## 视频图像指纹算法工具

一．算法原理

在TV端需截取以图像的中心点为中心截取图像，如全屏是1920\*1080，则截取800\*600的图像即可。之所以要截取图像中央区域，是为了排除视频周边可能出现的广告、字幕等附加内容的影响。截取区域可以在实际项目中调整。



1. 图像的编码如下所示：

我们对800\*600的图像进行缩放，缩放到140\*140，然后对该图像进行分块，分为7\*7的块，对这个块采用相应的逻辑结构进行编码，即得到88位的码值，为了更好的说明编码方式，我们这里采用了对分块为5\*5的图像进行编码，所描述的编码方式和7\*7的编码方式是一样的。

构建单张图像的编码：



对于单张图像的视屏指纹形成我们利用如下方式来实现，主要步骤如下：

1. 获取到图像，并从图像中截取到大小为（W\*H）的块，这里的W、H根据实际情况的应用需求而定，这里我们采用大小为140像素\*140像素的正方形图像块，对图像进行处理，得到其对应的灰度图像。
2. 对图像进行分块，把正方形图像的宽高两个方向上均分为（2\*N+1）个块，其中N∈(1,2,3…..)N的取值根据实际情况需要设置，这里我们采用N=2（为了更好的说明问题，在实际的应用中我们采用N=3）;
3. 对于上述的图1我们把大小获取到140像素\*140像素的图像划分为5\*5个小块，平均每个块的大小为28像素\*28像素，对每个块进行编号，所得编号如图1中所示从（1，1）到（5，5）。
4. 对于编码过程采用下面的方式来实现：
5. 从图像块的中心块开始（（N+1）,（N+1）），分别与其周围的左上、上、右上、左、右、左下、下、右下进行比对，对应的块的编号分别为（N，N）、（N,（N+1））、（N,（N+2））、（（N+1）,N）、（（N+1）,（N+2））、（（N+2）,N）、（（N+2）,（N+1））、（（N+2）,（N+2））。对应于图1中的是以（3，3）为中心，分别按照之前所说的顺序对每个块（2，2）、（2，3）、（2，4）、（3，2）、（3，4）、（4，2）、（4，3）、（4，4）进行比对，按照下面的法则进行编码：计算所对比的两个块的像素值，把周围的像素块与中心块进行比对，若中心块像素值的和大于等于其对应相比某一块像素值的和，则编码为1；若小于则编码为0，按照这个法则，可以对中心块（（N+1）,（N+1））进行编码，得到8个编码。
6. 分别以（N，N）、（N,（N+2））、（（N+2）,N）、（（N+2）,（N+2））为中心，在图1中体现为以（2，2）、（2，4）、（4，2）、（4，4）为中心。同样采取八个方向，只是在这里的时候把有重复块比对的去掉，保留其中的一个，比如对于图1 中（2，2）为中心的八个方向，其中有一个方向块（2，2）与（3，3）之间的编码在（A）中已经有了说明，所以这里可以把这个对比所得的编码去掉，其它三个中心点的依次类推。这里根据图1我们得到的是28个编码。
7. 依次采用上述（B）方法对图像进行比对，把重复编码的去掉。在图1中所示的垂直和水平方向的块与块之间的编码信息没有被表示，因此如在向上的垂直方向上有（N-k）,（N+1））与（（N-k+1）,（N+1））比对、（（N-k+1）,（N+1））与（（N-k+2）,（N+1））比对……其中k∈(1,2,3…..N-1)，采用这种方法依次对水平向左方向，水平向右方向，和垂直向下方向进行编码。在本图1中垂直向上体现为块（1，3）与块（2，3）的比对，水平向左体现为块（3，1）和块（3，2）的比对，水平向右体现为块（3，4）和（3，5）的比对，垂直向下体现为块（4，3）与块（5，3）的比对，这里我们得到的是4个编码。

