数字视音频处理（视频部分）实验报告

实验名称：视频镜头检测

姓名：

学号：

专业：数字媒体技术

电子邮箱：

联系电话：

指导老师：杨洋

一、实验目的及要求

实验目的：用两种以上算法编写程序进行视频镜头检测。

实验要求：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（这里我输出镜头切换处的帧图片）。

二、实验的开发与运行环境

Windows 10

Python 3.7.4

Opencv 3.4.2

三、实验介绍

视频镜头的过渡形式可以分类为硬剪切（Hard Cuts），淡入淡出（Fades），溶解（Dissolves），擦拭（Wipe）四种，硬剪切指从一个场景到下一个场景的瞬时过渡，淡入淡出指一个场景和下一个场景之间的渐进过渡（淡出）或下一个场景和本场景之间的渐进过渡（淡入），溶解指从一个场景到另一个场景的逐渐过渡，其中第一个场景淡出，第二个场景淡入。擦拭指有一条线在屏幕上移动，新场景出现在该线的后面的过渡形式。

镜头检测，即给定包含n个镜头的视频V，找到每个镜头的开始和结尾。镜头检测也称为镜头边界检测或过渡检测。它对任何类型的视频分析和视频应用程序都至关重要，因为它可以将视频分割成其基本组成部分：镜头。

镜头边缘检测常用的算法有：绝对帧间差法，图像像素差法，图像数值差法，颜色直方图法，压缩域差法，矩不变量法，边界跟踪法，运动矢量法，双阈值算法。

本次实验我用了图像像素差法，矩不变量法，颜色直方图法和双阈值算法。

四、实验的算法、步骤及其基本思路

1.首先将视频所有的帧提取出来。使用cv2.VideoCapture()函数读取视频，读取其每一帧，给其设置编号，并将其保存为图片存在指定文件夹下。

相关代码如下：

```
#得到视频的每一帧，将其转换为图片
video = 'video.mp4'
cap = cv2.VideoCapture(video)
id = 1
while cap.isOpened():
    ret,frame = cap.read()
    if ret == False:
        break
    else:
        str_id = str(id)
        str_id_final = '{0:0>4}'.format(str_id)
        cv2.imwrite('./image/shot/test/'+'shot_'+str_id_final+'.png',frame)
        id += 1
```

2.接下来介绍视频镜头检测的四种算法，其读入均为视频提取出来的图片帧的文件名列表。

（1）图像像素差法

判断相邻图像帧中像素点发生变化的多少，达到视频镜头边缘检测的目的，这是基于图像像素差法进行镜头边缘检测的基础。把图像转为灰度图像，计算相邻图像帧中像素点发生变化的多少，统计两幅图像对应像素变化超过设定的第一个阈值的像素点个数。

关键代码如下：

```
        pixel_difference = image2 - image1
        pixelNumber_list.append(len(pixel_difference[pixel_difference >=
threshold1]))
```

然后将变化的像素点个数与第二个设定的阈值比较，如超过范围，则认为这两帧之间发生较大变化，判断其为镜头边界。

关键代码如下：

```
    for i in range(len(pixelNumber_list)):
        if pixelNumber_list[i] > threshold2:
            imageID_list.append(i+1)
            temp = cv2.imread(shot_dir+image_list[i+1])
            cv2.imwrite(result_dir+image_list[i+1],temp)
```

（2）矩不变量法

图像矩不变量具有比例、旋转和过渡不变性的特点，是用来表示图像帧的好方法，所以可以用来进行镜头边缘检测。

对于灰度分布为f(x,y)的而图像，其(p+q)阶图像矩的定义为（如图Fig-1）

$$m_{pq} = \sum_{x}\sum_{y} x^{p} \cdot y^{q} \cdot f(x,y)$$

Fig-1

计算归一化中心矩的方法如图（Fig-2）

$$n_{pq} = \frac{1}{m_{00}^{r}} \cdot \sum_{x} \sum_{y} (x - \overline{x})^{p} \cdot (y - \overline{y})^{q} \cdot f(x, y)$$

Fig-2

其中几个变量的定义如图（Fig-3）

$$r = 1 + (p + q)/2 \qquad \bar{x} = m_{10}/m_{00} \qquad \bar{y} = m_{01}/m_{00}$$

