Configuration Manual

MSc Research Project
Video Summarisation based on key shots selection by using attention-based LSTM technique

Arghadeep Chowdhury
Student ID: X20189940

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Arghadeep Chowdhury		
Student ID:	X20189940		
Programme:	MSc Data Analytics	**Year:**	2021-2022
Module:	Research Project		
Lecturer:	Dr. Christian Horn		
Submission Due Date:	15th August 2022		
Project Title:	Video Summarisation based on keyframe selection using connectivity centroid clustering method		
Word Count:	**1415**	**Page Count: 14**	

I hereby certify that the information contained in this (my submission) is information about research I conducted for this project. All information other than my contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other authors' written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Arghadeep Chowdhury

Date: 14th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on the computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Arghadeep Chowdhury
Student ID: X20189940

1 Introduction

There are detailed specifications in the Configuration Manual to replicate the research and its outcomes in the individual environment. In addition to the cross-validation and evaluation of all the models built, the software and hardware requirement, data import and exploratory data analysis, data pre-processing, label encoding, feature selection, and the model-built Cross Validation and Evaluation. As shown in Section 2, the report provides information on the configuration of the environment.
Section 3 tells us about all the data collection. Section 4 is data exploration consisting of Frame Extraction. Key Frame extraction is interpreted in section 5. Section 6 contains all the information about video summarization. Section 7 provides the details about the video summarization of VSumm data. Section 8, explains how results are computed.

2 Environment

Details about the required hardware and software are provided in this section.

2.1 Hardware Requirements

Figure 1 talks about the hardware specifications. Intel i5-1135G7 with the 11th Generation Intel Core CPU @ 2.40 GHz, 8 GB installed DDR4 RAM Memory at speed of 2419 Mhz, 64 Bit Windows 11 Operating System.

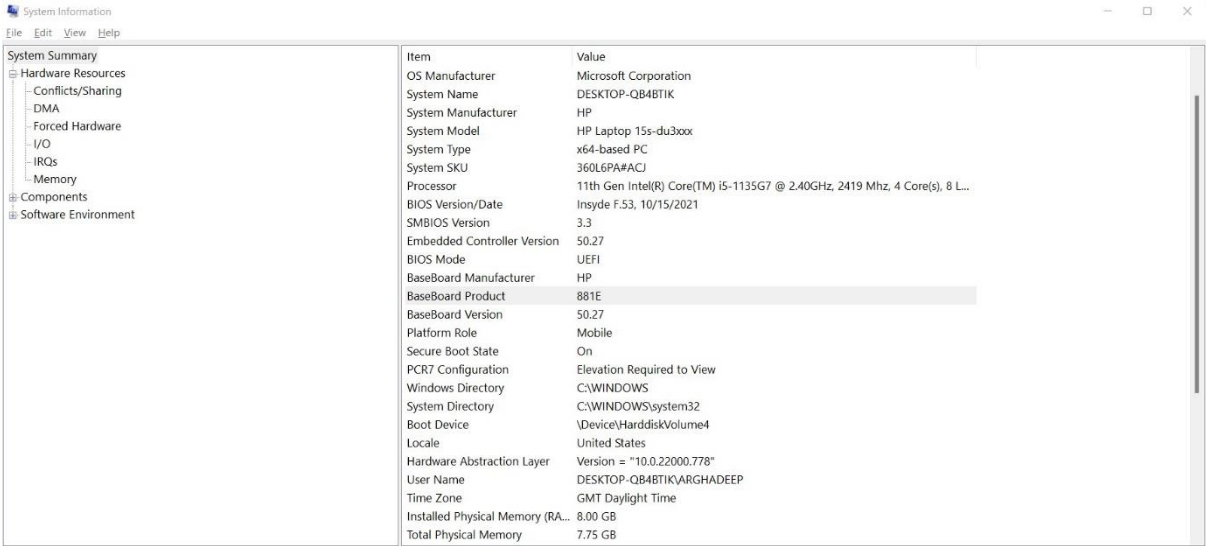

Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda 3 for Windows (Version 4.8.0)
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.7.6)

3 Data Collection

The data is collected from https://sites.google.com/site/vsummsite/home
This data consists of several videos along with their keyframes images.

