武汉大学计算机学院
本科生课程设计报告

数字图像处理课程报告

专 业 名 称 ：软件工程

课 程 名 称 ：数字图像处理

指 导 教 师 ：韩镇

学 生 学 号 ：2018302110116

学 生 姓 名 ：廖必彦

二〇二〇年十二月

郑 重 声 明

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名：廖必彦　　日期：2020.1.6

目录

一、参考出处

本次课程设计选择的是对图片的灰度图像直方图进行均衡化的基准算法。

参考网页：

https://baike.baidu.com/item/%E7%81%B0%E5%BA%A6%E5%8O%BC

https://www.cnblogs.com/dearzhoubi/p/8625757.html

https://baike.baidu.com/item/%E4%B8%AD%E5%8O%BC%E6%BB%A4%E6%B3%A2

本实验仅参考其中的算法。

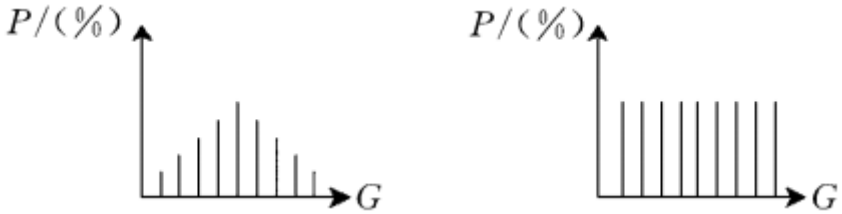
二、基准算法

1.算法原理

直方图均衡化(Histogram Equalization) 又称直方图平坦化,实质上是对图像进行非线性拉伸,重新分配图像象元值,使一定灰度范围内象元值的数量大致相等。这样,原来直方图中间的峰顶部分对比度得到增强,而两侧的谷底部分对比度降低,输出图像的直方图是一个较平的分段直方图:如果输出数据分段值较小的话,会产生粗略分类的视觉效果。

直方图是表示数字图像中每一灰度出现频率的统计关系。直方图能给出图像灰度范围、每个灰度的频度和灰度的分布、整幅图像的平均明暗和对比度等概貌性描述。灰度直方图是灰度级的函数， 反映的是图像中具有该灰度级像素的个数， 其横坐标是灰度级 r， 纵坐标是该灰度级出现的频率(即像素的个数) pr(r) ， 整个坐标系描述的是图像灰度级的分布情况， 由此可以看出图像的灰度分布特性， 即若大部分像素集中在低灰度区域， 图像呈现暗的特性； 若像素集中在高灰度区域， 图像呈现亮的特性。

图 1 所示就是直方图均衡化， 即将随机分布的图像直方图修改成均匀分布的直方图。基本思想是对原始图像的像素灰度做某种映射变换， 使变换后图像灰度的概率密度呈均匀分布。这就意味着图像灰度的动态范围得到了增加， 提高了图像的对比度。



图一

通过这种技术可以清晰地在直方图上看到图像亮度的分布情况，并可按照需要对图像亮度调整。另外,这种方法是可逆的，如果已知均衡化函数，就可以恢复原始直方图。

设变量 r 代表图像中像素灰度级。对灰度级进行归一化处理，则 0≤r≤1，其中 r= 0 表示黑，r= 1 表示白。对于一幅给定的图像来说，每个像素值在[0,1] 的灰度级是随机的。用概率密度函数 $p_r(r)$ 来表示图像灰度级的分布。为了有利于数字图像处理，引入离散形式。在离散形式下，用 r^k 代表离散灰度级,用 $p_r(r^k)$ 代表 $p_r(r)$,并且下式成立： $p_r(r^k)=\frac{n_r k}{n}$ ，其中，0≤ r^k ≤1，k=0，1，2，…，n-1。式中 n_r 为图像中出现 r^k 这种灰度的像素数，n 是原灰度图像中的像素总数，而 $\frac{n_r k}{n}$ 就是概率论中的频数。图像进行直方图均衡化的函数表达式为:

