Report

Student Name: Zhiqiang Cheng

Student Number: 22322639

Content

Report....1
Part1:....2
- Histogram Equalization....2
- Gamma Transformation....4
- Log Transformation....5
- Linear Transformation....7
- Grayscale Grading....9
- Conclusion....10

Part2....11
- Gaussian Filter....11
- Mean Filtering....13
- Medium Filtering....14
- Conclusion for Blur....15
- Laplace Filter....15
- USM....17
- Conclusion for Sharpening....18

Part3....19
- BiBubic....19
- Nearest Neighbor....21
- Bilinear Interpolation....22
- Max Pooling....26
- Mean Pooling....27
- Conclusion....29

Part1:

Histogram Equalization

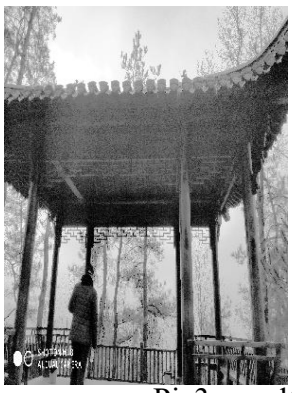
Pic1

Pic2

Pic3

Pic1_result

Pic2_result

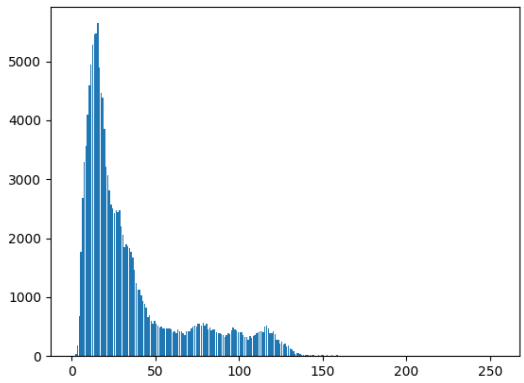


Pic3_result

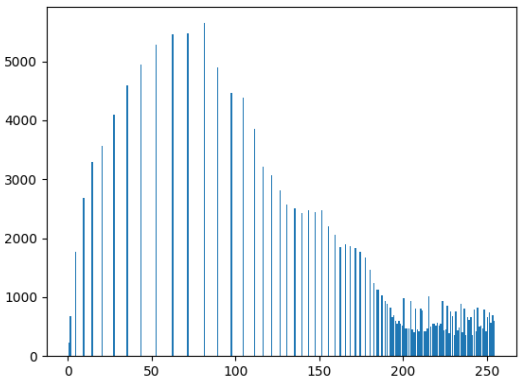
Histogram equalization refers to adjusting the dark, bright or concentrated gray histogram of an image to a histogram with uniform gray distribution to enhance image contrast.

After histogram equalization of the three figures, it can be observed that the heads of the characters in Pic1 that cannot be seen due to low brightness become visible, the features of the same characters in Pic2 become clearer, and the structure of the roof that cannot be seen due to low brightness can be seen in Pic3.

Take Pic2 as an example, its original gray level histogram has the following changes after equalization:



Original Figure



Figure

The abscissa of the gray level histogram is the gray level, and the ordinate is the occurrence rate of the gray level. The original grayscale image shows that the image is dark and the image brightness is too concentrated.

In order to make the image uniform, its histogram distribution needs to be approximately uniform through the cumulative distribution function. In order to expand the brightness range of the original image, a mapping function is needed to evenly map the pixel values of the original image to the new histogram. At the same time, this mapping function cannot change the size order of the original pixel values, and the value range of the pixel mapping function must be between 0 and 255.

The cumulative distribution function is a monotone increasing function with a range of 0 to 1, which formula is:

$$S_k = \sum_{j=0}^{k} \frac{n_j}{n} \quad , k = 0,1,2, \dots , L-1$$

This function is implemented by code in the form of:

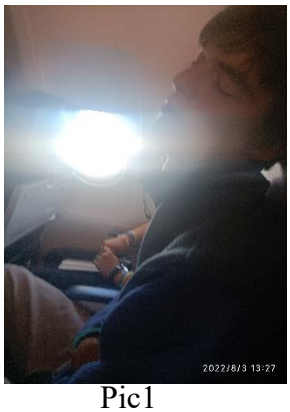
```
H, W = img.shape
    S = H * W * 1.
```

First, convert the total pixels of the image to Calculate the total number of pixels of the original image in the float form and calculate the percentage of the number of pixels of each gray level in the whole image. Specify sum_h start from 0, calculate the cumulative distribution of each gray level (i=0~255) of the image, and finally calculate the gray value result of the new image.

```
    for i in range(1, 255):
```

```
            ind = np.where(img == i)
            sum_h += len(img[ind])
            z_prime = z_max / S * sum_h
            result[ind] = z_prime

    result = result.astype(np.uint8)
```

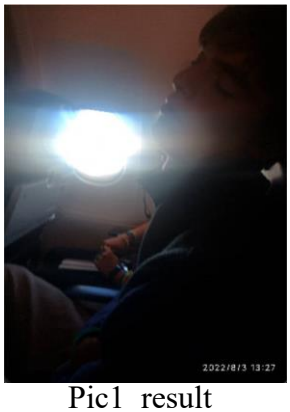
Gamma Transformation



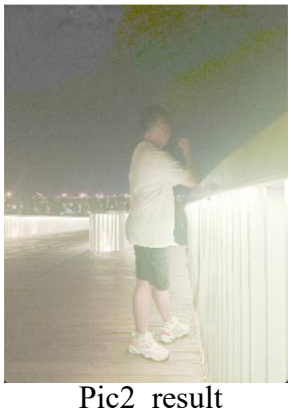
Pic1


Pic2


Pic3



Pic1_result


Pic2_result

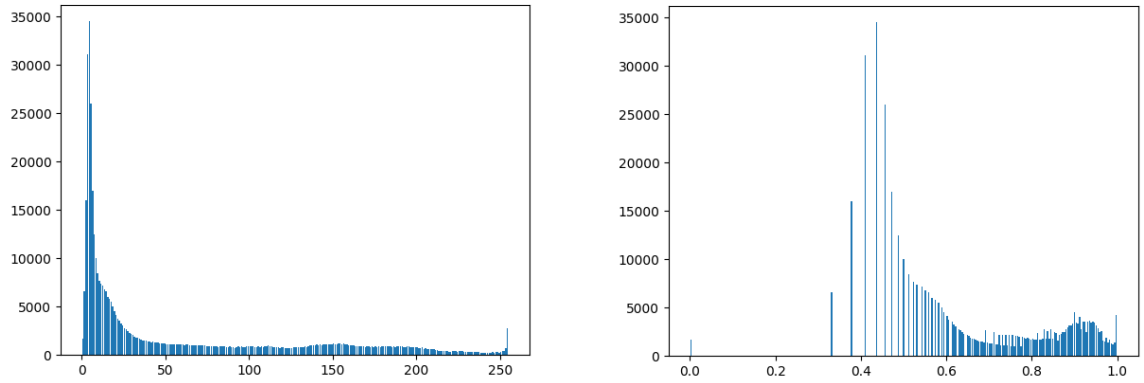

Pic3_result

From the above picture, it can be seen that after the gamma transformation of Pic1, the range of too bright light sources is reduced, the overall brightness is reduced, and the dim characters in Pic2 become bright, so that the upper body of the characters can be observed. The illumination of the brighter lights in Pic3 is reduced, which is more consistent with the normal brightness at night.

