

统计作业3 - PCA

学号：16337060

姓名：丰泽霖

代码见随附的压缩包，或者[GitLab](#)

此次使用 Haskell + C++ + MatLab 完成。

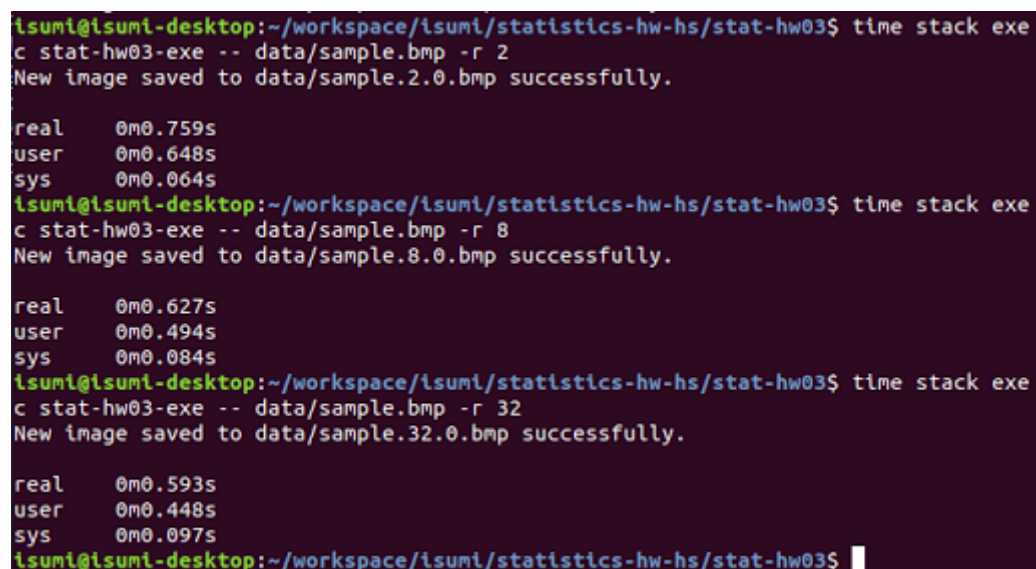
结果展示

编译

```
stack build stat-hw03
```

运行

```
cd stat-hw03  
stack exec stat-hw03-exe -- -r 8 data/sample.bmp
```



```
lsuni@lsuni-desktop:~/workspace/lsuni/statistics-hw-hs/stat-hw03$ time stack exe  
c stat-hw03-exe -- data/sample.bmp -r 2  
New image saved to data/sample.2.0.bmp successfully.  
  
real    0m0.759s  
user    0m0.648s  
sys     0m0.064s  
lsuni@lsuni-desktop:~/workspace/lsuni/statistics-hw-hs/stat-hw03$ time stack exe  
c stat-hw03-exe -- data/sample.bmp -r 8  
New image saved to data/sample.8.0.bmp successfully.  
  
real    0m0.627s  
user    0m0.494s  
sys     0m0.084s  
lsuni@lsuni-desktop:~/workspace/lsuni/statistics-hw-hs/stat-hw03$ time stack exe  
c stat-hw03-exe -- data/sample.bmp -r 32  
New image saved to data/sample.32.0.bmp successfully.  
  
real    0m0.593s  
user    0m0.448s  
sys     0m0.097s  
lsuni@lsuni-desktop:~/workspace/lsuni/statistics-hw-hs/stat-hw03$
```

独立的两个短横线之后的是命令行参数。你可以使用 `--help` 查看格式，即：`stack exec stat-hw03-exe -- --help`。参数 `-r` 表示压缩率，位置参数（positional argument）表示输入的文件路径，可以是相对于当前工作目录。

生成的三张图片以及原图见随附的目录 `pictures`

Notes

经过测试的平台：Ubuntu 18.04, stack 1.9.1, MATLAB R2018b, gcc-7.3.0

请先阅读README中Configure一节

另外，编译过程中可能提示缺少一些C语言库，那是某些Haskell函数库想要调用的，`stack` 不负责解决这些依赖。使用 `apt` 等包管理器，依照错误信息中提到的库名，安装那些库即可。

运行前，需要设 `LD_LIBRARY_PATH` 环境变量，以告知系统的动态链接器 `ld` MATLAB动态库的位置。例如：

```
export LD_LIBRARY_PATH=/home/isumi/opt/MATLAB/R2018b/extern/bin/glnxa64
```

不建议把这个环境变量添加到 `.bashrc` 等位置，否则会污染其他程序

另外，建议预先启动MATLAB并执行 `matlab.engine.shareEngine()`，再运行本程序。这样，本程序会使用这个共享计算引擎。否则，程序将在后台启动一个新的MATLAB引擎，那是很慢的，相当于打开一次桌面版MATLAB。

