Project Name

序号	学号	专业班级	姓名	性别
	3190105609	混合1901	汪辉	男

1. Project Introduction

- 实验要求

  编写程序进行视频镜头检测（2种及以上算法）

  输入：视频片段（或者是解码后的图像序列）

  输出：发生镜头切换处的帧编号

- 工作内容

  实现绝对帧间差法和图像像素差法

  基于压缩域差法实现了类哈希的检测算法

  提出并实验验证了通过海明码检测的算法

- 开发环境及系统运行要求，包括所用的开发工具、开发包、开源库、系统运行要求等

  开发环境：Python3.9.8

  开源库：opencv，numpy

2. Technical Details

- **引言**

镜头切换的实质是在播放连续图像——帧时，改变镜头连续的帧，基于这个基本概念，所有的边缘检测算法都需要对比前后两帧的内容来确定下一帧是否是新的镜头的内容。

镜头边缘检测算法的实质就是要找到一种或几种好的视频图像特征，通过判断相邻图像帧之间的特征是否发生剧烈变化，来完成视频镜头边缘检测任务。因此好的检测算法就意味着足够好的能表征镜头信息的图像（帧）特征。

完整的镜头边缘检测算法包括以下步骤：定义特征、基于这样的特征定义其相似度函数、遍历帧集合找到剧烈变化的帧。

- **精确率和返回率**

为了衡量算法的结果，需要引入两个指标：返回率（Recall）和精确率（Precision），按照如下定义：

$$Recall = correct/(correct + missed), Precision = correct/(correct + false)$$

返回率等于返回的正确的帧数除以全部的目标帧数，精确率等于返回的正确帧数除以返回的全部帧数。

- **阈值的设定**

阈值的确定对于检测的结果至关重要，阈值设定的越低，返回率越高而精确率越低，反之阈值设定的越高，结果的精确率越高而返回率越低。为了综合衡量两个指标，精确率或返回率都极端低的情况是不合适的，因此需要测试一系列的阈值以找到合适的范围。

- **绝对帧间差法和图像像素差法**

实验过程中首先实现了两个简单易理解的算法：绝对帧间差法和图像像素差法。绝对帧间差法计算一帧的所有像素点的像素和，比较相邻两帧的像素和，若差值大于设定的阈值，则认为发生了镜头切换。图像像素差法计算相邻两帧每个像素点的对应像素差，然后将相减的结果取绝对值后求和，若结果大于阈值，则认为发生了镜头切换。

Absolute Difference : $diff = \left|\sum_{i=0}^{count} pixel_{pre} - \sum_{j=0}^{count} pixel_{cur}\right|$

Image Pixel Difference : $diff = \sum_{i=0}^{count} |pixel_{pre} - pixel_{cur}|$

上述两个算法在实现时是相似的，原理较为简单。

```
# Absolut Difference
    img1 = cv2.imread("database/"+str( count ) +".jpg", 0)
    img2 = cv2.imread("database/"+str( count+1 ) +".jpg", 0) # load continuous
two frames
    difference = abs( int(np.sum(img2)) - int(np.sum(img1)) ) # absolute
difference of two frames
    if difference > threshold : # threshold is set indavance
        collection.append(count+1) # collect the switching shot
```

```
# Image Pixel Difference
    difference = np.sum( np.abs( cv2.subtract(img2,img1)) ) # sum of differences
of every two pixel
```

- **类哈希算法**

顾名思义，类哈希算法是类似哈希的原理来计算每一帧的特征的算法。事实上，也可以说绝对帧差法和图像像素差法是一种简易的哈希算法，只是并没有计算每帧的哈希值而只计算了相邻帧的差。为了能够使得镜头连续的帧具有相似的哈希值，必须严密地设计哈希函数。考虑使用opencv库提供的转换函数，我进行了以下尝试。