Gamma transform is mainly used for image correction. The gray value of darker areas

in the image can be enhanced by correcting the bleached image or the image that is too dark, and the gray value of the area with too large gray value in the image can be reduced.

Take Pic2 as an example, its original gray level histogram has the following changes after Gamma Transformation:



It can be observed from the above figure that the brightness of the grayscale image after gamma transformation is higher and the brightness concentration is reduced.

The Function of Gamma Transformation is:

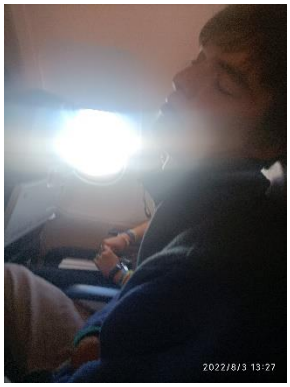
$$S = c * r^{\gamma}$$

It can be explained that r is the original gray value, the value range is 0 to 1, and c is the gray scale scaling factor, γ Controls the scaling of the transform. When the γ value is greater than 1, the gray scale of the image is compressed, and the image brightness decreases. When the γ value is less than 1, the contrast of the image will be enhanced.

The power function is calculated by np.power, the implementation in code list below:

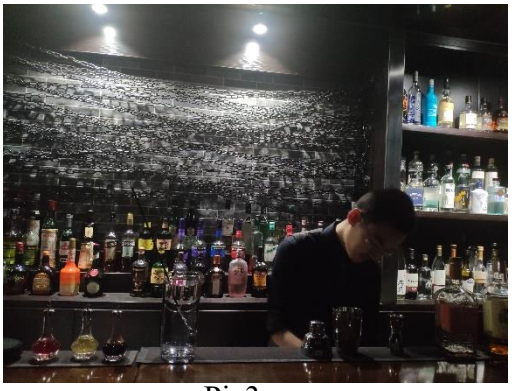
```
img1 = np.power(img/float(np.max(img)), 0.2)
```

Log Transformation

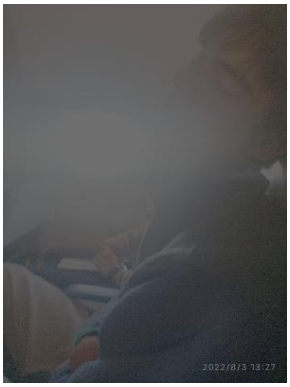


Pic1

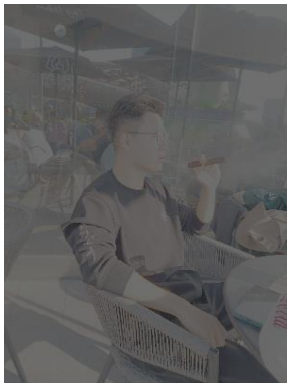
Pic2



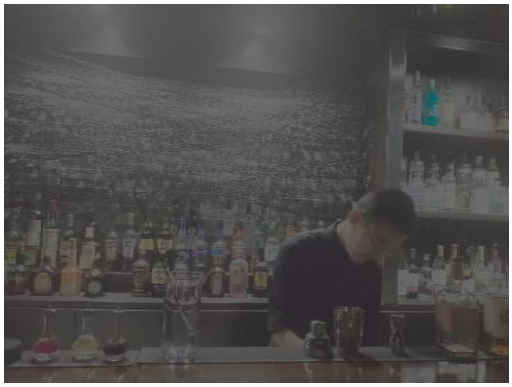
Pic3



Pic1_result



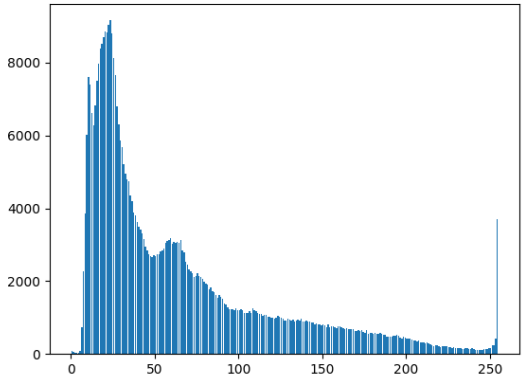
Pic2_result

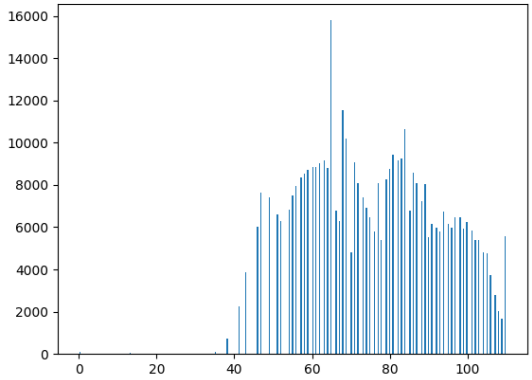


Pic3_result

Log transformation expands the low gray value part of the image and compresses the high gray value part to emphasize the low gray value part of the image, and it can also compress the dynamic range of the image with large pixel value changes. It can be seen from the above figure that the strong light source is darkened, while the part with lower gray value is expanded, and the difference between high and low gray values is reduced..

Take Pic3 as an example, its original gray level histogram has the following changes after Log Transformation:





Original Figure Figure

It can be seen that the gray level image that was originally dark and had too concentrated brightness in some places is normalized and no longer concentrated after Log transformation.

