

# 数字图像处理第一次大作业报告

计 43 2014011330 黄家晖

2017 年 4 月 24 日

## 1 基础部分：点处理

### 1.1 实现方式

该部分仅需要读取图片中的每个点的颜色数据进行相应的处理即可。我实现了明度、对比度、Gamma 值、直方图均衡化和直方图匹配，并附加实现了饱和度调整。

在编程中，我使用了 Python 语言，利用 `scipy` 库用于图片数据的读入，并使用 `numpy` 库进行数组操作。虽然使用了这两个科学计算库，但是所有的操作均只使用了简单的循环、访问等操作，不涉及更高级的库调用。

### 1.2 运行说明

运行环境为 Windows10，并配有 Anaconda3（对应 Python 3.5）版本作为 Python 的默认解释器。只要在命令行中输入 `python pointProcessing.py <args>` 即可运行或查看帮助：

```
1 >python pointProcessing.py -h
2 usage: pointProcessing.py [-h] [--brightness amount] [--contrast amount]
3                               [--gamma amount] [--histogramEQ HISTOGRAMEQ]
4                               [--histogramMatch HISTOGRAMMATCH]
5                               [--saturation amount]
6                               filename output
7
8 positional arguments:
9   filename           The filename of the input picture
10  output            The filename of the output picture
11
12 optional arguments:
13  -h, --help          show this help message and exit
14  --brightness amount
15  --contrast amount
16  --gamma amount
17  --histogramEQ HISTOGRAMEQ
18  --histogramMatch HISTOGRAMMATCH
19  --saturation amount From -1 to 1
```

### 1.3 实验结果

**明度变化** 明度变化效果图如图 1 所示。左图增加了 80 明度，右图减小了 80 明度。

**对比度变化** 对比度变化如图 2 所示。对比度斜率小于 1 则表示减小对比度，大于 1 表示增加对比度。



图 1: 明度变换演示效果图



图 2: 对比度变换演示效果图

**Gamma 变化** Gamma 变化如图 3所示。当  $\gamma > 1$  的时候图像变得更加明亮，反之图像变得更加暗淡，但是最亮处和最暗处极值处得到了保留。

**直方图均衡化** 对于图片的每一个通道进行直方图均衡化，效果如图 4所示。

**直方图匹配** 在网上寻找了一张与原图色调搭配不尽相同的图片，与鹦鹉图片进行直方图匹配，最终结果如图 5所示，可见鹦鹉图片变得与目标图色彩分布更加接近。

**饱和度变化** 饱和度变化比较复杂一些，但基础思想是使得每个颜色分量的色域变宽或变窄，使得图片中的颜色更加丰富或是更加暗淡（趋于灰度），具体实现在 `change_saturation`函数中。最终效果如图 6所示。

## 2 高级部分 1：图像重采样

### 2.1 实现方式

在本部分中，我实现了最近邻采样、双线性采样、三次线性插值采样的上采样和下采样方法，并附加实现了基于 Lanczos 滤波器的采样方式（选择的参数  $a = 3$ ）。使用 PSNR 进行度量，定量分析采样方法优劣。



图 3: Gamma 变换演示效果图



图 4: 直方图均衡化效果图

## 2.2 运行说明

运行环境与上一节相同。只要在命令行中输入 `python resampling.py <args>` 即可运行或查看帮助：

```

1 >python resampling.py -h
2 usage: resampling.py [-h] filename output method r c
3
4 positional arguments:
5   filename      The filename of the input picture
6   output        The filename of the output picture
7   method        nn, bi, cu or la
8   r             height
9   c             width
10
11 optional arguments:
12   -h, --help    show this help message and exit

```

诚然，使用 Python 编写的代码运行速度相比于 C++ 要慢上一个数量级，每次缩放图片（尤其是三次插值和 Lanczos 滤波器）需要耐心等待片刻。

## 2.3 实验结果

在本部分的实验中，对于大小为  $288 \times 288$  的原图，首先通过各种方法进行降采样得到  $128 \times 128$  和  $64 \times 64$  的图像，再使用相同的方法上采样到  $288 \times 288$ ，并与原图做 PSNR 的



