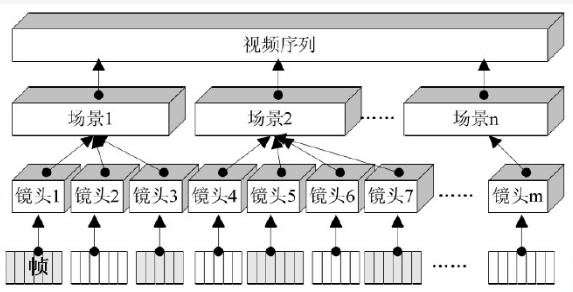
# 镜头边界检测常用技术

——1120152035王奥博

本文中所有代码和测试视频均能在

我们知道，一段有意义的视频是由不同的场景组成的，而一个场景是由不同的镜头组成的，同一个镜头通过快速展现连续图像帧（20 – 60 f/s）使观众产生图像在动的直观感觉。视频，镜头，场景和帧的关系如下：



接下来我们讨论关于镜头边界检测的问题。镜头用于表示一个时间段或相同地点连续的动作，由一系列连续的图像帧组成，一般是由摄像机一次摄像的开始和结束所决定。由此可见，镜头的边界往往代表了一些视频的关键信息，而提取这些镜头的边界可以帮助快速获取一个视频中的关键信息。

两个镜头发生转换时，连续的两帧之间往往会出现相应位置像素点的突变，那么通过检测连续两帧之间像素点的变化来判断是否为镜头边界就是一种很直观的方法。接下来介绍的**连续帧相减，直方图相减**均是基于检测相邻两帧之间像素点变化的方法。

连续帧相减：

人眼是以像素为单位来辨别两幅图像相似度即差异的，两幅图像在同一位置相同的像素点个数越多，两幅图像就越相似。同样，在同一位置不同的像素点越少，两幅图差异越大。

**连续帧相减**即基于这个原理，对一段视频的相邻帧做差，如果差值大于某个阈值即可说明镜头发生了切换，接下来结合代码讨论这种方法。

通过opencv的VideoCapture方法可以提取视频中的图像帧，首先我们提取视频中的相邻两帧：

\_,lastFrame=self.video.read()

whileself.video.isOpened():

notLast,frame=self.video.read()

对两帧做差，判断差值是否达到阈值（通过多次实验，当差值与原图的比值大于10时较为合理）：

ifnp.sum(self.subImages(lastFrame,frame))/frame.size>10:

self.flag=True

self.showImage(frame)

self.st+=1

两幅图做差的具体实现：

defsubImages(self,img1,img2):

# in case of InterOverflow

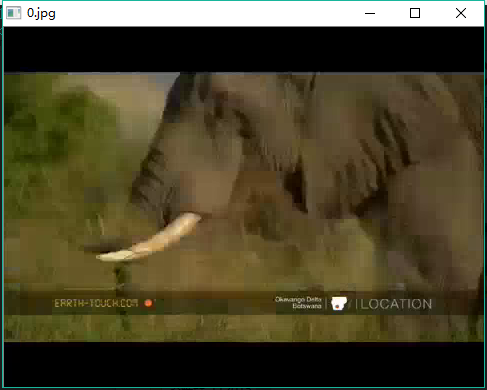
img1=img1.astype(np.int)

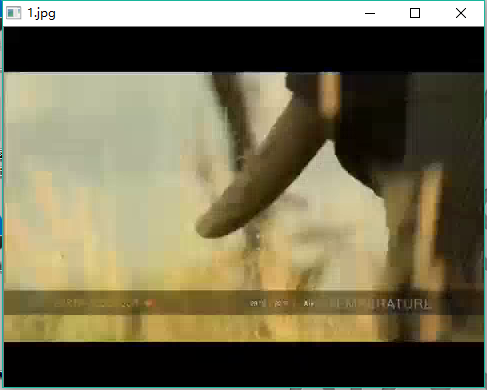
img2=img2.astype(np.int)

sub=abs(img1-img2).astype(np.uint8)