离散余弦变换——Discrete Cosine Transformation，图像频域变换的一种，具有压缩图像的特点。变换后DCT系数能量主要集中在左上角，其余大部分系数接近于零，常用于对信号和图像(包括静止图像和运动图像)进行有损数据压缩。进行了DCT变换前，通过一次图像转换压缩图像体积，这样使得DCT变换的结果更易于提取，最后取DCT变换后的图片的左上角部分作为特征提取部分。

$$Mat = leftup(contract(dct(imag)))$$

```
img1 = cv2.imread("database/"+str( count ) +".jpg", -1)
img1 = convertImage(img1) .astype(float)
dct_img1 = cv2.dct( cv2.dct(img1) )[:CHARACSIZE,:CHARACSIZE]
# 两次DCT变换能够使特征增强，减小提取的特征矩阵的维度
```

利用帧经过上述三次变换后的矩阵，计算这一帧的哈希值，具体的计算过程为以二进制重新编码每一个像素点。这里又需要一个阈值来衡量每个像素点的位结果，简易的处理可以直接使用矩阵的平均值来作阈值，若大于平均值则像素点为1否则为0。得到的0-1矩阵作为特征矩阵用于和相邻帧进行比较。

$$CharacterMatrix = [Mat[i][j] > mean?1 : 0]i, j = 0...size$$

```
# 实验中为了计算方便和算法效率直接取两个矩阵的平均值
avg = ( np.mean(dct_img2) + np.mean(dct_img1) )/2
# 直接提取每两帧的哈希省略存储
img1_list = ['0' if i > avg else '1' for i in dct_img1.flatten()]
img2_list = ['0' if i > avg else '1' for i in dct_img2.flatten()]
# 结果转化成一维向量 方便后续比较
```

- **海明码算法**

使用海明码编码图片和比较相邻帧的想法完全来自于实现了上述的伪哈希算法后，既然最终目标是构建0-1特征矩阵来进行比较，那么DCT变换的作用在哪里？省略DCT变换会对检测结果带来什么样的影响？

已经提到DCT变换起到了聚集特征的作用，能够缩小特征矩阵维度，相应地提升了算法效率，但是对于能否更加准确地判断镜头切换帧，似乎需要经过进一步验证。于是尝试新的想法，直接海明编码帧的图像，然后运用海明距离判断相邻帧的差异。

```
ham_img1 = convertImage(img1) .astype(float)
ham_img2 = convertImage(img2) .astype(float)
avg = ( np.mean(ham_img2) + np.mean(ham_img1) )/2
# directly code each frame
img1_list = ['0' if i > avg else '1' for i in ham_img1.flatten()]
img2_list = ['0' if i > avg else '1' for i in ham_img2.flatten()]
# implement hammingDist in src
difference = hammingDist(img1_list,img2_list)
```

考虑到原始帧的大小达到240*240，而计算海明距离对0-1矩阵每一位都进行了比较，适当的压缩是必要的，而且找到一个合适的压缩大小也和找到合适的阈值一样，实在是个费时费力的工作。

