中山大學
SUN YAT-SEN UNIVERSITY

《计算机视觉与模式识别》
Experiment 2
Testing Report

学院名称	:	数据科学与计算机学院
学生姓名	:	吴若宇
学号	:	14301030
专业（班级）	:	14级计算机科学与技术
时间	:	2017 年 3 月 19 日

PART 1

灰度图像和彩色图像的直方图均衡

1、 测试环境

1. 操作系统：Ubuntu 15.04
2. IDE：Sublime Text3 + Terminal

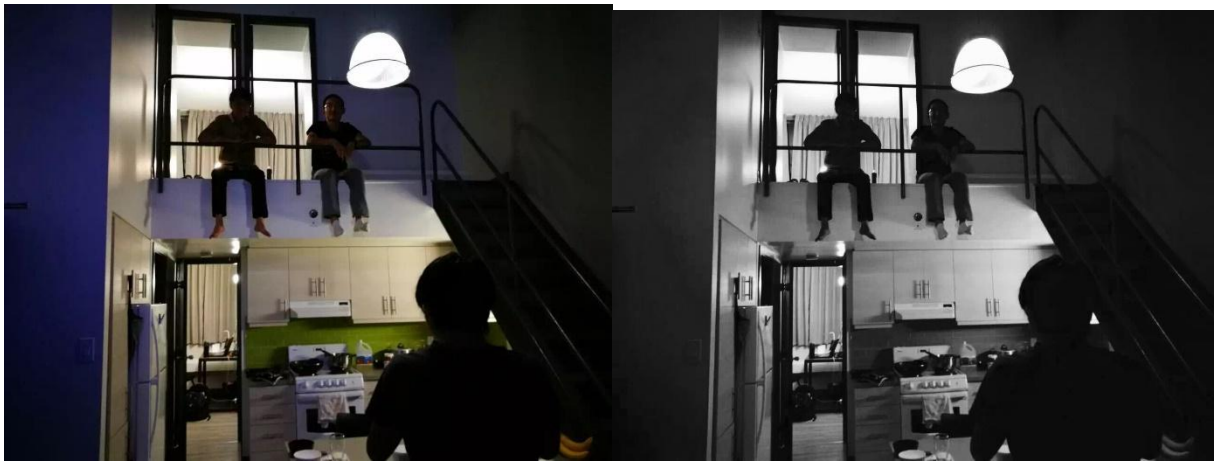
2、 测试数据及其灰度图

为了让实验具有普遍性，所以挑取了不同光照条件下的照片（都是用我手机拍的）。为了让实验效果更好，所以选取的很多都是拍的不好的照片。

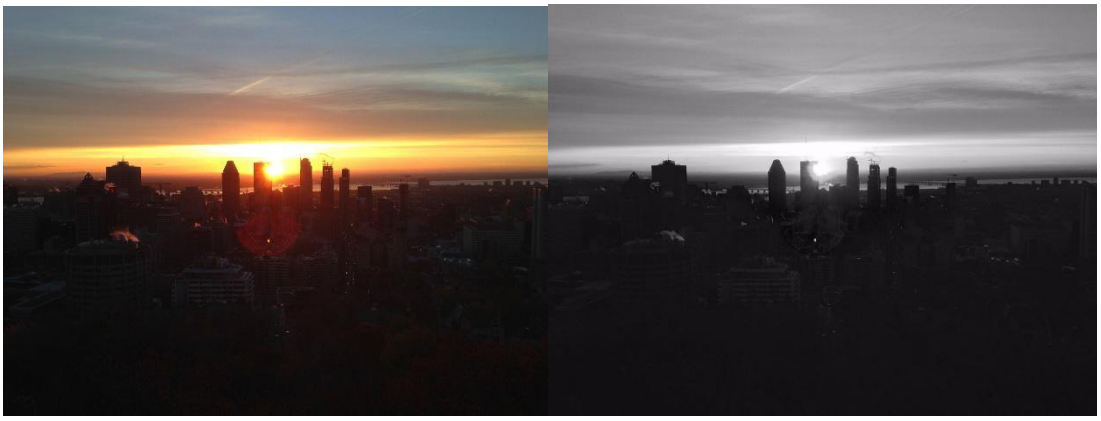
其中 1、9、11 是在正常光照条件下的图片。2、3、4、5、6、7、8 是光线比较暗或者是单点光源下的照片，10 是曝光时间过长的照片。