4 Frame Extraction

This section covers the code done to extract frames from the video. To import video, we are using the cv2 library and all the frames are extracted and saved in a folder as an image.

```
import numpy as np
import pandas as pd
import time, glob, shutil
import matplotlib.pyplot as plt
%matplotlib inline
import cv2, os, math
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import svds, eigs
import hashlib
from skimage.measure import compare_ssim
from skimage.metrics import structural_similarity
```

Figure 2: Required Python Libraries

```
try:
    shutil.rmtree('frames')
except:
    print("Existing frames removed")
try:
    os.mkdir('frames')
except:
    print("Frames folder created")
```

Figure 3: Creating a folder to save frames

Figure 4 illustrates the code to read the video and the global variables required to process the frames of the video. After importing the video file we are initializing a 1944 dimensional array to store 'flattened' color histograms, and a dictionary to store the original frame as an array.

```
video = cv2.VideoCapture('database/database/v21.mpg')          #loading the videos
fname = os.path.basename('database/database/v21.mpg').split(".")[0]

array = np.empty((0, 1944), int)   # creating an array to store the original frames
frames=dict()
numFrames=0
start_time = time.time()
try:
    os.mkdir('frames/' + fname)
except:
    print("frames folder created")
```

Figure 4: Video loading

Figure 5 represents the read the video and reading each frame from the video and saving it as an image. The frame image is then processed, and all the frames are normalized. Then we read the video file and check if it got frames. If true, then we rearrange the frames to get frames in RGB order since cv reads frame in bgr order. After storing each frame (array) to D, so that we can identify key frames later we divide a frame into 3*3 i.e 9 blocks. Then we find histograms for each block and flatten the histogram to a one-dimensional vector to generate the feature vector. The arr is created by vertically stacking i.e. appending each one-dimensional vector to generate an N*M matrix (where N iseveralof frames and M is 1944). All frames are transposed into columns by transposing the array i.e M*N dimensional matrix.

```
while video.isOpened():
    ret, frame = video.read()
    if ret == True:
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        name = "frames/" +fname +'/'+ str(numFrames) + '.jpg'
        cv2.imwrite(name,frame_rgb)
        frames[numFrames] = frame_rgb
        height, width, channels = frame_rgb.shape
        if height % 3 == 0:
            hBlock = int(height/3)
        else:
            hBlock = int(height/3) + 1

        if width % 3 == 0:
            wBlock = int(width/3)
        else:
            wBlock = int(width/3) + 1
        h=0
        w= 0
        feature_vector = []
        for a in range(1,4):
            h_window = hBlock*a
            for b in range(1,4):
                frame = frame_rgb[h : h_window, w : wBlock*b , :]
                hist = (cv2.calcHist(frame, [0, 1, 2], None, [6, 6, 6], [0, 256, 0, 256, 0, 256])).flatten()
                feature_vector += list(hist)
                w = wBlock*b

            h = hBlock*a
            w= 0
        array =np.vstack((array, feature_vector))
        numFrames+=1
    else:
        break
```

Figure 5: Frame Extraction

```
print("--- %s seconds ---" % (time.time() - start_time))
print(len(frames))
final_array = array.transpose()
print(final_array.shape)
print(numFrames)
```

```
--- 54.37127494812012 seconds ---
3291
(1944, 3291)
3291
```

```
In [6]: framesList = glob.glob("./frames/" +fname +"/*.jpg") #storing the frames
        framesList
```

```
 './frames/v21\\1803.jpg',
 './frames/v21\\1804.jpg',
 './frames/v21\\1805.jpg',
 './frames/v21\\1806.jpg',
 './frames/v21\\1807.jpg',
 './frames/v21\\1808.jpg',
 './frames/v21\\1809.jpg',
 './frames/v21\\181.jpg',
 './frames/v21\\1810.jpg',
 './frames/v21\\1811.jpg',
 './frames/v21\\1812.jpg',
 './frames/v21\\1813.jpg',
 './frames/v21\\1814.jpg',
 './frames/v21\\1815.jpg',
 './frames/v21\\1816.jpg',
 './frames/v21\\1817.jpg',
```

