数字图像处理 Homework1

518030910320 支珑润

March 2021

1 Task1

1.1 问题描述

- 输入一张灰度图（rose.tif），对其进行 1.2 倍，2 倍，4.5 倍，8 倍的下采样，并对下采样的图像用双线性插值放大回原始图像分辨率，计算对应的 PSNR 值；
- 要求：下采样、双线性插值、PSNR 计算都要自己编程实现，并与 matlab 自带的库函数处理结果进行对比，分析结果。

1.2 方法原理

1.2.1 下采样

通过下采样可以实现图像的缩小，得到原图像的缩略图。对尺寸为 M*N 的图像进行 k 倍的下采样，当 k 为整数时，即在原图像中在水平和垂直方向各每隔 k 个像素点取一个点作为新图像的像素点，最后得到尺寸为 (M/k)*(N/k) 的新图像。当 k 为有理数 P/Q（P,Q 为整数）时，可以先对图像做 Q 倍上采样再做 P 倍下采样。

1.2.2 双线性插值

插值是指在处理图片放大时，目标图像像素点在原图中对应像素点坐标并非整数点，因此需要通过原图的像素点计算出该非整数位置的像素值。双线性插值则是利用非整数点所处正方形四角的像素点，通过两次线性插值计算出目标点的像素值。具体原理如下所示：

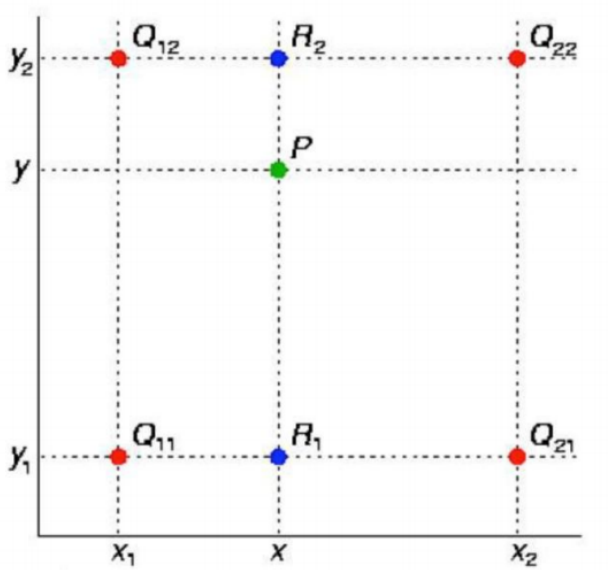

图 1: 双线性插值

需要求 P 点的插值，首先利用线性插值求出位于正方形边上、与 P 同水平坐标的两点 R_1, R_2 插值，然后继续通过线性插值求出 P 的插值。公式如下：

$$
\begin{aligned}
f(R_1) &= \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \\
f(R_2) &= \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \\
f(P) &= \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)
\end{aligned}
$$

1.2.3 PSNR

Peak Signal Noise Ratio，即峰值信噪比 PSNR，用于评价处理后的复原图与原图相比的复原程度，其定义为

$$
PSNR = 10 \times \log_{10} \frac{255^2}{\frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} [f(m,n) - g(m,n)]^2}
$$

式中 $f(m,n)$ 为原图，$g(m,n)$ 为处理后的复原图；若 $g(m,n)$ 越接近 $f(m,n)$ 则 PSNR 越高，复原程度越好。

1.3 求解过程

1.3.1 下采样

原图 rose.tif 的尺寸为 1024×1024，因此对于 2 倍和 8 倍下采样，得到的宽和高均是整数，比较好处理。在 matlab 中，只需创建一个相应尺寸的零矩阵，然后遍历原图，以 2 和 8 为步长将原像素填入新位置即可：

```
% 2倍下采样
height2 = ceil(height/2);
width2 = ceil(width/2);
img2 = zeros(height2, width2);
x = 1; y = 1;
for i=1:2:width
    for j=1:2:height
        img2(x, y) = img(i,j);
        y = y + 1;
    end
    x = x + 1;
    y = 1;
end
```

