西南科技大学

高级图像处理程序设计报告
基于 CLAHE 的图像增强技术

姓　　名：冯　　波

学　　号：7220180157

专　　业：控 制 工 程

指导老师：路锦正老师

基于 CLAHE 的图像增强技术

摘要：图像增强。目的是要改善图像的视觉效果, 针对给定图像的应用场合, 有目的地强调图像的整体或局部特性, 将原来不清晰的图像变得清晰或强调某些感兴趣的特征, 扩大图像中不同物体特征之间的差别, 抑制不感兴趣的特征, 使之改善图像质量, 丰富信息量, 加强图像判读和识别效果, 满足某些特殊分析的需要.

摘要：图像增强、AHE、CLAHE

一、 引言

图像增强的方法是通过一定手段对原图像附加一些信息或变换数据, 有选择地突出图像中感兴趣的特征或者抑制(掩盖)图像中某些不需要的特征, 使图像与视觉响应特性相匹配. 在图像增强过程中, 不分析图像降质的原因, 处理后的图像不一定逼近原始图像. 图像增强技术根据增强处理过程所在的空间不同, 可分为基于空域的算法和基于频域的算法两大类.

图像增强可分成两大类:频率域法和空间域法.基于空域的算法处理时直接对图像灰度级做运算. 基于频域的算法是在图像的某种变换域内对图像的变换系数值进行某种修正, 是一种间接增强的算法.

频率域法把图像看成一种二维信号, 对其进行基于二维傅里叶变换的信号增强.采用低通滤波(即只让低频信号通过)法, 可去掉图中的噪声; 采用高通滤波法, 则可增强边缘等高频信号, 使模糊的图片变得清晰.

基于空域的算法分为点运算算法和邻域去噪算法. 点运算算法即灰度级校正, 灰度变换和直方图修正等, 目的或使图像成像均匀, 或扩大图像动态范围, 扩展对比度. 邻域增强算法分为图像平滑和锐化两种. 平滑一般用于消除图像噪声, 但是也容易引起边缘的模糊. 常用算法有均值滤波, 中值滤波. 锐化的目的在于突出物体的边缘轮廓, 便于目标识别. 常用算法有梯度法,算子,高通滤波, 掩模匹配法, 统计差值法等。

二、 CLAHE 原理

自适应直方图均衡（AHE）是一种用于改善图像对比度的计算机图像处理技术。它与普通直方图均衡的不同之处在于自适应方法计算几个直方图，每个直方图对应于图像的不同部分，并使用它们来重新分配图像的亮度值。因此，它适合于改善局部对比度并增强图像的每个区域中的边缘的定义。

然而，AHE 倾向于过度简化图像的相对均匀区域中的噪声。自适应直方图均衡的变体称为对比度受限自适应直方图均衡（CLAHE）通过限制放大来防止这种情况。

普通直方图均衡使用从图像直方图导出的相同变换来变换所有像素。当整个图像中像素值的分布相似时，这很有效。然而，当图像包含比大多数图像明显更亮或更暗的区域时，这些区域中的对比度将不会充分增强。

自适应直方图均衡化（AHE）通过利用从邻域区域导出的变换函数变换每个像素来改进。最初开发用于飞机驾驶舱显示器。如下图所示。从直方图推导出变换函数与普通直方图均衡完全相同：变换函数与邻域中像素值的累积分布函数（CDF）成比例。

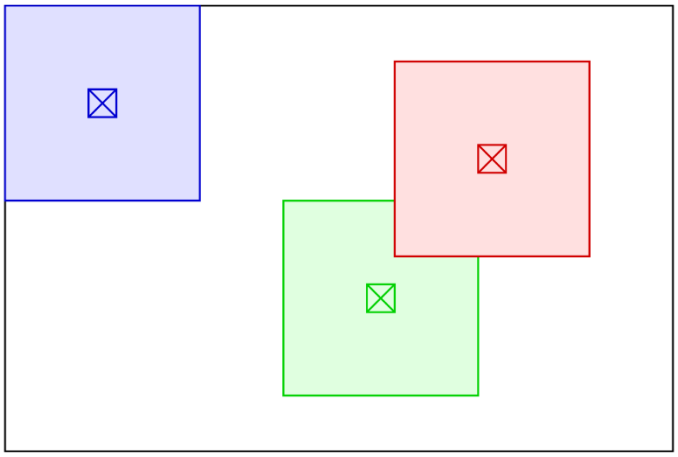


图 1.像素位置

对于图像边界附近的像素必须特殊处理，因为它们的邻域不完全位于图像内。比如图中蓝色像素左侧或上方的像素。可以通过相对于图像边界镜像像素行和列来扩展图像来解决。简单地复制边框上的像素线则不太合理，因为这样处理会导致高度锐化的邻域直方图。

