

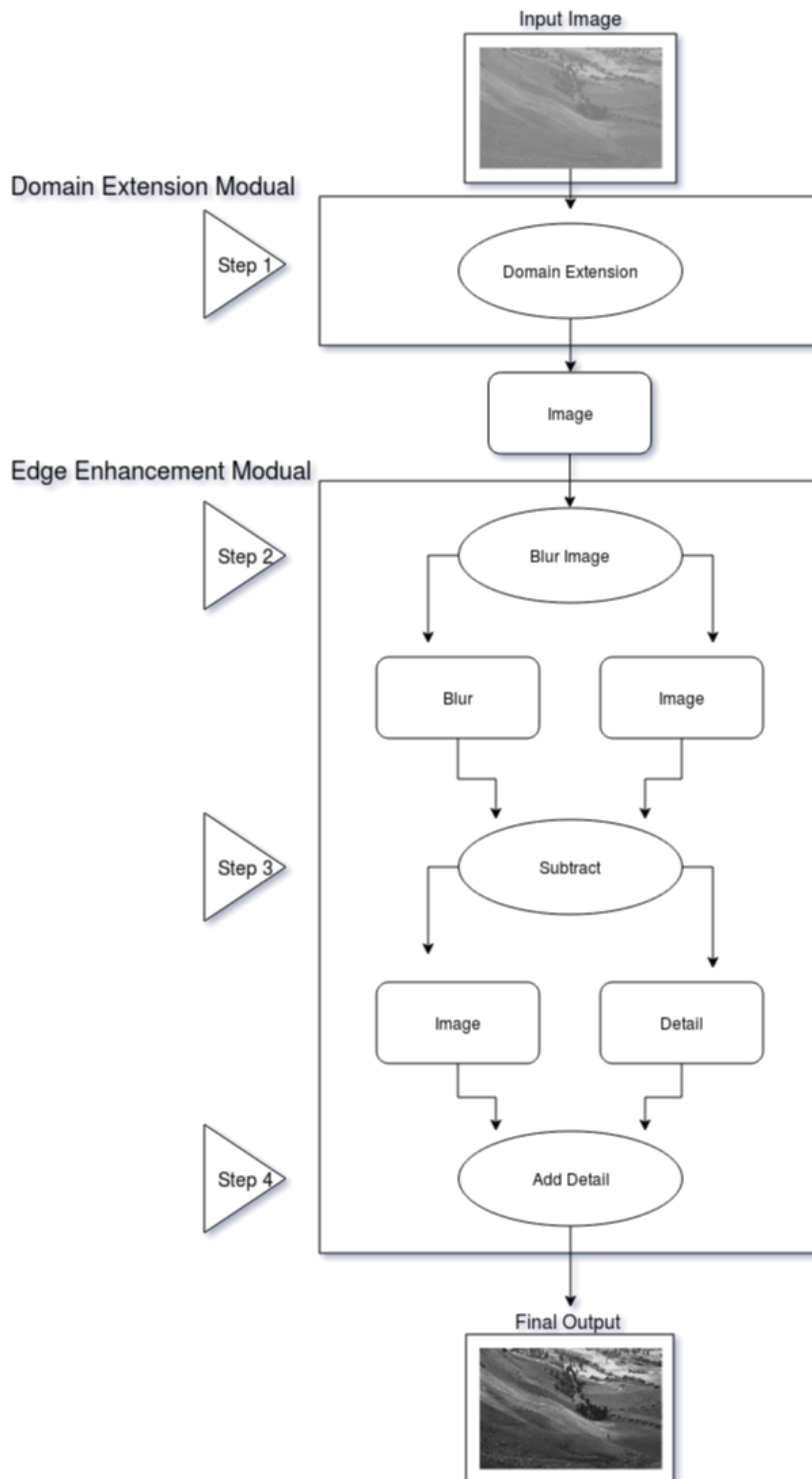
# Photo Enhance Using High School Math

Aidan Coopman

March 14, 2019

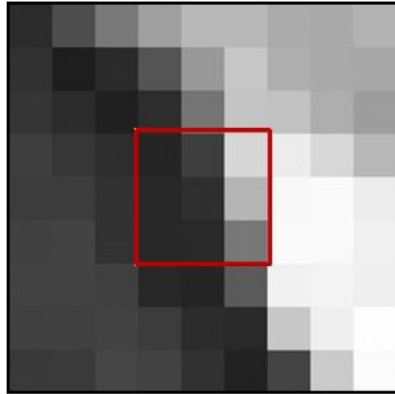
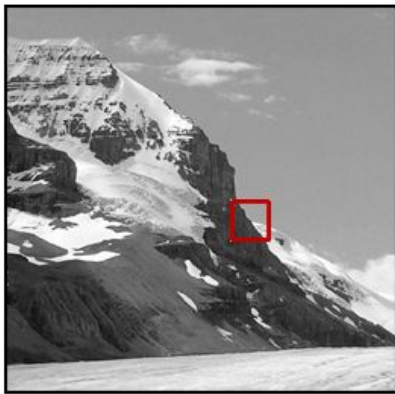
## Introduction:

From a young age, I have always been interested in photography. Taking photos is one thing, but editing them is another. I used programs such as Adobe Photoshop to edit and enhance my photos and I have always wondered about the technology of the software. I have been involved with computer science for a few years and decided to figure out if I could make a program that does the same thing that Photoshop does. I did some extensive research and found some great examples of how other people do photo enhancement, but I wanted to make a program on my own, not using someone else's work. Below is the outline/pipeline I used to enhance an image and each step will be described in detail.