(a) 2 倍

```
% 8倍下采样
height4 = ceil(height/8);
width4 = ceil(width/8);
img4 = zeros(height4, width4);
x = 1; y = 1;
for i=1:8:width
    for j=1:8:height
        img4(x, y) = img(i,j);
        y = y + 1;
    end
    x = x + 1;
    y = 1;
end
```

(b) 8 倍

图 2: 整数下采样

对于 1.2 倍和 4.5 倍下采样的处理要麻烦一点，我的处理方法是先进行整数倍的上采样再进行整数倍的下采样。例如，对于 1.2 倍我们可以先做 5 倍的上采样。这里有一个问题，我们在上采样是直接用原像素进行 $1 \to 5 \times 5$ 的扩充还是需要插值处理。在尝试了直接扩充和双线性插值这两种方法后，最后经过完整流程的 PSNR 显示直接扩充（$PSNR = 34.1581$）的效果要比插值（$PSNR = 29.1270$）要好，因此这里选择直接扩充。

在 5 倍上采样之后，就要进行 6 倍下采样。这里的问题是下采样后的尺寸不是整数，我的处理办法是向上取整。这样带来的效果是插值复原后的尺寸要比原图大一些，但是考虑到这张图像的特点（边缘均是纯黑像素），我们可以直接对其边缘做裁剪，这样裁剪掉的像素对图片的特征几乎不会有影响。这样我们就得到了非整数下采样后的图像，代码流程如下（以 1.2 倍为例，4.5 倍几乎一样）：

```
% 1.2倍下采样
% 先做5倍上采样

img1_1 = zeros(height*5, width*5);

for x=1:width*5
    for y=1:height*5
        i = ceil(x/5);
        j = ceil(y/5);
        img1_1(x, y) = img(i,j);
    end
end
%img1_1 = BL(img, height, width, 5);
```

(a) 5 倍上采样

```
% 再做6倍下采样
height1 = ceil(height*5/6);
width1 = ceil(width*5/6);
img1 = zeros(height1, width1);

x = 1; y = 1;
for i=1:6:width*5
    for j=1:6:height*5
        img1(x, y) = img1_1(i,j);
        y = y + 1;
    end
    x = x + 1;
    y = 1;
end
```

(b) 6 倍下采样

图 3: 非整数下采样

1.3.2 双线性插值

为了解决在索引原图像时的边界溢出问题，首先我们需要在原图边界加墙，像素值与边界值相同：

```
function [outimg] = BL(inimg,sh,sw,n)
ima2=zeros(sh+2,sw+2);
ima2(1,2:sw+1)=inimg(1,:); %原图像上边加墙，灰度值与边界一致
ima2(sh+2,2:sw+1)=inimg(sh,:); %原图像下边加墙，灰度值与边界一致
ima2(2:sh+1,2:sw+1)=inimg; %将原图像赋值给中心部分
ima2(2:sh+1,1)=inimg(:,1); %原图像左边加墙，灰度值与边界一致
ima2(2:sh+1,sw+2)=inimg(:,sw); %原图像右边加墙，灰度值与边界一致
ima2(1,1) = inimg(1,1);
ima2(1,sw+2) = inimg(1,sw);
ima2(sh+2,1) = inimg(sh,1);
ima2(sh+2,sw+2) = inimg(sh,sw); %四角赋值
```