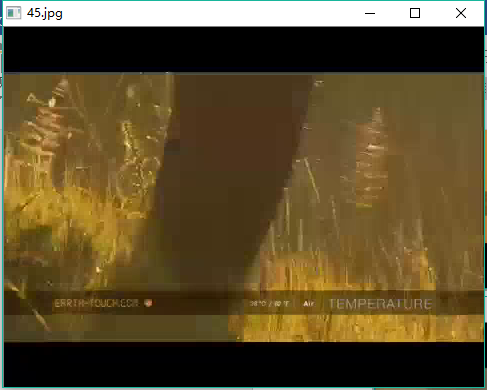
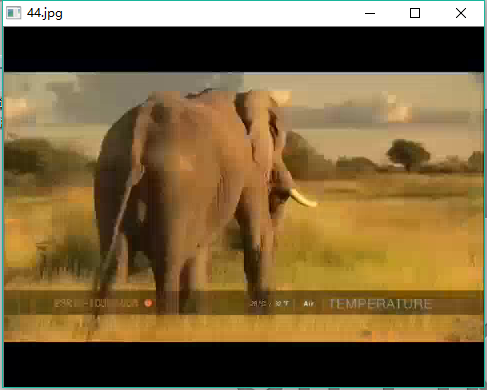
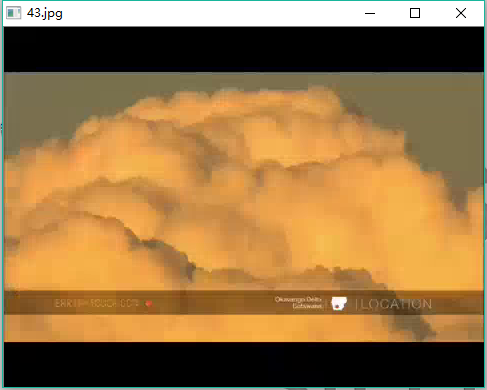
returnsub

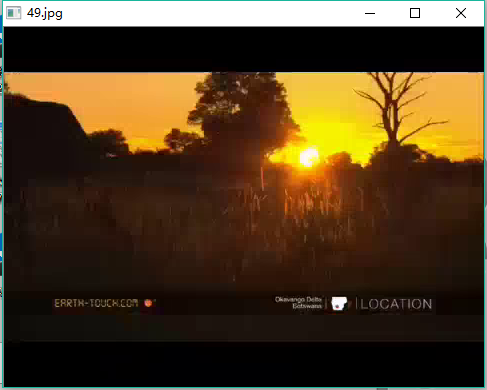
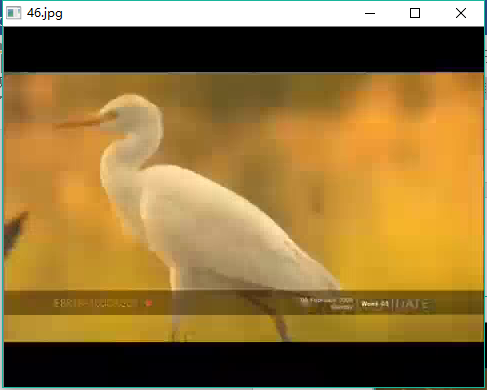
接下来我们用一段视频test1.avi进行测试（视频来源youtube）,当使用**连续帧相减法**时，检测出的镜头边界如下：





…





共检测出51个镜头



而从实际效果来看，当镜头出现明显变化时，是可以检测出的（如43-49副图），但同时也发现了一个很明显的问题：如果连续两帧只是做了轻微的缩放，也会被认为是镜头边界。从原理分析，当连续两帧做了轻微的缩放时，确实会造成相同位置像素点较大的差异，因此会被检测为镜头边界，但人眼基本分辨不出两帧之间的区别。

用test2.avi进行测试，检测出的镜头边界如下：





…





共找到49个镜头边界，但很明显误报率较高，这是由算法的原理决定的。Test2.avi是一段渐变切换的视频，图像帧之间的切换较为平稳，因此同一镜头内两帧之间的差值达到阈值时，就会被误认为不同的镜头

通过上述实验和查阅资料，可以对**连续帧相减法**作出总结：

**缺点：**

对摄像机的运动比较敏感。当摄像机做了拉远，推进平移等操作时，很容易被误检测为镜头边界。

阈值确定较为困难。通过实际测试，三段不同的视频用了三个不同的阈值。

对渐变式场景切换误报率较高。如上述实验中的test2.avi。

**优点：**

速度快。只做了两幅图相减的运算，上述实验对一段长达3:45的视频进行检测，实际检测时间不到10s。

对突变式场景切换检测较为准确。镜头有明显切换时，识别的准确率极高。

通过上述分析，我们可以提出改进**连续帧相减法**的方案：以画面缩放为例，人眼几乎分辨不出连续两帧之间的差别，说明两帧在同一位置附近的像素点基本相同，我们即可利用这一特性，不再直接对两图做差，而是对两图同一位置附近像素点的平均值做差。如取同一位置像素点3×3的领域求平均值再做差，实现代码类似图像旋转中的求邻域平均值。

直方图相减法：

针对**连续帧相减法**对误报画面平移的缺点，引入一种改进的方法**直方图相减法**。

连续帧相减法之所以会误报画面平移是因为当画面发生平移时，同一位置点的像素差值很有可能发生变化，两张图的差值就会长生较大差异，而此时两张图的直方图是基本类似的，因此可以很自然的想到通过两张图直方图的差值来确定是否发生镜头切换。