因此采用上述步骤（A）、（B）、（C）对于图1中说获取到的800像素\*600像素的图像，进行5\*5的块分割，得到的编码个数为：8+28+4=40，所以对于该图像产生了一个40位的0101的编码。对于7\*7的块我们同样采用上述的方式可以得到，在这里之所以选择5\*5是为了更块更方便的把编码的方式说明清楚。

我们可以对视频的每帧图像提取图像指纹码，但由于影视节目的复杂性，有以下几个问题需要注意。

* 1. 电影公司片头、电视剧片头等视频图像，需要分割出来，单独作为一个内容来处理，否则重复的太多。
  2. 视频内容在某些场景下，会有图像指纹重复的情况，如静止场景、回放场景等。所以，我们要处理相同指纹对应多个图像的问题。相同指纹问题，可以先使用multimap合并重复指纹值，再使用unordered\_map(hash\_map)做快速查找，如有重复指纹值，则返回多个图像信息。

f7fffff8ffff6fff7faffb

f7fffffcffff6fff7faffb

f7effffcffff6fff7faffb

fffffffcffff6fffffaffb

fffffffcffff6fffffaffb

fffffffcffff6fffffaffb

fffffffcffff6fffffaffb

fffffffcffff6fffffaffb

bffffffcffff6fffffaffb

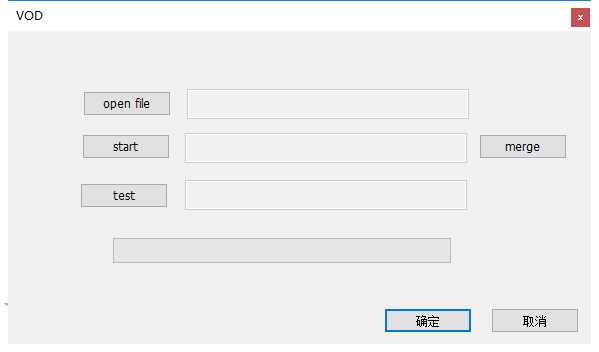
bffffffcffff6fffffaffb

bffffffcffff6fffffaffb

* 1. 图像指纹比对时，如完全匹配，则可以使用

unordered\_map(hash\_map)加快查找速度；如由于电视图像偏移拉伸等问题，导致完全匹配成功率低，则要考虑汉明距离近似匹配，那么大规模并发请求下，计算量会非常巨大。

二．工具说明

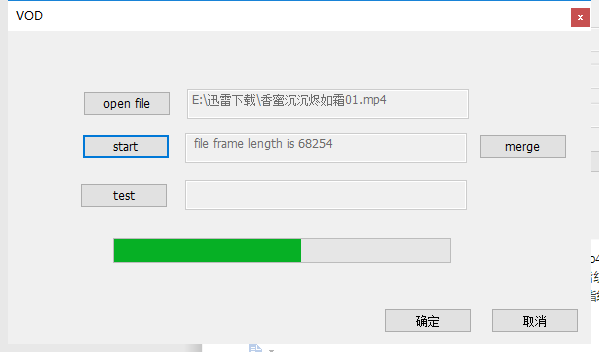


Open file：打开需要提取视频指纹码的视频(mp4格式)