1. lena.jpg

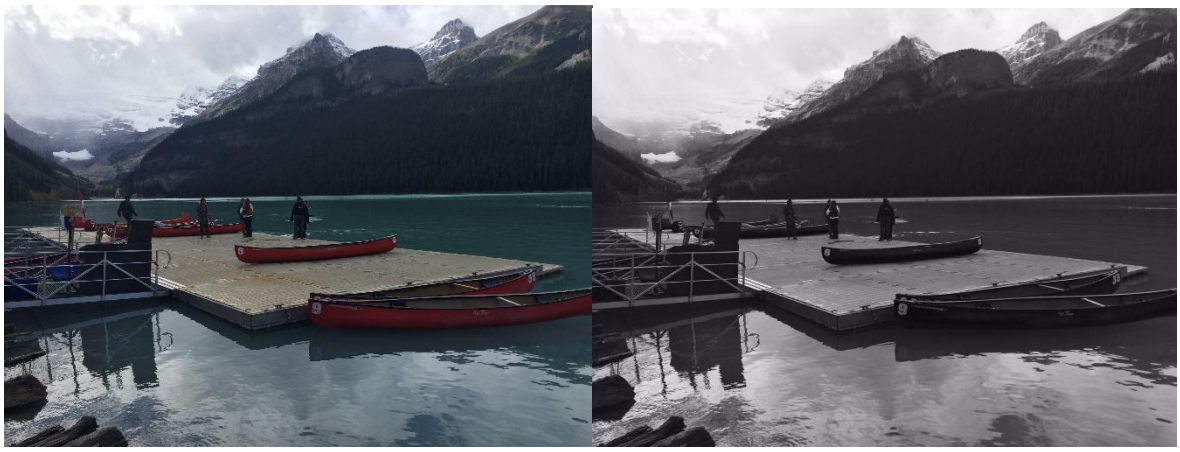
2. Dormitary.jpg



3. Montreal2.jpg



4. Lake Louis1

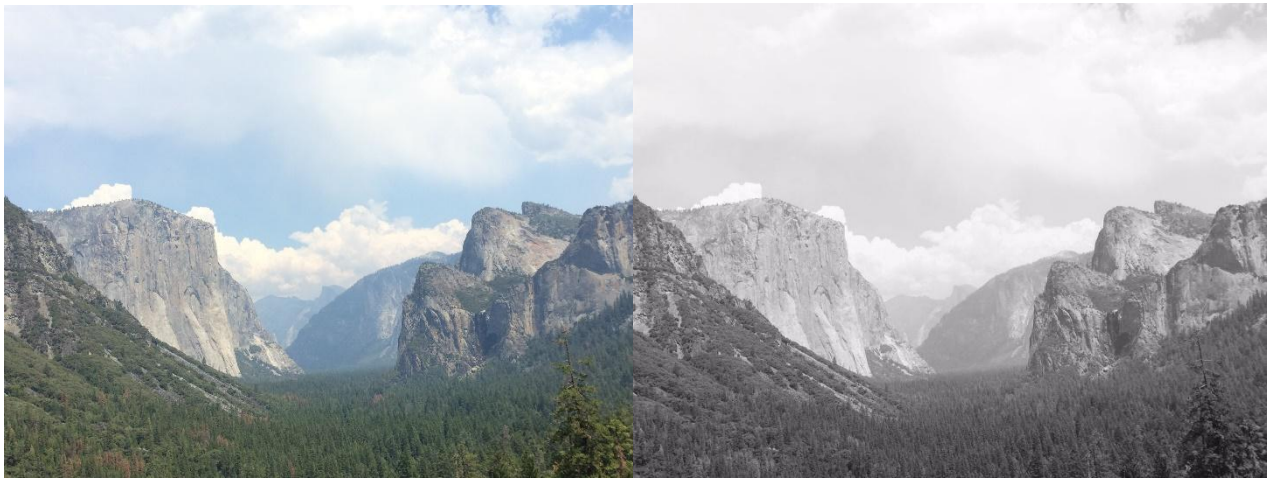


5. Tree before Cameron

6. Aquarium

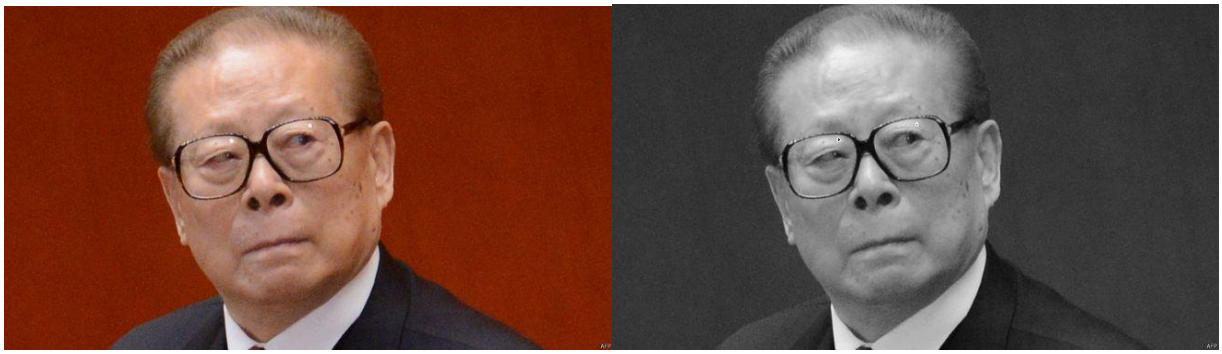
7. Notre-Dame Basilica

8. Yosemite1



9. Road in UBC

10. Old man



3、 测试过程

1.用 CImg 的接口写直方图均衡的代码

```
static std::array<int, 256> getHistogram(CImg<double>& img ,int channel = 0) {
    std::array<int, 256> histogram = {0};
    cimg_forXY(img, x, y) {
        histogram[img(x, y, channel)] ++;
    };
    return histogram;
}

static CImg<double>& equalize(CImg<double>& img) {
    for (int channel = 0; channel < img.spectrum(); channel ++) {
        std::array<int, 256> histogram = getHistogram(img, channel);
        unsigned long cumulated = 0;
        for (int index = 0; index < 256; index ++) {
            cumulated += histogram[index];
            histogram[index] = cumulated;
        }
        cimg_forXY(img, x, y) {
            img(x, y, channel) = (255 * histogram[img(x, y, channel)]/(double)cumulated);
        };
    }
    return img;
}
```

