

# 数字图像处理第一次作业报告

姓名：邓浩然

学号：516030910401

邮箱：[dhrdeng@sjtu.edu.cn](mailto:dhrdeng@sjtu.edu.cn)

## 第一题：下采样、双线性插值与PSNR

### 一、实验原理

#### 1. 下采样

图像下采样可以通俗地理解成缩小图像，又称为降采样。其目的有两个：1) 使得图像符合显示区域的大小；2) 生成对应图像的缩略图。假设我们有一幅图像 $X$ ，其大小为 $M \times N$ ，并且我们想对该图像进行 $s$ 倍下采样，即使图像缩小 $s$ 倍，这里要在长宽上同时缩小 $s$ 倍。那么最终得到的下采样图像的大小就为 $(M/s) \times (N/s)$ 。

一般情况下， $s$ 是 $M$ 与 $N$ 的公约数，那只需要对原图进行采样即可。若 $s$ 不是 $M$ 与 $N$ 的公约数，那则需要对原图先抽样，再插值。比如原图的分辨率为 $4 \times 4$ ，要求降采样得到 $3 \times 3$ 的图像，则需要先通过简单的采样得到 $2 \times 2$ 的图像，再通过插值得到 $3 \times 3$ 的图像。

#### 2. 上采样与双线性插值

图像上采样就是放大图像，也可以将之称为图像插值，其主要目的是放大原图，从而使得图像可以显示在更高分辨率的显示设备上。

在数学上，双线性插值是有两个变量的插值函数的线性插值扩展，其核心思想是在两个方向分别进行一次线性插值，如图1：

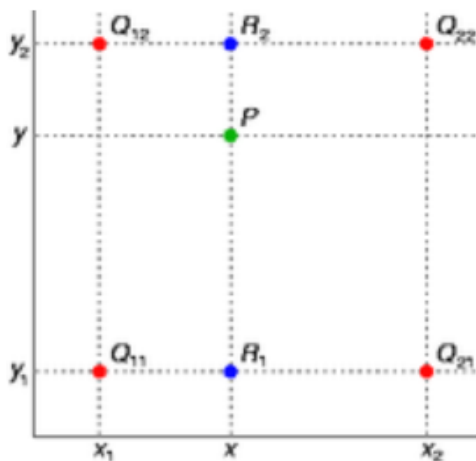


图1 双线性插值原理图

所以双线性插值的结果为：

$$f(x, y) \approx \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y) + \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1)$$

#### 3. PSNR

峰值信噪比经常用作图像压缩等领域中信号重建质量的测量方法，它常简单地通过均方差（MSE）进行定义。两个 $m \times n$ 单色图像 $I$ 和 $K$ ，如果一个为另外一个的噪声近似，那么它们的均方差定义为：

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \|I(i, j) - K(i, j)\|^2$$

峰值信噪比定义为：

$$PSNR = 10 \cdot \log_{10} \left( \frac{MAX_I^2}{MSE} \right) = 20 \cdot \log_{10} \left( \frac{MAX_I}{\sqrt{MSE}} \right)$$

其中， $MAX_I$ 是表示图像点颜色的最大数值，如果每个采样点用 8 位表示，那么就是 255。

## 二、算法实现

### 1. 降采样

为了将图像转为(M/s)×(N/s)大小，我使用了两种方法，一种是用Matlab自带的函数imresize(); 另一种是自己实现的方法，即在原图上取样。比如：

```
1 rose = imread('rose.tif');
2 [row,line]=size(rose);
3 % downsampling by 2
4 rose2 = rose(1:2:row, 1:2:line);
```

由于题目给的rose.tif图片分辨率是1024\*1024，并且题目要求的降采样的倍数为2、4、8、16、32（都是1024的因数），我可以对原图在行和列两个空间进行简单的取样。

### 2. 双线性插值

双线性插值也有两种方法来实现，一种就是用Matlab自带的函数imresize(); 另一种就是自己写的实现的双线性插值函数**bilin.m**。

该函数的实现分为四个部分：

1. 根据差值之后的图像的尺寸创建新图像；
2. 将待插值的原图扩展一圈：即将原图的最外围的四条边都平移至扩展出的那圈的四条边上，然后还剩四个角，就用原图的四个角来补充。我这么做的原因就是为了处理边缘插值的情况。
3. 通过对新的图像的每个像素点进行遍历，然后映射到待插值的图的某个“正方形”中，并确定该像素点在这个“正方形”中的具体位置，然后就可以插值了。具体的映射方法见我的具体代码**bilin.m**。
4. 量化为灰度值。

