Fox's Video Library (Version 2.0)

Ludmila I Kuncheva

November 14, 2021

```
%========================================================================
% (c) Fox's Vis Toolbox                                          ^--^
% DD.MM.YYYY ------------------------------------------------    \oo/
% ------------------------------------------------------------------\/-%
```

Contents

1 Introduction

This library was developed in the process of preparing several publications on video processing and summarisation.

There are many (too many?) methods for video summarisation using keyframes. The irony is that random frame selection (as the candidate summary) is not that bad at all! There is very little rigour in comparing one summary with another. Even to the human eye, a random summary could be just as good as a summary created through the most sophisticated method based on semantic feature selection using convolutional neural networks. The summaries are typically compared with a "ground truth" collection, which is a human-devised summary. But different users pick different frames! Whose ground truth are we matching? Besides, the matching metrics (e.g., F1 measure) don't offer sufficient fidelity to distinguish between good and bad candidate summaries. Scrap that, the difference between a good and a bad summary is in the eye of the beholder and can hardly be quantifies with a rigorous metric. Then why are there so many video summarisation methods? Honestly, I don't know. :)

We studied various methods for comparing two summaries, regarding both as *sets* of frames (not a storyboard or a sequence). Assuming that each frame is represented in some numerical feature space, we considered 6 matching methods and 3 distance metrics.

Our work went further into semantic summarisation where the user asks for a particular term, for example, 'phone', 'food', or 'car' (work mostly by Dr Paria Yousefi) []. The summary is then prepared only from the frames related to the concept of interest, and this time, it is a time-related. We developed a "compass" summary which shows the whole video as a 360deg circle, and the keyframes are positioned alongside the circle.

The second direction of our work was budget online video summarisation (work mostly by Dr Clare Matthews) []. Imagine that we are given a budget of 20 frames, and the video is streaming through our system. We don't know in advance how long the video is, but we are not allowed to exceed the budget. In other words, we must keep the 20 most representative frames at all times, and dynamically decide which ones to replace if need be. At the end of the video we need to present the 20-keyframe summary to the user.

This library contains some of the tools for extracting features from images (Section 2), comparison of two keyframe summaries (Section 3).

2 Feature extraction

```
function x = fox_get_features(im,fstr,blocks,bins)
%=========================================================================
%fox_get_features Returns a set of features extracted from an image
%   x = fox_get_features(im,fstr,blocks,bins)
%   Five different feature types are encoded.
%
%   Input -----
%      'im': RGB input image
%      'fstr': feature string - one of these - RGB, HSV, CHR, OHT, H
%      'blocks': number of blocks to split the image into, e.g., 1, 4, 9
%                (should be a square number)
```

```
%       'bins': number of bins for histogram features
%
%   Output -----
%       'x': vector-row with the extracted features
%==========================================================================
```

The following feature spaces are available from this function:

1. *RGB moments.* (`fstr = 'RBG'`) The function returns the mean and standard deviation of the red (R), green (G) and blue (B) channels for each the image ($n = 6$ features).

2. *HSV space.* (`fstr = 'HSV'`) The function returns the mean and standard deviations of the three HSV components ($n = 6$ features).

3. *Chrominance.* (`fstr = 'CHR'`) The function returns the mean and standard deviation of Chrominance components C_1 and C_2 ($n = 4$ features) calculated as:

$$C_1 = \frac{R}{q}, \;\; C_2 = \frac{G}{q}, \;\; q = \sqrt{R^2 + G^2 + B^2},$$

4. *Ohta space.* (`fstr = 'OHT'`) The function returns the mean and standard deviation of features I_1, I_2' and I_3' of Ohta space ($n = 6$ features) calculated as

$$\begin{aligned} I_1 &= \frac{1}{3}(R + G + B) \\ I_2' &= R - B \\ I_3' &= \frac{1}{2}(2G - R - B) \end{aligned}$$

The image can be processed as a whole (`'blocks' = 1`) or split into $K \times K$ uniformly sized cells (`'blocks' = K^2`). The number of elements of `x` is $K^2 \times n$, where n if the number of features for a single image. So, for $K = 3$ (`'blocks' = 9`), and feature space RGB (`fstr = 'RBG'`) the total number of features is $9 \times 6 = 54$.