2.写 python 脚本对图片进行自动处理，分别把灰度图像和彩色图像进行直方图均衡。

```
import os
import subprocess


def main():
    sourceImagePath = "../image/"
    targetImagePath = "../equalizedImage/"
    for imageName in os.listdir(sourceImagePath):
        args = ['./Ex3']
        args.append(sourceImagePath + imageName)
        args.append(targetImagePath + imageName)
        subprocess.call(args)


if __name__ == "__main__":
    main()
```

3.得到测试结果

4、 实验结果分析

1. 灰度图像的直方图均衡效果

1） lena.jpg

脸庞处的对比度显著的变高了,由于原本的图像偏暗，所以整体都变亮了，但是个人感觉图像效果没有变好而且有点不自然

2） Dormitory.jpg

由于原本的图像偏暗，所以整体都变亮了。在直方图均衡化之后，由于灰度级减少，所以可以看到斑状的东西。这幅图可以看到直方图均衡的优点。原本宿舍楼上两个人都看不清，但是处理过之后原本一片漆黑的脸部信息都显得非常明显。

3） Montreal2.jpg

和上图一样，整体变得清晰了，能看清城市的轮廓，在处理前基本上什么都看不清，但同样因为灰度级减少出现了黑斑

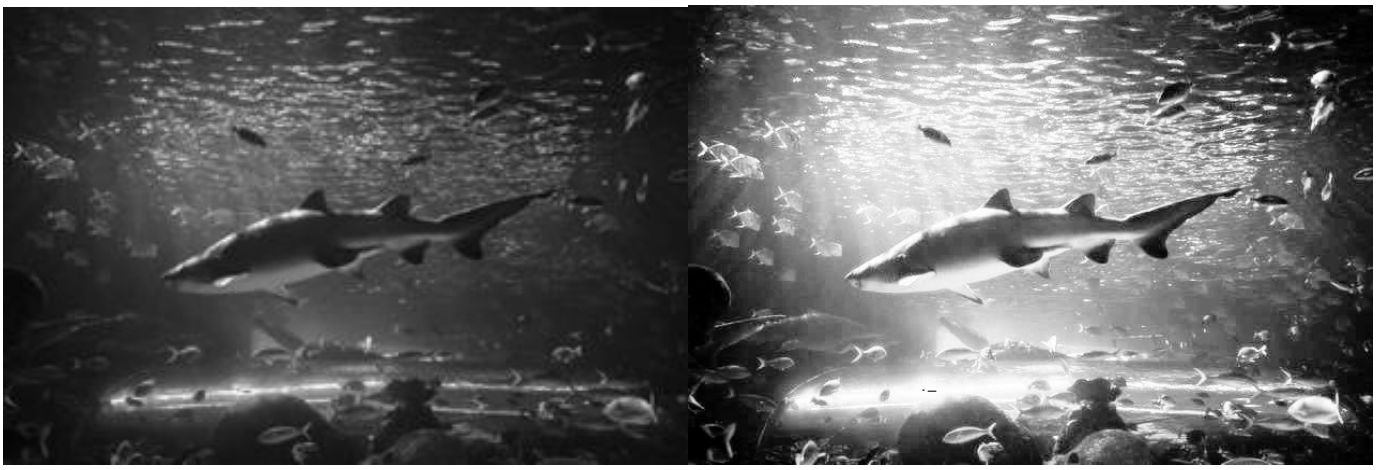
4） Lake Louis1.jpg