2.1 算法概述

普通的直方图均衡算法对于整幅图像的像素使用相同的直方图变换，对于那些像素值分布比较均衡的图像来说，算法的效果很好。然后，如果图像中包括明显比图像其它区域暗或者亮的部分，在这些部分的对比度将得不到有效的增强。

AHE 算法通过对局部区域执行响应的直方图均衡来改变上述问题。该算法首先被开发出来适用于改进航天器驾驶舱的显示效果。其最简单的形式，就是每个像素通过其周边一个矩形范围内的像素的直方图进行均衡化。均衡的方式则完全

同普通的均衡化算法：变换函数同像素周边的累积直方图函数（CDF）成比例。

图像边缘的像素需要特殊处理，因为边缘像素的领域不完全在图像内部。这个通过镜像图像边缘的行像素或列像素来解决。直接复制边缘的像素进行扩充是不合适的。因为这会导致带有剑锋的领域直方图。

2.2 AHE 的属性

（1）病毒感染领域的大小是该方法的一个参数。领域小，对比度得到增强，领域大，则对比度降低。

（2）当某个区域包含的像素值非常相似，其直方图就会尖状化，此时直方图的变换函数会将一个很窄范围内的像素映射到整个像素范围。这将使得某些平坦区域中的少量噪音经 AHE 处理后过度放大。

2.3 限制对比度自适应直方图均衡（Contrast Limited Adaptive histgram equalization/CLAHE）

CLAHE 同普通的自适应直方图均衡不同的地方主要是其对比度限幅。这个特性也可以应用到全局直方图均衡化中，即构成所谓的限制对比度直方图均衡（CLHE），但这在实际中很少使用。在 CLAHE 中，对于每个小区域都必须使用对比度限幅。CLAHE 主要是用来克服 AHE 的过度放大噪音的问题。

这主要是通过限制 AHE 算法的对比提高程度来达到的。在指定的像素值周边的对比度放大主要是由变换函数的斜度决定的。这个斜度和领域的累积直方图的斜度成比例。CLAHE 通过在计算 CDF 前用预先定义的阈值来裁剪直方图以达到限制放大幅度的目的。这限制了 CDF 的斜度因此，也限制了变换函数的斜度。直方图被裁剪的值，也就是所谓的裁剪限幅，取决于直方图的分布因此也取决于领域大小的取值。