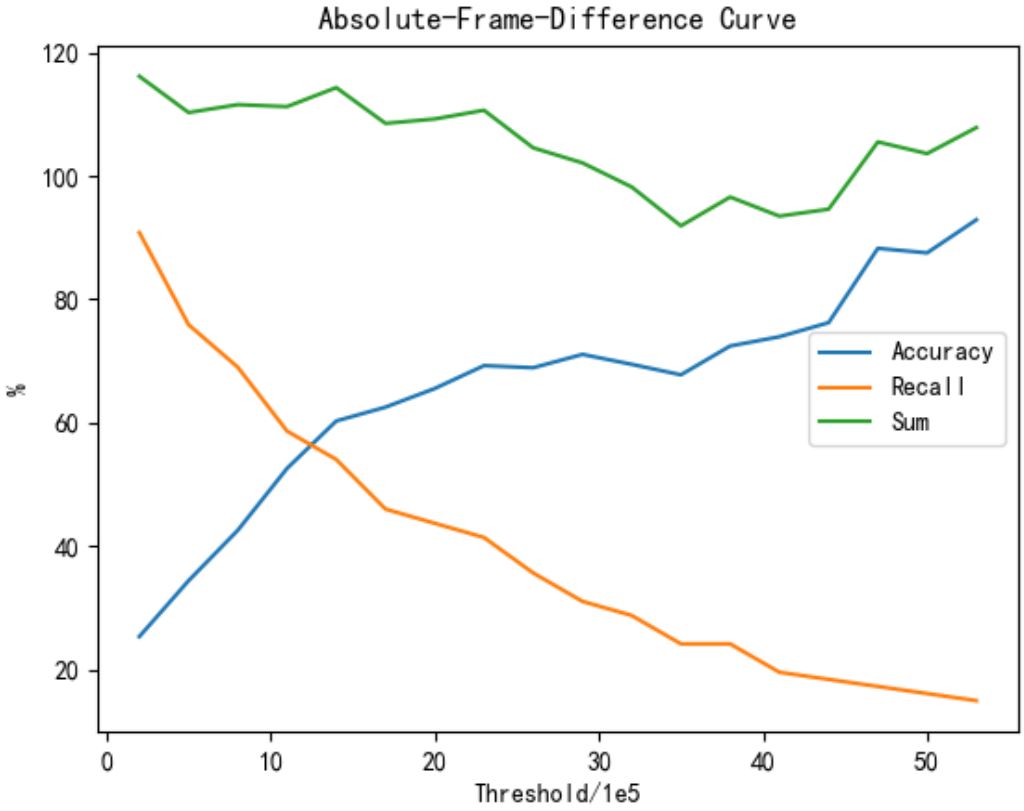
通过反复地调试参数和计算结果，我从这个算法中得到了不错的精确率和返回率，在后续实验结果中将详细讨论。

3. Experiment Results

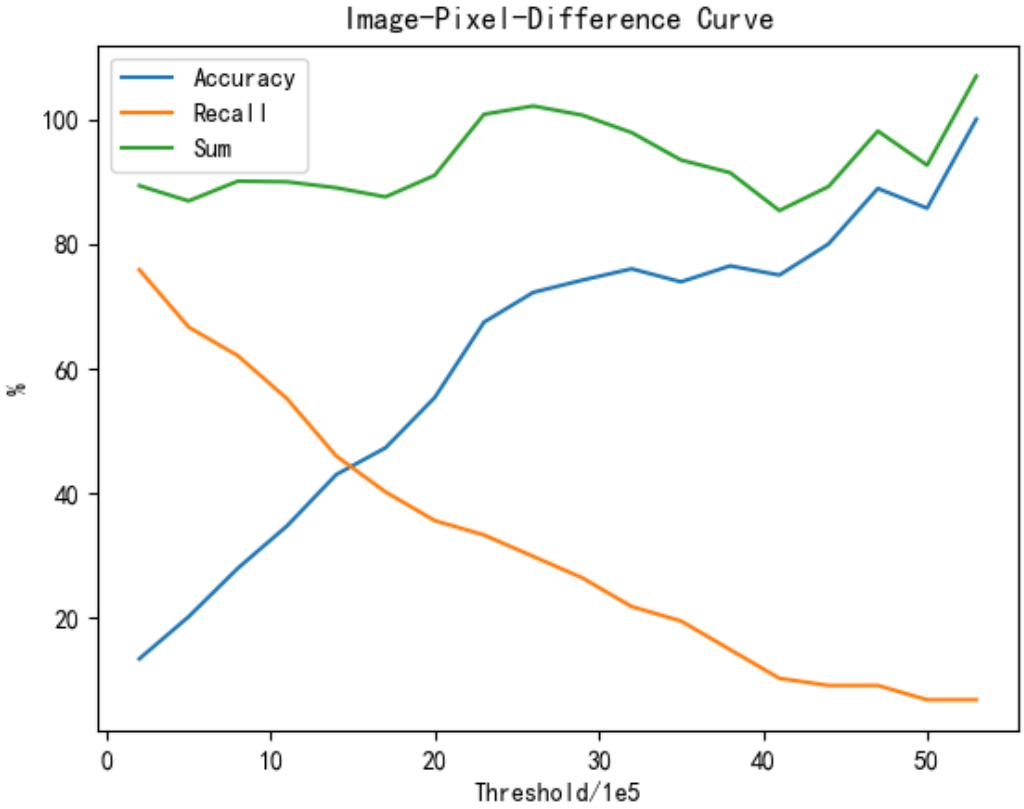
实验测试文件为老师提供的movie.mp4文件，首先简要地调用用opencv的一些方法生成了对应的帧集合，然后再测试每一个算法的表现。考虑到阈值对于算法表现影响很大，对于每一种算法，都测试了大量的阈值，得到一系列的检测结果。此外，精确率和返回率两者是负相关，为了衡量总体的表现如何，简要的取二者的和sum方便观察算法表现。

