DIP HW2 Report

電子所 郭家均 313510171

Low-luminosity Enhancement

1. Preprocessing

To avoid color shift in the image, I transfer the RGB channel to YCbCr channel to process the image, and I only do the enhancement on the luma channel Y.

$$Y = 0.299R + 0.587G + 0.114B$$
$$Cb = 0.568(B\text{-}Y) + 128 = -0.172R - 0.339G + 0.511B + 128$$
$$Cr = 0.713(R\text{-}Y) + 128 = 0.511R - 0.428G - 0.083B + 128$$

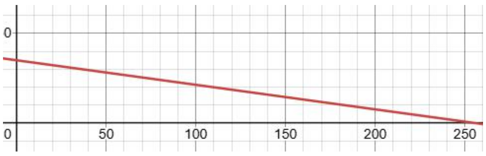
$$R = Y + 1.371(Cr - 128)$$
$$G = Y - 0.698(Cr - 128) - 0.336(Cb - 128)$$
$$B = Y + 1.732(Cb - 128)$$

The transferring steps involve multiplying the RGB value with a transfer matrix. To avoid overflow in the pixel value, I use a clamp function to limit the pixel value.

```
double IMAGE::Clamp(double d, double low, double high) {
    if(d < low) return low;
    if(d > high) return high;
    return d;
}
```

2. Linear-Scaled constant enhancement

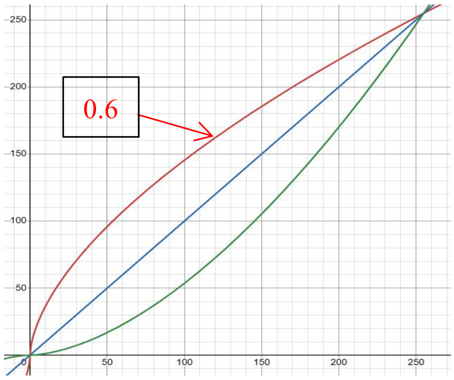
```
void IMAGE::EnhanceLuma() {
    for(int i = 0; i < H; i++) {
        for(int j = 0; j < W; j++) {
            pixel[i][j].Y += 35 * (255-pixel[i][j].Y)/255;
        }
    }
}
```



With the original pixel's Y adding 35*(255-Y)/255 constant, the darker part will add more value, and the lighter part will add less value. The plot above shows the constant value from 0 to 255.

3. Gamma correction (gamme = 0.6)

```
void IMAGE::EnhanceLuma(double gamma) {
    for(int i = 0; i < H; i++) {
        for(int j = 0; j < W; j++) {
            pixel[i][j].Y = 255 * pow(pixel[i][j].Y / 255, gamma);
        }
    }
    To_RGB();
}
```



With original Y = 255*pow(Y/255, gamma), the darker area will stretch to a lighter part, with lighter area suppressed.

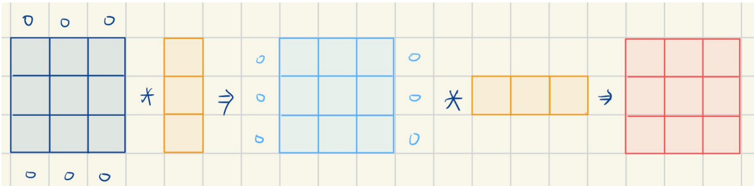
Linear-Scaled constant Enhancement	Gamma correction (0.6)
	

Gamma correction seems to enhance even better, so I choose gamma correction in the end.

Sharpness Enhancement

1. Low-pass filter: Gaussian smoothing

We can think of the image as a signal, after passing through LPF the high-frequency noise or transition in an image will be reduced. Since Gaussian smoothing is separable, I implement it in 2 times 1D convolution with zero padding.



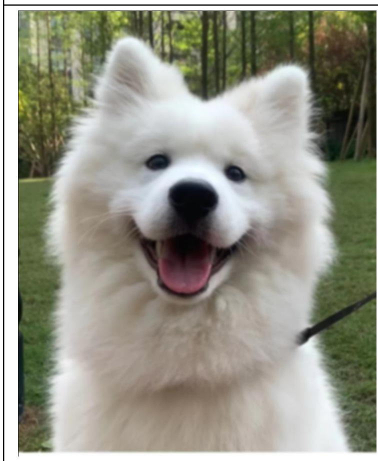
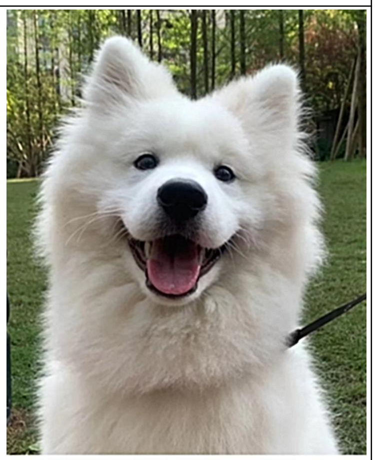
2. Obtain high-frequency element

We simply subtract the original image with LPF image. To observe the high frequency element in the photo, I scale the high frequency value below. We can clearly see wherever a great intensity transition on the left hand side, the right hand side will show white lines indicate high frequency element exist there.