图 4: 边界加墙

接下来只需要按照 1.2.2 中的公式计算即可，如图 5 所示。

```
dw=sw*n; %计算缩放后的图像的宽
dh=sh*n; %计算缩放后的图像的高
dw1=round((sw+2)*n); %计算加墙后缩放的图像的宽
dh1=round((sh+2)*n); %计算加墙后缩放的图像的高
outimg=zeros(dh1,dw1); %创建新图像的矩阵
%从不是“墙”的位置开始计算缩放后的图像的各点灰度值
%考虑缩小图像时，输入的缩放倍数是小数，需进行取整
start=round(n+1);
endI=round(dh+n);
endJ=round(dw+n);
for i=start:endI
    for j=start:endJ
        %缩放后的图像坐标在原图像处的位置
        tx=i/n; ty=j/n;
        %得到小数坐标
        tdx=tx-floor(tx); tdy=ty-floor(ty);
        %确定临近四个角的坐标
        Q11x=tx-tdx;Q11y=ty-tdy;    %Q11点
        Q12x=tx-tdx;Q12y=Q11y+1;    %Q12点
        Q21x=Q11x+1;Q21y=Q11y;      %Q21点
        Q22x=Q11x+1;Q22y=Q11y+1;     %Q22点
        %根据双线性内插算法，算出缩放后的图像在（i,j）点处的灰度值
        outimg(i,j)=tdx*tdy*ima2(Q11x,Q11y)+(1-tdx)*tdy*ima2(Q12x,Q12y)...
            +tdx*(1-tdy)*ima2(Q21x,Q21y)+(1-tdy)*(1-tdx)*ima2(Q22x,Q22y);
    end
end
end
```

图 5: 双线性插值

1.3.3 PSNR 计算

在计算 PSNR 时，需要保证变换后的图像与原图像尺寸完全相同。按照我们之前的取整处理，得到的图像总会比原图像更大，因此我们只需裁剪部分边界像素。在裁剪时，我们也需要注意保持两边的裁剪程度一致，这样可以尽量保持图像特征和原图的相对位置一致。在获取相同尺寸的图像后，我们就可以按照公式计算 PSNR 值。

```
function [output] = PSNR(img1,img2)
[width, height] = size(img1);
sum = 0;
for i=1:width
    for j=1:height
        sum = sum + (img1(i,j)-img2(i,j))^2;
    end
end
output = 10 * log10(255*255*width*height/sum);
end
```

图 6: PSNR 计算

1.4 结果分析

经过上述操作最后得到的四种不同倍数下采样复原图片如下所示:

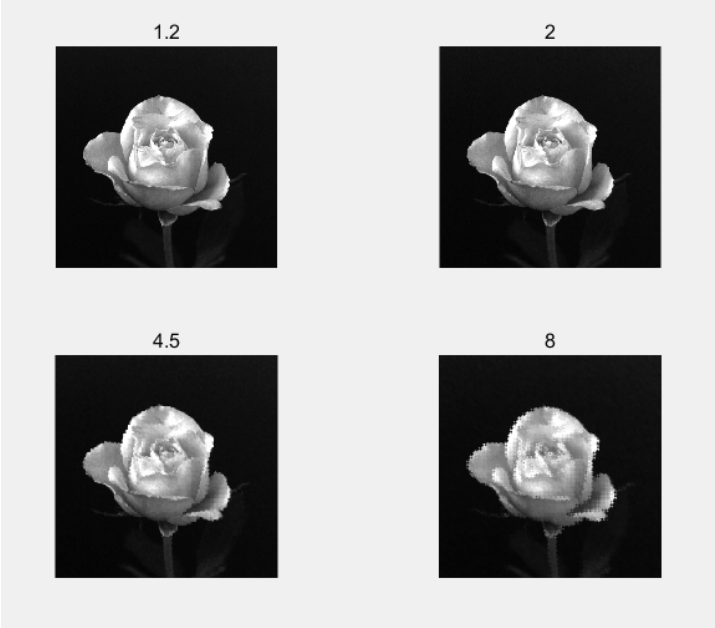

图 7: 不同倍数下采样复原图像

从中可以直观看出，随着采样倍数逐渐增加，经过复原的图像质量也逐渐降低。但显然我们的方法是存在一定缺陷的，通过调用 matlab 自带的 imresize 函数，可以直接实现非整数的下采样和双线性插值。分别计算我们得出的图像以及标准方法得出的图像与原图的 PSNR 数值，结果如下:

PSNR	自我实现	标准实现
1.2 倍	34.1581	40.8431
2 倍	31.9202	37.7519
4.5 倍	26.6568	33.7129
8 倍	22.4556	30.4543