Start：开始提取指纹码，下方进度条显示提取指纹码的进度

Test：提供测试帧图(jpg格式)，匹配文件中的指纹码。若匹配成功，会显示出相应的视频名\_时间戳

Merge：提供多个视频的指纹码合并操作

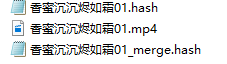


测试选择图片，如果匹配成功会显示具体的视频名\_时间戳，后期可以从中提取出视频名进行后续操作。

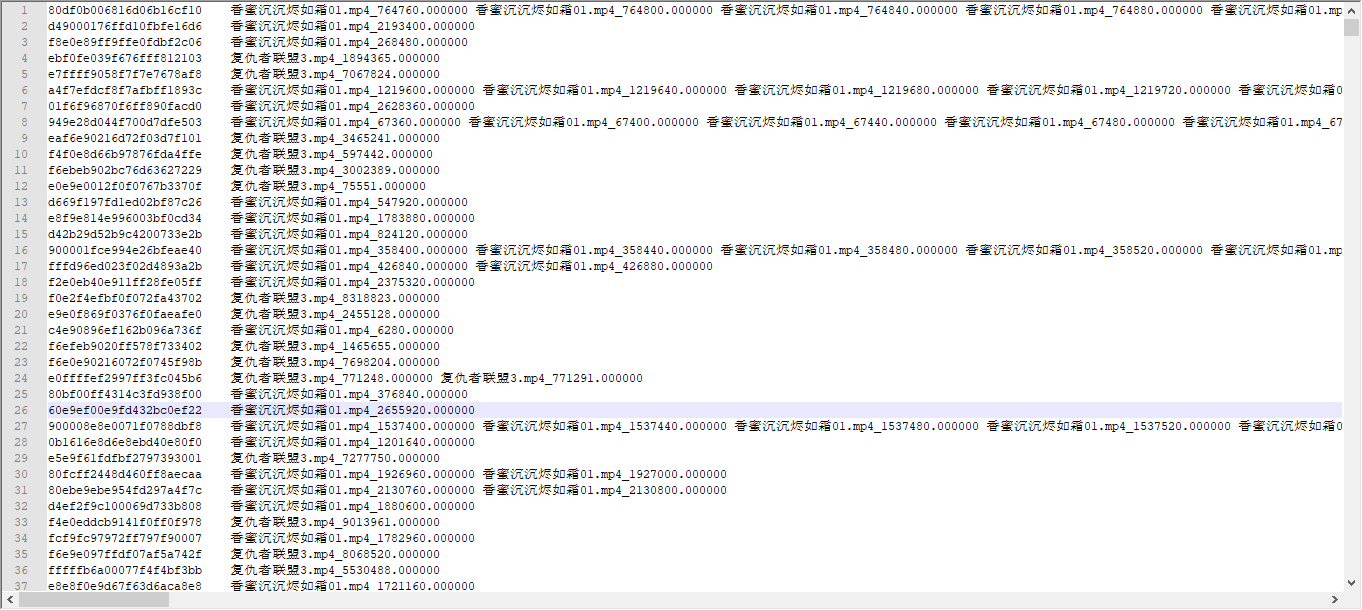
ps:测试所使用的图片如果是截图会产生一定的误差问题，主要原因是截图的尺寸和边缘像素问题，通过截取图像中心关键信息部分可以解决。



单个视频生成视频指纹码过程会在视频原目录下创造两个hash文件，\_.hash文件是未合并相同指纹码的文件，...\_merge.hash是合并了相同指纹码时间戳的文件。



合并多个视频的指纹码文件：



截图指纹码和帧图指纹码的误差：

第一例：





第二例：





可以看出只有几位的不同，思考过后可以考虑用取关键信息部分取消除这部分截图误差。

三．日后需要解决的问题

1.截取帧图尺寸问题：如果截取的过多，会丢失一些特殊场景的重要信息； 如果截取的过小，那么截取就没有意义，反而会占用内存。

测试用的截取范围：

帧图中心点(imgCentrex,imgCentrey)

截取起始点

(imgCentrex-cvRound(imgCentrex/2),imgCentrey-cvRound(imgCentrey/2))

截取长度(imgCentrexWidth,imgCentreHeigh)





现在的代码中是用全图操作的代码，裁剪代码已注释。

**存在问题：如果一帧图的中心部分是偏近于黑屏的暗色区域(比如黑洞)，而周围要裁剪的部分是信息部分，如何裁剪？**

2.执行效率问题：程序内有多次文件读写操作，还有大量的C++容器迭代，拖慢运行效率，后期需要优化。

3.测试时使用15s的短视频进行测试，发现opencv的异常问题，较短的小视频无法计算灰度直方图，但不影响电视剧和电影等长视频的计算。

4.具体的运行时间要看视频的大小。

5.test,merge按钮是在主线程进行操作的，可以将改为子线程，避免与主线程相互阻塞造成程序假死状态。