能看清背景中山上的树了，效果很好

5） Tree before Cameron.jpg

看清了树的细节，但也因为均衡化之后亮度提升中间反而看不清了。

6） Aquarium.jpg



处理的效果感觉不错，除了有点太亮了其余都挺完美的

7） Notre-Dame Basilica.jpg

效果非常的差，尤其是在画面中部，显著降低了有效信号的对比度。

这是因为原图背景非常的暗，不仅灰度级减少的比较多，而且与

原图色彩的差异也非常大。黑色的穹顶都变成白色的了。

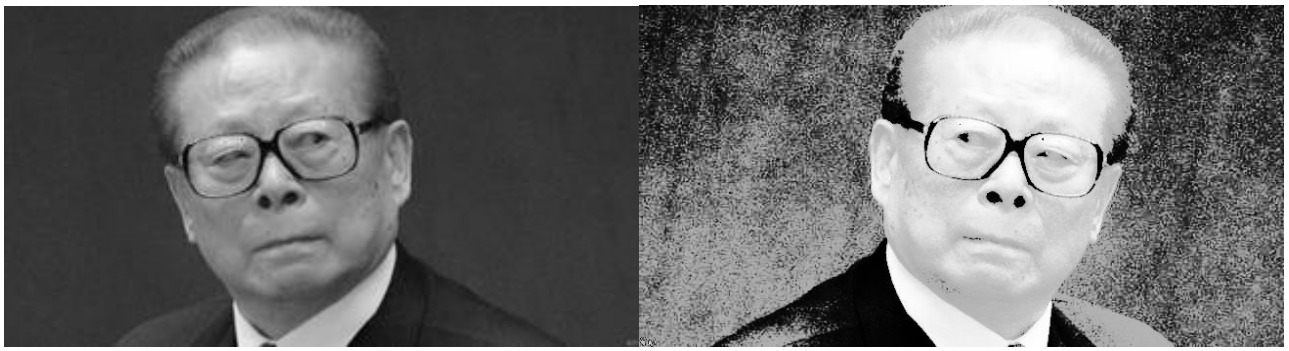
8） Yosemite1.jpg

这张图没有必要进行直方图均衡化处理，虽然对比度高了但是图片效果并没有提升。

9） Road in UBC.jpg

这张图是选取的唯一一张曝光过度的照片。可以看到，原本的图片除了太阳和路基本上什么都看不清。经过直方图均衡之后路面的情况和右边的草地都看清了。

10） Old man.jpg



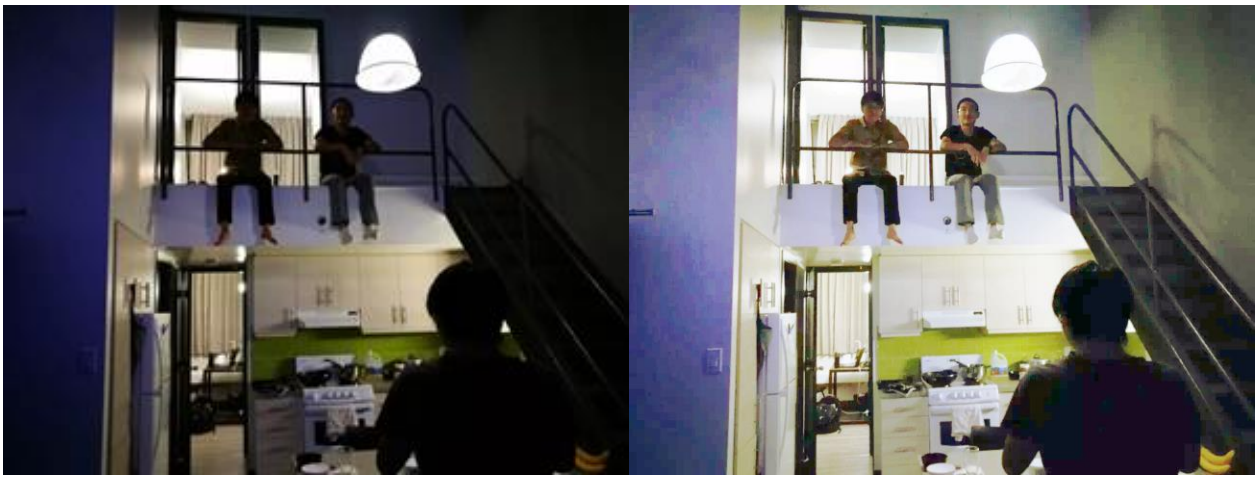
以长者的气场才不需要什么直方图均衡呢哼！

2. 彩色图像的直方图均衡效果

1) lena.jpg

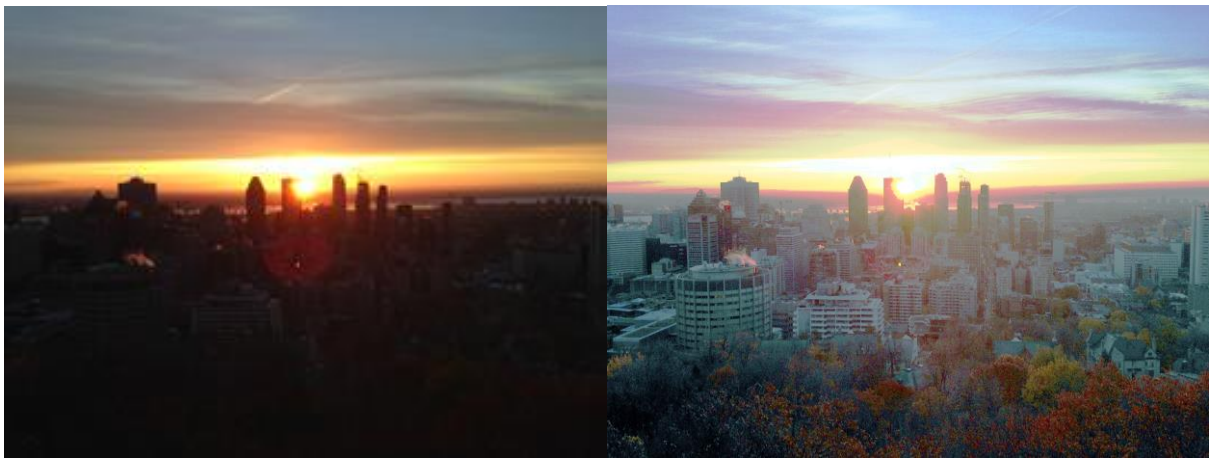
整个图像由暖色调变为冷色调，这是因为原本 R 通道的均值偏高 B 通道的均值偏低。而在均衡化之后，由于各个通道是被单独的处理，所以各个通道的均值都应该在 256/2 = 128 附近。通过程序验证，在处理前通道的均值为（134.889， 81.1391，68.2223），在处理之后通道的均值为（127.712， 128.152，128.378）。除了没有维持原图片的调之外没有什么不足的地方。