The function of Log transformation is list below, r is the original gray value, the value range is 0 to 1, and c is the gray scale scaling factor.

$$s = c * \log(1 + r)$$

The equation is implemented in the code using np.log:

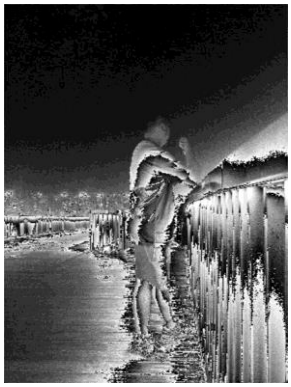
result = c*np.log(1.0+img)

Linear Transformation

The image enhancement linear transformation can adjust the contrast and brightness of the image. The linear transformation formula is as follows, Where a controls the contrast of the image and b affects the brightness of the image.

$$s = a * r + b$$

Pic1



a=-1 b=255

a<0 b=0

It can be seen from the above figure that when a=- 1 and b=255, the image is reversed, the light source is darkened, and the brightness of the dark place is increased. When a<0 and b=0, the dark area of the image is lightened, and the bright area is darkened, but not completely reversed.

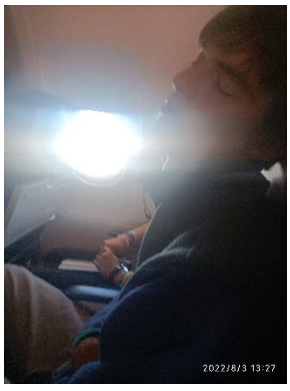
I classify different situations through conditional statements. The process in the code is as follows:

```
A=[-0.5,1.5,0.5,1,-1]
B=[0,20,-50,255]
def LinearTransformation(a,b,img):
    for a in range(len(A)):
        for b in range(len(B)):
            aa = A[a]
            bb = B[b]
            print(aa,bb)
            if aa <0 and bb==0:
                new_img = np.ones((img.shape[0], img.shape[1]), dtype=np.uint8)
                for i in range(new_img.shape[0]):
                    for j in range(new_img.shape[1]):
                        new_img[i][j] = img[i][j]*a + b
                cv2.imshow('a<0 and b=0', new_img)
                cv2.waitKey(0)

            elif aa>1:
                new_img = np.ones((img.shape[0], img.shape[1]), dtype=np.uint8)
                for i in range(new_img.shape[0]):
                    for j in range(new_img.shape[1]):
                        if img[i][j] * a + b > 255:
                            new_img[i][j] = 255
                        else:
                            new_img[i][j] = img[i][j] * a + b
                cv2.imshow('a>1', new_img)
                cv2.waitKey(0)

            elif aa <1:
                new_img = np.ones((img.shape[0], img.shape[1]), dtype=np.uint8)
                for i in range(new_img.shape[0]):
                    for j in range(new_img.shape[1]):
                        new_img[i][j] = img[i][j] * a + b
                cv2.imshow('a<1', new_img)
                cv2.waitKey(0)
```

```
            elif aa== -1 and bb != 0:
                new_img4 = np.ones((img.shape[0], img.shape[1]),
dtype=np.uint8)
                for i in range(new_img4.shape[0]):
                    for j in range(new_img4.shape[1]):
                        pix = img[i][j] * a + b
                        if pix > 255:
                            new_img4[i][j] = 255
                        elif pix < 0:
                            new_img4[i][j] = 0
                        else:
                            new_img4[i][j] = pix
                cv2.imshow('a=-1,b!=0', new_img)
                cv2.waitKey(0)

            elif aa== -1 and bb == 255:
                new_img5 = 255 - img
                cv2.imshow('a=-1,b=255', new_img)
                cv2.waitKey(0)

            else:
                break

    return new_img
```

Grayscale Grading

Pic1



Pic2

Pic3

Pic1_result

Pic2_result

Pic3_result

The gray level grading can freely select the gray level area that you want to highlight in the image. The gray level of other areas can be reduced after highlighting [A, B], or it can remain unchanged. As can be seen from the above figure, I specified the part with lower brightness in Pic1, so that the picture has this effect. In Pic2, I chose the part with higher brightness to continue to highlight, which produced this effect. In Pic3, because the overall picture brightness is low, the grayscale of the building rises after the grayscale layering.

```
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if r_left <= img[i, j] <= r_right:
            level_img[i, j] = r_max
        else:
            level_img[i, j] = img[i, j]
```

Through traversal in the code, if the selected grayscale is within the grayscale range of the image, I will increase the grayscale of the range to the maximum.

Conclusion

It is not possible to judge the advantages and disadvantages simply by the simplicity and complexity of some image enhancement principles. Every image enhancement has its own application occasions, even if the time linear transformation is the same. Only

when the requirements for image change are clear can appropriate image enhancement methods be properly selected. Among these methods, gamma transformation and histogram equalization have more obvious effects for the selected photos.

Part2

Gaussian Filter

Pic1

Pic2

Pic1_result

Pic2_result

From the above figure, it can be observed that the image with Gaussian filtering has obvious blur effect, and the details of the image are relatively complete. Gaussian filtering has a good suppression effect on noise with normal distribution.

For a two-dimensional image, its Gaussian function formula is as follows:

$$f = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2-y^2}{2\sigma^2}}$$

First, it is necessary to generate the Gaussian kernel, determine the distance from each point in the neighborhood of the Gaussian kernel to the center point according to the selected filtering radius and standard deviation, and then substitute it into the upper function to solve the position values of the Gaussian kernel, and then conduct normalization and other processing. The process in the code is as follows:

```
K = np.zeros((K_size, K_size), dtype=np.float)

    for x in range(-pad, -pad + K_size):

        for y in range(-pad, -pad + K_size):
            K[y + pad, x + pad] = np.exp(-(x ** 2 + y ** 2) / (2 * (sigma ** 2)))

    K /= (2 * np.pi * sigma * sigma)

    K /= K.sum()

    tmp = result.copy()
```

Then fill the boundary according to the size of Gaussian kernel, Because there are not enough points in the neighborhood of the points near the boundary for filtering, it is necessary to expand the upper, lower, left and right boundaries before filtering. The width of each boundary expansion is the size of the filtering radius.

```
pad = K_size // 2

    result = np.zeros((H + pad * 2, W + pad * 2, C), dtype=np.float)

    result[pad: pad + H, pad: pad + W] = img.copy().astype(np.float)
```

For each element P in the image, align the center of the Gaussian kernel with the space of the element P to be processed, multiply and add the weights in the Gaussian kernel and the elements in the neighborhood of P, and then operate in step 3 as the output value of P (excluding the elements with extended boundaries). The result is the result of Gaussian filtering.

```
    for y in range(H):

        for x in range(W):

            for c in range(C):
                result[pad + y, pad + x, c] = np.sum(K * tmp[y: y + K_size, x: x +
K_size, c])
```

```
result = np.clip(result, 0, 255)

result = result[pad: pad + H, pad: pad + W].astype(np.uint8)
```