$$S_i = T(r_i) = \sum_{i=0}^{k-1} \frac{n_i}{n}$$

其中，k 为灰度级数，相应的反变换为

$$r_i = T^{-1}(S_i)$$

2. 算法步骤

根据原理，进行直方图均衡化的基准步骤可分为三步

A）计算原图像各灰度级像素个数，即求图像 f(x,y)中各个像素 r 的分布概率密度 $p_r(r)$

B）计算累积分布函数

C）映射变化，即将累积分布函数缩放至原灰度级数范围[0， 2^k]内

3. 算法实现

为了方便计算，本实验采用了平均值法求灰度值：

```
//平均值法求灰度值
/* 1.浮点法：Gray=R*0.3+G*0.59+B*0.11
 * 2.整数法：Gray=(R*30+G*59+B*11)/100
 * 3.移位法：Gray =(R*77+G*151+B*28)>>8;
 * 4.平均值法：Gray= (R+G+B) /3;
 * 5.仅取绿色：Gray=G;
 * 6.Gamma校正算法：[(R^2.2+(1.5*G)^2.2+(0.6*B)^2.2)/(1+(1.5)^2.2+(0.6)^2.2)]^(-2.2);
 */
private void GetGrayBitmap()
{
    label3.Text = "图像初始化准备中";
    for (int x = 0; x < currentBitmap.Width; x++)
    {
        for (int y = 0; y < currentBitmap.Height; y++)
        {
            Color currentColor = currentBitmap.GetPixel(x, y);
            int r = (currentColor.R + currentColor.G + currentColor.B) / 3;
            currentBitmap.SetPixel(x, y, Color.FromArgb(r, r, r));
        }
    }
    label3.Text = "";
}
```

如图，我们对图像的 bitmap 进行了重赋值，将 rgb 三值均设置为其平均值（即原像素灰度值）

然后，进行步骤 A 灰度分布计算，得到数组 ori_Pixel[256]：

```
label3.Text = "正在生成直方图";
//计算原图像各灰度级像素个数
int[] ori_Pixel = new int[256];
for (int x = 0; x < currentBitmap.Width; x++)
{
    for (int y = 0; y < currentBitmap.Height; y++)
    {
        Color currentColor = currentBitmap.GetPixel(x, y);
        int temp = (currentColor.R + currentColor.G + currentColor.B) / 3;
        ori_Pixel[temp]++;
    }
}
```

之后，进行步骤 B,计算累积分布函数：

```
//计算累计分布函数
int[] temp_Pixel = new int[256];
for(int i = 0; i < 256; i++)
{
    if (i != 0)
    {
        temp_Pixel[i] = ori_Pixel[i] + temp_Pixel[i - 1];
    }
    else
    {
        temp_Pixel[i] = ori_Pixel[i];
    }
}
```

最后，进行灰度映射

```
newBitmap = CopyBitmap(currentBitmap);
//灰度映射
int[] new_Pixel = new int[256];
for(int i = 0; i < 256; i++)
{
    new_Pixel[i] = (int)(255.0 * temp_Pixel[i] / (currentBitmap.Width * currentBitmap.Height) + 0.5);
}
chart2.Series[0].Points.DataBindY(new_Pixel);
for (int x = 0; x < newBitmap.Width; x++)
{
    for (int y = 0; y < newBitmap.Height; y++)
    {
        int i = (newBitmap.GetPixel(x, y).R + newBitmap.GetPixel(x, y).G + newBitmap.GetPixel(x, y).B) / 3;
        newBitmap.SetPixel(x, y, Color.FromArgb(new_Pixel[i], new_Pixel[i], new_Pixel[i]));
    }
}
```

三、改进算法

1. 改进思路

直接使用基准直方图均衡化算法进行图像预处理时，得到的效果不够理想故采用一种对比度增强算法。该算法首先对直方图进行平滑处理以消除直方图中因噪声而引入的随机干扰点，然后进行直方图均衡化，增大图像的对比度，接着在整个显示范围内对图像灰度级进行等间距排列，使输出图像的灰度动态范围达到最大灰度变化范围，最后对整幅图像中值滤波处理，滤除图像中被增强的噪声。