The returned vector `x` contains the means followed by the standard deviations. For example, a 4-block RGB feature vector will be:

$$\begin{aligned} \mathbf{x} = [&m_{r1}, m_{g1}, m_{b1}, s_{r1}, s_{g1}, s_{b1}, m_{r2}, m_{g2}, m_{b2}, s_{r2}, s_{g2}, s_{b2}, ... \\ &m_{r3}, m_{g3}, m_{b3}, s_{r3}, s_{g3}, s_{b3}, m_{r4}, m_{g4}, m_{b4}, s_{r4}, s_{g4}, s_{b4}], \end{aligned}$$

where m denotes mean, s denotes standard deviation, rgb stand for red, green, blue, and the number tag is the block number.

For feature string 'H', the function will return a hue value histogram in the specified number of bins. The bins span the whole range of possible hue values. Thus, a solid red image will be represented with a histogram containing 1 in the first bin (after scaling to sum 1), and zeros elsewhere.

Python implementation of the feature extraction function is provided in folder `python_feature_extraction`. The functions are stored in library `feature_extractor.py`.

To select features from the whole video, use:

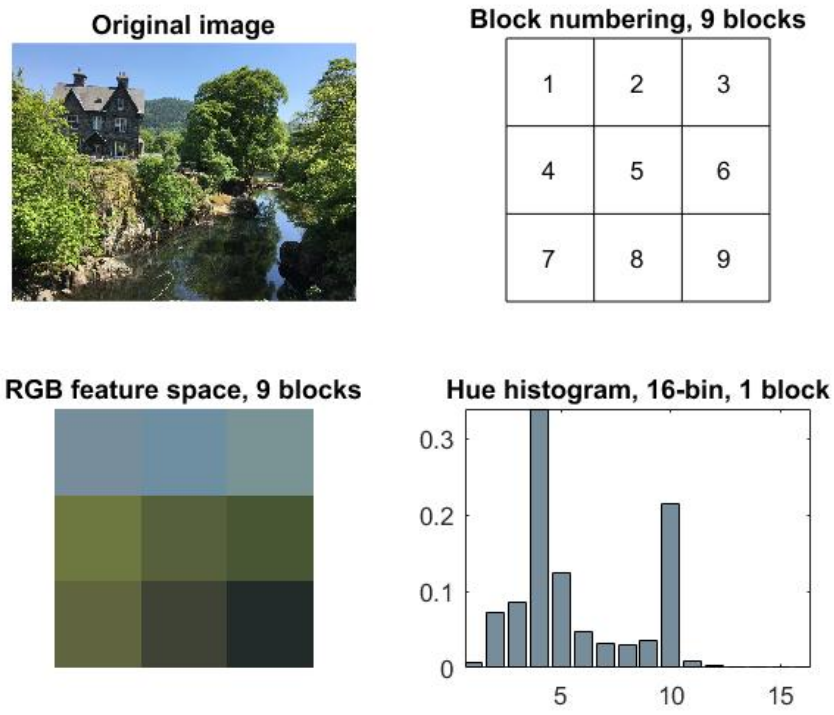


Figure 1: Example of feature extraction.

```
function data = fox_features_from_video(fn,feature_string, blocks, bins)
%=========================================================================
%fox_features_from_video Calculate the features of the frames in a video.
%   data = fox_features_from_video(fn,feature_string, blocks, bins)
%   extracts the features of the frames in a video and saves the data in
%   a mat file of the name of the video.
%
%   Input -----
%      'fn': the name of the video file (full path)
%      'feature_string': One of the following - RGB, HSV, CHR, OHT, H
%                        (default 'RGB')
%      'blocks': number of blocks to split the frames (default 9)
%      'bins': number of bins for histogram features (default 8)
%
%   Output -----
%      'data': an N-by-M array with data, where N is the number of frames
%      of the video and M is the number of extracted features.
%=========================================================================
```

`FoxsVideoToolboxTester_Features` demonstrates feature extraction through function `fox_get_features`. The code uploads an image, extracts RGB features using 9 blocks (3 by 3), and plots the colours of the respective blocks as in Figure 1. The numbering of the cells is also shown. Next, a 16-bin histogram of the hue values is extracted and shown in the bottom right sub-figure.