通常，直接忽略掉那些超出直方图裁剪限幅的部分是不好的，而应该将这些裁剪掉的部分均匀的分布到直方图的其他部分。如图 2 所示。

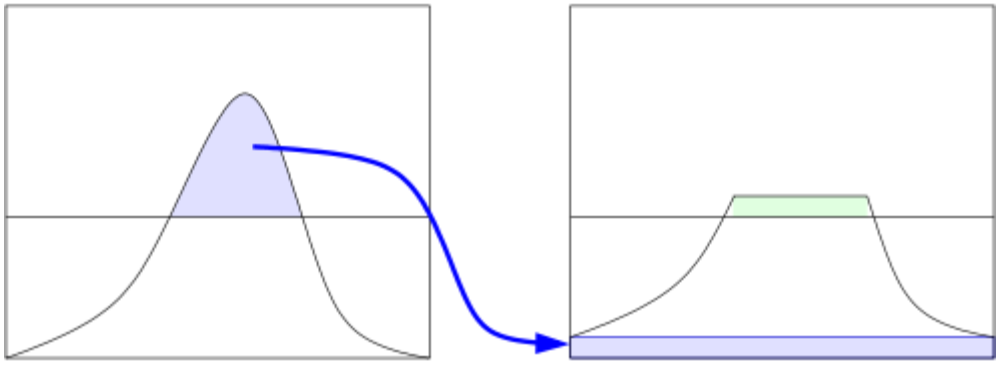


图 2. 直方图裁剪限幅

这个重分布的过程可能会导致那些被裁剪掉的部分由重新超过了裁剪值（如上图的绿色部分所示）。

2.4. 通过插值加快计算速度

如上所述的直接的自适应直方图，不管是否带有对比度限制，都需要对图像中的每个像素计算器领域直方图以及对应的变换函数，这使得算法及其耗时。

而插值使得上述算法效率上有极大的提升，并且质量上没有下降。首先，将图像均匀分成等份矩形大小，如图 3 的右侧部分所示（8 行 8 列 64 个块是常用的选择）。然后计算个块的直方图、CDF 以及对应的变换函数。这个变换函数对于块的中心像素（下图左侧部分的黑色小方块）是完全符合原始定义的。而其他的像素通过哪些于其临近的四个块的变换函数插值获取。位于图中蓝色阴影部分的像素采用双线性查插值，而位于便于边缘的（绿色阴影）部分采用线性插值，角点处（红色阴影处）直接使用块所在的变换函数。