图 5: 直方图匹配效果图



图 6: 饱和度变换演示效果图

对比, 得出采样方法的还原度。同时, 还对我的算法的花费时间进行了测量。为了还原图片质量, 在 pdf 中输出了图片原始大小。

**近邻采样** 使用近邻采样效果如图 7所示。上采样之后, 出现了明显的马赛克现象, 还原度不高。

**双线性采样** 使用双线性采样效果如图 8所示。上采样之后, 与近邻采样相比, 图片曲线更加顺滑。



图 7: 近邻采样效果图

**三次线性插值采样** 三次插值采样得到的图片如图 9所示。由于三次线性插值充分考虑到了插值点的导数平滑性，因此图片相比二次插值更加平滑。

**Lanczos 滤波器采样** 使用了 Lanczos 滤波器采样结果如图 10所示。与之前的采样方法不同的是，Lanczos 滤波器使用卷积的方法进行采样，并且根据二维性质可以先对行卷积、再对列卷积，所以可以加快代码运行的速度。使用了这种卷积核之后图像与三次插值比较接近。但可能由于图片原因，PSNR 的值并没有比三次插值高。

**量化分析对比** 四种插值算法的分析对比如表 1和表 2所示。在我的实现中，三次插值是求解矩阵计算的，所以运行起来速度比较慢，如果改用卷积核的方式理论上可以大大加快速度。从 PSNR 分析中可以看出，其数值大概与人眼感知相近。对于最近邻算法，放大之后会