3 Comparing two summaries

3.1 Frame matching

```
function [match, value] = fox_match_two_vectors(a,b,threshold,d)
%=========================================================================
%fox_match_two_vectors Match two vectors
%   [match, value] = fox_match_two_vectors(a,b,threshold,d) calculates
%   the distance between two vectors and returns a logical value 'match'
%   as well as the value of the distance.
%
%   Input -----
%      'a': 1-by-N vector with image representation
%      'b': 1-by-N vector with image representation
%      'threshold': threshold for the match; match is declared if
%                   distance is <= threshold
%      'd': distance type
%           1 Euclidean, 2 Minkowski (L1 norm), 3 Cosine (angular),
%           4 SURF
%
%   Output -----
%      'match': match output - 0 no match, 1 match (logical variable)
%      'value': distance value
%=========================================================================
```

Function `fox_match_two_vectors` matches two frames, $\mathbf{a} = [a_1, \ldots a_n]^T$ and $\mathbf{b} = [b_1, \ldots b_n]^T$, represented as vectors in some n-dimensional feature space. The two vectors must have the same number of elements. Each vector can be either a row or a column. The following distances are included:

- (1) Euclidean

$$v = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2},$$

- (2) Minkowski

$$v = \sum_{i=1}^{n} |a_i - b_i|,$$

- (3) Cosine (angular distance)

$$v = \frac{2}{\pi} \cos^{-1} \left(\frac{\mathbf{a}^T \mathbf{b}}{||\mathbf{a}||\ ||\mathbf{b}||} \right).$$

Function `fox_match_two_frames_surf` matches two frames, A and B, represented as images. SURF descriptors are used.

```
function [match, value] = fox_match_two_frames_surf(A,B,threshold,vis)
%=========================================================================
%fox_match_two_frames_surf Calculate the match between two images using
%SURF features.
%   [match, value] = fox_match_two_frames_surf(A,B,threshold,vis) uses
```

```
%   SURF features to calcuate the match between two images
%
%   Input -----
%      'A': image 1
%      'B': image 2
%      'threshold': thresholf for the match; match is declared if
%                   distance is <= threshold
%      'vis': 0/1 visualisation flag
%
%   Output -----
%      'match': match output - 0 no match, 1 match (logical variable)
%      'value': distance value
%=========================================================================
```

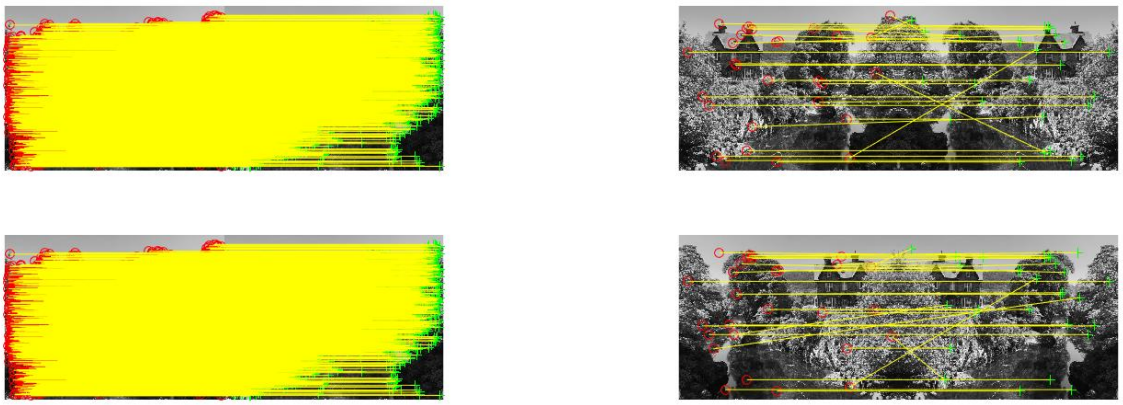
`FoxsVideoToolboxTester_MatchingFrames` demonstrates the use of the matching functions by comparing an image with its reverse, both with 1 block (there should be no difference), and 9 blocks. Note that calling the functions for every pairwise comparison is slow. For the experiments in the related papers, we wrote bespoke code which did the calculations in bulk. Due to its problem-woven nature, however, this code is not easily usable as a part of a library. The output is shown below and in Figure 2 for the SURF feature matching:

```
Testing Frame Matching (RGB), Euclidean

Threshold 1.2247
Original vs reversed - 1 block: 1, value 0.0000
Original vs reversed - 9 blocks: 1, value 0.6092

Testing Frame Matching (H-histograms, 32 bins) Minkowski

Threshold 0.5000
Original vs reversed - 1 block: 1, value 0.0000
Original vs reversed - 9 blocks: 0, value 0.7491


Testing Frame Matching SURF

Threshold 0.5000
Original vs Original 1, value 0.0000
Original vs Reverse 0, value 0.9885
```