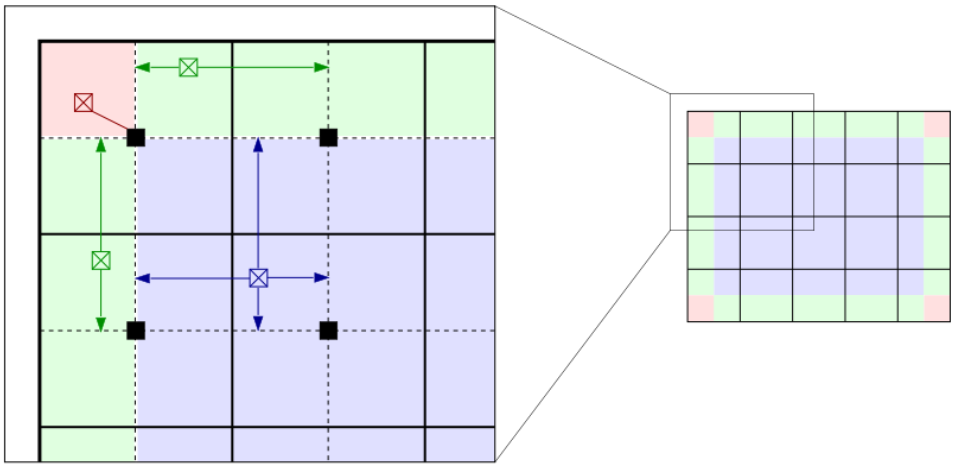


图 3.插值示意图

这样的过程极大的降低了变换函数需要计算的次数，只是增加了一些双线性插值的计算量。

三、 CLAHE 算法步骤

（1）图像分块，以块为单位；

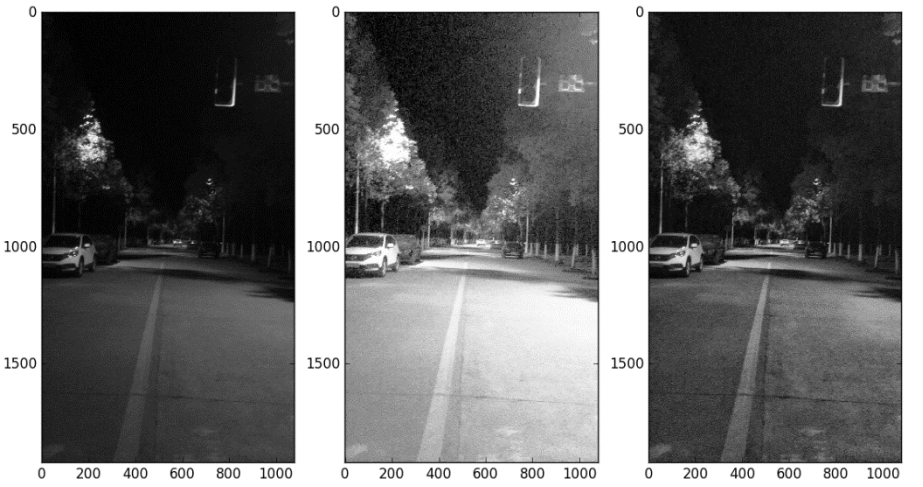
（2）先计算直方图，然后修剪直方图，最后均衡化；

（3）遍历操作哥哥图像块，进行块间双线性插值；

（4）与原图做图层滤色混合操作。（可选）

四、 实验结果

分别采用 python、C++对灰度图像和彩色图像进行 CLAHE 增强实验，实验结果如图 4 所示。



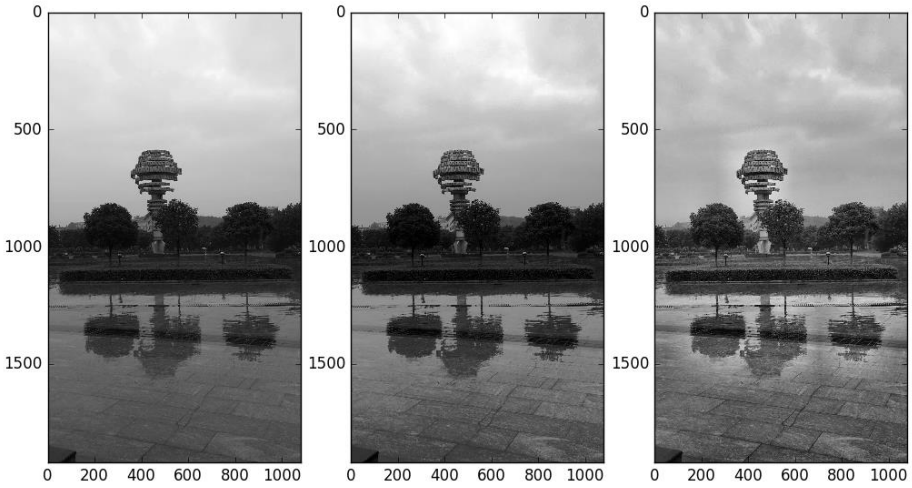


图 4.CLAHE 增强效果图

从图四可以看出 CLAHE 增强效果还是比较明显。对于处理速度，这个函数由于只有一些分加插值计算，速度很快。如果图像不能倍分成整块，一般需要扩充边缘，opencv 的处理方式是上下左右四条边对称镜像扩充。这种方式很合理。

附录：

C++：

```
#include "opencv2/opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

void main()
{
    Mat inp_img = cv::imread("test1.jpg");
    if (!inp_img.data) {
        cout << "Something Wrong";
    }
    namedWindow("Input Image", CV_WINDOW_AUTOSIZE);
    imshow("Input Image", inp_img);

    Mat clahe_img;
    cvtColor(inp_img, clahe_img, CV_BGR2Lab);
    vector<cv::Mat> channels(3);
    split(clahe_img, channels);

    Ptr<cv::CLAHE> clahe = createCLAHE();
    // 直方图的柱子高度大于计算后的 ClipLimit 的部分被裁剪掉，然后将其平均分配给整张直方图
    clahe->setClipLimit(4.); // (int)(4.*(8*8)/256)
    //clahe->setTilesGridSize(Size(8, 8)); // 将图像分为 8*8 块
    Mat dst;
    clahe->apply(channels[0], dst);
    dst.copyTo(channels[0]);
    //
    clahe->apply(channels[1], dst);
    dst.copyTo(channels[1]);
    clahe->apply(channels[2], dst);
    dst.copyTo(channels[2]);
    merge(channels, clahe_img);

    Mat image_clahe;
    cvtColor(clahe_img, image_clahe, CV_Lab2BGR);

    namedWindow("CLAHE Image", CV_WINDOW_AUTOSIZE);
```

```
    imshow("CLAHE Image", image_clahe);
    imwrite("out.jpg", image_clahe);
    waitKey();
    system("pause");
}
```

Python：

```
-------------------
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('test1.jpg',0) #直接读为灰度图像
res = cv2.equalizeHist(img)

clahe = cv2.createCLAHE(clipLimit=2,tileGridSize=(10,10))
cl1 = clahe.apply(img)

plt.subplot(131),plt.imshow(img,'gray')
plt.subplot(132),plt.imshow(res,'gray')
plt.subplot(133),plt.imshow(cl1,'gray')

plt.show()
```