## Understanding an Image:

A standard image is represented by pixels or bytes which has the range of 0 to 255. 255 is white and 0 is black. Looking at the image below, a 3x3 square is taken from the image on the left and blown up. The middle image shows the zoomed in red square from the far left image. The far right image shows the pixel values of the 3x3 square. These numbers represent the pixel values. As one can see the leftmost column has the lowest pixel values, and in the image the darkest pixels. As one moves further to the right, the pixel values increase. When the pixel values increase, one can see that the image has more white pixels.



43	102	169
35	58	191
38	44	155

In the next few pages, I will describe the step-by-step procedure of photo enhance.

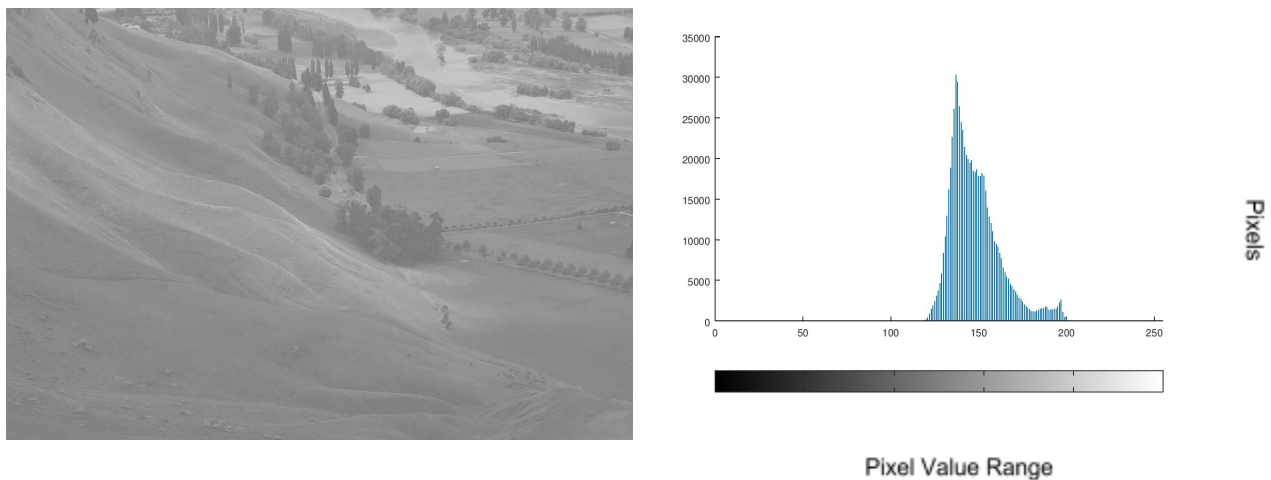
## Pixel Value Extension Module:

In this module, the goal is to extend the pixel value (intensity) range so the image covers the whole range of pixel values. The human eye sees contrast, which is the difference in pixel values. The way to maximize the differences is to make sure the pixels use the full spectrum of values, between 0 and 255.