3.2 Pairing (summary matching) methods

To compare two keyframe summaries, C (candidate) and GT (ground truth), we need a method to pair frames, one from each summary. Function `fox_pairing_frames` offers 6 methods [1, 2]. An example of the use of the function is given in `FoxsVideoToolboxTester_MatchingSummaries`.

```
function [F,number_of_matches,mcs,mgt] = fox_pairing_frames(...
    matchMatrix,threshold,pairingMethod)
```



(a) Matching self
(`match = 1, value = 0`)

(b) Matching reverse image
(`match = 0, value = 0.9885`)

Figure 2: SURF feature matching results.

```
%=========================================================================
%fox_pairing_frames Pair frames from two summaries
%   [F,number_of_matches,mcs,mgt] = fox_pairing_frames(matchMatrix, ...
%   threshold,pairingMethod) calculates the statistics of the paired
%   frames from two candidate keyframe summaries.
%
%   Input -----
%      'matchMatrix': matrix with the pairwise *distances* between the
%                     frames of summaries A and B. The size of the
%                     matrix is M-by-N where M is the number of frames
%                     in summary A, and N is the number of frames in
%                     summary B.
%                     +++++ Note: A is the candidate summary and B is
%                     the ground truth.+++++
%      'threshold': threshold for the matching; values in matchMatrix
%                   smaller than threshold are declared matches
%      'pairingMethod': must be one of the following [1,2]
%                   1: Naive Matching
%                   2: Greedy Matching
%                   3: Hungarian Matching
%                   4: Mahmoud Method
%                   5: Kannappan Method
%                   6: Maximal Matching (Hopcroft-Karp)
%
%   Output -----
%      'F': F-measure (see the Note above)
%      'number_of_matches': number of matched frames
%      'mcs': indices of matched frames from the candidate summary
%      'mgt': indices of matched frames from the groun truth
%
% -------------
% [1] Kuncheva L. I., P. Yousefi and I. A. D. Gunn, On the Evaluation
% of Video Keyframe Summaries using User Ground Truth,
% arXiv:1712.06899, 2017.
% [2] Gunn I. A. D., L. I. Kuncheva, and P. Yousefi, Bipartite Graph
% Matching for Keyframe Summary Evaluation, arXiv:1712.06914, 2017.
%=========================================================================
```