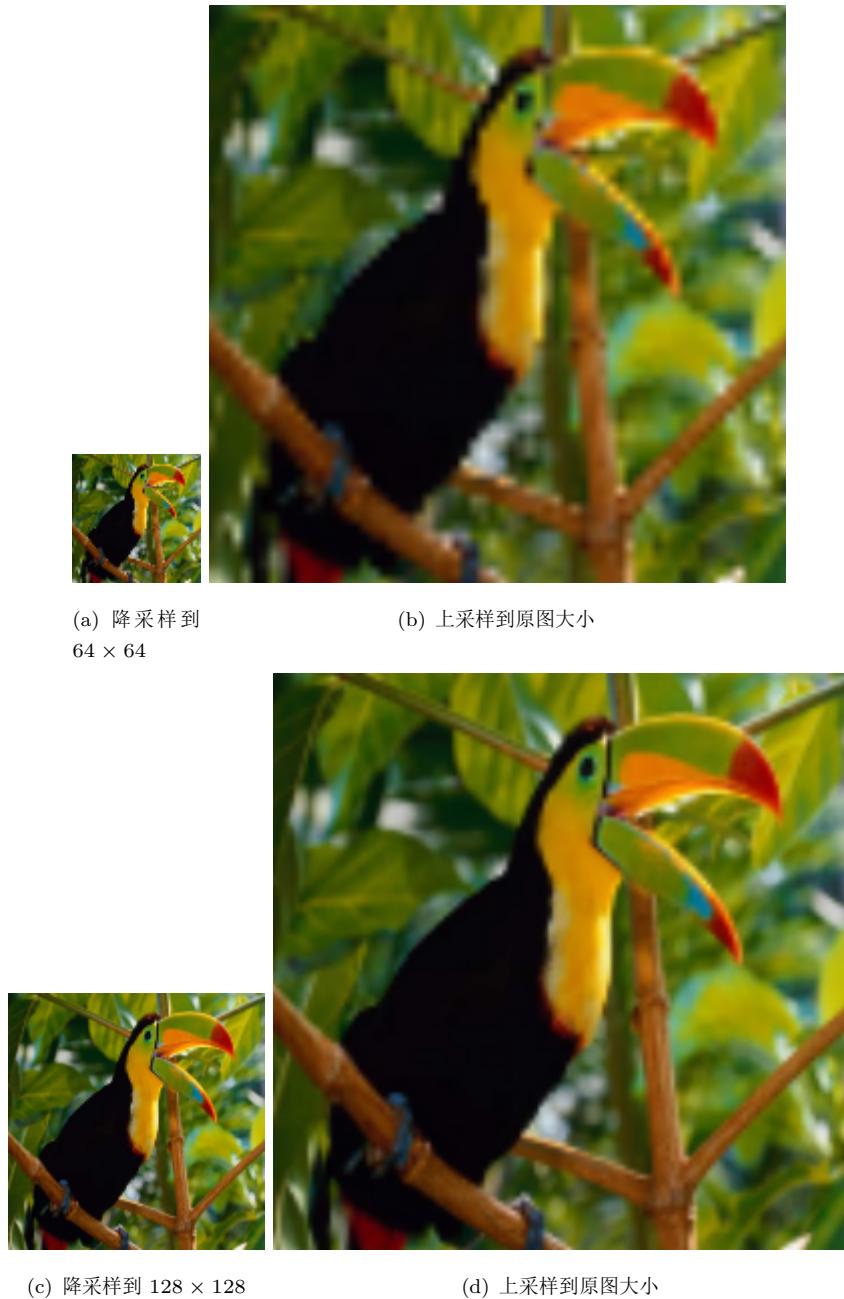


图 8: Bilinear 效果图

有明显的颗粒感，还原度不高，因此其峰值信噪比也低一些，说明与原图有一定差距。而二次插值到最复杂的 Lanczos 滤波器方法都对图像的平滑性进行了一定的考虑，因此也更接近原图，信噪比较高。

另外，从  $64 \times 64$  的图上采样和从  $128 \times 128$  的图上采样之后得到的目标图片 PSNR 差距在 2dB-3dB 之间（大约是 100-1000 倍），这说明下采样之后图片的信息损失会大大增多。

当然，PSNR 的值并不能完全客观正确地反映一个缩放方法的好坏，只能作为一个不错的参考标准。设想如果一个纯杂色的图片进行下采样和上采样，那样平滑过渡的上采样方法不一定能够捕捉高频的颜色变化，导致 PSNR 的值领先不明显。



图 9: Bicubic 效果图

花费时间 (下采样/上采样)	Nearest Neighbour	Bilinear	Bicubic	Lanczos Filter
64 × 64	0.01s / 0.14s	0.05s / 0.98s	0.76s / 14.81s	2.22s / 10.33s
128 × 128	0.03s / 0.14s	0.19s / 0.96s	2.88s / 14.36s	5.38s / 12.21s

表 1: 各个采样方法花费时间对比

### 3 高级部分 2: 伪造识别

#### 3.1 实现方式

实验要求是找出照片中丢失的人，我的思路大概分为如下几步：



图 10: Lanczos 采样效果图

PSNR	Nearest Neighbour	Bilinear	Bicubic	Lanczos Filter
64 × 64	31.62 dB	32.07 dB	32.48 dB	32.13 dB
128 × 128	33.49 dB	35.33 dB	36.68 dB	35.83 dB

表 2: 各个采样方法 PSNR 对比

- 找出照片中的所有人；
- 对含有人的选框找出不变性特征；
- 对找出的特征进行匹配。

在实现找出照片中所有人的时候，我首先对照片进行人脸检测，采用的是 Viola-Jones 的机器学习方法。人脸数据库采用了 Yale 大学的 155 张人脸数据，非人脸数据根据从互联网搜集空教室关键词爬取并提取的一些小型 patch，为了完成本任务，部分人脸数据和非人脸数据取自于 `left.jpeg` 和 `right.jpeg`，所使用的数据请参见 `Dataset` 文件夹，其中 Yale 大学人脸数据在 `data.txt` 文件内。训练和检测使用的 Haar 特征为 Viola 等人 2001 年论文中提出的 4 个特征形状，为了简化实现，`Adaboost` 算法中我并没有实现论文中的 Cascade，直接使用 80 个弱分类器线性组成强分类器，所以在人脸检测的时候比较慢。训练过程大约持续 8 个小时左右（如果用 C++ 实现可能更快）。

