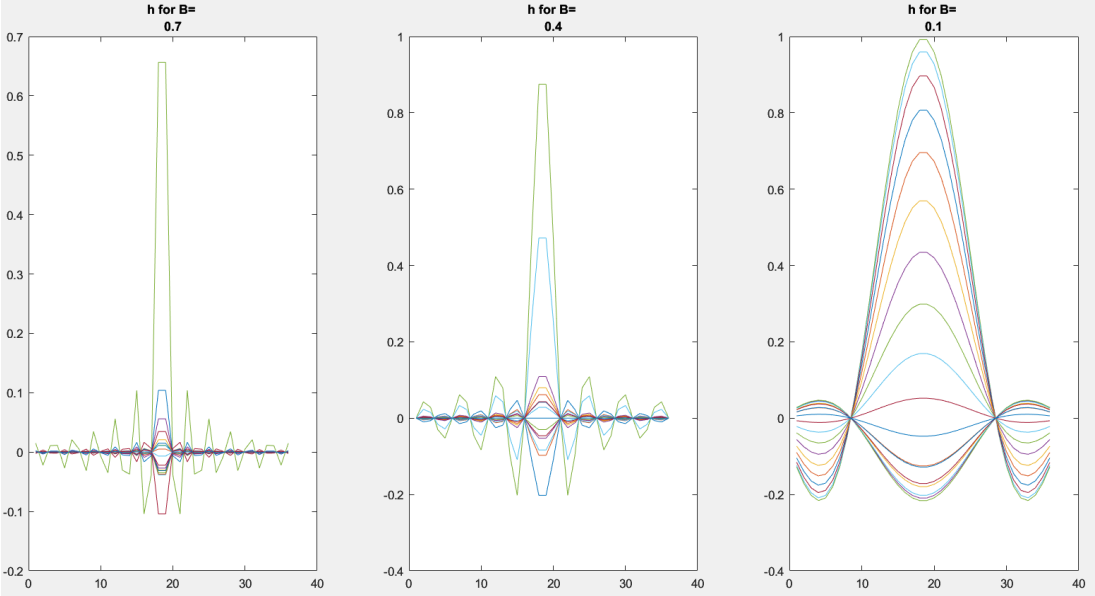

Figure 4: Different Bandwidths for Different B values

As B’s value gets higher, we can see that we get a sharper figure. In other words, the bandwidth of our filter gets smaller. Moreover, the values further than the central point of LPF, which is 0 in this case, gets closer to 0 faster allowing us to kill high frequencies faster.

One loss in this case is if the image that we are trying to display has frequencies that are on the outside of the bandwidth it is likely there will be a loss on the resolution of the image. Therefore, one can say that B = 0.4 is more likely to give a higher resolution image compared to B = 0.1 and B= 0.7 where one has a wide bandwidth and the other has a narrow bandwidth, respectively.

After observing this point here is the code and plot for this part:

```
x=ReadMyImage('Part4.bmp');
D7= rem(22002861,7); %mod7 part= 6
Mh= 30+ D7;
Nh= 30+ D7;
subplot(2,2,1), hold on, title("Original Figure(x)"),DisplayMyImage(x),
Bi=[0.7 0.4 0.1];
% We form the plots with 3 iterations here anda calculate h for different B values
for a=1:3
        B= Bi(a);
        for m=0:Mh-1
                for n=0:Nh-1
                h(m+1,n+1)=sinc(B*(m-(Mh-1)/2)) * sinc(B*(n-(Nh-1)/2));
                end
        end
        y=DSLSI2D(x,h);
        str1= ["Response when B= " num2str(B)];
        subplot(2,2,a+1),hold on, title(str1),DisplayMyImage(y)
end
```