需要另外说明的是，由于所有帧是实验前事先生成好的，而用于判断镜头切换帧是否返回正确的正确结果集也是我提前人工筛选好并写好在文件expection里的，故实验结果不适用于其他任何视频生成的帧，以下镜头切换帧测试结果的数据只在本人实验文件夹里有效。

- **绝对帧间差法**



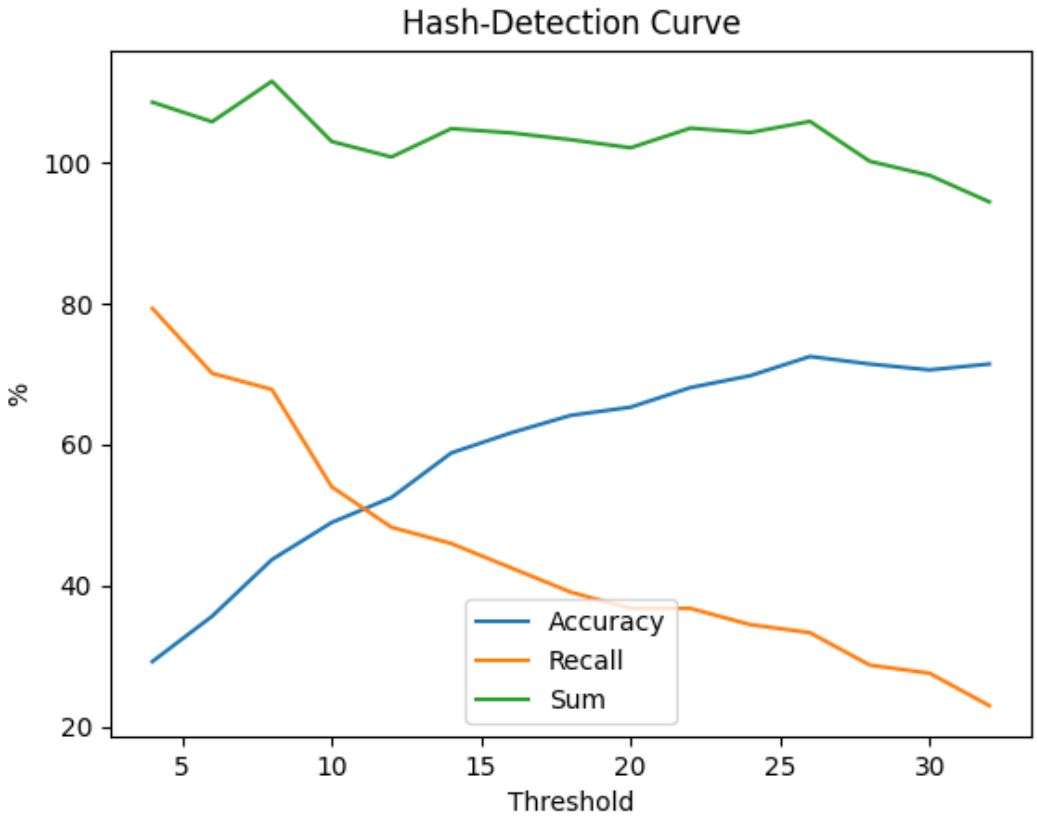
- **图像像素差法**