Fig-3

直接用普通矩或中心矩进行特征表示不能使特征同时具有平移、旋转和比例不变性。如果利用归一化中心矩，则特征不仅具有平移不变性，而且还具有矩阵不变性。

通过上面几个公式，利用二阶和三阶中心矩可以构造七个不变矩，他们在连续图像变化下可保持平移、缩放和旋转不变，当我们对视频镜头边缘检测进行评估时，则使用前三个矩不变量（如图Fig-4）。从相邻图像帧提取矩不变量特征后，就可以计算这些矩不变量特征的欧拉距离（如图Fig-5），如果欧拉距离超过一定阈值，则认为相邻图像帧之间出现了镜头转换。

$$\Phi_1 = n_{20} + n_{02}$$

$$\Phi_2 = (n_{20} - n_{02})^2 + 4n_{11}^2$$

$$\Phi_3 = (n_{30} - 3n_{12})^2 + (3n_{21} - n_{03})^2$$

Fig-4

$$d(f, f') = \left| \overrightarrow{\sigma_f} - \overrightarrow{\sigma_{f'}} \right|^2, \text{其中} \vec{\sigma} = \{\Phi_1, \Phi_2, \Phi_3\}$$

Fig-5

关键代码如下：

①计算不变矩

```
    #计算图像的0阶几何矩
    m00 = gray.sum()
    m10 = m01 = 0
    #  计算图像的二阶、三阶几何矩
    m11 = m20 = m02 = m12 = m21 = m30 = m03 = 0
    for i in range(row):
        m10 += (i * gray[i]).sum()
        m20 += (i ** 2 * gray[i]).sum()
        m30 += (i ** 3 * gray[i]).sum()
        for j in range(col):
            m11 += i * j * gray[i][j]
            m12 += i * j ** 2 * gray[i][j]
            m21 += i ** 2 * j * gray[i][j]
    for j in range(col):
```

```
            m01 += (j * gray[:, j]).sum()
            m02 += (j ** 2 * gray[:, j]).sum()
            m30 += (j ** 3 * gray[:, j]).sum()
        # 由标准矩我们可以得到图像的"重心"
        u10 = m10 / m00
        u01 = m01 / m00
        # 计算图像的二阶中心矩、三阶中心矩
        y00 = m00
        y10 = y01 = 0
        y11 = m11 - u01 * m10
        y20 = m20 - u10 * m10
        y02 = m02 - u01 * m01
        y30 = m30 - 3 * u10 * m20 + 2 * u10 ** 2 * m10
        y12 = m12 - 2 * u01 * m11 - u10 * m02 + 2 * u01 ** 2 * m10
        y21 = m21 - 2 * u10 * m11 - u01 * m20 + 2 * u10 ** 2 * m01
        y03 = m03 - 3 * u01 * m02 + 2 * u01 ** 2 * m01
        # 计算图像的归格化中心矩
        n20 = y20 / m00 ** 2
        n02 = y02 / m00 ** 2
        n11 = y11 / m00 ** 2
        n30 = y30 / m00 ** 2.5
        n03 = y03 / m00 ** 2.5
        n12 = y12 / m00 ** 2.5
        n21 = y21 / m00 ** 2.5
        # 计算图像的不变矩
        h1 = n20 + n02
        h2 = (n20 - n02) ** 2 + 4 * n11 ** 2
        h3 = (n30 - 3 * n12) ** 2 + (3 * n21 - n03) ** 2
```

②计算相邻图像帧提取出的矩不变特征的欧拉距离

```
        #欧氏距离
        distance = (humoment_list[i][0] - humoment_list[i+1][0])**2 +
(humoment_list[i][1] - humoment_list[i+1][1])**2 + (humoment_list[i][2] -
humoment_list[i+1][2])**2
```

③最后将欧氏距离与设定的阈值进行比较，如果超过一定阈值认为发生了镜头转换。

```
    for i in range(len(d)):
        if d[i] > threshold:
            imageID_list.append(i + 1)
            temp = cv2.imread(shot_dir + image_list[i + 1])
            cv2.imwrite(result_dir + image_list[i + 1], temp)
```