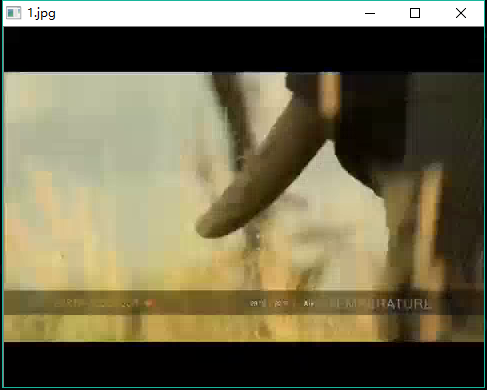
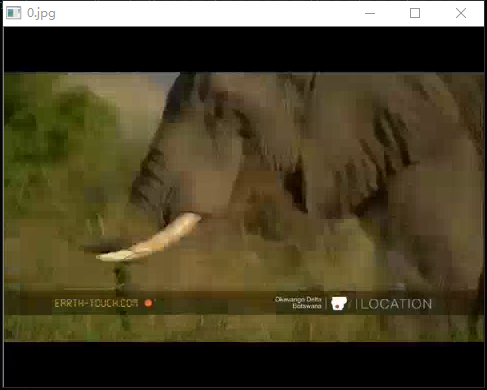
相比连续帧相减法，只需要在两图做差之前将两图转化为直方图即可，实现代码如下：

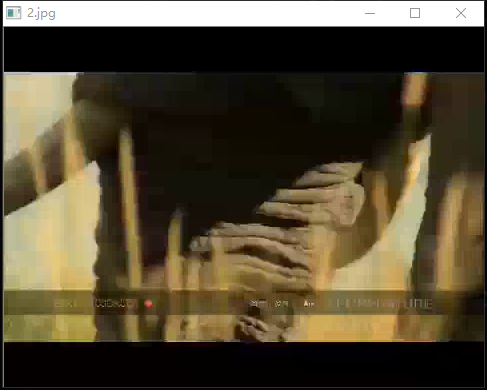
defconvertToGray(self,img):

#convert RGB to GRAY

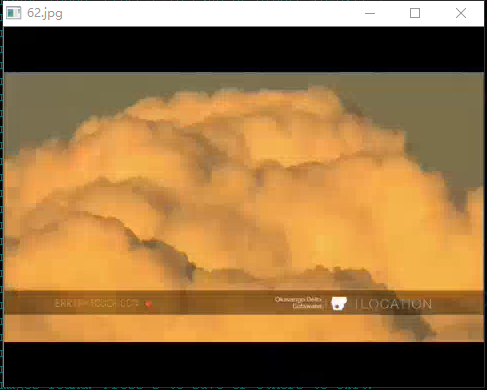
returncv2.cvtColor(img,cv2.COLOR\_RGB2GRAY)

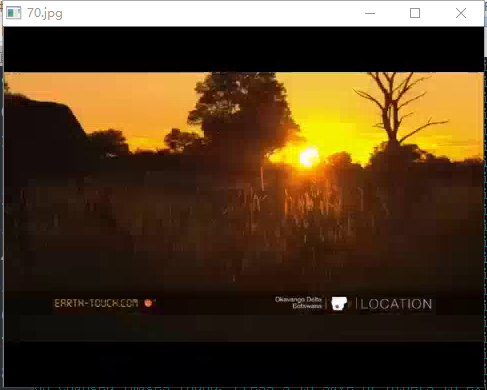
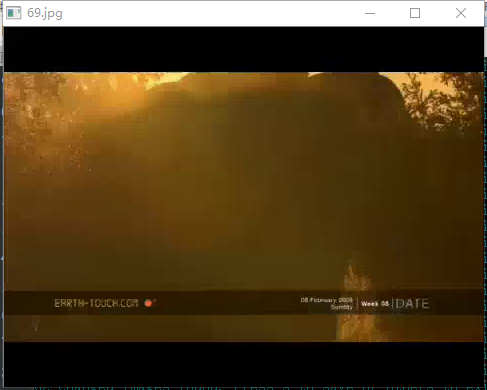
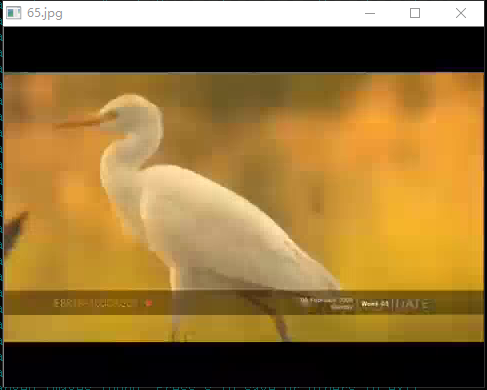
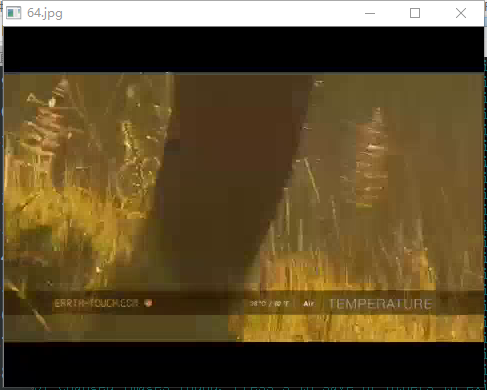
对test1.avi进行测试，测试结果如下：