2. 算法实现

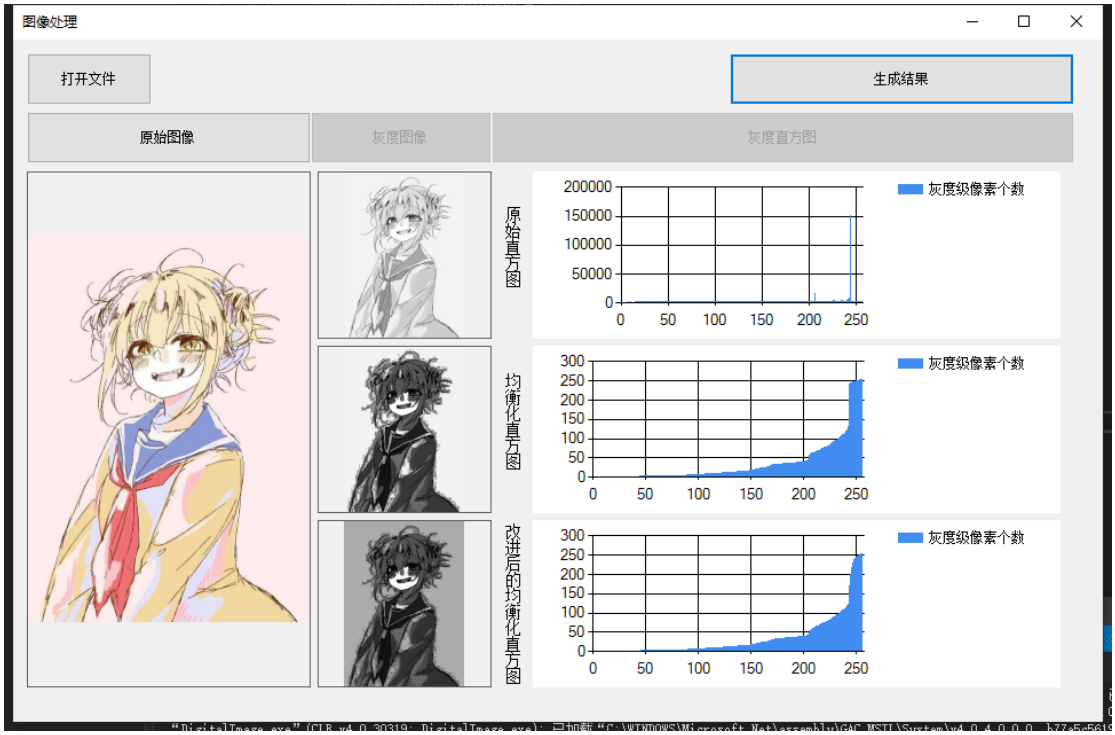
改进后的算法相当于在步骤 A 前先进行一个步骤 A-1，对原始图像进行平滑处理：

```
//先进行平滑处理
for (int i = 0; i < 256; i++)
{
    if (i != 0)
    {
        ori_Pixel[i] = (int)(0.4 * ori_Pixel[i] + 0.6 * ori_Pixel[i - 1] + 0.5);
    }
}
```

然后再在步骤 C 后补上步骤 C-2，即中值滤波处理：

```
//中值滤波处理
private void filter(Bitmap bitmap)
{
    int[] L = new int[9];
    for (int x = 0; x < bitmap.Width; x++)
    {
        for (int y = 0; y < bitmap.Height; y++)
        {
            if ((x == 0 && y == 0) || (x == 0 && y == bitmap.Height - 1) || (x == bitmap.W
            {
                bitmap.SetPixel(x, y, Color.FromArgb(0, 0, 0));
            }
            else if (x == 0)
            {
                L[0] = L[1] = L[2] = 0;
                L[3] = bitmap.GetPixel(x, y - 1).R;
                L[4] = bitmap.GetPixel(x + 1, y - 1).R;
                L[5] = bitmap.GetPixel(x, y).R;
                L[6] = bitmap.GetPixel(x + 1, y).R;
                L[7] = bitmap.GetPixel(x, y + 1).R;
                L[8] = bitmap.GetPixel(x + 1, y + 1).R;
                Array.Sort(L);
                bitmap.SetPixel(x, y, Color.FromArgb(L[4], L[4], L[4]));
            }
            else if (x == bitmap.Width - 1)
```

四、实验结果



从图可以看出，改进后的图像取得了令人满意的效果，说明此算法对于低对比度图像处理效果好，算法具有较强的适应性。

教师评语评分

评语：

评分：

评阅人：

年　　月　　日