（3）颜色直方图法

图像颜色直方图特征在镜头边缘检测中被经常使用，基于颜色直方图特征、对比图像颜色分布间差异的方法就叫做颜色直方图法。

颜色直方图法主要通过计算两幅图像的相似度来判断相邻图像帧之间是否发生了镜头转换。图像的相似度的计算方法如图（Fig-6，Fig-7），其中H(f,j)指图像1或者图像2的直方图，这里我给其添加权重（h，s，v分别对应权重0.5，0.3，0.2）。其中如果相邻图像帧是同一张图片的话，计算出来的相似度为1。两张图片的相似度越接近1，其相似度越高。

$$d(f,f') = \frac{s(f,f')}{\sum_{j=0}^{N} H(f,j)}$$

Fig-6

$$s(f,f') = \sum_{j=0}^{N} \min(H(f,j), H(f',j))$$

Fig-7

关键代码示例如下：

①计算直方图

```
        image = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
        h,s,v = cv2.split(image)

        # 计算直方图
        hist_h = cv2.calcHist([h],[0],None,[256],[0,255])
        hist_s = cv2.calcHist([s], [0], None, [256], [0, 255])
        hist_v = cv2.calcHist([v], [0], None, [256], [0, 255])
        hist = 0.5*hist_h + 0.3*hist_s + 0.2*hist_v
        hist_list.append(hist)
```

②计算相似度

```
    for i in range(256):
        s += min(image1_hist[i],image2_hist[i])
        h += image2_hist[i]
    similarity = s/h
    return similarity
```

③当相似度小于阈值时，则认为发生了镜头切换

```
    for i in range(len(similarity_list)):
        if similarity_list[i] < threshold:
            imageID_list.append(i+1)
            temp = cv2.imread(shot_dir+image_list[i+1])
            cv2.imwrite(result_dir+image_list[i+1],temp)
```

（4）双阈值算法

根据前面的介绍，镜头边缘可以大体分为两大类型：突变和渐变。突变即简单的镜头切换（cut），而渐变是具有某种特殊效果的逐渐过渡（如fade、dissolve和wipe）。一般而言，只要设定合适的阈值就能检测出镜头突变，而对于复杂的渐变来说，连续两帧之间的差别不如突变明显。渐变引起的帧间差一般要远小于突变的帧间差。一种简单的解决办法是降低阈值，使得能够识别渐变的镜头变化。但是这种方法并不可行，因为渐变的差别可能会小于摄像机镜头的特殊效果造成的差别，比如对象

运动、摄像机镜头的拉伸。如果降低阈值，会得到很多错误的切分结果。因此调整单阈值的判断方法不是非常好。

双阈值的方法采用两个阈值: Tb, 类似于普通判断镜头切分的阈值; Ts, 较低的阈值，用以判断一些特殊效果。采用简单的直方图差法比较相邻帧的差别（计算公式如下图所示）：

$$SD_i = \sum_{j=1}^{g} |H_i(j) - H_{i+1}(j)|$$

Fig-8

Hi(j)表示第i帧的直方图，j 是G个可能的灰度级之一。SDi表示第i帧和第i+1帧的差别。如果SDi超过Tb，则认为是镜头的切分点；而小于Tb大于Ts的帧，标注为可能的渐变的开端(Fs)。然后该帧和后续帧比较，称为"累计比较"。渐变的过程中，" 累计比较差"会逐渐增加，如果帧间差减少到小于Ts，而累计差超过Tb，标注为渐变的结束(Fe) 。注意，只有当相邻帧差超过Ts时，才计算累计比较差。如果相邻帧差低于Ts，而累计比较差小于Tb，就放弃这个可能的渐变开始点(Fs)，寻找下一-个可能的渐变。双阈值的关键思路在于同时满足两个不同的阈值条件，不仅识别普通的镜头突变，也能识别渐变。

