监控照片对比度增强项目文档

项目背景

如今的社会中，摄像头扮演者越来越重要的作用，不仅为我们的社会治安带来了极大的方便，对于我们生活的其他各个方面都有着极其重要的意义。

行车记录仪是现在每台汽车都基本配备的组件，但是，夜间道路标志、路标和障碍物的可见度明显降低，对于行车记录仪这种普通的摄像头来说可能并不能太胜任。

为了帮助司机在黑暗的环境中安全行驶，我们可以对汽车前置摄像头拍摄的图像进行对比度增强，并将增强后的图像显示给司机，司机可以通过车载的屏幕辅助观察周围的环境，达到夜间安全驾车。

原始图像

我们提供了三个图片，都来自汽车的行车记录仪。可以看到在夜间照明不良的路口可见的视野范围十分有限，因此该项目通过多种方法进行对比度增强，达到辅助司机夜间安全驾驶的目的。

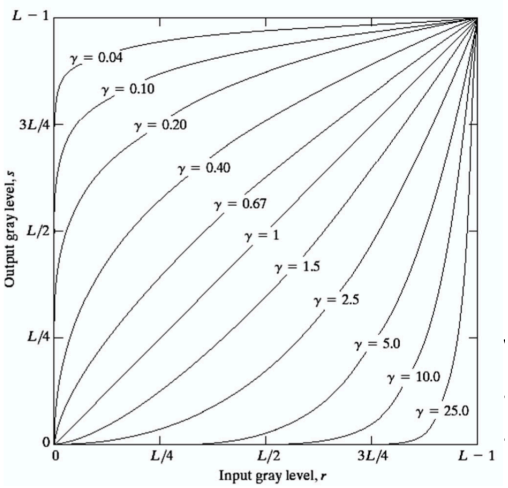
处理一：Gamma变换

Gamma变换在之前的项目中也有提及到，简单来说，就是对原图的像素进行Gamma变换，变换到新的像素点，新的像素点较之前更亮写。

在前面的实战中我们有学习过Gamma变换分为两类：

- Gamma > 1
- Gamma < 1

在行车记录仪的图像中，由于图像较暗（细节信息隐藏在黑色像素中），因此我们要对白色像素进行扩展，对黑色像素进行压缩，所以我们的Gamma应该小于1

> 🔍还记得是为什么嘛?
>
> hint
>
> 
>

代码实战

- Gamma变换核心代码

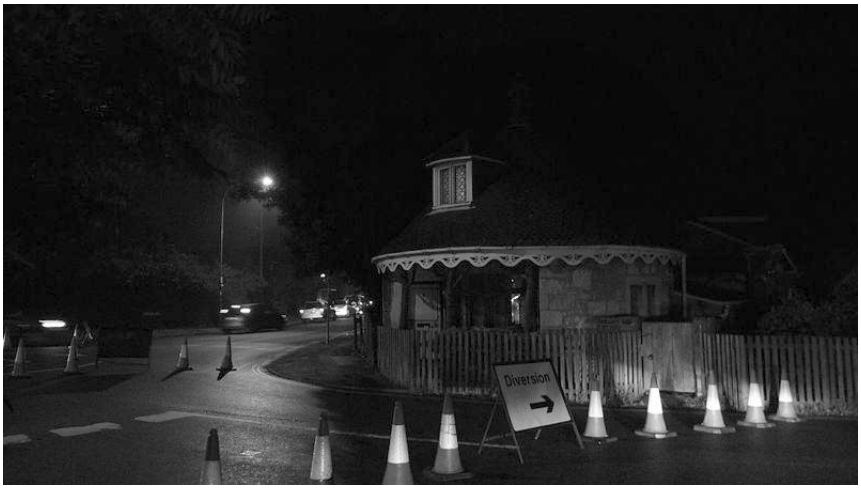
```
def GammaTransform(I, gamma):
    '''
    @description: 使用numpy直接对图片进行Gamma变换

    @params I: 原图
    @params gamma: Gamma变换参数

    @return : Gamma变换后的图像
    '''
    c = 255 / (255**gamma)
    J = c * I**gamma
    return J
```

```
J = GammaTransform(I, 0.7)
```

实验结果

从一下Gamma取0.7的实验结果可以很明显的看到，原来隐藏在黑色中的信息现在暴露出来了（比如图三中路旁的树和楼上的窗户等）

> 🔍聪明的你能发现都有哪些细节信息显现出来了嘛?