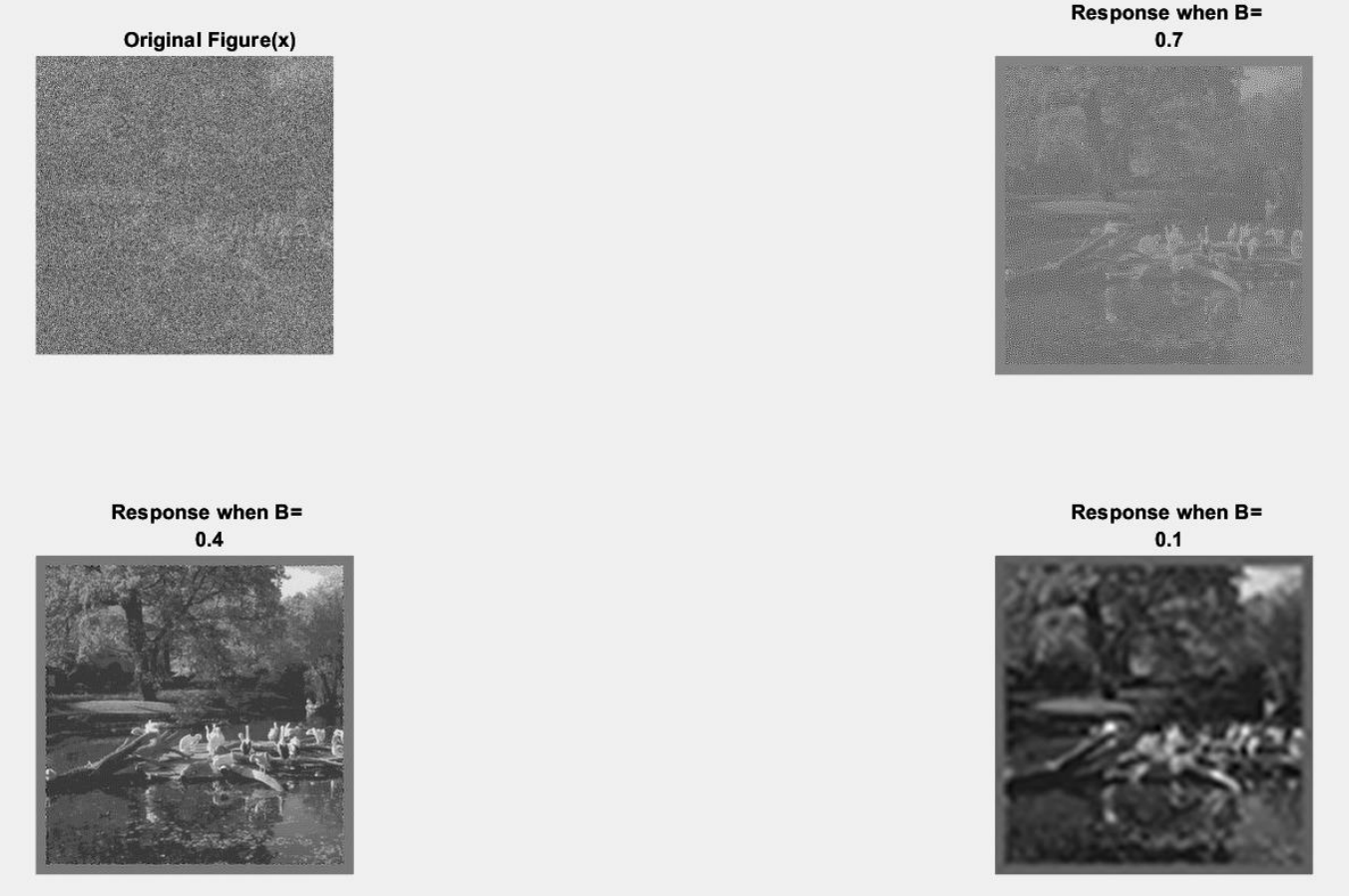

Figure 5: Comparison for different B Values and Original Image

As expected, the clearest image occurred when B = 0.4. This means the frequency content of this image is most suited to this filter. The difference between B = 0.1 and B = 0.7 can be expressed as blur and noise. On B = 0.7 some important low frequency data are cut off which causes a noisy image. In B = 0.1, there is a blur, or rather a noise caused by the high frequency signals.