这个过程可以解释为如下图所示（Fig-9）：

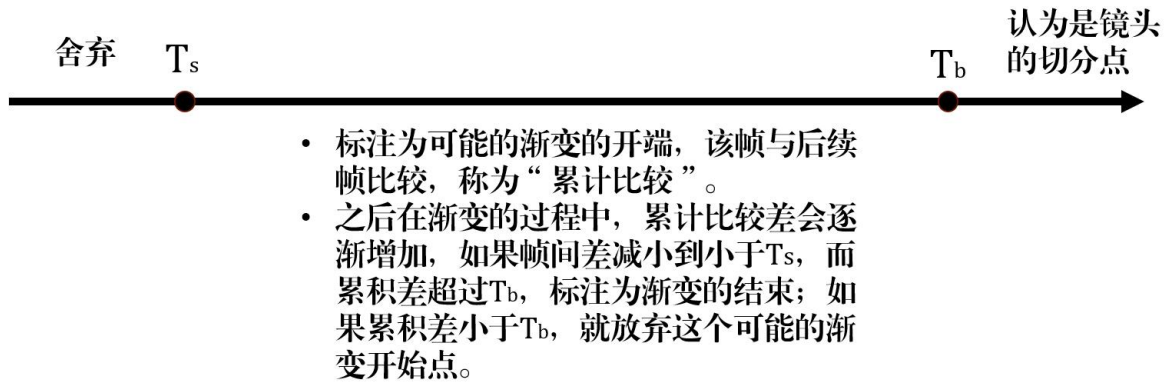


Fig-9

具体实现的关键代码如下：

①计算直方图的算法不再赘述，在上面已经有所提及

②双阈值计算的主要过程

```
        if difference_list[i] > Tb:
            imageID_list.append(i+1)
            temp = cv2.imread(shot_dir+image_list[i+1])
            cv2.imwrite(result_dir+image_list[i+1],temp)
        elif difference_list[i]>Ts:
            Fs = i
            isFs = True
        else:
            if isFs == True:
                isFs = False
                difference = getDifference(hist_list[Fs], hist_list[i])
                if difference > Tb:
                    imageID_list.append(Fs + 1)
                    temp = cv2.imread(shot_dir + image_list[Fs + 1])
                    cv2.imwrite(result_dir + image_list[Fs + 1], temp)
```

五、实验结果与分析

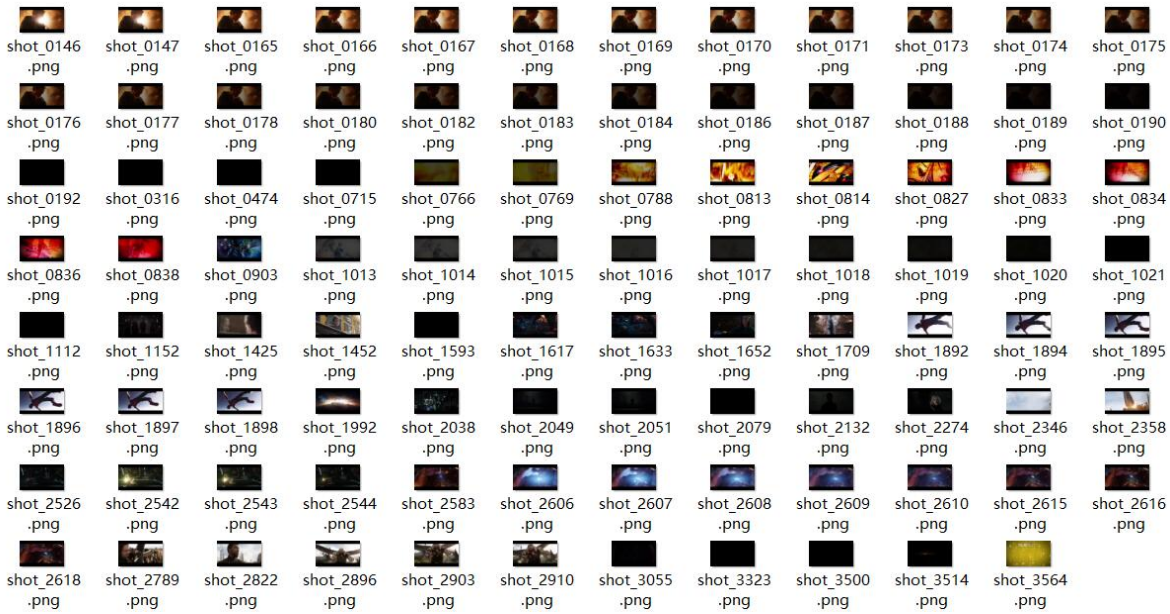
其中指定视频统计出镜头检测共56次，本人自己寻找的皮卡丘视频共统计出镜头检测共20次。

1.图像像素差法

指定视频共检测出镜头切换95次，检测正确的有24次，召回率为42.86%，精确率为25.26%。

本人自己寻找的皮卡丘视频检测出镜头切换9次，检测正确的有8次，召回率为40%，精确率为88.89%。

通过下面的结果图可以看到，该方法对镜头移动十分敏感，对噪声的容错性较差，检测出来其召回率（recall）和精确率（precision）是比较差甚至可以说是最差的。