### 3. PSNR的计算

直接利用原理中的公式，先计算两张图片的MSE，在再通过MSE计算相应的PSNR值。

具体的实现在我自己编写的函数**psnr\_calculator.m**里面。

## 三、结果分析

### 1. 使用自己实现的函数的程序的结果



图2 降采样2倍 (512x512)



图3 降采样2倍之后插值2倍 (1024x1024)









	
图4 降采样4倍 (256x256)	图5 降采样4倍之后插值4倍 (1024x1024)
	
图6 降采样8倍 (128x128)	图7 降采样8倍之后插值8倍 (1024x1024)
	
图8 降采样16倍 (64x64)	图9 降采样16倍之后插值16倍 (1024x1024)
	
图10 降采样32倍 (32x32)	图11 降采样32倍之后插值32倍 (1024x1024)

表1 经过不同处理的图像的PSNR值 (使用自己写的计算PSNR的函数)

通过某倍数的降采样与插值的图	2	4	8	16	32
PSNR值（使用自己实现的psnr计算）	28.33	24.90	21.76	18.86	16.70
PSNR值（使用Matlab自带psnr计算）	28.33	24.90	21.76	18.86	16.70

2. 使用Matlab自带函数的程序的结果







	
图12 降采样2倍（512x512）	图13 降采样2倍之后插值2倍（1024x1024）
	
图14 降采样4倍（256x256）	图15 降采样4倍之后插值4倍（1024x1024）
	
图16 降采样8倍（128x128）	图17 降采样8倍之后插值8倍（1024x1024）

	
图18 降采样16倍 (64x64)	图19 降采样16倍之后插值16倍 (1024x1024)
	
图20 降采样32倍 (32x32)	图21 降采样32倍之后插值32倍 (1024x1024)

表2 经过不同处理的图像的PSNR值

通过某倍数的降采样与插值的图	2	4	8	16	32
PSNR值 (使用Matlab自带psnr计算)	37.68	34.31	30.42	26.80	23.35
PSNR值 (使用自己实现的psnr计算)	37.68	34.31	30.42	26.80	23.35

### 3. 实验结果分析

针对降采样、线性插值与PSNR的分析 (不包括实现方式之间的对比)：

- 1. 通过对比上述实验结果，不难看出，对于同一张灰度图，下采样的倍数越大，分辨率越低，图像越模糊。
- 2. 通过对比上述实验结果，不难看出，双线性插值能让图片更加平滑。
- 3. 对于同一张图片做的不同倍数的下采样得到的图片做相应倍数的双线性插值，对于倍数越高的图片，双线性插值的平滑效果越明显。
- 4. 由于下采样本身是一个丢弃信息的过程，所以无论做多少倍的双线性插值，都无法让图片恢复原来的信息。
- 5. 对于同一张图片做的不同倍数的下采样得到的图片做相应倍数的双线性插值，对于倍数越高的图片，相应于原图的PSNR值越低，也就代表失真越多。很显然，下采样的倍数越大，丢失的信息越多，也就失真越多，PSNR值越低。

对比两种实现方式：

- 1. 当下采样的倍数比较低的时候，下采样的结果看起来几乎没有区别。然而当下采样率非常大的时候，如下图所示，我自己实现的下采样的结果并不如Matlab自带的imresize()函数下采样的结果光滑。我的分析是，因为imresize()函数的scale参数既可以大于1，也可以小于1，也就是说这个函数既可以下采样，也可以（双线性）插值。那么它在下采样的时候，并不是像我一样单纯的隔若干行，隔若干列地对原图采样，而是加上了一些插值的操作。因此虽然两种方式得到的结果分辨率相同，但是图片的光滑程度不同（imresize()的结果更平滑）。



图22 自己实现的16倍下采样 (64x64)



图23 Matlab自带的16倍下采样 (64x64)