Part 5 – Edge Detection

In this part we are asked to take vertically, horizontally and combination of both. Here is the h1 part:

Figure 6: Overlap between Vertical Details and Original Image

Code:

```
x=ReadMyImage("Part5.bmp");
subplot(1,2,1),DisplayMyImage(x),title("Original Image");
h1= [0.5 -0.5];
y1= DSLSI2D(h1,x);
s1= y1.*y1;
subplot(1,2,2), DisplayMyImage(s1), title("Vertical details")
```

Essentially the vertical filter we apply allows us to heighten the vertical edges. When we apply an edge detection kernel to a low frequency image the sum of gradual change results in a small value hence a blackened area. The edges which have high frequencies in such case results in a larger value compared to low frequency area which results in lighter ”white” area.

Moving on to h2 part:

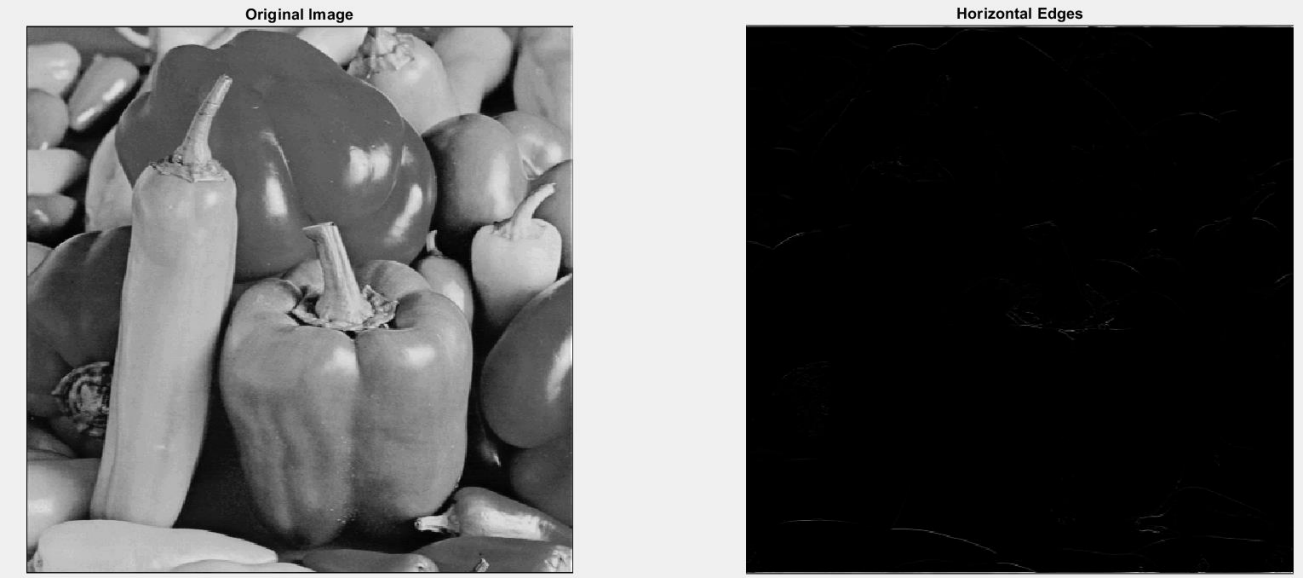

Figure 7: Overlap Between Horizontal Edges and Original Image

Code:

```
x=ReadMyImage("Part5.bmp");
subplot(1,2,1),DisplayMyImage(x),title("Original Image");
h2 = [0.5; -0.5];
y2 = DSLSI2D(h2,x);
s2 = y2.*y2;
subplot(1,2,2),DisplayMyImage(s2), title("Horizontal Edges")
```

Explanation is the same for the most part but the only difference we have in this case the edge detection kernel we use is horizontal which allows one to see horizontal edges in the image.

Moving on to h3 part:

Figure 8: Overlap Between Edges in Both Direction and Original Image

Code:

```
x=ReadMyImage("Part5.bmp");
subplot(1,2,1),DisplayMyImage(x),title("Original Image");
h3= 0.5*h2 + 0.5*h1;
y3 = DSLSI2D(h3,x);
s3=y3.*y3;
subplot(1,2,2),DisplayMyImage(s3), title("Both Edges")
```

h3 is sum of horizontal and vertical kernel and has the values 0.5 and –0.5 respectively in its diagonal points. As a result, convolution sum of this kernel with image x will allow us to see edges in both directions.