由于 `Adaboost` 的性质，训练集内的准确率会不断上升，输出如下：

```

1 2017-04-18 22:28:25 INFO: Trying to add a new classifier. Iteration 14
2 2017-04-18 22:28:25 INFO: 0 Features finished for this iteration
3 2017-04-18 22:29:35 INFO: 20000 Features finished for this iteration
4 2017-04-18 22:30:44 INFO: 40000 Features finished for this iteration
5 2017-04-18 22:31:53 INFO: 60000 Features finished for this iteration
6 2017-04-18 22:32:08 INFO: Current correct rate = 0.9775
7 2017-04-18 22:32:08 INFO: Newly added feature id = 18473
8 2017-04-18 22:32:08 INFO: Trying to add a new classifier. Iteration 15
9 2017-04-18 22:32:08 INFO: 0 Features finished for this iteration
10 2017-04-18 22:33:16 INFO: 20000 Features finished for this iteration
11 2017-04-18 22:34:25 INFO: 40000 Features finished for this iteration
12 2017-04-18 22:35:34 INFO: 60000 Features finished for this iteration
13 2017-04-18 22:35:49 INFO: Current correct rate = 0.9894
14 2017-04-18 22:35:49 INFO: Newly added feature id = 27299
15 2017-04-18 22:35:49 INFO: Trying to add a new classifier. Iteration 16
16 2017-04-18 22:35:49 INFO: 0 Features finished for this iteration
17 2017-04-18 22:37:00 INFO: 20000 Features finished for this iteration
18 2017-04-18 22:38:08 INFO: 40000 Features finished for this iteration
19 2017-04-18 22:39:16 INFO: 60000 Features finished for this iteration
20 2017-04-18 22:39:31 INFO: Current correct rate = 0.9894
21 2017-04-18 22:39:31 INFO: Newly added feature id = 41690

```

实际测试中，可能由于数据集选取的原因，人脸检测的准确率并不高，因此继续对数据进行处理，考虑到中国人脸的色相在 0 到 15 之间（240 一个循环），因此对色相不符合要求的小块进行筛选。最后对重叠的小块进行去重，就能得到人脸的包围框。为了增强匹配结果，又手动偏移了一个人脸检测框的位置。

接着，需要为每个框选出的同学提取相应的特征，这里为了简化没有实现 Sift。在特征选取的时候使用的是 FAST 算法，仅仅考虑每个像素和周围的 9 个点（FAST-9）的插值确定点的重要性；提取特征使用了 SIFT 的一部分，对于每个特征点周围的  $16 \times 16$  个像素计算了梯度图，并分箱成  $4 \times 4$  的网格，每个网格含有 8 个主方向。因此，本特征提取仅仅考虑到了亮度、对比度不变性，并没有考虑到尺度不变性，这是因为考虑到本实验中尺度变化不大所使用的近似。另外，分箱能在一定程度上解决拍摄角度不一样造成的仿射变换。选出的 FAST 特征点如图 12 所示。

在比较特征的时候，一个选框内有许多特征，因此设立阈值，在两个选框内相似的特征个数大于这个阈值的时候，就认为这两个选框匹配（是同一个人）。每个框将与其匹配值和最大的对应框进行匹配。

最终检测出的同学和匹配情况如图 13 和 14 所示。其中框的位置即是人脸的位置，红色框则是两幅图中匹配的同学，蓝色框是未匹配的同学，可以认为是右图中出现，但左图中“消失”的同学。如果按照 `readme.txt` 中的要求，不考虑左图左半部分和右图右半部分匹配错误的情况，仅考虑重叠区域，算法能够较好工作——算法找出了四个消失的同学（参见