Mean Filtering

Pic1

Pic2

Pic1result

Pic2result

As can be seen from the above figure, the image is obviously blurred. The difference

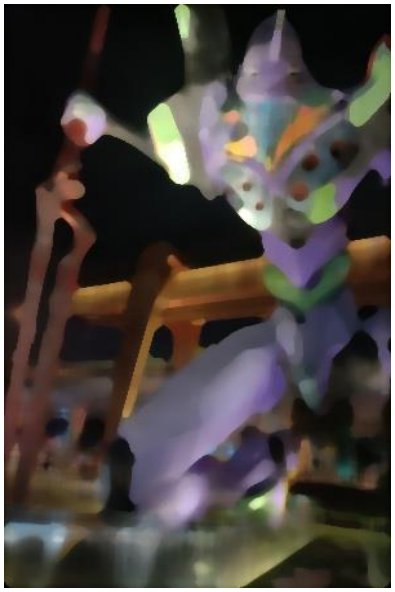
between the mean filter and the Gaussian filter is that the template coefficients of the mean filter are the same, which is 1. The template coefficient of the Gaussian filter decreases with the increase of the distance from the center of the template, and follows the distribution of the Gaussian function.

First fill the image edge in the code:

```
# Expand the incoming image with the size of num
    img = cv2.copyMakeBorder(img, num, num, num, num, cv2.BORDER_REPLICATE)
    h1, w1 = img.shape[0:2]
```

then the points in the image are processed with kernel one by one

Medium Filtering



It can be seen from the above figure that the image after the median value is blurred basically loses the image details, and only the general outline of the object in the picture can be seen. The median blur is different from the Gaussian filter and the mean filter. It is a nonlinear filter. It takes the median of the field as the gray value of the current point. It arranges adjacent pixels according to their sizes, and take the value in the middle of the sorted pixels as the pixel value of median filtering, as shown in the following figure.

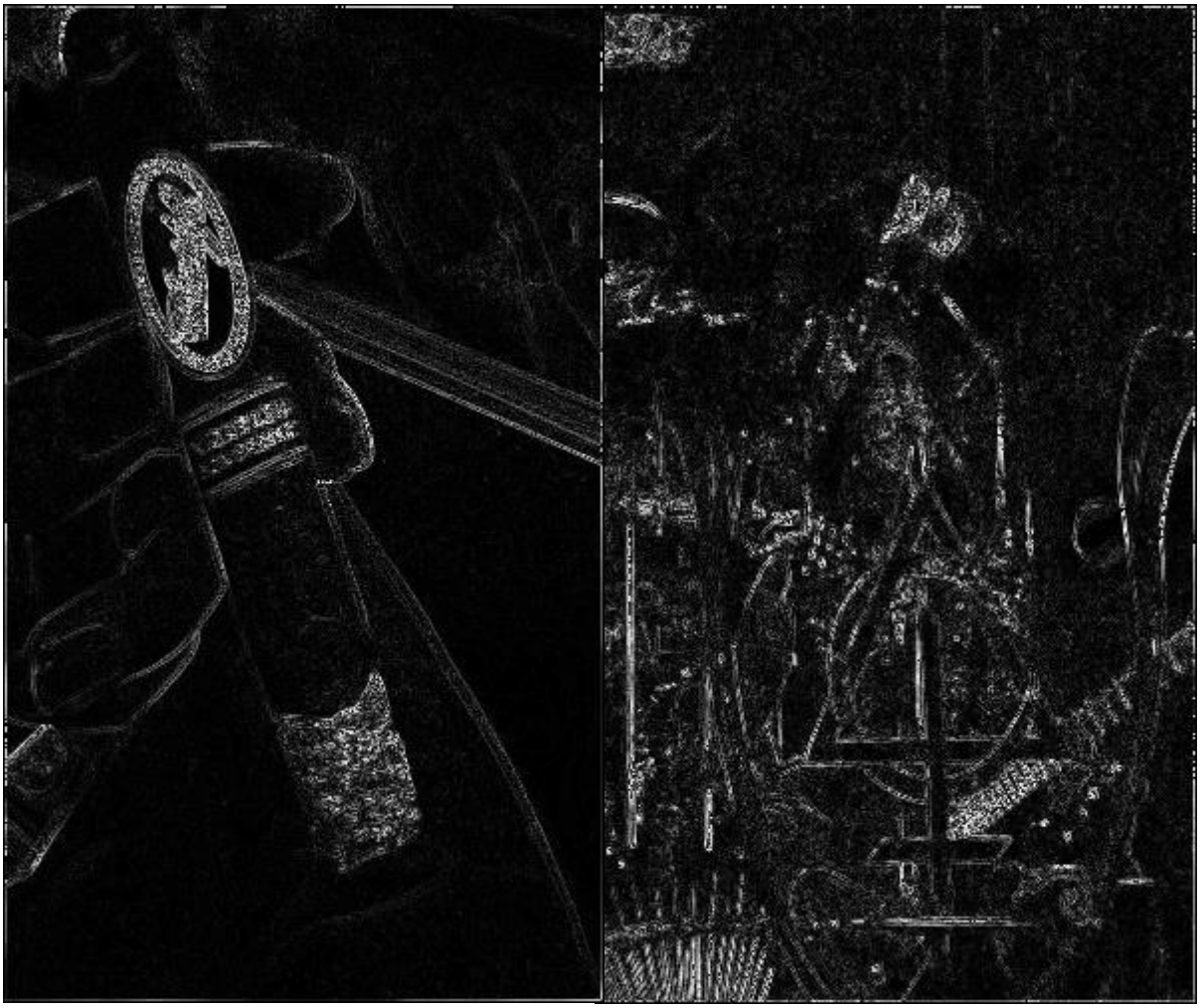
It is also necessary to fill the edge of the image first, then, the pixels of the selected area and their adjacent pixels are arranged by size through no. medium, and then the median value is taken.

result[pad+y,pad+x,ci] = np.median(tmp[y:y+K_size,x:x+K_size,ci])

Conclusion for Blur

After the same picture is blurred, it can be found that Gaussian blur can retain more details of the characters, and median blur is used as a nonlinear filter. It simply selects the median of the pixel points in the region as the gray value of the whole picture, which makes the blur effect most obvious, but there is no way to reflect the details. Mean filtering is to take the mean value of the pixel points in the corresponding area for the convolution kernel, so the blurred details are not as good as Gaussian filtering.

Laplace Filter



It can be seen from the above figure that the edge of the image is strengthened after the processing of Laplace operator, while retaining other details. Compared with Sobel operator, Laplace operator actually has second order differentiation, which uses the second order difference to calculate the edge of the image. Laplace operator is a high pass filter, which is used to retain the high frequency component (the part with sharp changes) of the image and suppress the low frequency component (the part with slow changes) of the image, so it can be used to detect edges. However, it can be found that the edge is not the only part of the Laplace enhancement, indicating that the method is relatively sensitive to noise.