Original Gray-Scale image	Image through HPF
	

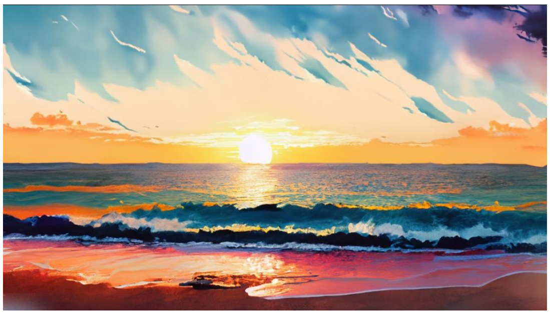
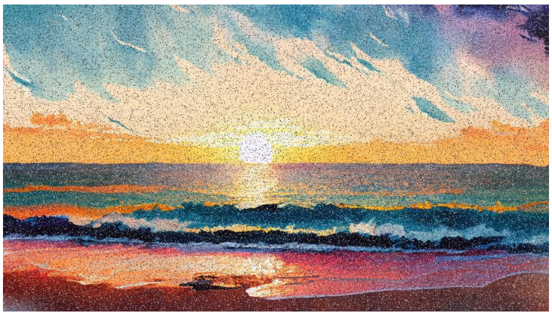
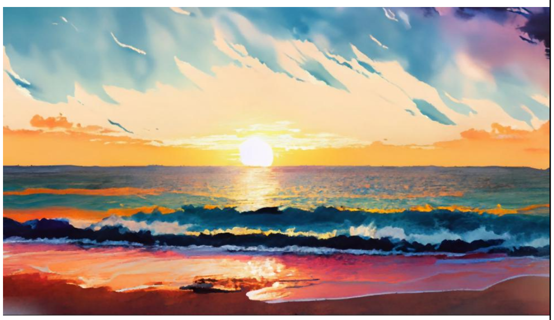
3. Add scaled high frequency details onto original image

With Y = Y + level * (High frequency element) formula, we can enhance the image overall sharpness. I pick level = 5 and 10 to showcase different level of sharpness enhancement below.

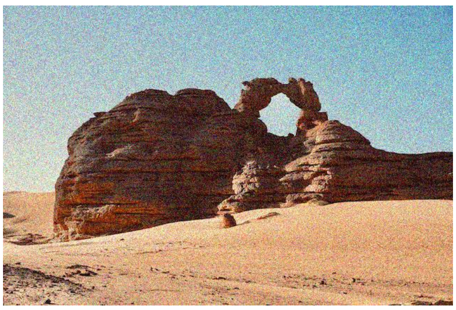
Original Image	Enhance with level = 5	Enhance with level = 10
		

Denoise Salt-and-pepper noise / Gaussian noise

For the first image, I use median filter(3*3) to remove salt and pepper noise. The overall logic is to sort out pixel value in a region, then pick the median to avoid extreme value of the pixel. Furthermore, with a little bit of sharpness enhancement, the SSIM value will increase.

Ground truth	Noisy image
	
Median filter (SSIM 0.947512)	Median w/ sharpness (SSIM 0.948055)
	

For the Gaussian noise image, I use the Gaussian smoothing filter on it, with the target being only Y channel or separately on the RGB channel. After applying Gaussian smoothing filter, I add two additional median filter (3*3) for higher SSIM. We'll discuss the image quality next page.

Ground truth	Noisy image
	
Gaussian smooth on Y (SSIM 0.77242)	Gaussian smooth on RGB (SSIM 0.77236)
	

Further Discussion

SSIM evaluation

Since we evaluate image only on Gray scale channel, the implementation is as such:
Get the mean, variance, and covariance of the channel pixel. Then simply use the given formula.

```cpp
double C1 = 6.5025, C2 = 58.5225;

double mean1 = computeMean(img1, x, y, patchSize);
double mean2 = computeMean(img2, x, y, patchSize);

double variance1 = computeVariance(img1, x, y, patchSize, mean1);
double variance2 = computeVariance(img2, x, y, patchSize, mean2);

double covariance = computeCovariance(img1, img2, x, y, patchSize, mean1, mean2);

return ((2 * mean1 * mean2 + C1) * (2 * covariance + C2)) /
        ((mean1 * mean1 + mean2 * mean2 + C1) * (variance1 + variance2 + C2));
```

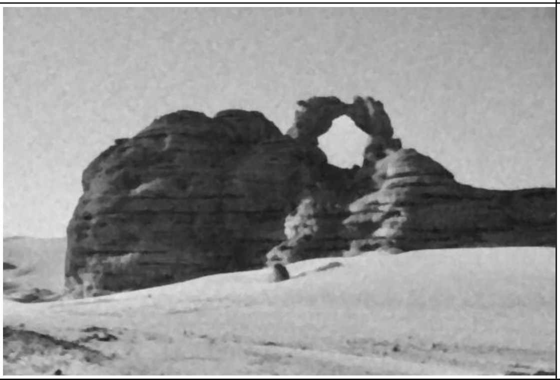
```cpp
// Slide the window over the image
for (int y = 0; y <= H - patchSize; y += patchSize) {
    for (int x = 0; x <= W - patchSize; x += patchSize) {
        // Calculate SSIM for Y channel and sum up
        ssimSum += calculateSSIMForY(pixel, origin, x, y, patchSize);
        ++numPatches;
    }
}

// Average SSIM for Y channel
double ssimY = ssimSum / numPatches;

// Output SSIM score
std::cout << "-- SSIM Score (Y channel): " << ssimY << std::endl;
```

Color blob in Y-channel only Gaussian smoothing

In the denoise section, we can see the image quality of applying Gaussian filter only on Y channel (case1) is worse than apply it to RGB channel separately (case2). The reason is the noise in RGB channels are independent. In the Luma channel where we evaluate the SSIM score, we can see that case1 is indeed smooth out in the Y channel with better contrast than case2.

Gray-Scale Y channel Gaussian smoothing	Gray-Scale RGB channel Gaussian smoothing
	

If we use SSIM evaluate on RGB channel the result will be different, with the case1 worse than case2. Since case2 treated noise in RGB channel independently.

Zoom-in Y channel Gaussian smoothing	Zoom-in RGB channel Gaussian smoothing