表 1: PSNR 结果对比

从表中看出，下采样后复原的图像与原图的相似程度会随着采样倍数增加而逐渐降低，这显然是由于随着采样倍数增加，原图丢失的特征越来越多。而我们的方法显然与标准库函数相比要逊色一些，这可能是由于我们的一些处理并不严谨，例如在上采样时采用像素直接扩充，在处理复原图像与原图尺寸不一致时直接裁剪边界，双线性插值的处理也比较简单。但总的来说，与标准库函数相比，我们的处理效果尚可。

2 Task2

2.1 问题描述

- 输入一张 RGB 彩色图像（iris.tif），转换到 HSI 空间，并调整 H、S、I 的值，说明 H、S、I 变化对图像主观质量的影响；
- 选定一种 RGB 到 YUV 的空间转换并实现，并采用 YUV444，YUV422，YUV420 采样格式，用 YUV2RGB 转换回 RGB，查看其主观质量，分析人眼视觉对于 Y、U、V 的视觉特性。
- 要求：RGB 与 HSI 的转换可直接调用库函数，但 RGB 与 YUV 的转换自己编程实现，YUV 采样后的插值调用问题（1）中实现的双线性插值。

2.2 方法原理

2.2.1 HSI 颜色空间

HSI 颜色空间利用三个参量来表示彩色光：

- 色调 H：颜色的种类；

- 色饱和度 S：彩色光的纯度。若 S=1，则为高饱和度的彩色光，如深红；若 S<1，则为低饱和度的彩色光，如淡红，可由深红与白光配成；若 S=0，则为中性色，如白，灰，黑等；
- 亮度 I：与光的强度有关，与彩色无关。
- 色调和色饱和度合称“色度”，可用色环表示。

可用纺锤形来表示彩色光的亮度和色度。椎体中心饱和度为 0，到中心轴的距离表示饱和度，外表面饱和度最高为 1，从下到上亮度逐渐增大，圆锥体横断面用来表示色度，按照红绿蓝的顺序。

RGB 到 HSI 的转换可用下列公式表示：

$$
\begin{aligned}
H &= \begin{cases} \theta & B \le G \\ 360 - \theta & B > G \end{cases} \\
\theta &= \arccos\left\{ \frac{\frac{1}{2}\left[(R-G)+(R+B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{\frac{1}{2}}} \right\} \\
S &= 1 - \frac{3}{R+G+B}\min(R,G,B) \\
I &= \frac{1}{3}(R+G+B)
\end{aligned}
$$

相应的,HSI 转 RGB 的公式可反向推出。对于不同的 H（不同的 θ 范围），RGB 的变换公式各有不同：

- 在 RG 扇形区域 $[0, 120)$：

$$
\begin{aligned}
B &= I(1-S) \\
R &= I\left[1 + \frac{S\cos H}{\cos(60-H)}\right] \\
G &= 3I - (R+B)
\end{aligned}
$$

- 在 GB 扇形区域 $[120, 240)$：

$$
\begin{aligned}
R &= I(1-S) \\
G &= I\left[1 + \frac{S\cos(H-120)}{\cos(180-H)}\right] \\
B &= 3I - (R+G)
\end{aligned}
$$

- 在 BR 扇形区域 $[240, 360)$：

$$
\begin{aligned}
G &= I(1 - S) \\
B &= I\left[1 + \frac{S\cos(H - 240)}{\cos(300 - H)}\right] \\
R &= 3I - (G + B)
\end{aligned}
$$

2.2.2 YUV 颜色空间

YUV 颜色空间利用三个参量来表示彩色光：亮度 Y，色度 U，浓度 V。人眼对亮度更敏感，实际应用中允许降低色度的带宽，而不影响视觉质量，从而减少数据量。YUV 有多种采样格式，如下图所示：