思路

本次作业目标是使用主成分分析来有损压缩图片。作业提示说了要把输入的图片切割成 16×16 的小块，每块是一个observation，块中的每个位置是一个variable，即共有256个variable。

对图像的分割不是困难的事。我们首先来看最核心的用 `pca` 减少变量数目以及复原的步骤。

用MATLAB表达的话，压缩过程即为

```
[coeff, score] = pca(ingredients, 'NumComponents', 2)
avgs = mean(ingredients)
```

这里 `ingredients` 是11行4列的矩阵。

`coeff`, `score`, `avgs` 三者都需要存储。不过，随样本规模增加，`coeff` 和 `avgs` 的字节

数不会增加，所以主要存储的是 `score`，它占用的存储空间是 $\frac{\text{input size}}{2}$ 。所以总体的压缩率近似为2。

解压过程只是简单的矩阵运算：

```
score * coeff' + repmat(avgs, size(score, 1), 1)
```

到这里为止，核心思路已经很清楚了。另外需要注意一点：我们只需要 `pca` 这个函数即可，其他函数用Haskell的 `hmatrix` 中提供的即可。

实现

为了从Haskell调用MATLAB的 `pca` 函数，我们需要借助C++，因为MATLAB提供了C++编程接口。代码层面，Haskell代码借助Foreign Function Interface调用满足c calling convention的函数，C++代码通过 `extern "C"` 提供C接口，同时在C++代码内部调用MATLAB的C++ API。最终，Haskell和C++代码在编译期静态链接在一起，并动态链接MATLAB提供的若干shared object。

C++ 对Haskell提供的函数如下：

```
extern "C" _Bool isumi_stat_pca(  
    const double *data,  
    size_t variables,  
    size_t observations,  
    size_t num_components,  
    double *coeff,  
    double *score  
);
```

对应的Haskell binding

```
foreign import ccall unsafe "isumi_stat_pca"  
c_pca :: Ptr CDouble -- ^data  
      -> CSize -- ^variables  
      -> CSize -- ^observations  
      -> CSize -- ^num components  
      -> Ptr CDouble -- ^(result) coeff  
      -> Ptr CDouble -- ^(result) score  
      -> IO CBool
```

经过封装，最终得到函数 `pca` 和 `pcaInv`

```
pca :: Matrix Double -- ^observations
    -> Int -- ^number of components to reduce to
    -> PCAResult
pca m n = PCAResult coeff score
  (LA.fromList . fmap (mean . LA.toList) . LA.toColumns $ m)
  where
    (coeff, score) = pca' m n
```

```
pcaInv :: PCAResult
    -> Matrix Double
-- score * coeff' + repmat(avgs, size(score, 1), 1)
pcaInv (PCAResult coeff score inputAvg) =
  (score <> LA.tr coeff) `LA.add`
  LA.fromRows (replicate (LA.rows score) inputAvg)
```

以上部分位于 `src/Isumi/StatisticsHW/Ch03/PCA.hs` 以及 `csrc/`

接下来实现对图片的压缩和复原。这里面有一个问题，就是图片分块之后，在使用PCA压缩之前要把这些块串起来，也就是会丢失掉图片中一行有多少块这一信息。所以在压缩时把原图宽度一并存储。

```
data CompressedMat
  -- | (result of PCA, columns of original matrix)
  = CompressedMat PCAResult Int
```

这里增加的 `Int` 就是原图宽度。根据这个信息便可以复原图片。

输入的图片是RGB8类型，对每种颜色的矩阵单独压缩，单独解压，再合并到一起。我这里 `JuicyPixel` 库直接读入的类型是RGB8，另外用Linux命令行工具 `identify` 检测文件类型也是 `sRGB` 类型，所以我就把这个图片当做RGB处理了，虽然看起来它只是个灰度图。当然，可以强制转换成灰度图，但是既然不同的工具都认为它是RGB，我也不再做预处理。

```
lsuni@lsuni-desktop:~/workspace/lsuni/statistics-hw-hs$ identify stat-hw03/data/sample.bmp
stat-hw03/data/sample.bmp BMP3 512x512 512x512+0+0 8-bit sRGB 256c 263KB 0.000u 0:00.009
```

最后得到这两个函数：

```
-- | Compress an RGB8 image
compressImage :: Image PixelRGB8
    -> Double -- ^compression ratio
    -> Maybe CompressedImage
compressImage pcas r = fmap CompressedImage
  . traverse (compressMatrix r) . rgb8ImageToMatrices $ pcas
```

```
-- | Decompress an RGB8 image
decompressImage :: CompressedImage -> Maybe (Image PixelRGB8)
decompressImage (CompressedImage cms) =
    matricesToRGB8Image =<< traverse decompressMatrix cms
```

主程序中，组合这两个函数就可以实现压缩并重建的过程了：

```
compressImage img compressionRatio >=> decompressImage
```