…







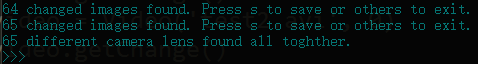
共找到72个镜头切换，通过对这些边界图像的分析，可以明显的发现相对于**连续帧相减法**，**直方图相减法**已经很少误报图像的平移，但对对焦过程中的误报率仍然较高，因为在对焦的过程中往往会出现图像直方图的剧烈变化，因而产生误报。

我们再用test2.avi进行测试，测试结果如下：



…





共找到65个镜头边界点。同样可以看出，**直方图相减法**仍然对渐变式的镜头切换没有太好的处理方法。

通过上述实验总结**直方图相减法**的优缺点：

**优点：**

1. 对镜头的平移不敏感。因为转化为直方图后，平移前后的两幅图像相差并不大。
2. 运算速度较快。相对连续帧相减法，直方图相减法仅仅多了一步转化为直方图的运算。

**缺点：**

1. 对焦过程中易发生误报。对焦过程中，图像的直方图发生了较大改变。
2. 对有类似直方图的图片会发生误报。可以想象两张图片直方图相似但内容完全不同时，直方图相减法还是会识别为同一场景。

接下来，我将简短的介绍**感知哈希法**。感知哈希法对视频中的每一帧做哈希运算生成一个哈希值，通过比较相邻图片的哈希值比较图片的差异性，当差异性大于阈值时，即可认为图片发生了镜头切换。因此hash算法的选取就至关重要。常用的哈希算法由以下步骤组成：

* 1. 对图片进行缩放
  2. 计算DCT（将图像信号转换到频域）
  3. 计算hash

通过上述分析，不同的边界检测算法有不同的优缺点，因此在不同的场景之下可以根据需要选取合适的边界检测算法。但无论那种算法都有缺点，改进这些算法的缺点可以是未来图像处理的一个方向。

完整代码：

#coding=utf-8

\_\_Auther\_\_="M4x"

importnumpyasnp

importcv2

importpdb

classVideo(object):

def\_\_init\_\_(self,file,method):

print"Only \*.avi is supported"

self.method=method# 0: Subtracting successive frames; 1: Subtracting histograms

self.st=0

self.flag=False

self.video=cv2.VideoCapture(file)

# if self.video.isOpened():

# print self.video.read()

# else:

ifnotself.video.isOpened():

print"BAD VIDEO FORMAT! Only \*.avi is supported!"

defsubImages(self,img1,img2):

# in case of InterOverflow

img1=img1.astype(np.int)

img2=img2.astype(np.int)

sub=abs(img1-img2).astype(np.uint8)

returnsub

defshowImage(self,img):

print"%d changed images found. Press s to save or others to exit."%(self.st+1)

cv2.imshow(str(self.st)+".jpg",img)

ifcv2.waitKey(0)&0xff==ord('s'):

cv2.imwrite(str(self.st)+".jpg",img)

cv2.destroyAllWindows()

defgetChange(self):

\_,lastFrame=self.video.read()

whileself.video.isOpened():

notLast,frame=self.video.read()

# print frame.size

ifnotnotLast:

ifnotself.flag:

print"The whole video is in the same camera lens."

cv2.destroyAllWindows()

self.video.release()

else:

print"%d different camera lens found all toghther."%self.st

break

#Subtracting successive frames

elifself.method==0:

ifnp.sum(self.subImages(lastFrame,frame))/frame.size>10:

self.flag=True

self.showImage(frame)

self.st+=1

#Subtracting histograms

elifself.method==1:

ifnp.sum(self.subImages(self.convertToGray(lastFrame),self.convertToGray(frame)))/frame.size>2:

self.flag=True

self.showImage(frame)

self.st+=1

lastFrame=frame

defconvertToGray(self,img):

#convert RGB to GRAY

returncv2.cvtColor(img,cv2.COLOR\_RGB2GRAY)

# video = Video('test1.avi', 0) # Subtracting successive frames

# video.getChange()

# video = Video('test2.avi', 0)

# video.getChange()

# video = Video('test1.avi', 1) # Subtracting histograms

# video.getChange()

video=Video('test2.avi',1)

video.getChange()