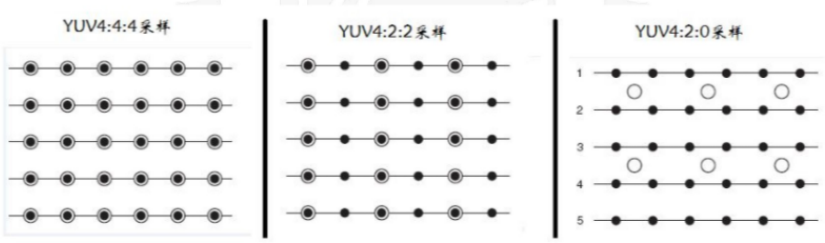

图 8: YUV 采样格式

- YUV444: Y、U、V 三个信道的采样率相同，因此在生成的图像里，每个像素的三个分量完整。
- YUV422: 每个色度信道的采样率是亮度信道的一半，所以水平方向上的色度采样率只是 4:4:4 的一半。
- YUV420: 对于每个色度分量来说，水平和垂直方向的的采样率都是 2:1，因此是四个像素点共用一个 UV 分量。

RGB 与 YUV 的转换有多种不同的格式，对应的公式也不同。在本任务中，采用在电脑显示器中的转换格式，一般是全范围取值，YUV 范围也在 $[0, 255]$。转换公式如下：

$$
\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}
$$

YUV 到 RGB 的转换也可以通过上述公式进行逆变换:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.4017 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.7722 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U-128 \\ V-128 \end{bmatrix}$$

2.3 求解过程

2.3.1 RGB 与 HSI 的转换

RGB 与 HSI 的相互转换按照上面的公式，分别通过函数 RGB2HSI, HSI2RGB 实现，需要注意其中 H、S、I 的取值范围，分别为 $[0, 360)$, $[0, 1]$, $[0, 1]$，具体计算过程详见代码 RGB2HSI.m，HSI2RGB.m，此不赘述。

2.3.2 改变 HSI 对图像主观质量的影响

用函数 RGB2HSI 将原图片转换成 HSI 格式后，我们分别获得 H、S、I 分量，考虑到它们的取值范围有限，我们将它们各自减半，然后用函数 HSI2RGB 恢复成 RGB 格式，与原图进行比较。代码如下所示：

```
% RGB与HSI的转换
HSIimg = RGB2HSI(img);
H = HSIimg(:,:,1);
S = HSIimg(:,:,2);
I = HSIimg(:,:,3);
% 改变H：取值范围是0-360
H_RGBimg = HSI2RGB(cat(3,H/2,S,I));
S_RGBimg = HSI2RGB(cat(3,H,S/2,I));
I_RGBimg = HSI2RGB(cat(3,H,S,I/2));
% 展示改变HSI后的复原结果
figure;
subplot(2,2,1),imshow(img),title('原图');
subplot(2,2,2),imshow(H_RGBimg),title('减半H');
subplot(2,2,3),imshow(S_RGBimg),title('减半S');
subplot(2,2,4),imshow(I_RGBimg),title('减半I');
```

图 9: 改变 HSI 代码

2.3.3 RGB 与 YUV 的转换及 YUV 采样

RGB 转换成 YUV 由函数 RGB2YUV 实现，具体计算过程与上述公式一致，此不赘述。重点在于 YUV 的三种采样格式。YUV444 不需要另外的操作，在 YUV422 采样时，我们采用的方案是用一个一个高度不变，宽度减半的双信道 UV 矩阵来保存色度分量。扫描每一行像素，只保留奇数列的 UV 分量，这样我们就实现了每一行的色度分量采样率为 1/2。在 YUV420 采样时，我们需要一个高度宽度都减半的双信道 UV 矩阵来保存每个 2×2 像素群中左上角像素的色度分量。这两种采样具体实现如下所示，分别通过函数 YUV422sample 和 YUV420sample 实现：

```
function [Y,UV] = YUV422sample(YUVimg)
Y = YUVimg(:,:,1);
U = YUVimg(:,:,2);
V = YUVimg(:,:,3);
[height,width,~] = size(YUVimg);
U1 = zeros(height,floor(width/2));
V1 = zeros(height,floor(width/2));
for i=1:height
    for j=1:floor(width/2)
        U1(i,j) = U(i,2*j-1);
        V1(i,j) = V(i,2*j-1);
    end
end
UV = cat(3,U1,V1);
end
```