如果我们再将Gamma调的小一点，比如0.5

我们可以看到，天空中明显出现了失真的效果，但是其他部分的细节信息我们可以看到的更多了。

所以，我们在进行数字图像处理的时候，很多时候我们要自己调节参数的值以达到最满足我们视觉效果的方案。

处理二：直方图均衡化

直方图均衡化是我们进行图像对比度增强的另一种常用手段，直方图简单来说就是统计了图像中不同像素的个数。而从直方图中能明显反应出图像的一些特性：

- 直方图分布越宽 -> 图像对比度越明显
- 直方图分布越窄 -> 图像对不度越小（图像视觉效果显得灰蒙蒙的）

代码实战

- 直方图均衡化

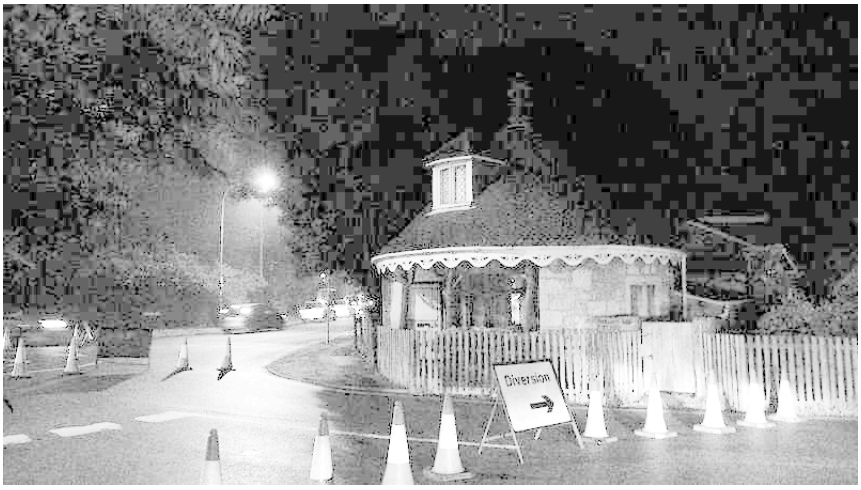
```
def histeq(img):
    '''
    @description: 对图像进行直方图均衡化

    @params : 图像

    @return : null
    '''
    histogram = get_histogram(img)

    pr = {}  # 建立概率分布映射表

    for i in histogram.keys():
        pr[i] = histogram[i] / (img.shape[0] * img.shape[1])

    tmp = 0
    for m in pr.keys():
        tmp += pr[m]
```

```
        pr[m] = max(histogram) * tmp

    new_img = np.zeros(shape=(img.shape[0], img.shape[1]), dtype=np.uint8)

    for k in range(img.shape[0]):
        for l in range(img.shape[1]):
            new_img[k][l] = pr[img[k][l]]

    return new_img
```

- 计算直方图

```
def get_histogram(img):
    '''
    @description: 计算图像的直方图信息

    @params : 图像

    @return : null
    '''
    # 建立原始图像各灰度级的灰度值与像素个数对应表
    histogram = {}
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            k = img[i][j]
            if k in histogram:
                histogram[k] += 1
            else:
                histogram[k] = 1

    sorted_histogram = {}               # 建立排好序的映射表
    sorted_list = sorted(histogram)     # 根据灰度值进行从低至高的排序

    for j in range(len(sorted_list)):
        sorted_histogram[sorted_list[j]] = histogram[sorted_list[j]]

    return sorted_histogram
```

实验结果



可以看到，普通的直方图均衡化的效果非常差，虽然我们可以很清晰的看到大量原先隐藏在黑色中的信息，但是图片变得视觉效果很差，有很多黑白交替的瓦片状，因此我们采用第三种处理解决这个问题。