The function of Laplace is:

$$\nabla 2f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

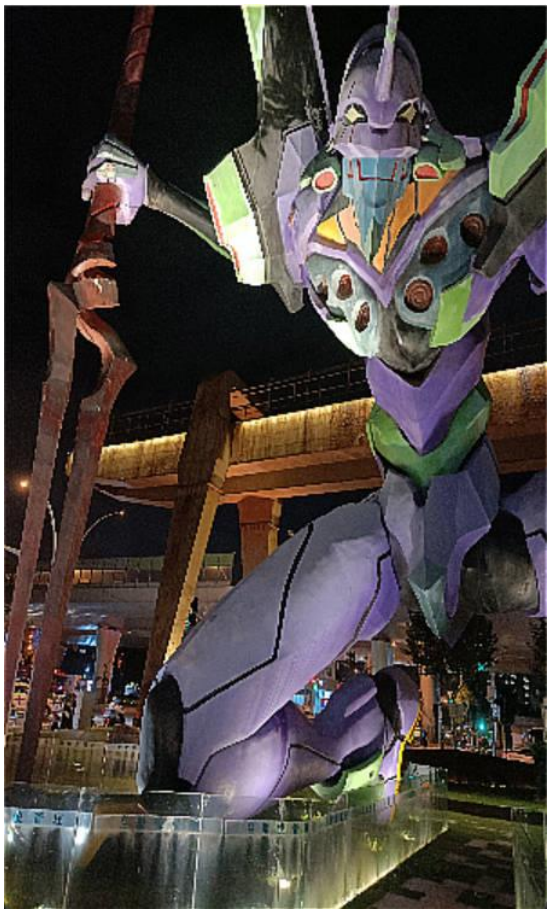
Use Laplacian filter to get the region of gray mutation in the image $\nabla 2f(x,y)$

laplace[row_i, col_j] * img2[i + row_i, j + col_j]

Use the original image plus? 2f (x, y), as follows:

sum = laplace[row_i, col_j] * img2[i + row_i, j + col_j] + sum

USM



It can be seen that the edge of the image after USM sharpening can be displayed more clearly. If the image is enlarged, it can be observed that the image after USM sharpening has added a black outline on the edge. This sharpening method is to first make a Gaussian blur on the original image, then subtract a coefficient from the original image and multiply it by the Gaussian blur image, and then scale the value to 0~255 RGB pixel value. The method based on USM sharpening can remove some small interference details and noise, which is more authentic than the image sharpening result obtained by using convolution sharpening operator directly.

```
result = result*1.0
alpha = 1.5

img_out = (img - result) * alpha + img

img_out = img_out / 255.0

#Saturation treatment
mask_1 = img_out < 0
mask_2 = img_out > 1

img_out = img_out * (1 - mask_1)
img_out = img_out * (1 - mask_2) + mask_2
```

Conclusion for Sharpening

Traditional sharpening generally uses operators to obtain high-frequency components, and the effect of sharpening is very obvious after obtaining high-frequency components through this filter operator. However, one side effect is that it is very sensitive to noise, and it is easy to generate bright pseudo edges at the edge of image texture. The USM obtains the high pass result through Gaussian filtering. If the absolute value of the high pass value is greater than the specified threshold, the pixel will be sharpened. Otherwise, no sharpening will be performed, and the intensity of sharpening is controlled by the quantity value Amount. By adjusting the threshold value, it is determined to what degree of contrast pixels need to be enhanced.

Part3

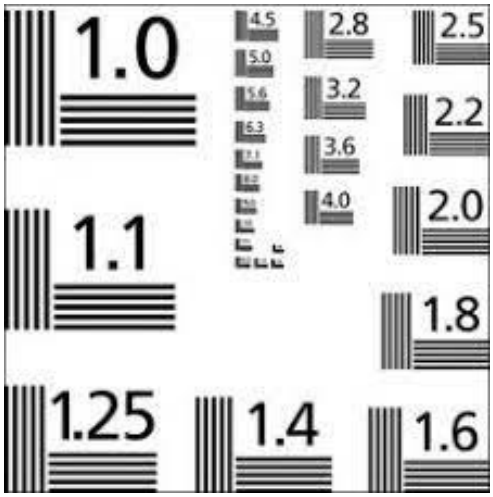
BiBubic

Original

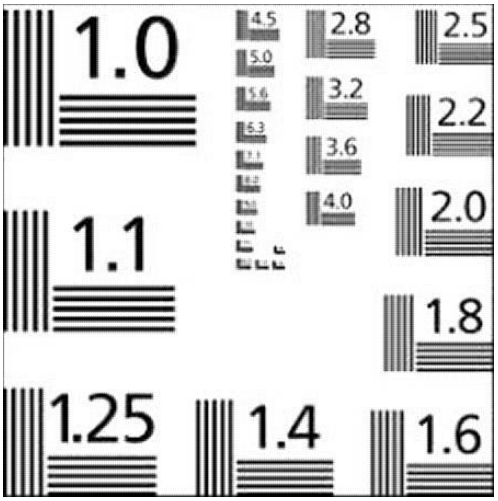
*2

*4

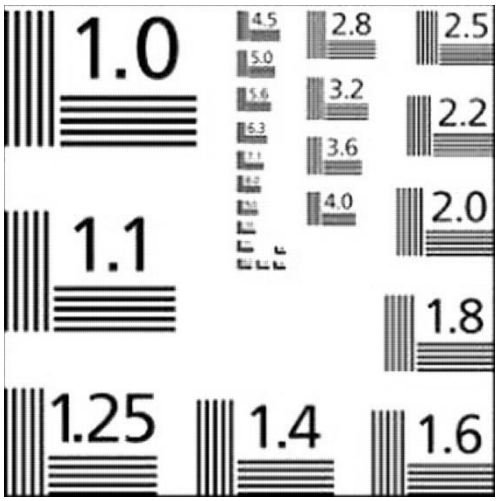
*8



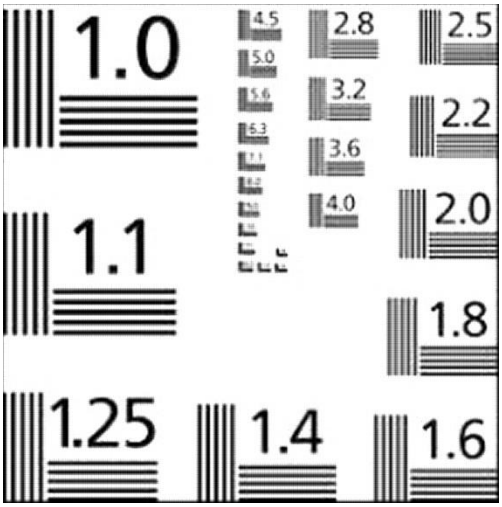
Original



*2



*4



*8

The algorithm uses the gray values of 16 points around the sampling point for cubic interpolation, which takes into account not only the influence of the gray values of 4 directly adjacent points, but also the influence of the change rate of gray values between adjacent points. The third operation can get a magnification effect closer to the high-resolution image, but it also leads to a sharp increase in the amount of operation. As can be seen from the figure, when the image is magnified to 8 times, the grid like lines begin to appear in the image. And the running time is very long.