(a) YUV422 采样

```
function [Y,UV] = YUV420sample(YUVimg)
Y = YUVimg(:,:,1);
U = YUVimg(:,:,2);
V = YUVimg(:,:,3);
[height,width,~] = size(YUVimg);
U1 = zeros(floor(height/2),floor(width/2));
V1 = zeros(floor(height/2),floor(width/2));
for i=1:floor(height/2)
    for j=1:floor(width/2)
        U1(i,j) = U(2*i-1,2*j-1);
        V1(i,j) = V(2*i-1,2*j-1);
    end
end
UV = cat(3,U1,V1);
end
```

(b) YUV420 采样

图 10: YUV 采样格式

2.3.4 YUV 转 RGB

上面提到的 YUV 转 RGB 公式只有在每个像素的 Y、U、V 分量都完好时才能使用。YUV444 可以直接用公式计算。对于 YUV422 和 YUV420，像素的色度分量是缺失的，我们首先需要通过上采样补全缺失的 U、V 值。按照我们之前的做法，YUV422 用的是原图尺寸的单信道矩阵，以每行的两个相邻像素为单位，我们让它们共享奇数列像素的色度分量；而 YUV420 的矩阵像素点位置位于原图的四个像素点中间，这时候需要用到 Task1 中的双线性插值。在得到完整的 Y、U、V 矩阵，我们可以通过上面的公式计算出 RGB 格式，具体实现过程如下（RGB 计算过程省略）：

```
for i=1:height
    for j=1:width
        if (rem(j,2)==0)
            U(i,j) = UV(i,j/2,1);
            V(i,j) = UV(i,j/2,2);
        else
            U(i,j) = UV(i,min((j+1)/2,floor(width/2)),1);
            V(i,j) = UV(i,min((j+1)/2,floor(width/2)),2);
        end
    end
end
% RGB计算
```

(a) YUV422 转 RGB

```
U0 = UV(:,:,1);
V0 = UV(:,:,2);
U = BL(U0, floor(height/2),floor(width/2),2);
V = BL(V0, floor(height/2),floor(width/2),2);
U = cut(U,height,width);
V = cut(V,height,width);
% RGB计算
```

(b) YUV420 转 RGB

图 11: YUV 转 RGB

2.4 结果分析

2.4.1 改变 HSI 对图像主观质量的影响

将 H、S、I 分量分别减半后，得到的图像与原图对比结果如下：

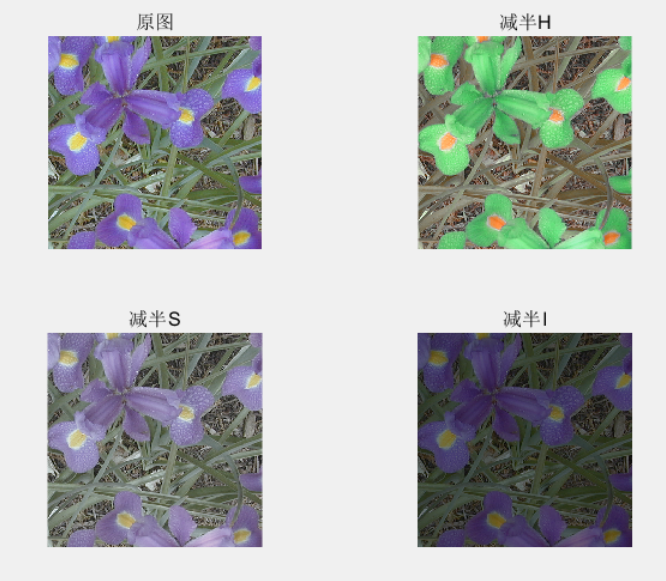

图 12: 分别减半 HSI 得到的图像

在减半 H 分量后，我们发现花瓣的整体色调黄绿，这是因为我们通过减半将 H 分量的范围减小到 (0,180)，而从色调的分布我们可以发现，原本偏蓝色调的 H 减半后刚好位于偏绿的部分，所以我们可以清晰的发现图像中蓝紫