Part 6 – Pattern Recognition

In this part we use an image of Turkish National Football team and inverted face to make a pattern recognition test. Taking convolution sum and the absolute value for different powers we get:

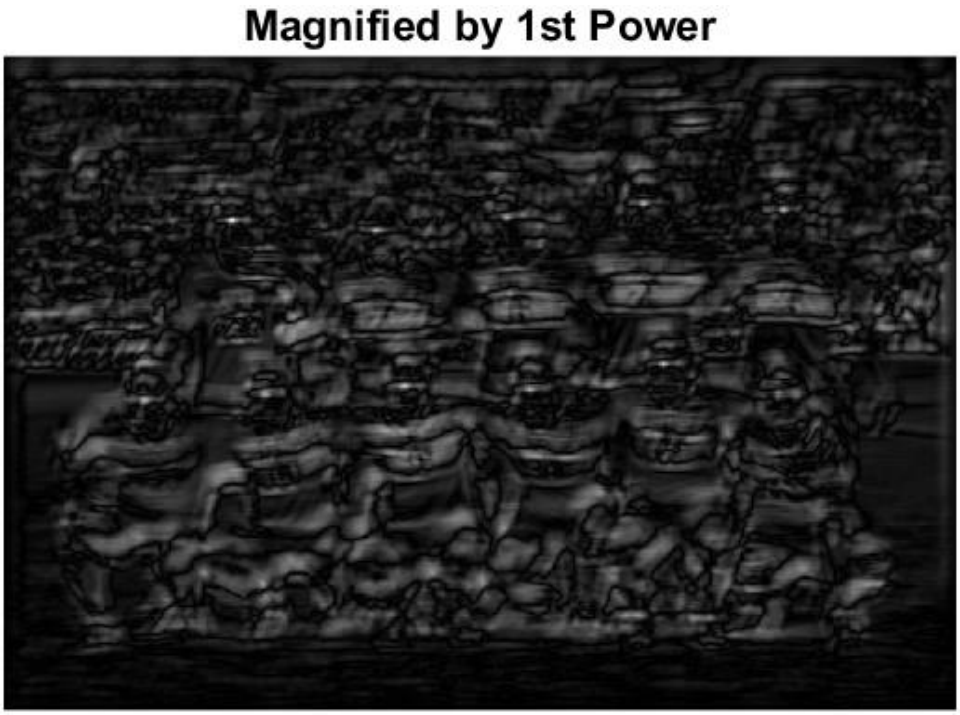

Figure 9: Image of abs(y)

It can be clearly seen that faces can be included as bright points but there are other bright points as well. This means there are some overlaps between those points and the filter we applied which is the inverted face in this case.