Detail

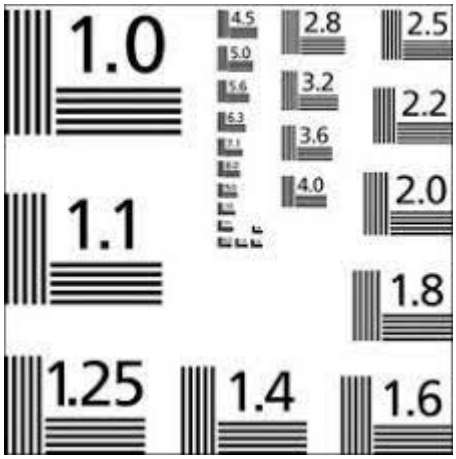
Nearest Neighbor

Original

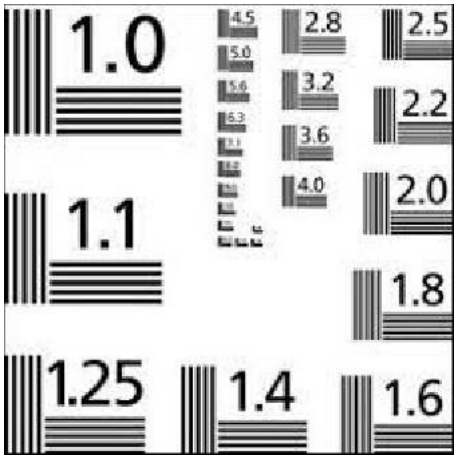
*1.5

*2

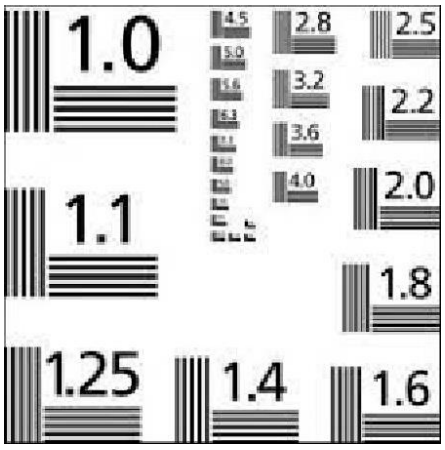
*3



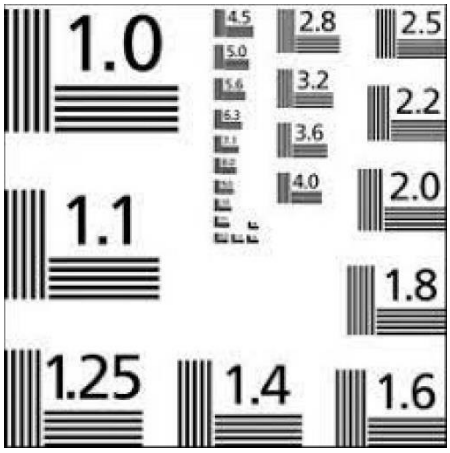
Original



*1.5



*2



*3

Detail

Nearest neighbor interpolation is a relatively simple method of using interpolation convolution amplification, but it can be seen from the graph that the resolution after the method decreases. The principle is to make the gray value of the transformed pixel equal to the gray value of the nearest input pixel. The nearest neighbor element method requires less calculation, but it may cause discontinuity in the grayscale of the image generated by interpolation, and there may be obvious sawtooth in the place where the grayscale changes.

Bilinear Interpolation

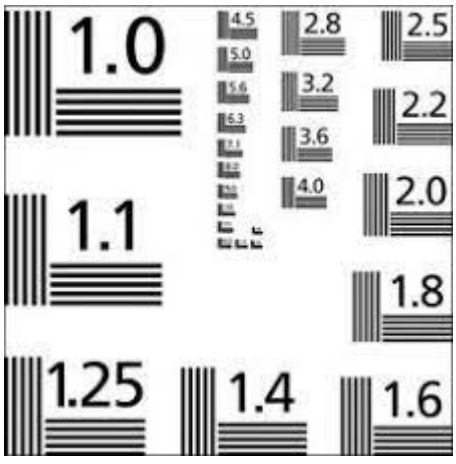
Magnification

Original

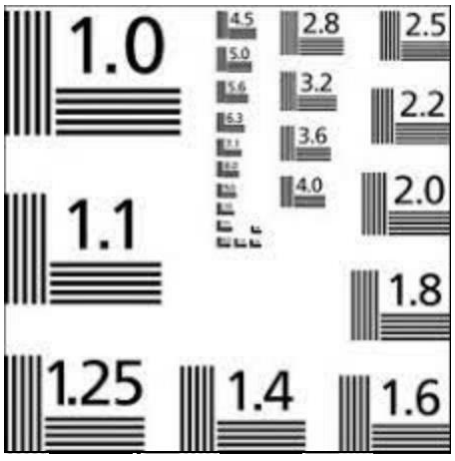
*2

*4

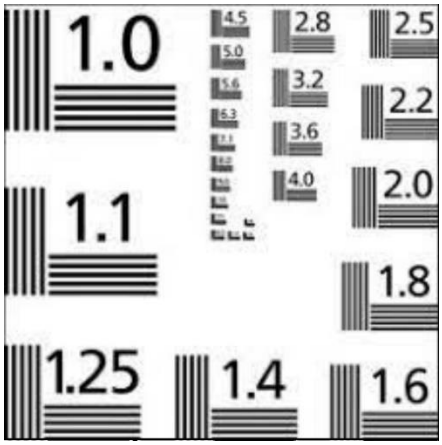
*8



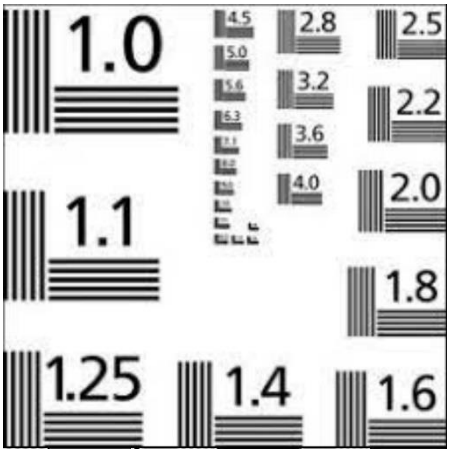
Original



*2



*4



*8

The time and speed of bilinear interpolation to enlarge the image is longer than that of nearest neighbor interpolation, but shorter than that of bicubic interpolation. At the same time, the effect of zooming in the image is better than its bicubic interpolation. After zooming in the details of the image, we can find that the zooming in result of bilinear interpolation is basically the same as the original image. Linear interpolation is to calculate (x, y) according to two points (x0, y0) and (x1, y1). Bilinear interpolation is linear interpolation in two directions.