Figure 6: Details of all the frames

```
img = cv2.imread(framesList[10])
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```

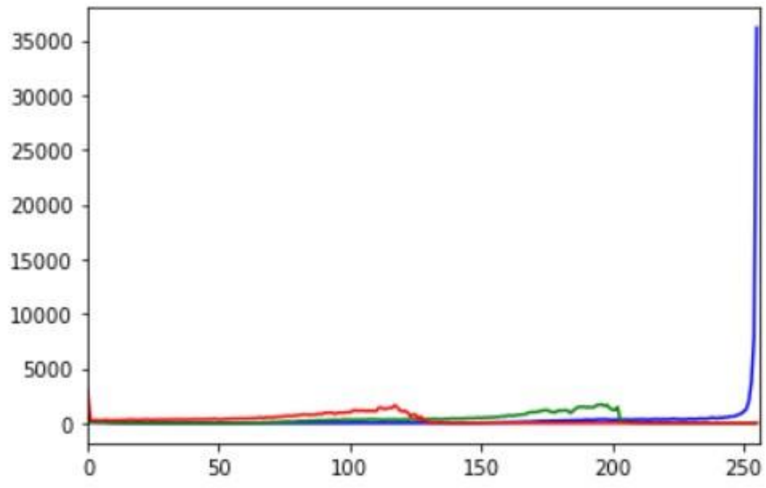

Figure 7: Histogram plot for the frames

5 Key Frame Selection

Key frame extraction is done using the Connectivity centroid algorithm for keyframe extraction. A sparse matrix is initialized, and the top 96 singular values of the vector generate projections. Projections are the column vectors i.e.; the frame histogram data has been projected onto the orthonormal basis formed by vectors

Connectivity Clustering method

```
k = 96
```

```
u, s, vect = svds(csc_matrix(final_array, dtype=float), k) #sparse matrix initialised
print(u.shape, s.shape, vect.shape)
```

```
(1944, 96) (96,) (96, 3291)
```

```
vectTranspose = vect.transpose() #generating projections
projections = vectTranspose @ np.diag  (s)
print(projections.shape)
```

```
(3291, 96)
```

Figure 8: Data splitting

Figure 9 below shows illustrates the use of dynamic clustering to find similar frames in a projected frame histogram, i.e. to make shots. Frame cluster is generated to store frames in the respective cluster and add the first two projected frames in the first cluster. Then the centroids of each cluster are stored, and the mean is taken to find the center of the centroid. A cosine similarity metric is used to quantify how similar is one vector to other.

```
frameCluster = dict()  # dynamic clustering of projected frame histograms to find all the frames that are similar
for i in range(projections.shape[0]):
    frameCluster[i] = np.empty((0,k), int)

frameCluster[0] = np.vstack((frameCluster[0], projections[0]))
frameCluster[0] = np.vstack((frameCluster[0], projections[1]))

clusterCentroid = dict()
for i in range(projections.shape[0]):
    clusterCentroid[i] = np.empty((0,k), int)

clusterCentroid[0] = np.mean(frameCluster[0], axis=0)

count = 0
for i in range(2,projections.shape[0]):
    similarity = np.dot(projections[i], clusterCentroid[count])/( (np.dot(projections[i],projections[i]) **.5) * (np.dot(cluster
    if similarity < 0.9:
        count+=1
        frameCluster[count] = np.vstack((frameCluster[count], projections[i]))
        clusterCentroid[count] = np.mean(frameCluster[count], axis=0)
    else:
        frameCluster[count] = np.vstack((frameCluster[count], projections[i]))
        clusterCentroid[count] = np.mean(frameCluster[count], axis=0)
```

Figure 9: Cluster centroid

Our next step is to determine how many data points are contained in each cluster. We can assume that sparse clusters indicate the transition between shots so we will ignore these frames which lie in such clusters and wherever the clusters are densely populated indicates they form shots and we can take the last element of these shots to summarize that particular shot where we find 0 in cluster data points indicates that all required clusters have been formed, so we can delete these from copied points. Last is the size of each cluster. So, a total of 90 shots is required.