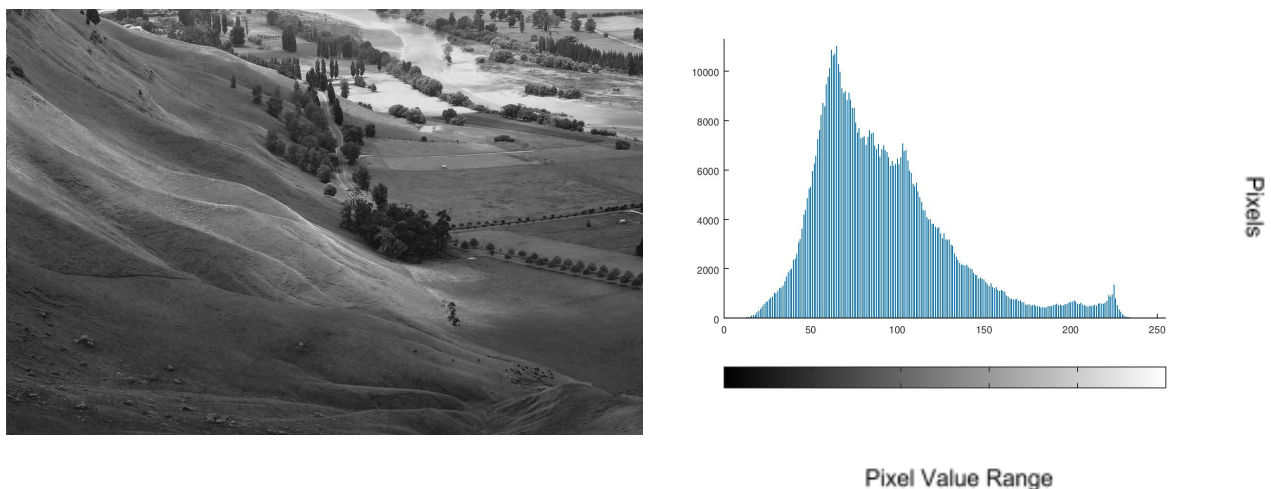
### Step 1: Pixel Value Extension

The first step in enhancing an image is a pixel value extension. This spreads out the pixel values to cover the whole spectrum from values 0 to 255. The images below have a histogram of pixel intensity values. The first image has a minimum pixel values of 114, and a maximum of 208. The maximum intensity difference is  $208 - 114 = 94$ . The pixel values can be between 0 and 255, making for a maximum intensity difference of 255. One could say that the first image only uses  $94/255 = 37\%$  of the visual spectrum.

**Before pixel value extension**



**After pixel value extension**



**Pixel value extension is done by:**

- 1) Find the minimum and maximum pixel values from the input image
- 2) Subtract the minimum from each pixel of the image. This shifts the spectrum over so it starts at zero
- 3) Find the scale factor to multiply the pixels by. The max value (after the above shift) times the scale factor has to equal 255. Hence the scale factor =  $255/\text{max}$
- 4) Multiply all the pixels values by the scale factor

## Code For Pixel Value Extension

Below is the code for the pixel value extension algorithm:

```
void pixel_value_pixels(uint8_t * image, int height, int width)
{
    int min=256;
    int max=0;

    //STEP 1
    for (int i = 0; i<(height*width); i++)
    {
        if(image[i] < min)
            min = image[i];

        if(image[i] > max)
            max = image[i];
    }

    //STEP 2
    for (int i = 0; i<(height*width); i++)
        image[i] -= min;

    float range=max-min;

    //STEP 3
    float scale = 255.0/new_max;

    //STEP 4
    for (int i = 0; i<(height*width); i++)
        image[i] = image[i]*scale;
}
```

## Edge Enhancement Module:

In this module the image is enhanced by means of blurring and subtracting images.