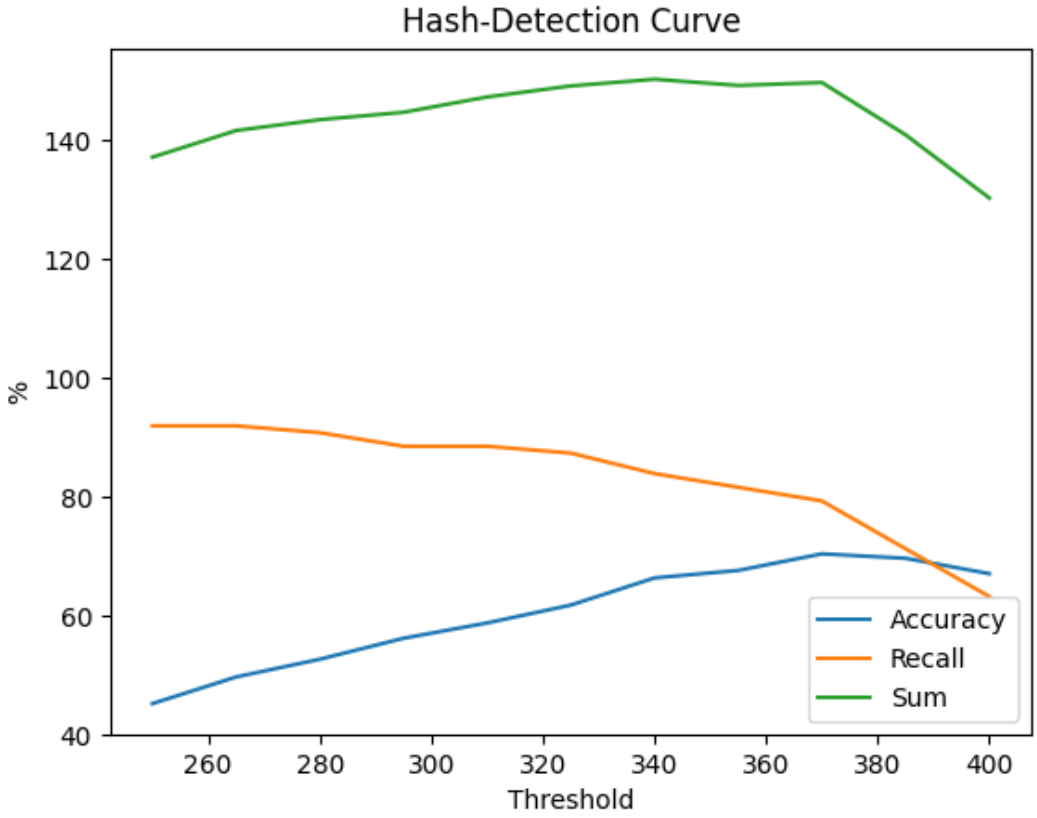
绝对帧间差法和图像像素差法的表现非常不好，综合考虑精确率和返回率，在最平衡的情况下，可以看到，AFD的精确率和返回率都不超过60%，而IPD的两者甚至不超过50%。AFD的表现略好于IPD，因为测试文件《复仇者联盟》宣传片包含大量快速运动的镜头，这些非边缘的镜头会被IPD轻易地判定为镜头切换的帧。