色的花瓣变成了绿色，而对于其他特征，图像的明亮程度和颜色的“浓度”没有变化，这证明了 H 分量主要影响的图像的色调，即颜色的种类。

在减半 S 分量后，我们发现图像的颜色种类和明亮程度没有变化，唯一变化的是颜色的“浓度”，可以明显发现颜色变淡了，这说明 S 分量主要影响的是图像颜色的饱和度，S 越小，颜色饱和度越低，看上去也就越淡。

在减半 I 分量后，我们发现图像的颜色性质包括种类和浓度没有变化，只是图像整体变暗了，说名 I 分量影响的是图像的明亮程度，而与具体颜色无关。

总的来说，HSI 格式的图像，三个分量对图像主观质量有着不同方向上的、独立的影响，也给我们呈现了另一种维度的图像性质。

2.4.2 不同采样格式得到的 RGB 复原质量

利用 YUV444，YUV422，YUV420 采样后，经过上采样再转换成 RGB 后，得到的图像如下所示：

图 13: 不同采样格式得到的 RGB 复原质量

从中可以发现，经过三种采样格式得到的 RGB 图与原图几乎没什么区别。因此，我们可以发现人眼对亮度，也就是 Y 分量更敏感，而适当的降低色度（U）和浓度（V）的带宽，并不会影响图像的视觉质量。所以我们在实际应用中可以通过 YUV 采样减少数据量，从而降低数据传输的成本。

3 Task3

3.1 问题描述

- 输入一张灰度图（lena.bmp），设计三种不同类型的图像处理方法，使得处理后的图 PSNR 相近，但主观视觉效果完全不同，计算对应的 SSIM 值，并分析 PSNR 与 SSIM 的质量评价特点。
- 要求：给出三种图像处理方法的作用（可用库函数实现），算出 PSNR 值（PSNR 的计算可调用已实现的 PSNR 计算函数）和 SSIM（自己编程实现）。

3.2 方法原理

3.2.1 PSNR

见 1.2.3

3.2.2 SSIM

Structural Similarity Index，即结构相似性，视亮度、对比度和结构三个不同因素的组合，可以较好地反映人眼主观感受。值越大，视频质量越好。给定两张图片 x 和 y，其计算公式如下：

$$SSIM(x,y) = \frac{(2\mu_X\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

其中 μ_x 和 μ_y 分别是 x 和 y 的均值，σ_x^2 和 σ_y^2 分别是 x 和 y 的方差，σ_{xy} 是 x 和 y 的协方差，$C_1 = (k_1L)^2$ 和 $C_2 = (k_2L)^2$ 是两个常数，L 是像素值的范围，$k_1 = 0.01$ 和 $k_2 = 0.03$ 是两个默认值。在实际计算时，我们在图片上取一个 $N \times N$ 的窗口，不断滑动窗口进行计算，最后取平均值作为整张图片的 SSIM 值。对于 RGB 格式的图片，我们分别计算三个 channel 的 SSIM 值然后取平均值。

3.3 求解过程

3.3.1 三种图像处理方法

本任务选择了以下三种图像处理方式：

- 加高斯白噪声：高斯白噪声指的是概率分布服从高斯分布的一类噪声，其幅度分布服从高斯分布，而功率谱分布是均匀分布的。在图像上添加高斯白噪声会使得图像质量变差，对原像素的视觉效果产生较大干扰。在 matlab 中，对图像添加高斯白噪声可通过函数 imnoise，选定参数为“gaussian” 实现。

- 对比度拉伸：可以通过函数 imadjust(I,[lowin; highin],[lowout; highout]) 实现，实现的功能是将原图像处于 [lowin: highin] 区间的像素值映射到 [lowout: highout]，二区间外的像素值按大小二值化为 (lowout, highout)，这样实现的效果是图像像素间的差异被放大，也就是增强了图像的对比度，原图一些对比度较低的细节得以被增强。

- 散焦模糊：可以通过对图像施加圆形区域均值滤波实现，可借助函数 fspecial，指定滤波类型为’disk’。添加了散焦模糊的图像顾名思义会呈现出焦点发散式的模糊效果。