### Step 2: Blur the image

The first step in enhancing the edges was to get rid of the edges by blurring the image. This blurring of the image is done by using a Gaussian blur. This sounds counter-intuitive but when there are no edges or detail in the image, one can then subtract the images to determine where the details are in the image.

To create the Gaussian 3x3 kernel, this equation was used:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)}$$

$G(-1,-1)$	$G(-1,0)$	$G(-1,1)$
$G(0,-1)$	$G(0,0)$	$G(0,1)$
$G(1,-1)$	$G(1,0)$	$G(1,1)$

=

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

=
(1/16) •

1	2	1
2	4	2
1	2	1

0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

\*


1	0	1
0	1	0
1	0	1

=

1	4	3	4	1
1	2	4	3	3
1	2	3	4	1
1	3	3	1	1
3	3	1	1	0

**I**
**K**
**I \* K**


Here's an example of a simple blur



$\frac{1}{9}$ 

•1	•1	•1
•1	•1	•1
•1	•1	•1

=



Original
Blur (with a box filter)

## Blurred image:

The photo on the left is the original image with the pixel value extension. The right image is the blurred version of the left image.

Original image



Blurred Image



## Code for blurring the image

```
void blur_image(uint8_t * image, uint8_t * out_image, int height, int width)
{
    //first row
    int a,b,c;
    int aa,bb,cc;
    int aaa,bbb,ccc;

    for (int i = 0; i < height-3; i++)
    {
        for( int j =0; j< width-3; j++)
        {
            //this is first row of kernel
            a = image[j+i*width];
            b = image[j+i*width+1];
            c = image[j+i*width+2];
            //2nd
            aa = image[j+(i+1)*width];
            bb = image[j+(i+1)*width+1];
            cc = image[j+(i+1)*width+2];
            //3rd
            aaa = image[j+(i+2)*width];
            bbb = image[j+(i+2)*width+1];
            ccc = image[j+(i+2)*width+2];

            //gaussian filter
            out_image[j+i*width]=(a+2*b+c+2*aa+4*bb+2*cc+aaa+2*bbb+ccc)/16;

        }
    }
}
```

### Step 3: Subtract the blur from the image to get detail

After obtaining both the pixel value extended image and the blurred image, the next step was to subtract the pixel value extended image with the blurred image. This provides the detail in the image.

**Original(Pixel Value Extended) Image**



-

**Blurred Image**



=

**Detail**





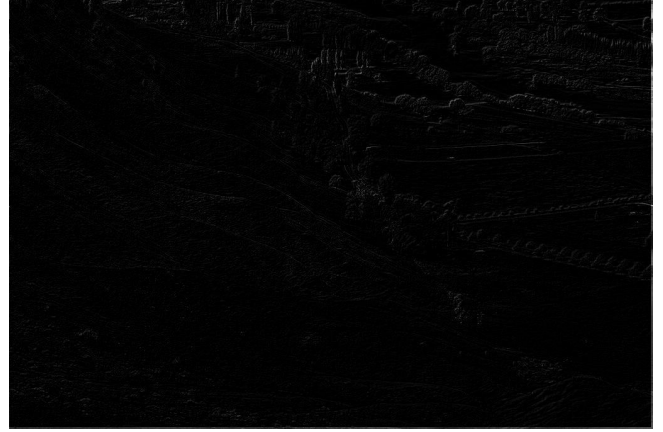
#### Step 4: Add the detail to the image

The next step in enhancing the image is to add the detail into the image. This was done by combining the pixel value extended image and the detail that was obtained in the last step.