```
clusterDataPts = [] #finding the number of data points in each cluster formed

for i in range(projections.shape[0]):
    clusterDataPts.append(frameCluster[i].shape[0])

last = clusterDataPts.index(0)
```

```
res = [index for index, value in enumerate(clusterDataPts) if value >= 5]
print(len(res))
```

```
90
```

Figure 10: Cluster data projections

```
for i in range(last):    #append each cluster to get multidimensional array of dimension
    points= np.repeat(i, clusterDataPts[i]).reshape(clusterDataPts[i],1)
    frameCluster[i] = np.hstack((frameCluster[i],points))
```

```
clusterArray=  np.empty((0,k+1), int)
for i in range(last):
    clusterArray = np.vstack((clusterArray,frameCluster[i]))
```

Figure 11: Cluster saved as an array

```
colnames = []       #converting the multidimensional array to a data frame
for i in range(1, k+2):
    col_name = "v" + str(i)
    colnames+= [col_name]
print(colnames)

clusterData= pd.DataFrame(clusterArray, columns= colnames)
```

```
['v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'v9', 'v10', 'v11', 'v12', 'v13', 'v14', 'v15', 'v16', 'v17', 'v18', 'v19', 'v
20', 'v21', 'v22', 'v23', 'v24', 'v25', 'v26', 'v27', 'v28', 'v29', 'v30', 'v31', 'v32', 'v33', 'v34', 'v35', 'v36', 'v37', 'v3
8', 'v39', 'v40', 'v41', 'v42', 'v43', 'v44', 'v45', 'v46', 'v47', 'v48', 'v49', 'v50', 'v51', 'v52', 'v53', 'v54', 'v55', 'v5
6', 'v57', 'v58', 'v59', 'v60', 'v61', 'v62', 'v63', 'v64', 'v65', 'v66', 'v67', 'v68', 'v69', 'v70', 'v71', 'v72', 'v73', 'v7
4', 'v75', 'v76', 'v77', 'v78', 'v79', 'v80', 'v81', 'v82', 'v83', 'v84', 'v85', 'v86', 'v87', 'v88', 'v89', 'v90', 'v91', 'v9
2', 'v93', 'v94', 'v95', 'v96', 'v97']
```

Figure 12: Cluster array converted to a pandas Data frame

Data frames are then created from multidimensional arrays. In the next step, we converted the cluster level from float to integer. Once the frames are filtered, only those that belong to required clusters or qualify for being in the shot are considered that have more than 90 frames in them. For each cluster /group take its last element which summarizes the shot i.e key-frame by finding key-frames (frame number so that we can go back to get the original picture) and output the frames in png format.

```
clusterData['v97']= clusterData['v97'].astype(int)  #converting the datatype
```

```
clusterData =  clusterData[clusterData.v97.isin(res)].groupby('v97').tail(1)['v97'].index
```

```
clusterData
```

```
Int64Index([  36,   76,  101,  122,  167,  219,  270,  321,  634,  755,  860,
             865,  887,  945,  951,  957,  962,  984, 1036, 1085, 1097, 1105,
            1113, 1119, 1165, 1176, 1221, 1230, 1239, 1336, 1395, 1445, 1486,
            1516, 1531, 1585, 1598, 1609, 1625, 1663, 1684, 1692, 1719, 1724,
            1734, 1744, 1795, 1832, 1844, 1851, 1860, 1885, 1909, 1916, 1957,
            1987, 2072, 2080, 2085, 2095, 2121, 2142, 2150, 2196, 2210, 2281,
            2289, 2294, 2375, 2461, 2547, 2649, 2691, 2729, 2767, 2782, 2792,
            2805, 2827, 2853, 2865, 2885, 2904, 2921, 2987, 2999, 3246, 3259,
            3266, 3290],
           dtype='int64')
```

Figure 13: Data Frame Processing