Original pic detail

Bilinear Interpolation resize detail

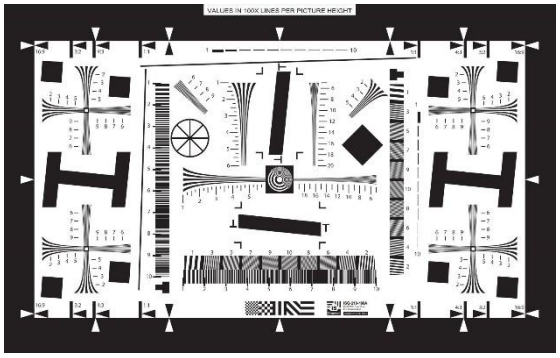
Minification

Original

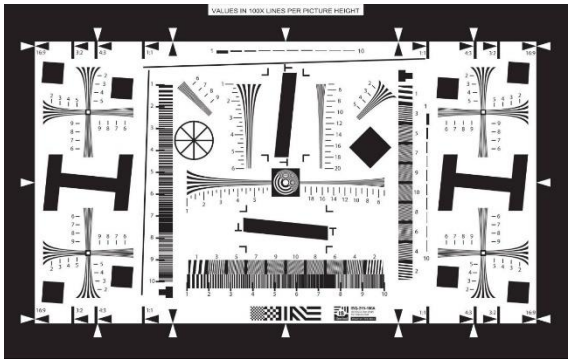
*0.8

*0.4

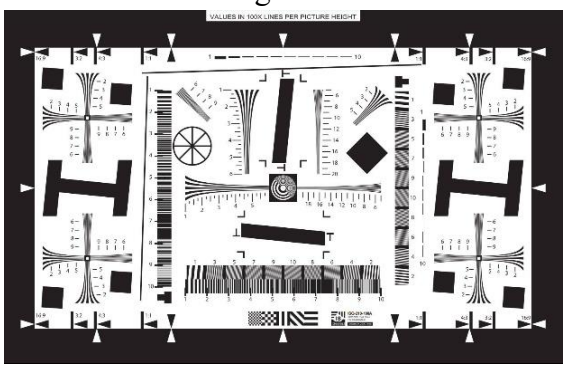
*0.2



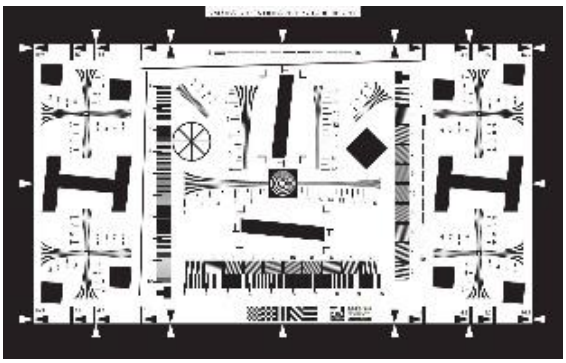
Original



*0.8



*0.4



*0.2

Using the bilinear interpolation method to sit down, it can be found that the image processing process decreases with the increase of the reduction times. It takes about 2 minutes to reduce by 2 times, and about 20 seconds to reduce by 4 times. However, with the reduction of the image, the resolution starts to decline, which is most obvious when the image is reduced by 8 times.