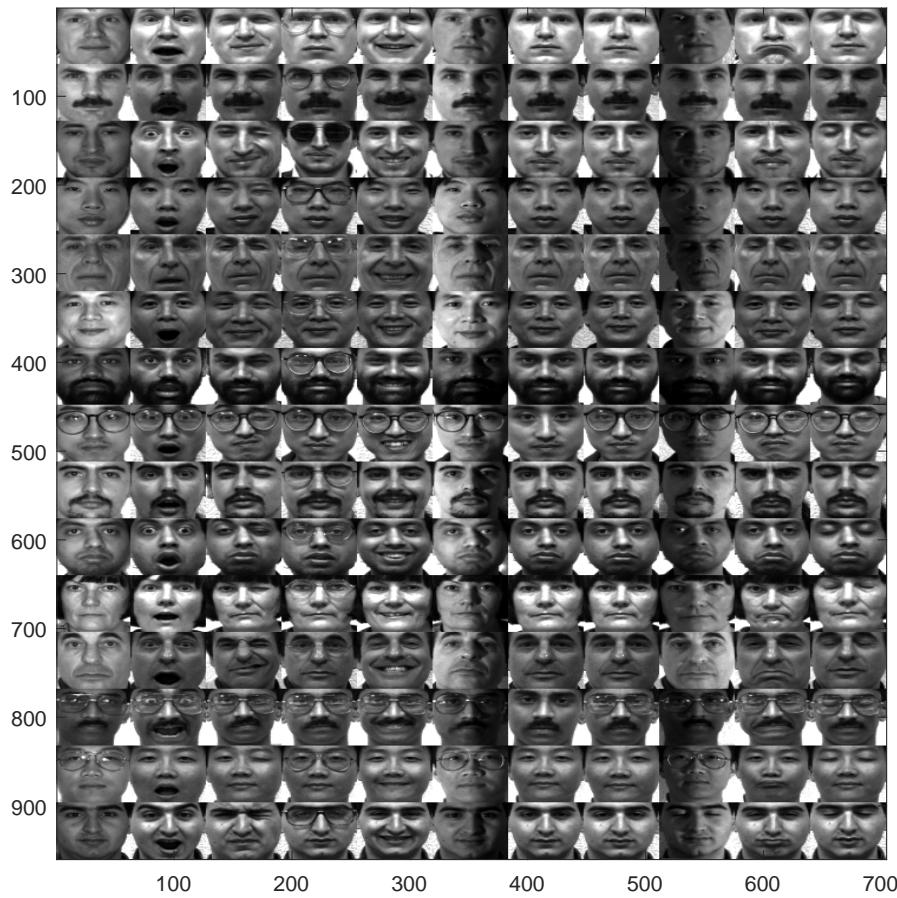


图 11: Yale 数据集

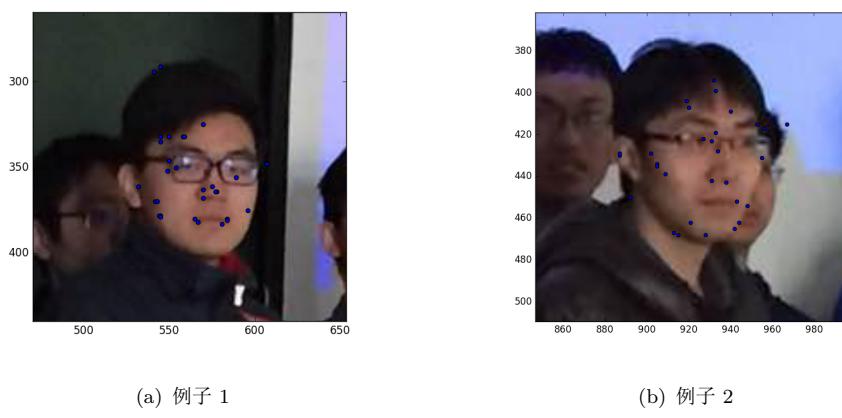


图 12: FAST 算法特征点

右图从左向右数 4 个蓝框), 仅有一个找错了; 但是如果整体来看, 匹配的效果则不是很好, 可以看出左图的左半部分有很多同学没有在右图出现, 却也被标记成了红色, 另外左图的右半部份也有许多同学的脸没有被检测出来。