result-test

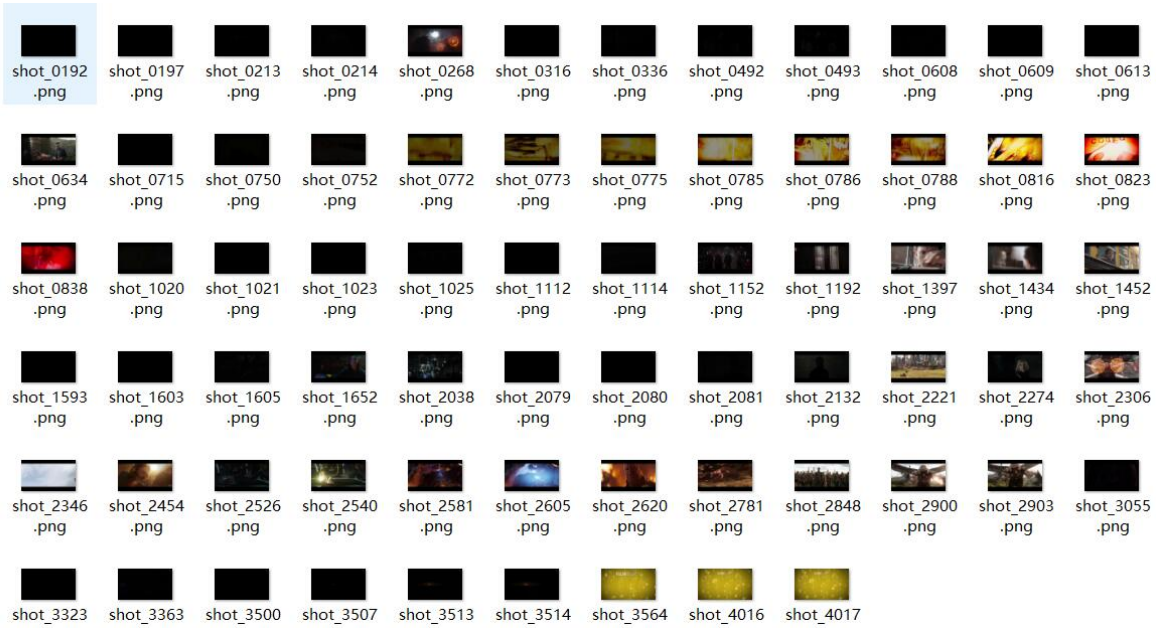
result-pikachu

2.矩不变量法

指定视频共检测出镜头切换69次，检测正确的有32次，召回率为57.14%，精确率为63.64%。

本人自己寻找的皮卡丘视频检测出镜头切换11次，检测正确的有7次，召回率为35%，精确率为63.64%。

矩不变量法计算的缺点在于计算量大，效率不高，在数据庞大、数据量增大的情况下，效果会变差。通过矩不变量法检测出来其召回率（recall）和精确率（precision）较差，但比通过图像像素差法得到的结果稍好一些。