2) Dormitory.jpg



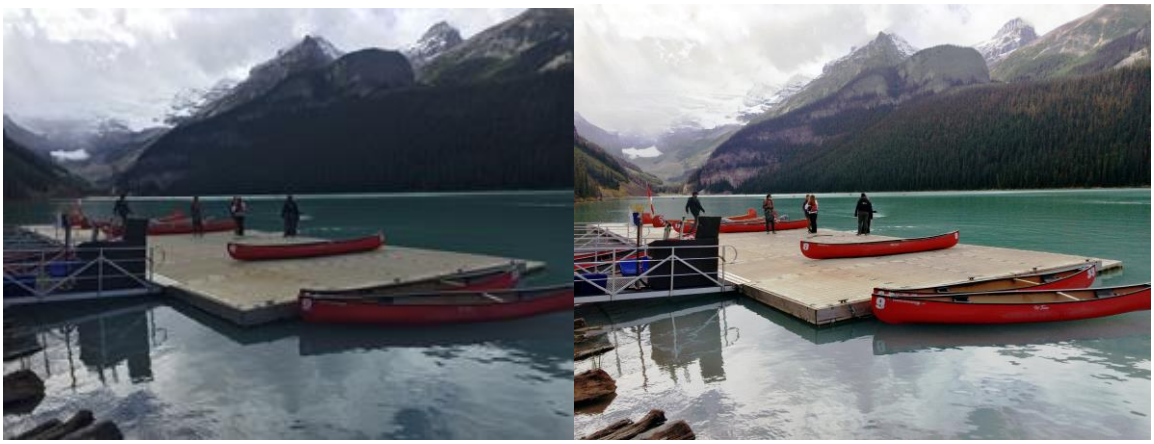
效果比较感人，把原本跪了的照片修复了，原本三个通道的均值都偏低，经过处理到 128 左右后，色彩明亮了且对比度比较高。楼上两个酥油和楼下的我都看得非常清晰，色彩也比较自然，体现了直方图均衡在**背景和前景都非常暗**的情况下的很好的效果。

3) Montreal2.jpg



同（2）一样效果感人，原因也一样

4) Lake Louis.jpg



同（2）一样效果感人，原因也一样

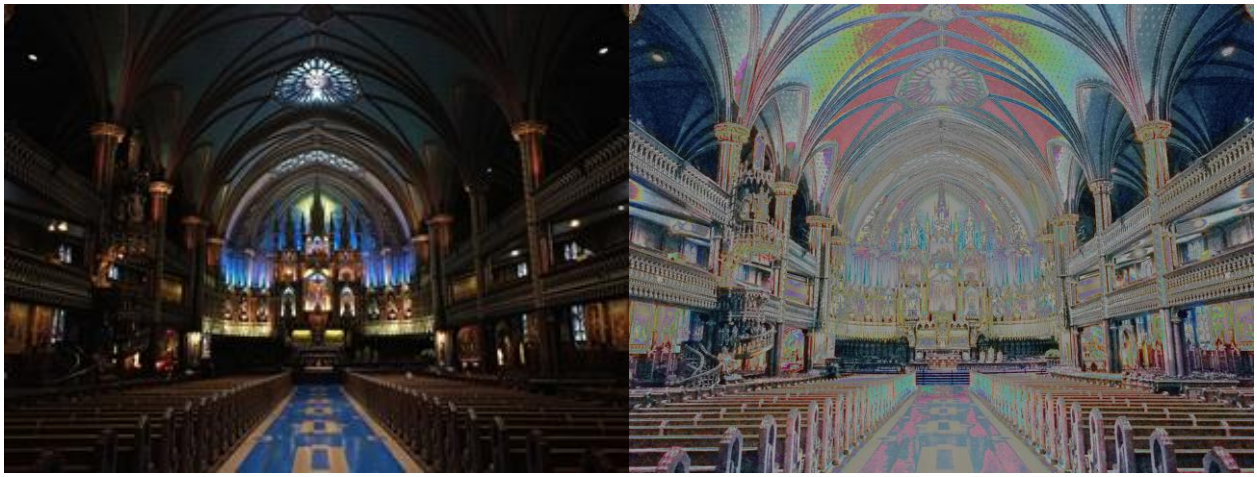
5) Tree before Cameron.jpg

注意背景处在处理后我们能看到天空了，在原图中树是偏白色的而在处理后的图像中能看出树是绿色的，这些是非常大的进步。但是因为原图中只有单点光源，在把周围变亮的同时也把中间调整的过亮，导致中间的效果不如处理前的，这是在之后的优化中可以考虑的