Original Image



Detail



+

=

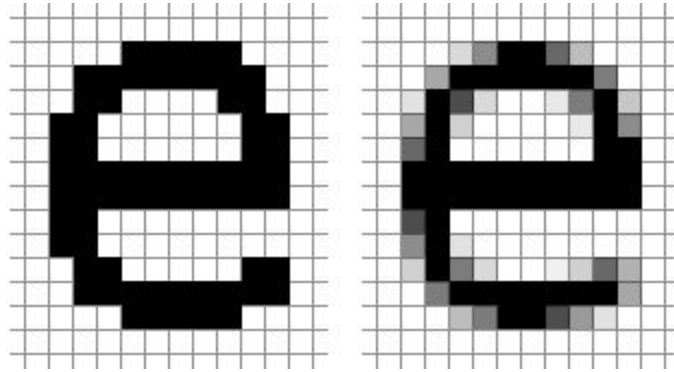
Edge Enhanced Image





## Measuring Detail in an Image:

As stated before, the human eye sees contrast which is essentially the difference in pixel values. The images below show a letter blown up in size and one can see that the left image is more sharp than the right image. This is because it has a larger pixel value difference between neighboring pixels. The image on the left has pure white pixels(pixel value = 255) directly beside black pixels(pixel value = 0) making the difference between neighboring pixels equal to 255(255-0=255). The image on the right has black, grey, then white. This makes the difference between neighboring pixels smaller around 255-128 = 127.



Comparing the difference between neighboring pixels is just a derivative. Hence, a way of measuring sharpness in an image is to take the derivative in the x and y direction and find the maximum value of the derivative.

$$f'(x,y) = f'(x) + f'(y)$$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$$f'(y) = \lim_{h \rightarrow 0} \frac{f(y+h) - f(y)}{h}$$

Applying this sharpness measure, the original image before edge enhancement was 361. The sharpness after being edge enhanced was 478.

**Original Image (Sharpness = 361)**



**Edge Enhanced Image (Sharpness = 478)**



### Code to find sharpness:

```
int find_sharpness(uint8_t * image, int height, int width)
{
    int a;
    int b;
    int sharp;
    int max_sharp_x=-255;
    int max_sharp_y=-255;
    int sharpness;

    //find partial derivative for x
    for (int i = 0; i < height; i++)
    {
        for( int j =0; j< (width-1); j++)
        {
            a= image[j  +i*width];
            b= image[j+1 +i*width];
            sharp = b-a;
            if(abs(sharp) > max_sharp_x)
                max_sharp_x = abs(sharp);
        }
    }

    //find partial derivative for y
    for (int i = 0; i < height-1; i++)
    {
        for( int j =0; j< width; j++)
        {
            a= image[j+i*width];
            b= image[j+(i+1)*width];
            sharp = b-a;
            if(abs(sharp) > max_sharp_y)
                max_sharp_y = abs(sharp);
        }
    }

    //add the 2 partial derivatives
    sharpness = max_sharp_x + max_sharp_y;

    return sharpness;
}
```

# Conclusion

The goal of this study was to be able to enhance images using high school math. Five photos have been run through the program and data was recorded on the table below. The results show that there was a large increase in pixel range and sharpness in all images, which proves the program works. Attached below are images for comparing the difference using ones eyes.

The code for this project can be found online at:  
<https://github.com/acoopman/PhotoEnhance>

Before Enhancement					After Enhancement			
Photo	Min pixel	Max pixel	Pixel range	Sharpness	Min pixel	Max pixel	Pixel range	Sharpness
TestInput.jpg	114	208	94	361	0	255	255	478
girl.jpg	34	255	221	251	0	255	255	375
underwater.jpg	19	254	235	230	0	255	255	350
eyeball.jpg	20	255	235	243	0	255	255	410
car.png	93	185	92	367	0	255	255	456





## **Bibliography:**

[http://gsp.humboldt.edu/OLM/Courses/GSP\\_216\\_Online/lesson3-1/raster-models.html](http://gsp.humboldt.edu/OLM/Courses/GSP_216_Online/lesson3-1/raster-models.html)

[https://www.researchgate.net/figure/An-example-of-convolution-operation-in-2D-2\\_fig3\\_324165524](https://www.researchgate.net/figure/An-example-of-convolution-operation-in-2D-2_fig3_324165524)

<http://ai.stanford.edu/~syueung/cvweb/tutorial1.html>

<http://annystudio.com/misc/anti-aliased-fonts-hurt/>