处理三：自适应直方图均衡化

自适应直方图均衡化和普通的直方图均衡化不同点在于，它是在一个小的局部求解直方图，并进行这一小范围的均衡化。这样的好处也很明显：图像中一个很小的区间像素值大致相同，因此进行这种小范围的重新调整亮度可以获得超出普通的视觉效果。

代码实战

- 自适应直方图均衡化

```
def adapthisteq(I, cliplimit=2.0, tilegridsize=(8,8)):
    '''
    @description: 对原图进行自适应直方图均衡化

    @params I: 原图
    @params cliplimit: 指定直方图均衡化的限幅(默认为2.0)
    @params tilegridsize: 指定自适应直方图均衡化的瓦片大小(默认为8*8的瓦片)

    @return : 自适应直方图均衡化后的图像
    '''


    # 进行正规化
    I_max = np.amax(I)
    I_min = np.amin(I)
    I_img = ((I - I_min) / (I_max - I_min)) * 255
    I_img = I_img.astype('uint8')

    clahe = cv2.createCLAHE(clipLimit=cliplimit,
tileGridSize=tilegridsize)
    J = clahe.apply(I)

    return J
```

实验结果

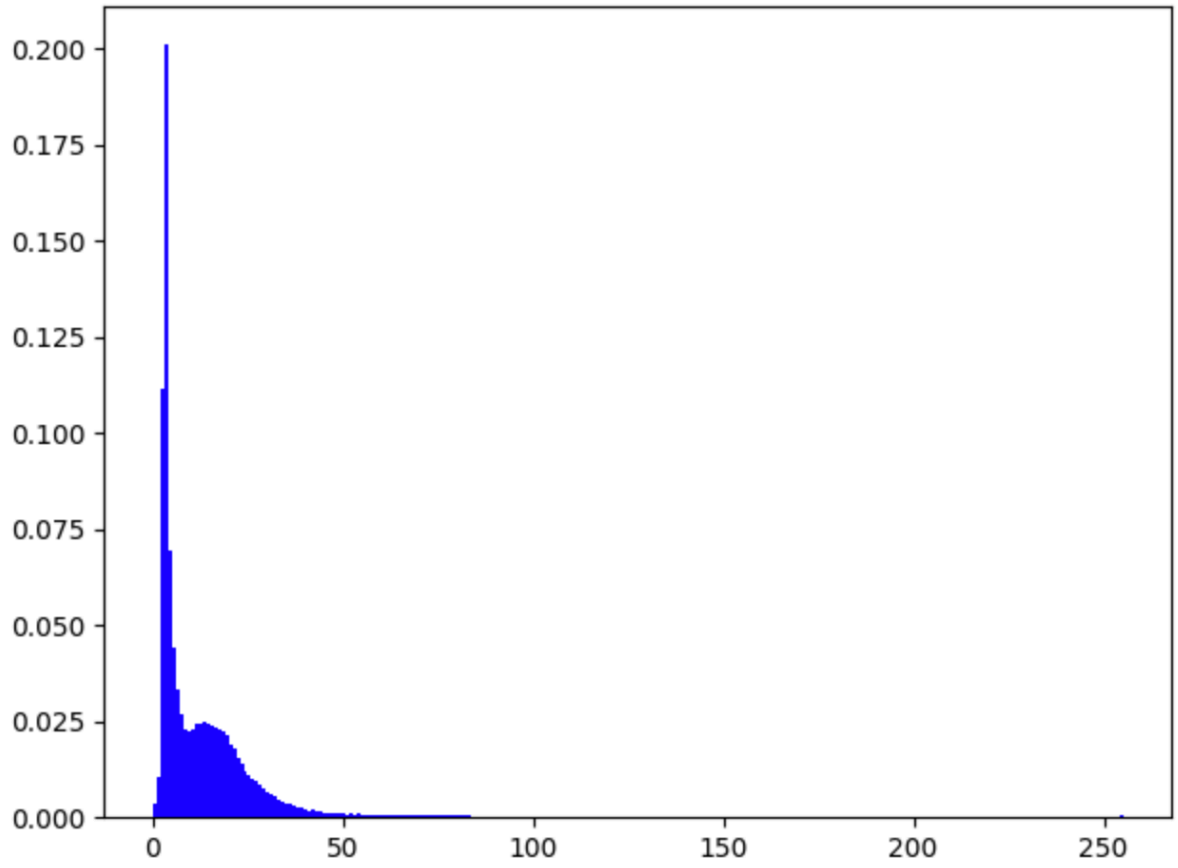
我们可以看到采用自适应直方图均衡化(老师的参数设置为cliplimit=2.0, tilegridsize=(8,8))，带来的视觉效果明显较普通的直方图均衡化有了质的提高。