6) Aquarium.jpg

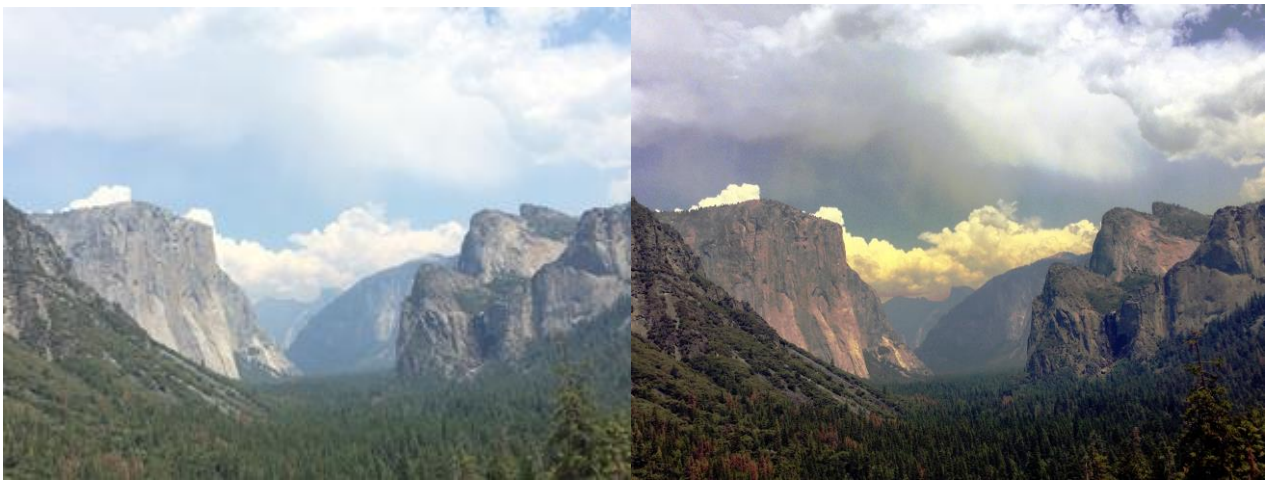
原图中 R 通道的值较低，G、B 通道的值较高，分别均衡化之后 R 通道高了很多所以效果变成了这样。这再次提醒我们直方图均衡应该考虑原图的色调，不能三个通道独立的处理。

7) Notre-Dame Basilica.jpg



也很毁，和上一张图的原因一样。同时直方图均衡降低了中间的对比度。

8) Yosemite1.jpg

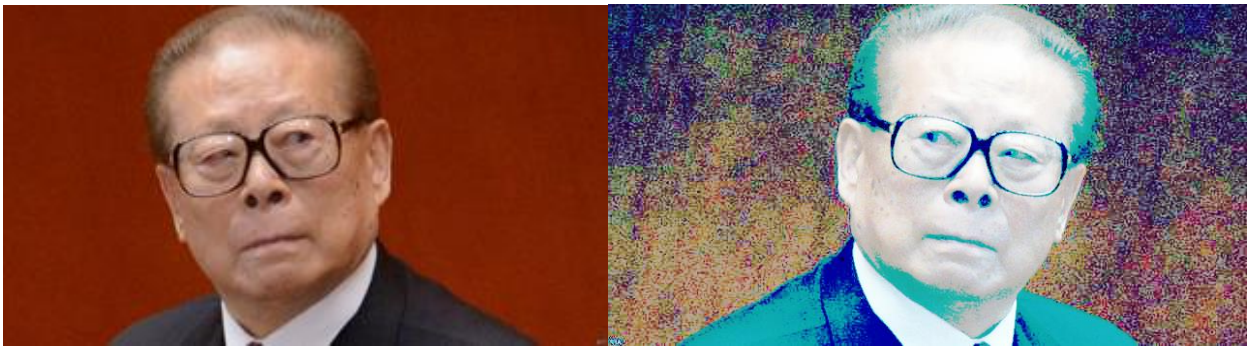


像这么正常的图片，没有必要使用直方图均衡。

9) Road in UBC.jpg

作为唯一一张曝光过足的图片，可以看到直方图均衡对于背景过亮的图片也非常有用，虽然阳光还是很闪，但是其他部分的对比度提高了变得好辨认了，路上的沟壑和路牌都变得清晰很多。