result-test

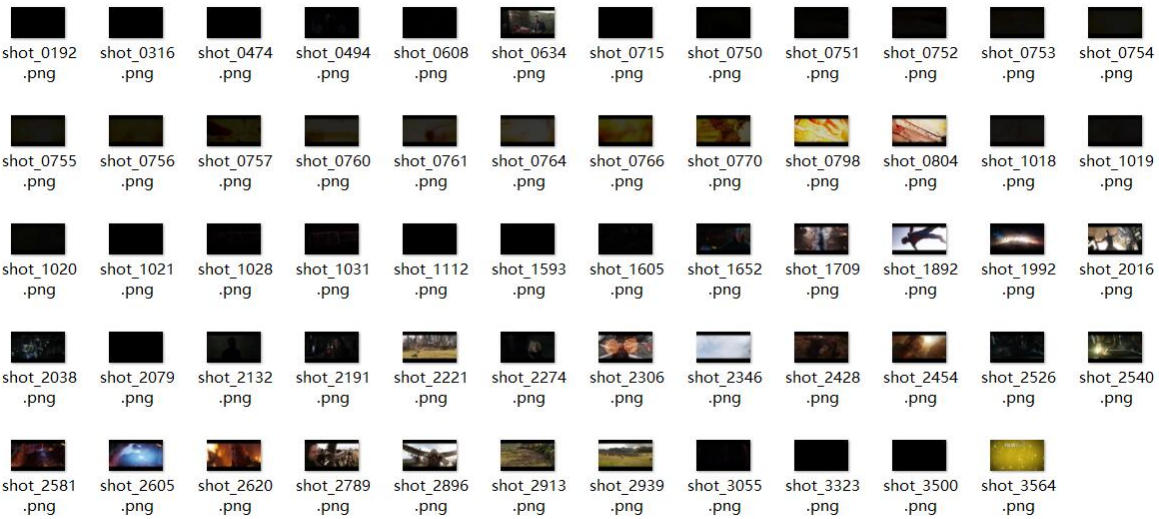
result-pikachu

3.颜色直方图法

指定视频共检测出镜头切换59次，检测正确的有35次，召回率为62.5%，精确率为59.32%。

本人自己寻找的皮卡丘视频检测出镜头切换12次，检测正确的有11次，召回率为55%，精确率为91.67%。

颜色直方图法实现比较简单，计算复杂度较低，比较适合一般镜头的切换判断，但是对于内容变化较大的视频可能会产生误检，而对于内容变化较小的视频则可能会产生漏检。通过颜色直方图法检测出来其召回率（recall）和精确率（precision）是目前的三种算法中效果比较理想的，但是对于渐变的镜头切换并不能很好的检测出来。



result-test

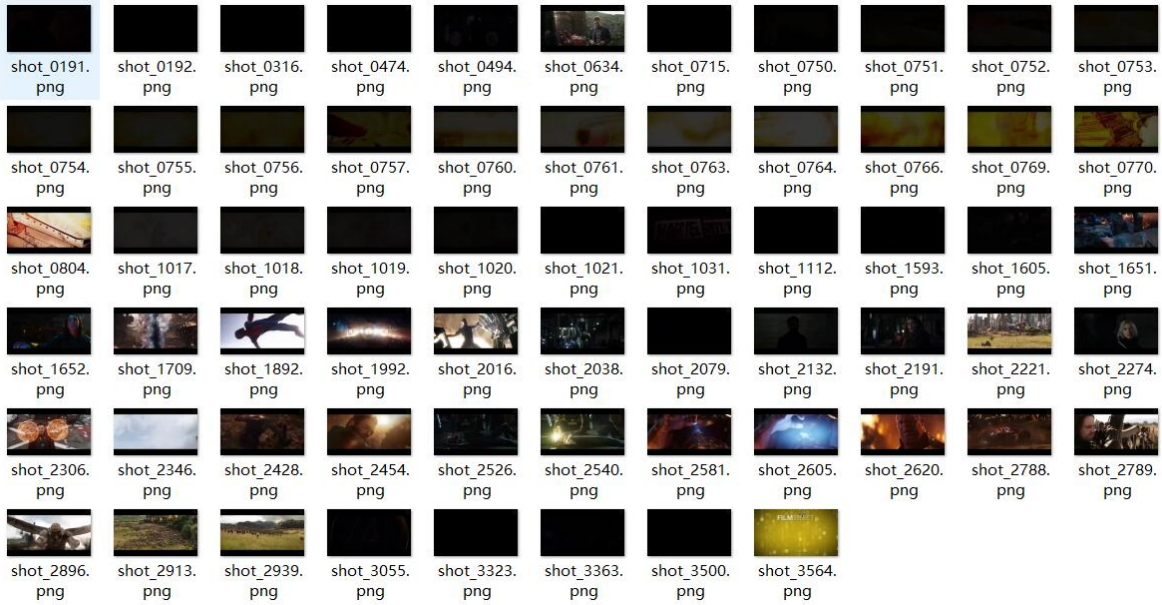
result-pikachu

4.双阈值算法

指定视频共检测出镜头切换63次，检测正确的有40次，召回率为71.43%，精确率为63.49%。

本人自己寻找的皮卡丘视频检测出镜头切换21次，检测正确的有17次，召回率为85%，精确率为80.95%。

双阈值算法基于颜色直方图法，比较适合镜头的切换判断，对于渐变和突变两种镜头切换形式都可以比较好的检测出来，但是也可能会发生一些误检。综上所述，通过双阈值算法检测出来其召回率（recall）和精确率（precision）是目前我所涉及的这四项算法中最理想的。



result-test

result-pikachu

六、实验心得

通过本次实验，我了解了视频镜头检测的算法，并实现了其中的四种算法，掌握了Opencv的基本操作及有关读取视频、将视频帧保存为图片的基本操作，自己动手实现了图像像素差法，矩不变量法，颜色直方图法和双阈值算法，通过对于四种算法实验结果的分析，深入了解了四种算法分别的优缺点，同时巩固了Python的语法及运用，实验的过程比较顺利，在不断的思考和实践中完成了本次作业，感觉收获很大，受益匪浅。