Original detail

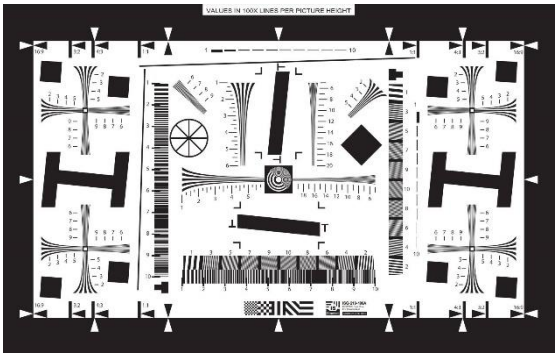
Bilinear Interpolation resize detail

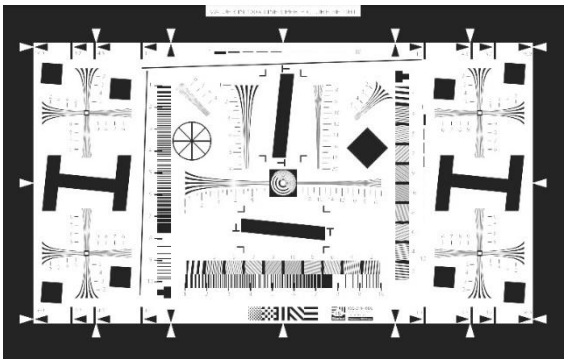
Max Pooling

Original

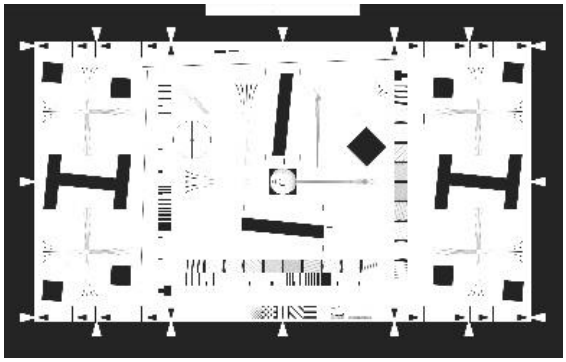
*0.8

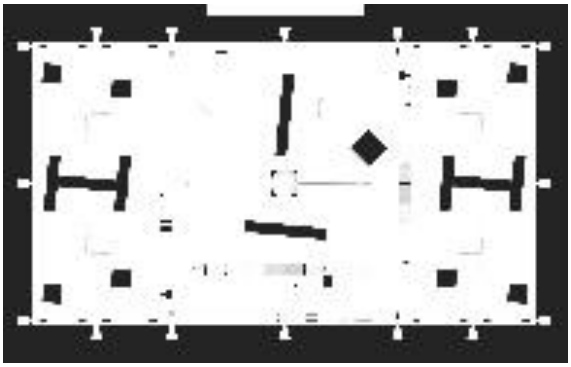
*0.4 *0.2





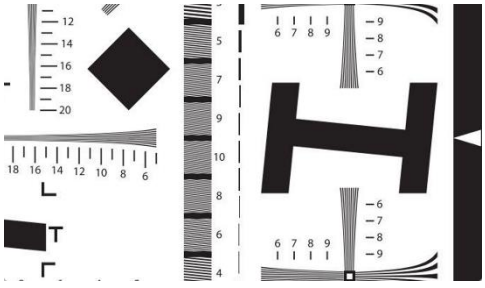
Original *0.8

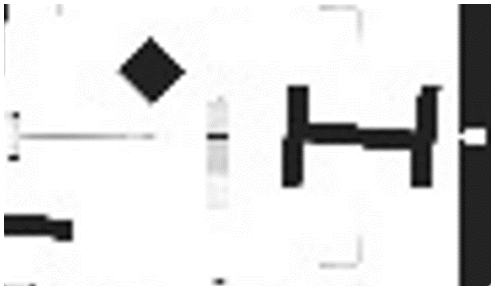




*0.4 *0.2

Pooling is to select an area on the image and take the maximum or average value of the pixel values in the area. Maximizing pooling is to maximize the pixel value. The maximum pooling can see that the resolution decreases significantly after zooming out, for example, when zooming out 8 times in the composite image, a lot of texture and content are lost. The resolution after zooming out with maximum pooling is worse than that of zooming out with bilinear interpolation.





Original Detail Max pooling Detail

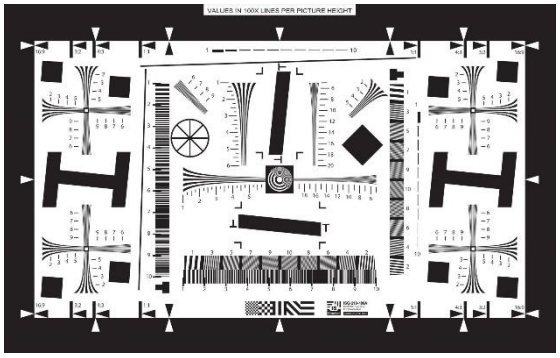
Mean Pooling

Original

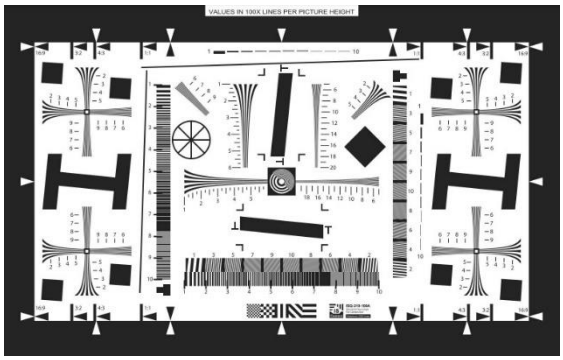
*0.8

*0.4

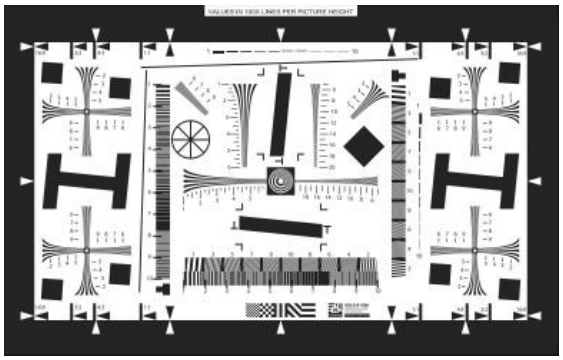
*0.2



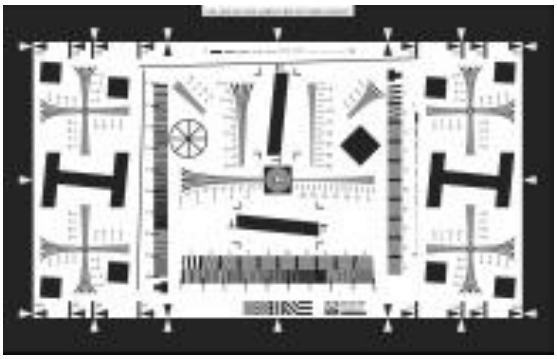
Original



*0.8

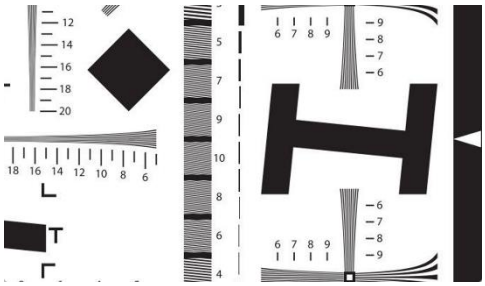


*0.4

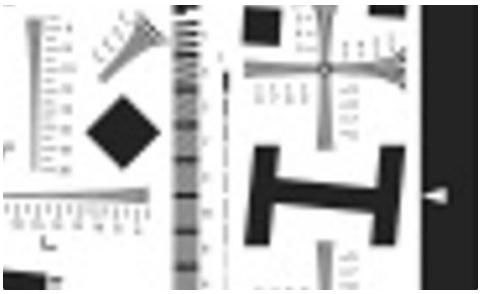


*0.2

Mean Pooling Average the pixels in the pooled area to reduce the resolution. It can be seen from the above figure that compared with the max pooling, the resolution after the average pooling is obviously due to the max pooling.



Original Detail



Max Pooling Detail

Conclusion

For Magnification:

Double legislative interpolation takes the longest time to process images, followed by bilinear interpolation, and the fastest is nearest neighbor interpolation.

However, the bicubic interpolation method still has the highest resolution despite the grid like texture after the image is enlarged. Second, use this method to enlarge the picture. As the image magnification increases, the resolution will be lost. The most serious loss of resolution is the image magnified by nearest neighbor interpolation.

For Minification:

Among the three methods, the bilinear interpolation method takes the longest time to reduce the image, but the processing time will decrease as the reduction ratio increases. There is basically no difference in processing time between average pooling and maximum pooling.

Among the three methods, maximum pooling will cause serious resolution loss, while dual legislative interpolation and mean pooling will cause less resolution loss.