```
try:    #creating a folder to store the keyframes
    os.mkdir('keyframesCluster')
except:
    print("keyframesCluster folder created")
try:
    os.mkdir('keyframesCluster/' + fname)
except:
    print(fname + " folder created")
```

```
keyframesCluster folder created
v21 folder created
```

Figure 14: Keyframe folder creation

```
for cluster in clusterData:
    frame_rgb = cv2.cvtColor(frames[cluster], cv2.COLOR_RGB2BGR)
    name = 'keyframesCluster/'+ fname+'/keyframe'+ str(cluster) +'.jpg'
    cv2.imwrite(name, frame_rgb)
```

```
keyframes = glob.glob("./keyframesCluster/" +fname +"/*.jpg")  #storing the keyframes in
keyframes
```

```
['./keyframesCluster/v21\\keyframe101.jpg',
 './keyframesCluster/v21\\keyframe1036.jpg',
 './keyframesCluster/v21\\keyframe1085.jpg',
 './keyframesCluster/v21\\keyframe1097.jpg',
 './keyframesCluster/v21\\keyframe1105.jpg',
 './keyframesCluster/v21\\keyframe1113.jpg',
 './keyframesCluster/v21\\keyframe1119.jpg',
 './keyframesCluster/v21\\keyframe1165.jpg',
 './keyframesCluster/v21\\keyframe1176.jpg',
 './keyframesCluster/v21\\keyframe122.jpg',
```

Figure 15: Saving Keyframes as images

```python
fig, ax = plt.subplots(2, 5, figsize=(12, 8))
for frame, ax in zip(keyframes, ax.flatten()):
    ax.imshow(plt.imread(frame))
```

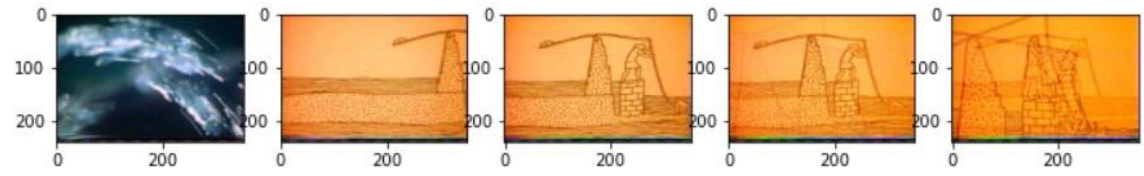

Figure 16: Visualising Keyframes

6 Video Summarisation

This section covers the code to generate the video from the extracted keyframes. The video is saved as mp4 files.

```python
def generateVideoSummary(frames, title, fps=10, fourcc=cv2.VideoWriter_fourcc('m', 'p', '4', 'v')): #
    img_array = []
    for filename in frames:
        img = cv2.imread(filename)
        height, width, layers = img.shape
        size = (width, height)
        img_array.append(img)
    out = cv2.VideoWriter(title, fourcc, fps, size)
    for i in range(len(img_array)):
        out.write(img_array[i])
    out.release()
```

```python
generateVideoSummary(keyframes, 'vidSummaryCluster.mp4') #generating the summarized video
```

Figure 17: Video summary creation

```python
generateVideoSummary(keyframes, 'vidSummaryCluster.mp4')
```

Figure 18: Implementation of Video Summary function

7 Video Summarisation of Vsumm

This section explains the process to generate the keyframes of the video data VSUMM. All the videos are looped through the steps of frame extraction, and keyframe extraction using connectivity centroid.

```python
def cleanSpace():
    try:
        shutil.rmtree('frames')
    except:
        print("Existing frames removed")
    try:
        os.mkdir('frames')
    except:
        print("Frames folder created")
```

Figure 19: Clearing space of folders created in the above steps