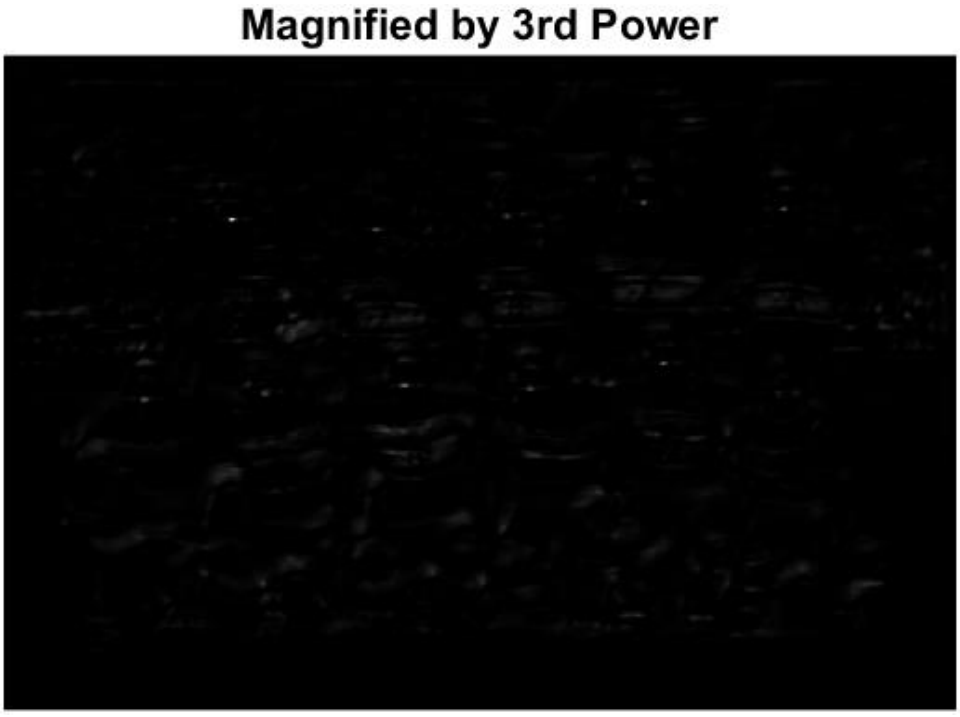

Figure 10: Image of abs(y).^3

When we take the third power of abs(y) essentially, we will be eliminating the bright points whose values in the matrix of y is between 0 and 1 and amplifying the ones whose values are higher than 1. As a result, we expect the brightest points to be around the faces. This can be seen from the visible bright dots from the figure.

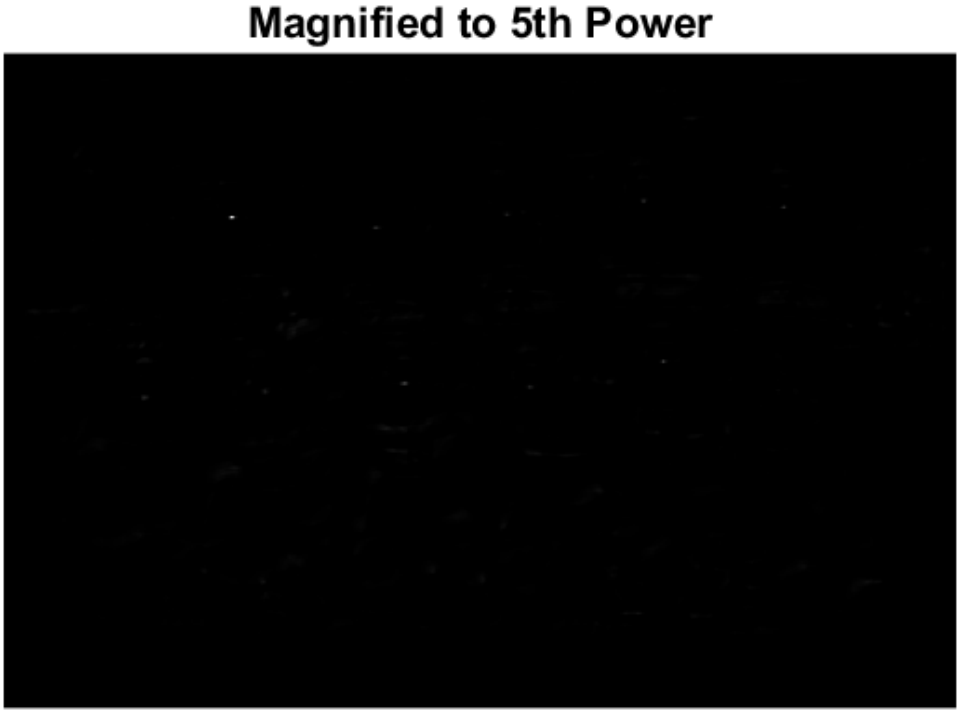

Figure 11: Image of abs(y).^5

In this case of n, the values of y which increases the most by the fifth power is displayed mostly as the other values essentially fall short. Hence even though we can see a few bright points around where the faces should be in the figure it is harder to spot them.

Therefore, I believe the image of abs(y).^3 is the most accurate depiction to spot the faces as there are fewer errors and faces are more visible compared to its fifth power.