10) Old man.jpg



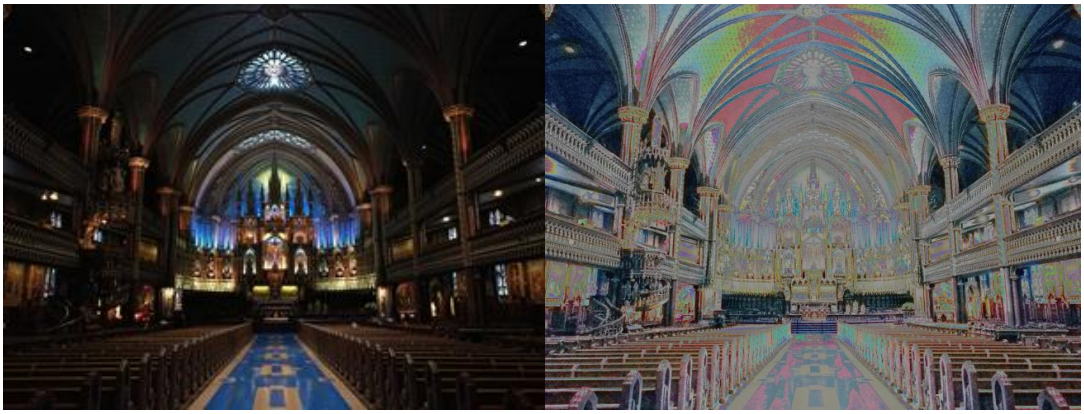
以长者的气场才不需要什么直方图均衡呢哼！

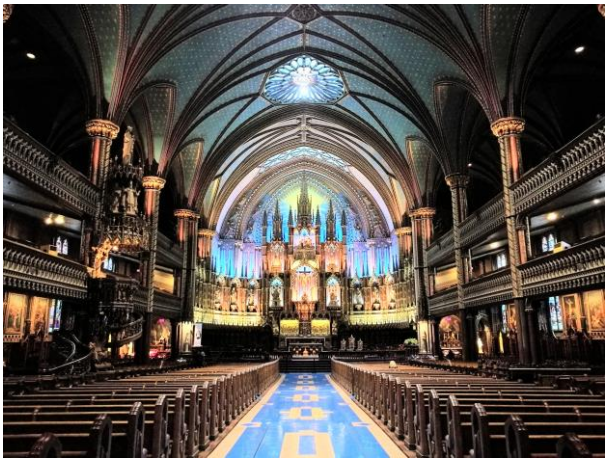
5、 总结及改进

直方图均衡这个方法首先足够简单，代码量不多就能实现。而且从上面的例子我们可以看出这种方法对于背景或者前景太亮或者太暗的图像非常有用，但是它在处理彩色图像的缺点还是比较显著的，它会改变原图片的色彩平衡，因为在 RGB 图像中三个通道是相互关联的，不满足分别作直方图均衡的前提条件。但是如果我们先把 RGB 图像转换为 Lab 颜色空间上的图像，a、b 两个通道与颜色有关而 L 通道代表亮度。如果我们只对亮度这一个通道进行直方图均衡就能做到在不影响图像的 hue 的基础上增加对比度。

我们对 Notre-Dame Basilica.jpg 进行处理，代码和得到的结果如下:

```
static CImg<double>& LabEqualize(CImg<double>& img) {
    img.RGBtoLab();
    std::array<int, 101> histogram = {0};
    cimg_forXY(img, x, y) {
        histogram[(int)img(x, y, 0)] ++;
    };
    unsigned long cumulated = 0;
    for (int index = 0; index < 101; index ++) {
        cumulated += histogram[index];
        histogram[index] = cumulated;
    }
    cimg_forXY(img, x, y) {
        img(x, y, 0) = 100 * histogram[int(img(x, y, 0))]/(double)cumulated;
    };
    img.LabtoRGB();
    return img;
}
```





第一张图片为原图，第二张为将 RGB 三个通道分别做直方图均衡的效果，第三张为转换成 Lab 再在 L 上做直方图均衡的结果。可以看到图三不仅效果比图二好很多，而且也比原图好很多，在没有改变 saturation 和 hue 的情况下使整张图片的对比度增大了，两侧都看得非常清晰（表示十分感动 0.0）。

参考： S. Naik and C. Murthy, "Hue-preserving color image enhancement without gamut problem," IEEE Trans. Image Processing, vol. 12, no. 12, pp. 1591–1598, Dec. 2003

PART 2

颜色转换

1、测试环境

1.操作系统：Ubuntu 15.04

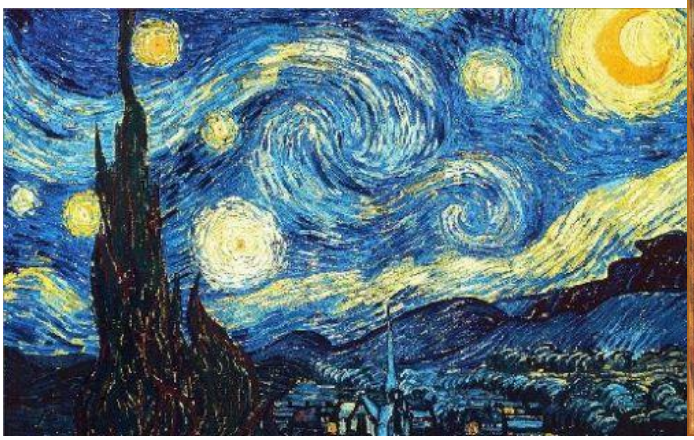
2.IDE：Sublime Text3 + Terminal

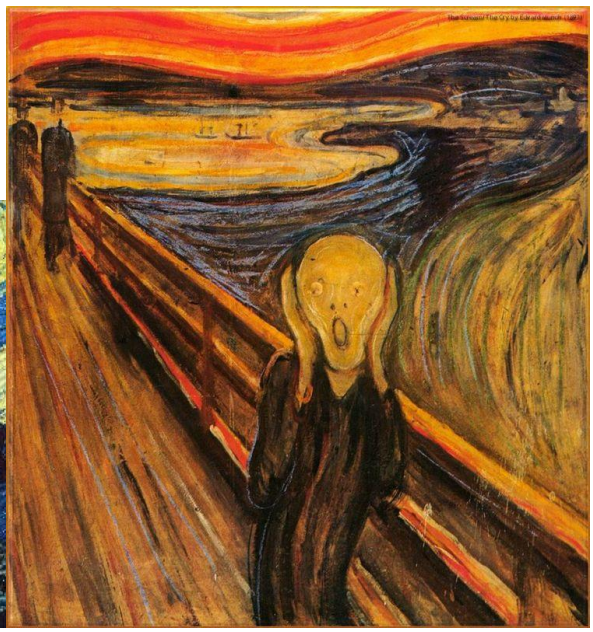
2、测试数据

根据论文 Color Transfer between Images[Reinhard et al. 2001], source image 指颜色空间分布来源的 image，target image 是要被改变颜色空间分布的 image。