- **类哈希算法**

  压缩图像到32位宽，经过2次DCT变换再提取8位宽特征矩阵得到的结果如下：



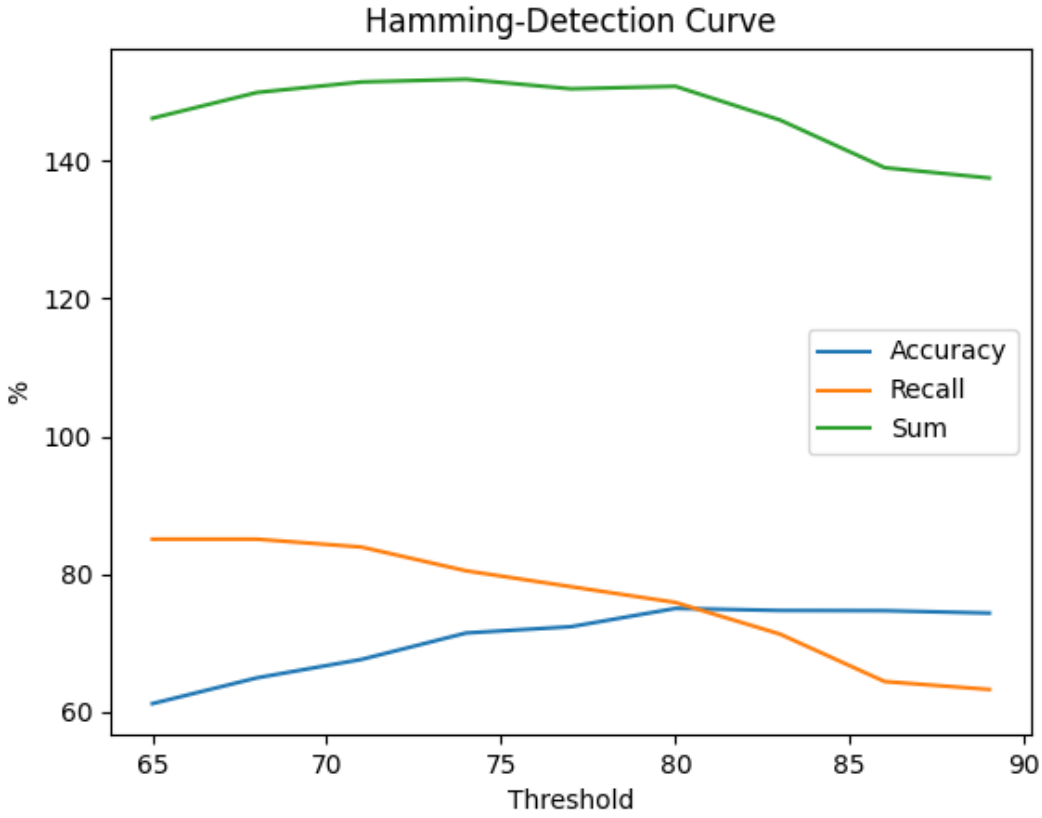
这个结果没有比AFD和IPD更好，于是修改了压缩和特征两个参数在进行测试，发现在有损压缩和特征提取的过程破坏了帧与帧之间的相似度关系。以下位压缩到48位宽并提取36位宽特征矩阵的结果：



至此，终于得到了具有突破性的检测结果，在370附近的阈值下，我们可以得到70的精确率和80的返回率！

```
Compression size: 48 Character size: 36 Threshold: 370
Accuracy: 70.41%
Recall: 79.31%
```

- **海明码算法**



压缩到16位宽的海明码算法也没有让我失望，在65到85的阈值范围内，精确率和返回率都相当可观！

```
Compression size: 16*16 Threshold: 80
Accuracy: 75.00%
Recall: 75.86%
```

到此，由于时间和其他因素，我没有再测试其他提高准确率和返回率的算法。由于四种算法都是适于静态镜头切换检测的算法，而实验测试视频文件包含大量快速运动的镜头，故算法最好也只达到75的准确率和返回率。针对运动物体的检测有理论上更好的运动矢量法，本实验没有涉及。

针对测试视频文件，实验结果图似乎分不出海明码算法和类哈希算法的优劣，但考虑到在类哈希算法测试得到的第二个结果中，压缩的宽度48位而海明码算法中压缩宽度为16位，仍然认为此哈希算法效果不如海明码。综上，可以得到以下实验结果：

$$HammingCode > PseudoHash > AbsoluteDifference > ImagePixelDifference$$

References:

python数字图像处理（二）关键镜头检测 - lynskylate - 博客园 (cnblogs.com)