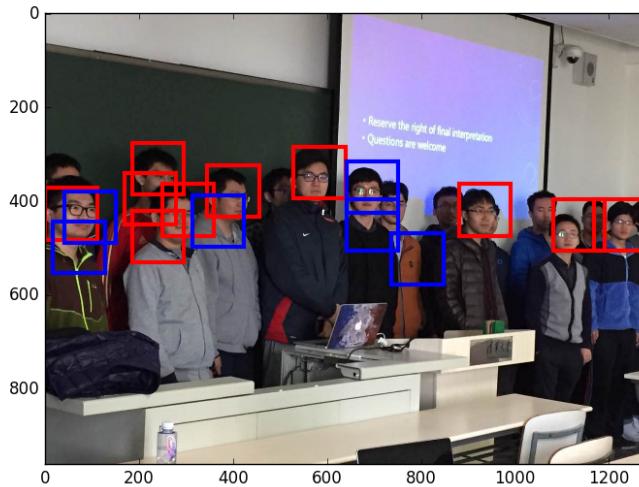


图 13: 左图中的匹配情况

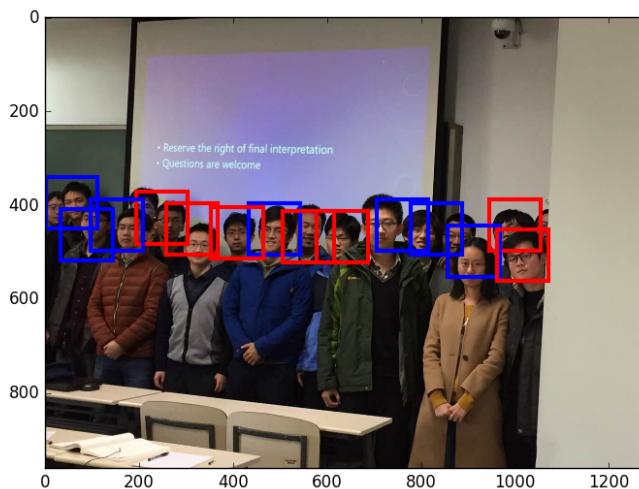


图 14: 右图中的匹配情况

正确的标记如图 15所示。如果要使得算法更加准确，可以考虑在检测人脸时采用可变的检测窗口大小，另外在选取特征点特征向量的时候也应该更加仔细（例如在分箱的时候加入权重等方法）。

在完成本实验时比较吃力，需要阅读许多文献资料和网上的开源实现，但诚然由于能力有限，最终完成的代码效率不高，运行一次（不含训练）需要 2 分钟左右，且效果也不是很理想。不过我也确实很努力尝试了许多新的方法，还希望助教给予鼓励。

### 3.2 运行说明

在提交的文件中包含了数据集，可以直接开始训练，不过训练的时间较长，如果需要的话，请输入：

```
1 > python forgeDetection.py train
```

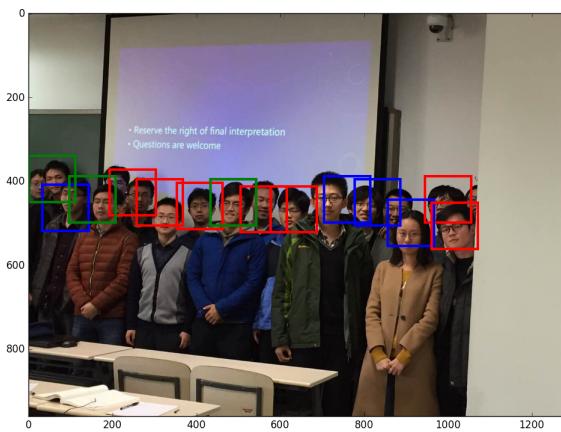


图 15: 正确匹配被绿色框框出

耐心等待，会生成 `snapshot.pkl`文件，即为分类器模型文件，里面包含了若干个弱分类器的线性组合。在提交的代码中，包含了已经训练好包含有 120 个分类器的模型文件，可以直接使用。

如果要开始运行代码，请输入：

```
1 > python forgeDetection.py test
```

程序即会调用 `matplotlib`依次绘出图 13、图 14和图 15。

如果不能运行，请联系我：[huangjh14@mails.tsinghua.edu.cn](mailto:huangjh14@mails.tsinghua.edu.cn)