分析两种直方图均衡化

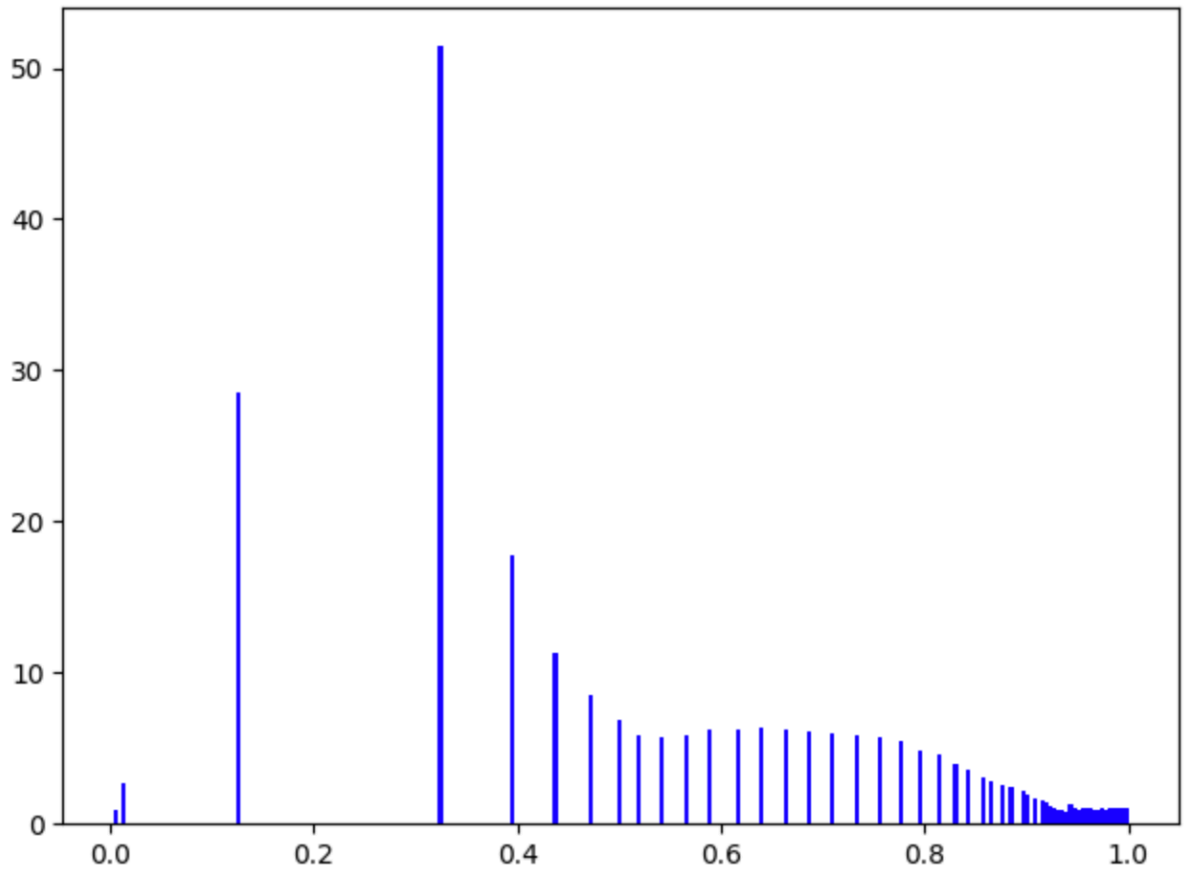
最后，我们从直方图的角度简单探究一下两种直方图均衡化的不同（普通直方图均衡化和自适应直方图均衡化）