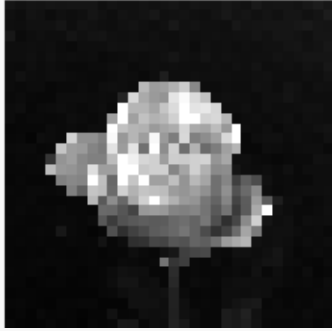


图24 自己实现的32倍下采样 (32x32)

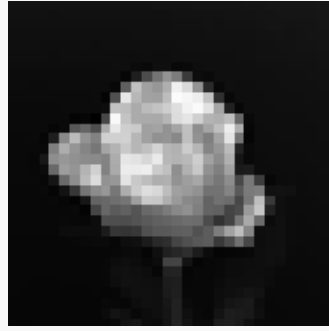


图25 Matlab自带的32倍下采样 (32x32)

2. 由于我两个实验的代码 (**t1.m**, **t1\_usingBuiltInFunc.m**) 并没有对同一张下采样之后的图进行不同的双线性插值 (比如说同样是16倍插值, 图9与图19的原图是不同的)。为了公平对比, 我同时对图18与图20进行两种不同的双线性插值, 一种是我自己实现的, 一种是Matlab自带的`imresize()`函数。结果如下图所示。可以发现, 两种方式的结果从肉眼几乎看不出区别, 严谨起见, 我计算了两种方式得出的图的PSNR值 (使用Matlab自带的`psnr()`函数计算), 结果如表3所示。可以看出, 使用Matlab自带的双线性插值算法得到的图片的PSNR值更高, 也就是失真越少, 效果更好。



图26 降采样16倍之后插值16倍（自己实现）



图27 降采样16倍之后插值16倍（Matlab自带）



图28 降采样32倍之后插值32倍（自己实现）



图29 降采样32倍之后插值32倍（Matlab自带）

表3 不同插值的PSNR值的比较

PSNR值	自己实现的双线性插值	Matlab自带的双线性插值
降采样16倍之后插值16倍	22.48	26.80
降采样32倍之后插值32倍	19.94	23.35

3. PSNR计算的结果刚开始令人出乎意料。从实验结果来看，我自己实现的PSNR值的计算函数与Matlab自带的psnr()函数有比较大的差异，具体体现在我实现的计算PSNR值的函数计算出的结果要大于Matlab自带的psnr()函数（输入的两个参数完全一样）。我不太能理解其中的原因，因为PSNR的计算公式是固定的，不应该出现偏差。后来我发现，我输入的图片都是灰度图，数据格式都是uint8，但是psnr的计算是需要用到除法，对数等运算的，所以我应该先把输入的两张图片变成double格式，才能精确计算。修改之后，果然结果就和Matlab自带的psnr()函数完全一样了。

```
1 % convert the input into double format
2 raw = double(raw);
3 new = double(new);
```

## 第二题：RGB、HSI与YUV的转换

### 一、实验原理

## 1. HSI空间

当人观察一个彩色物体时，用色调、饱和度、亮度来描述物体的颜色。HSI (Hue-Saturation-Intensity(Lightness), HSI或HSL) 颜色模型用H、S、I三参数描述颜色特性，其中H定义颜色的波长，称为色调；S表示颜色的深浅程度，称为饱和度；I表示强度或亮度。在HSI颜色模型的双六棱锥表示，I是强度轴，色调H的角度范围为 $[0, 2\pi]$ ，其中，纯红色的角度为0，纯绿色的角度为 $2\pi/3$ ，纯蓝色的角度为 $4\pi/3$ 。

从RGB空间到HSI空间的转换公式如图2所示：

### ■ RGB → HSI

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

图2 RGB->HSI转换公式

## 2. YUV空间

YUV，是一种颜色编码方法。常使用在各个视频处理组件中。YUV在对照片或视频编码时，考虑到人类的感知能力，允许降低色度的带宽。YUV是编译true-color颜色空间 (color space) 的种类，Y'UV, YUV, YCbCr, YPbPr等专有名词都可以称为YUV，彼此有重叠。“Y”表示明亮度 (Luminance、Luma)， “U”和“V”则是色度、浓度 (Chrominance、Chroma)。

为节省带宽起见，大多数YUV格式平均使用的每像素位数都少于24位。主要的抽样 (subsample) 格式有YCbCr4:2:0、YCbCr4:2:2、YCbCr4:1:1和YCbCr4:4:4。YUV的表示法称为A:B:C表示法：