Target image 就使用了 part1 的 10 张图片。

而 source image 我选取了梵高的两幅作品，分别是星夜(starry night.jpg)和呐喊(The scream.jpg)，如下：





选取这两幅图的原因是：首先两个图片艺术效果都很好，更重要的是星夜为冷色调而呐喊为暖色调，对图片进行处理的时候效果比较明显。

3、测试过程

下图为 colorTransfer 的代码，主要原理就是假设 Lab 三个通道为独立的高斯分布，然后将 target 的三个通道的高斯分布分别变换为和 source 三个通道的高斯分布特征一样的高斯分布。

```
static std::array<double, 3> getMean(CImg<double>& img) {
    std::array<double, 3> mean = {0, 0, 0};
    cimg_forXY(img, x, y) {
        for (int channel = 0; channel < 3; channel ++)
            mean[channel] += img(x, y, channel);
    };
    for (int channel = 0; channel < 3; channel ++)
        mean[channel] /= (img.width() * img.height());
    return mean;
}

static std::array<double, 3> getVariance(CImg<double>& img) {
    std::array<double, 3> variance = {0, 0, 0};
    std::array<double, 3> mean = imageSet::getMean(img);
    cimg_forXY(img, x, y) {
        for (int channel = 0; channel < 3; channel ++)
            variance[channel] += std::pow((img(x, y, channel) - mean[channel]), 2);
    };
    for (int channel = 0; channel < 3; channel ++) {
        variance[channel] /= (img.width() * img.height());
        variance[channel] = std::pow(variance[channel], 1/2);
    }
    return variance;
}

static CImg<unsigned char> colorTransfer(CImg<double>& source, CImg<double> target) {
    // convert rgb space to lab space
    source.RGBtoLab();
    target.RGBtoLab();

    // get stats from source image and
    std::array<double, 3> sourceMean = imageSet::getMean(source);
    std::array<double, 3> sourceVariance = imageSet::getVariance(source);
    std::array<double, 3> targetMean = imageSet::getMean(target);
    std::array<double, 3> targetVariance = imageSet::getVariance(target);

    cimg_forXY(target, x, y) {
        for (int channel = 0; channel < 3; channel ++) {
            // substract mean value from source
            target(x, y, channel) -= targetMean[channel];
            // scale
            target(x, y, channel) *= (targetVariance[channel] / sourceVariance[channel]);
            // add the mean value from target
            target(x, y, channel) += sourceMean[channel];
        }
    };

    target.LabtoRGB();
    return target;
}
```

然后通过脚本 transferRunScript.py 来跑程序得到测试结果，该脚本如下：

```
import os
import subprocess


def main():
    sourceImagePath = "../transferSource/"
    targetImagePath = "../image/"
    outputImagePath = "../transferImage/"
    for targetImageName in os.listdir(targetImagePath):
        for sourceImageName in os.listdir(sourceImagePath):
            args = ['./Ex3']
            args.append(sourceImagePath + sourceImageName)
            args.append(targetImagePath + targetImageName)
            args.append(outputImagePath + targetImageName + "_" + sourceImageName.strip('.jpg'))
            subprocess.call(args)


if __name__ == "__main__":
    main()
```

4、测试结果

该试验采用了两个 sourceImage，前五幅图会采用 Starrynight.jpg 作为 SourceImage，后五幅图采用 The scream.jpg 作为 SourceImage。下列结果中左边和右边分别为变换前的和变换后的图像。

1）lena.jpg

2）Dormitary.jpg

3)Montreal2.jpg

4)Lake Louis1.jpg

5)Tree before Cameron.jpg

6)Aquarium.jpg

7)Notre-Dame Basilica

8)Yosemite1.jpg

9)Old man.jpg

5、总结及改进

可以看到，测试结果基本符合我们的预期。在 sourceImage 为星夜（Starry Night.jpg）的时候，转换过的图像的色调与星夜较为类似，为冷色调。在 sourceImage 为呐喊（The Scream.jpg）的时候，转换过的图像的色调与呐喊较为类似，为暖色调。

但是，这种办法只是简单的把 targetImage 的颜色分布套在 sourceImage 上，只能做到 color transfer，而这种 transfer 多少会显得有些不自然。要做到一个 image 到另一个 image 的 transfer，我认为不仅仅要在 color 上进行 transfer，而且还应该要在 style 上 transfer。

我就此查了一下文献，发现了一些很有意思的文章和实现方法，其中一个是用卷积神经网络，能实现在风格（style）上的转变。

这个算法的思想比较简单，它用 CNN 进行特征提取，因为 CNN 低层提取的特征偏向于点线，而高层提取的特征偏向于纹理结构（texture），所以可以再用 CNN 进行重建，得到 transfer style 的效果。

参考：A Neural Algorithm of Artistic Style[Leon A. Gatys, Alexander S. Ecker, Matthias Bethge]