拿第一张图为例

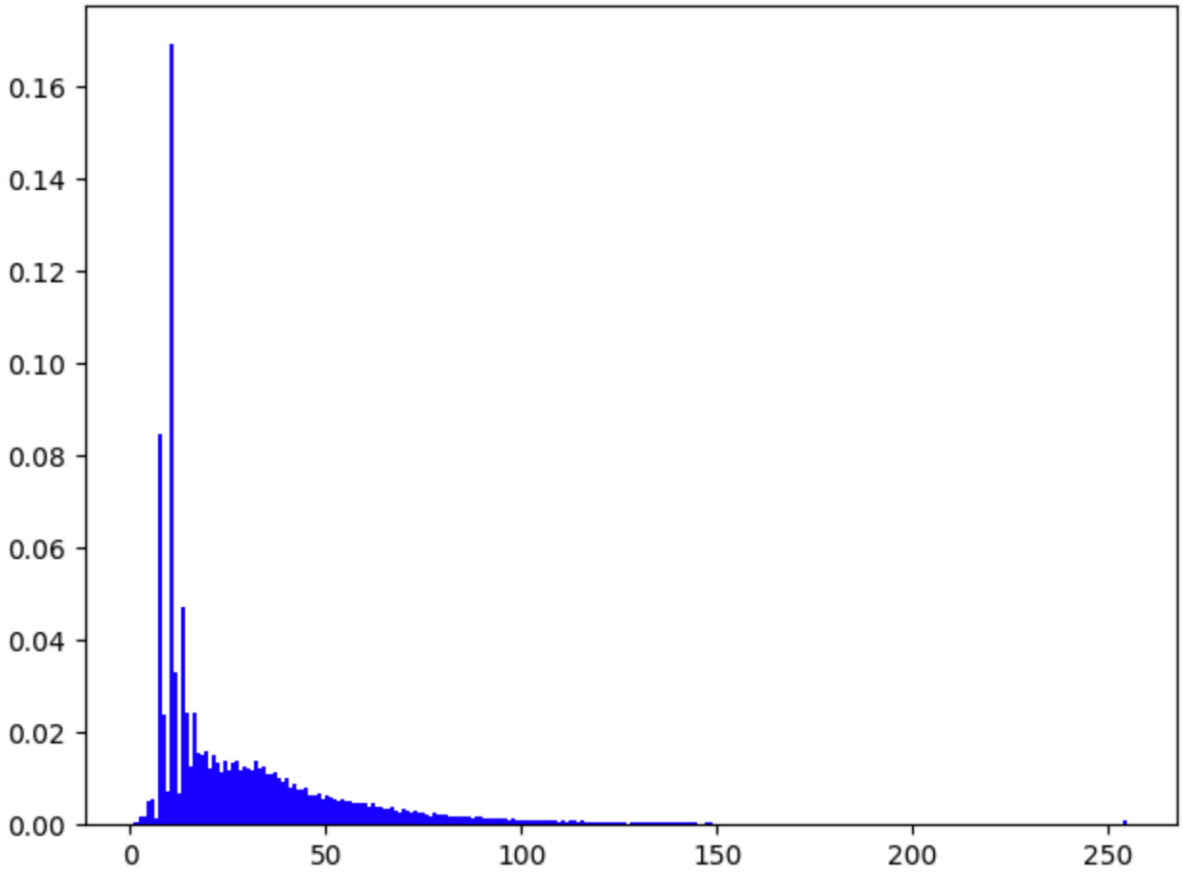
- 原图的直方图



- 普通直方图均衡化后的直方图



- 自适应直方图均衡化后的直方图



我们可以很明显的看到两种不同的直方图均衡化做的操作的不同：

- 普通的方法是在整个像素空间进行平均，而我们的原图几乎只在黑色部分才有像素，这一平均到白色区间，势必会造成处理后图像的失真
- 而自适应方法是移动一个小小的模板进行匹配，然后进行一个很小范围内的直方图均衡化，可以看到对原图中黑色的区间进行了均衡，这就使得处理后的图像视觉效果很棒