```python
def genKeyFrames(video):
    cleanSpace()
    fname = os.path.basename(video).split(".")[0]
    video = cv2.VideoCapture(video)

    array = np.empty((0, 1944), int)
    frames=dict()
    numFrames=0
    try:
        os.mkdir('frames/' + fname)
    except:
        print("frames folder created")

    while video.isOpened():
        ret, frame = video.read()
        if ret == True:
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            name = "frames/" +fname +'/'+ str(numFrames) + '.jpg'
            cv2.imwrite(name,frame_rgb)
            frames[numFrames] = frame_rgb
            height, width, channels = frame_rgb.shape
            if height % 3 == 0:
                hBlock = int(height/3)
            else:
                hBlock = int(height/3) + 1

            if width % 3 == 0:
                wBlock = int(width/3)
            else:
```

Figure 20: KeyFrame extractor function

```python
videos = glob.glob("datasets/database/*.mpg")
videos[:10]
```

```
['datasets/database\\v21.mpg',
 'datasets/database\\v22.mpg',
 'datasets/database\\v23.mpg',
 'datasets/database\\v24.mpg',
 'datasets/database\\v25.mpg',
 'datasets/database\\v26.mpg',
 'datasets/database\\v27.mpg',
 'datasets/database\\v28.mpg',
 'datasets/database\\v29.mpg',
 'datasets/database\\v30.mpg']
```

Figure 21: List of VSUMM videos

```python
for video in videos: #[:5]: #genera
    genKeyFrames(video)
```

```
v21 folder created
keyframes generated for v21
keyframes generated for v22
keyframes generated for v23
keyframes generated for v24
keyframes generated for v25
keyframes generated for v26
keyframes generated for v27
keyframes generated for v28
keyframes generated for v29
keyframes generated for v30
keyframes generated for v31
keyframes generated for v32
keyframes generated for v33
keyframes generated for v34
```

Figure 22: Generating keyframes of all the video

8 Model result

This section explains the performance of the keyframe extraction algorithm. The keyframes extracted by the algorithm implementation and the keyframes downloaded from the VSUMM are compared. They are compared on three criteria. First, that video is compared based on a perceptual hash that takes an image and returns a corresponding hash value. This works on the concept that there are different types of perceptual hashes, but a given image returns the same hash value, even if the image has been resized.

The second metric used is the total number of keyframes generated by both models.

The third metric is SSIM. The structural information model (SSIM) predicts that image degradation occurs when structural information changes. The idea of structural information is that pixels are strongly linked during the processing of an image, especially when they are spatially close. Information about the structure of objects in the visual scene is contained in these dependencies.

```
for video in videos: #[:5]:  #creating a function to find the similarity between the original and dataset keyframes
    fname = os.path.basename(video).split(".")[0]
    keyframes = glob.glob("E:/python/keyframesCluster/" +fname +"/*.jpg")
    name1= 'videoSummary' + fname + '.mp4'
    generateVideoSummary(keyframes, name1)
    keyframesvsum = glob.glob("E:/python/VSUMM2Summary/" +fname +"/*.jpeg")
    name2= 'videoSummaryVSUM' + fname + '.mp4'
    generateVideoSummary(keyframesvsum, name2)
    file1 = open(name1, 'r', encoding='cp437').read()
    file2 = open(name2, 'r', encoding='cp437').read()
    print('For summary of ' + fname)
    if hashlib.sha512(file1.encode('utf-8')).hexdigest() == hashlib.sha512(file2.encode('utf-8')).hexdigest():
        print ('They are the same')
    else:
        print ('They are different')
    if len(keyframes) == len(keyframesvsum):
        print("Number of keyframes is same")
    elif len(keyframes) < len(keyframesvsum):
        print("Number of keyframes is less")
    else:
        print("Number of keyframes is more")
    ssim =0
    for i in range(0,len(keyframesvsum)) :
        img = cv2.imread(keyframes[i])
        img_2 = cv2.imread(keyframesvsum[i])
        ssim= ssim+structural_similarity(img, img_2, multichannel=True)
    print("The similarity scores of the keyframes is" , ssim)
    print('-------------------------------------------------------')
```

Figure 23: Keyframes comparison

References

- https://www.sciencedirect.com/science/article/abs/pii/S0167865510002783?via%3Dihub
- https://www.imatest.com/docs/ssim/
- https://cse.hkust.edu.hk/~rossiter/mm_projects/video_key_frame/key_frame_index.html
- https://sites.google.com/site/vsummsite/home