# **数字图像处理期末大项目**

15331288 唐玄昭 教务四班

# PSNR 的实现过程

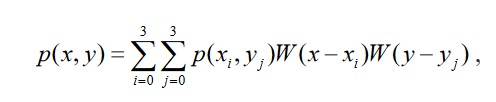
PSNR 其实仅仅根据所给的公式，带入计算即可，基本没有其他特殊问题

# SSIM 的实现过程

根据论文所给的公式，为了更好地计算 SSIM，我们这里采用加权平均数地方式来进行处理，即对于每一个窗口区间，分别求出这个窗口的 SSIM 值，利用高斯核赋予权值，本质上就是一个滤波的过程，在对每个窗口求出对应的SSIM值后，再整体相加取均值求得整张图片的 SSIM 值

**双三次插值简要思路**

开始的预处理阶段类似于双线性插值，即左上角对齐之后，将目标的像素点映射到原图的某一具体位置，此时可以通过



来对这个像素点进行赋值，通过取周围 16 个点，并取加权和，就可以得知目标的像素点的值，之后再进行直方图均衡化，便可以获得想要的结果。

# 超分辨率算法实现思路

其实所有的思路都是从作者的论文中获得的，这里我将分步骤将我所实现的流程描述出来：

# 将 HR 图像转成 LR 图像

这里的实现很简单，即利用论文的公式



将图片从高分辨率的图像经过滤波和下采样后，取得低分辨率的图片

# 取得 LR patch

从每一张图片中，我们都可以获得很多个 LR patch，它们是 7 \* 7 的像素块，即从最左上角的像素块出发，每次往后移动一个像素，从而获得无数多个 LR patch

# 获得 feature 和中心点

每一个 LR patch 去掉周围的 4 个点后，再减去均值，便可以得到对应的 feature，这个 feature 可以看作是 patch 的特征，之后会在聚类中使用到；而为了方便以后的使用，我把每一个 patch 的中心点保存下来，方便之后的使用，即方便之后寻找 LR 和 HR 的映射

# 聚类

考虑到电脑性能问题，这里选取第 3 步中获得的一部分样本进行聚类处理，并根据样本数目对每个类的中心点进行排序存储

# 获取映射函数

为了求得 LR 和 HR patch 之间的映射，这里对所有样本，取得对应地 LR 和 HR 的映射，并根据



求得回归系数，从而为之后的测试提供数据

# 测试数据

对于每一个图片，我们同样对图片进行 patch 划分，并将这里的每一个 patch 归类，根据所属类里面的回归系数，求出相应的高分辨率图像，由于高分辨率图像每个像素点可能会被重复计算，因此这里我们取均值进行还原。从论文中看，由于人眼对亮度的变化会对色彩更加敏感，因此我们对图像的亮度通道进行运算，其他的按照双三次插值求得对应 size 的图片即可

# 利用双三次插值得出的量化指标

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **PSNR(bicubic)** | **SSIM** | **PSNR(paper)** | **SSIM** |
| **Baboon** | 19.6493 | 0.6878 | 20.0809 | 0.7278 |
| **Barbara** | 23.2244 | 0.8243 | 24.4561 | 0.8716 |
| **Bridge** | 23.2377 | 0.8329 | 21.7358 | 0.7354 |
| **Coastguard** | 23.0952 | 0.5981 | 23.1233 | 0.6038 |
| **Comic** | 20.5337 | 0.7004 | 19.7205 | 0.6607 |
| **Face** | 25.8120 | 0.6916 | 25.2670 | 0.7281 |
| **Flowers** | 24.6430 | 0.7820 | 21.4602 | 0.6592 |
| **Foreman** | 26.2704 | 0.8834 | 23.7133 | 0.8730 |
| **Lenna** | 27.3890 | 0.9030 | 23.5090 | 0.8389 |
| **Man** | 24.4872 | 0.8681 | 22.0856 | 0.7835 |
| **Monarch** | 27.1593 | 0.9549 | 24.3900 | 0.9216 |
| **Pepper** | 26.3828 | 0.9202 | 23.5561 | 0.8581 |
| **Ppt3** | 20.6385 | 0.9261 | 19.2832 | 0.8900 |
| **Zebra** | 24.0677 | 0.8946 | 22.1743 | 0.8732 |
| **average** | 24.0422 | 0.8191 | 22.4682 | 0.7875 |

**\*上结果采用自带函数，我的函数和自带的有一点点偏差，为保证准确性，采用函数库的结果进行分析。**

# 结果分析

实验结果均保存在target文件目录下，详情请看readme。

在bicubic 的结果中，我发现结果与原图的差别很大，同时，越复杂的地方效果越差，而平滑一点的地方效果会较好。这与该算法的过程也相关。

而在论文提到的算法中，当取得的 cluster 数目越多，结果也会越好，细节部分也更好。从结果看来，效果较 bicubic 来说，还是更好一些的。同时，我也尝试了不同的 cluster 数目来计算，结果也十分显然，数目越多，还原度就越高，因为较多的数目会带来更多的分类，因此 patch 的所属类别也就越精细。 而虽然最后的均值小于bicubic，但是可以看到我这里的大部分数据还是优于bicubic的，同时，对比图片的视觉效果是更好的，然而可能因为开始时候进行的滤波操作，导致最后的结果在计算的数据结果上存在弱势。

在我实现的PSNR和SSIM代码中，虽然做到了每一步都严格按照公式和论文的结果，但是最后的结构却还是和自带的库的结果相差一点，这里分析原因应该是部分细节与实现不同，在测试时候发现我的ssim的结果，如果是两幅一样的图比较，结果仍然是1，所以说明我的代码依然具有参考价值。

# 改进

从自然景物来看，一个景物一定是连续的，因此一个点与周围的值一定是相关的，所以插值算法可以考虑周围更多的点来还原，而论文中的算法可以考虑对类别进行区分，比如动漫图片的还原不能使用自然景物的聚类来计算，效果一定是不好的，所以除了用更多的类别来计算，对图片类别的区分和利用也可以是较为重要的一方面。