- 4:4:4表示完全取样。
- 4:2:2表示2:1的水平取样，垂直完全采样。
- 4:2:0表示2:1的水平取样，垂直2:1采样。
- 4:1:1表示4:1的水平取样，垂直完全采样。

具体的采样格式如图 ()：



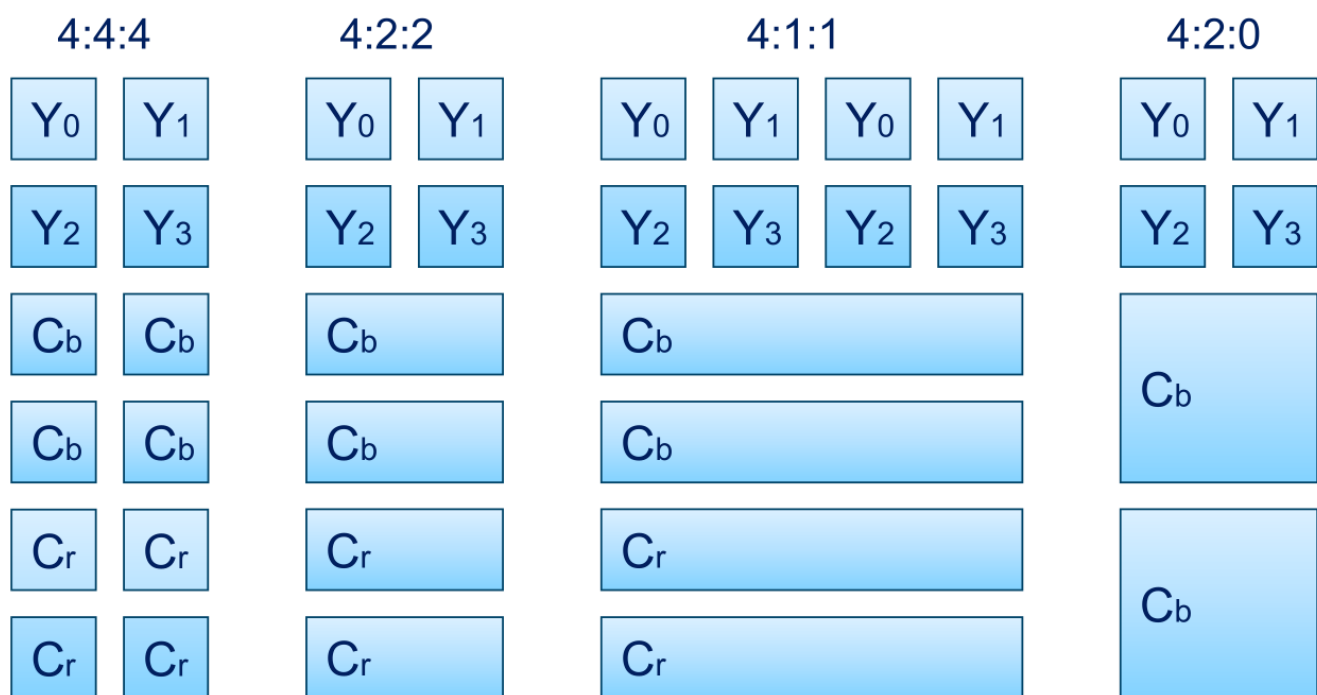


图 ( ) YUV444, 422, 411, 420采样

转换矩阵 (公式) :

$$Y = 0.30 \times R + 0.59 \times G + 0.11 \times B$$

$$C_B = 0.56 \times (B - Y)$$

$$C_R = 0.71 \times (R - Y)$$

## 二、算法实现

### 1. RGB到HSI的转换

题目不要求自己实现从RGB到HSI的转换，不过我还是实现了这个函数。利用实验原理中的转换规则，计算每个分量矩阵，得到了H、S、I矩阵，最后用cat()函数拼接成了完整的HSI空间的图像。

完整的代码我打包成了函数，在RGBtoHSI.m中。

### 2. RGB到YUV的转换

利用实验原理中的转换规则，计算每个分量矩阵，得到了Y, C<sub>B</sub>, C<sub>R</sub>矩阵。

## 三、结果分析

### 第三题：设计三种图像处理方法并分析

#### 一、实验原理

#### 二、算法实现

#### 三、结果分析