上述图像处理方式的实现代码如下所示：

```
% 处理方法一：加高斯噪声
img1 = imnoise(img,'gaussian',0.01,0.01);
img1_double = double(img1);
PSNR1 = PSNR(img_double, img1_double);
SSIM1 = SSIM(img, img1);

% 处理方法二：对比度拉伸
img2 = imadjust(img,[0.2 0.8],[0 1]);
img2_double = double(img2);
PSNR2 = PSNR(img_double, img2_double);
SSIM2 = SSIM(img, img2);

% 处理方法三：散焦模糊
PSF = fspecial('disk',10);
img3 = imfilter(img,PSF,'conv','symmetric');
img3_double = double(img3);
PSNR3 = PSNR(img_double, img3_double);
SSIM3 = SSIM(img, img3);
```

图 14: 三种图像处理方式

3.3.2 PSNR 和 SSIM 计算

PSNR 的计算我们可以直接调用 Task1 中的计算函数，我们需要注意的是要先把图像像素值转为浮点数。SSIM 的计算我们遵循 3.2.2 中的公式，用一个 8×8 的滑窗依次计算每个小窗的 SSIM 值，然后取平均。均值和方差的计算我们均按照其定义，利用两次 mean 可值计算出矩阵所有元素的均值。具体实现代码如下所示：

```
[height, width] = size(img1);
img1 = double(img1);
img2 = double(img2);
SSIM = 0; N=8;
C1 = (0.01*255)^2;
C2 = (0.03*255)^2;
total = (height+N-1)*(width+N-1);
% 用N*N滑窗分别计算局部SSIM并取平均值
for i=1:(height-N+1)
    for j=1:(width-N+1)
        window1 = img1(i:i+N-1,j:j+N-1);
        window2 = img2(i:i+N-1,j:j+N-1);
        mu_x = mean(mean(window1));
        mu_y = mean(mean(window2));
        sigma_x2 = mean(mean((window1-mu_x).^2));
        sigma_y2 = mean(mean((window2-mu_y).^2));
        sigma_xy = mean(mean((window1-mu_x).*(window2-mu_y)));
        SSIM = SSIM + (2*mu_x*mu_y+C1)*(2*sigma_xy+C2)/...
            ((mu_x^2+mu_y^2+C1)*(sigma_x2+sigma_y2+C2));
    end
end
SSIM = SSIM / total;
```

图 15: 三种图像处理方式

3.4 结果分析

在经过三种不同的图像处理方式后，我们得到了视觉效果完全不同的三张图片，如下所示：

图 16: 三种图像处理方式处理后的图片

我们分别计算它们与原图的 PSNR 和 SSIM，结果如下：

	加高斯噪声	对比度拉伸	散焦模糊
PSNR	20.0189	18.8161	23.0738
SSIM	0.2948	0.7707	0.5585

表 2: PSNR 与 SSIM

从表中可以看出，三张图片与原图的 PSNR 相近，但却呈现出了完全不同的视觉效果。这说明 PSNR 能够基于图像像素给出它们的相似度，但是并不能很好的符合人眼的视觉特性。例如，我们一般认为对比度拉伸得到的视觉效果要好于另外两种，但却获得了最低的 PSNR 值。相反地，SSIM 呈现的结果更符合我们的主观视觉评价标准，即加了噪声的图片与原图的差别是最大的，而对比度拉伸后的图片与原图最贴近，这说明 SSIM 的评价能较好的符合人眼的

视觉特性，相比 PSNR 能更好地反映人的主观评价。

4 总结与反思

通过本次作业的三个 task，我对图像处理与质量评价方面的知识有了更深入的理解，感受到一张图片不只是单纯的像素矩阵和 RGB 格式，其背后蕴含着更丰富的不同维度特性。利用不同图像处理方法，我们可以挖掘出图像主观视觉之外更丰富的信息。而且经过实际 matlab 操作，我在这过程中解决了许多图像处理的实际问题，包括图像的读取玉战士，数据类型、矩阵格式的转换，matlab 库函数的灵活运用，这进一步增强了我对 matlab 这一强大数学处理工具的使用熟练度，也收获了许多宝贵经验。