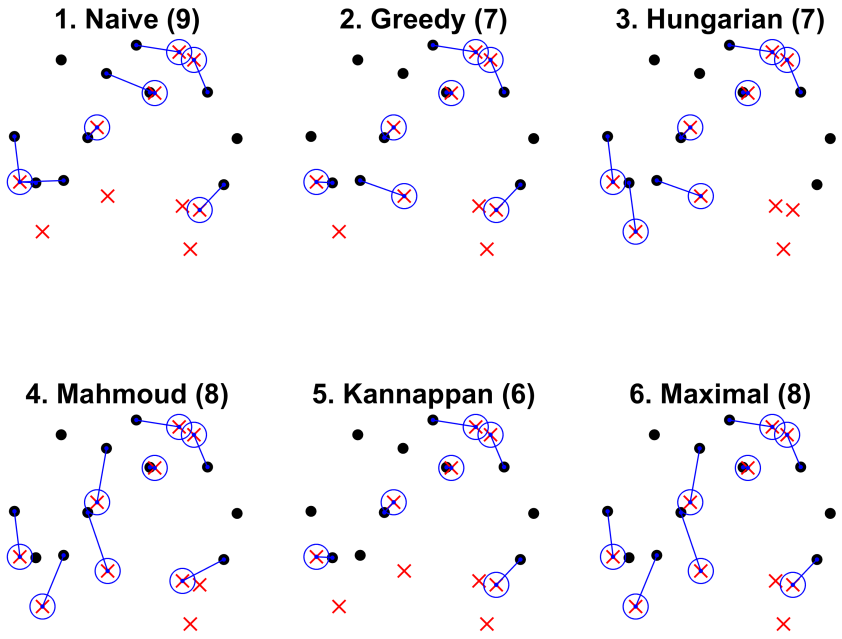


Figure 3: Results from the 6 matching (pairing) methods for two hypothetical summaries. The ground truth (10 frames) is shown with red crosses, and the candidate summary (11 frames), with black dots. The matching frames are joined with blue lines.

Two hypothetical summaries in the 2D space are shown in the plots in Figure 3. The frames for the ground truth (10 frames) are plotted with red crosses, and the ones for the candidate summary (11 frames), with black dots. The matching frames are joined with blue lines. The Ground Truth frame of the matched pair is also circled. The number of matches is given in brackets next to the method's name in the title of the plot. The function returns the F-values too. It is clear that while some matching methods give high number of matches, their results (pairs that have been matched) are not very convincing.

3.3 Example of the pipeline

`FoxsVideoToolboxTester_Pipeline` demonstrates the work of the entire pipeline of evaluating the similarity between two keyframe summaries. We made the following choices (the same as De Avila et al. [3]):

1. Feature space: H histogram, 16 blocks.
2. Distance: Minkowski (same as L_1 norm, Manhattan).
3. Threshold for frame similarity: 0.5.
4. Pairing method: Greedy

The example compares the user #1 summary (ground truth, 11 frames) of video v21 from

Ground truth

Candidate summary

Figure 4: Results from the match of the two summaries. The 6 matched frames are shown at the front (blue rims).

the VSUMM database[1] and the summary obtained using the OV method (also included in the database, 9 frames). The frames of the two summaries are stored in folder `KeyframeSummaries`.

The number of matches is 6, $F = 0.6$. The two summaries are shown in Figure 4. The matched frames are arranged at the front of the summary (not chronologically). The matched frames have dark blue rims.

4 Keyframe selection methods

4.1 Uniform summary

The uniform segmentation takes the total number of frames n and returns the indices of k uniformly spread frames. The video is split in k segments of equal length and the middle frame of each segment is returned. For example, if we have $n = 100$ frames and want $k = 10$-summary, the function returns frame indices:

6 16 26 36 46 55 65 75 85 95

```
function index = fox_uniform_summary(n,k)
%=========================================================================
%fox_uniform_summary Return the indices of k uniformly distributed frames
%within n video frames.
%   function index = fox_uniform_summary(n,k)
%
%   Input -----
%      'n': total number of frames in the video
%      'k': number of frames in the summary
%
%   Output -----
%      'index': the indices of the chosen frames
%
%   Example:
%      index = fox_uniform_summary(100,10)
%=========================================================================
```

[1] `https://sites.google.com/site/vsummsite/download`

The keyframes should be subsequently retrieved directly from the video or from a folder containing the individual frames. A function `fox_store_frames(fn,odir)` is provided in Section 5 to store all the frames of a video in a given folder. Function `fox_retrieve_frames(fn,index)` uses the index to retrieve the frames from the original video.

4.2 Segmentation summary

The segmentation summary takes the data extracted from the frames to find out which parts of the video (segments) should be represented by a singe keyframe. We followed the segmentation method described in the paper by Doherty et al. [4]. Subsequently, we take the frame closest to the centroid of each segment as advocated by Kuncheva et al. [5].

```
function frame_index = fox_segmentation_summary(data,par)
%=========================================================================
%fox_segmentation_summary Apply segmentation and keyframe extraction
% using the closest-to-centroid method for each segment.
%   frame_index = fox_segmentation_summary(data,par) segments the video
%   represented as data into a number of segments obtained from the
%   algorithm. The segmentation method follows [1]. Subsequently, we
%   take the frame closest to the centroid of each segment as advocated
%   in [2].
%
%   Input -----
%      'data': a data array of size N-by-M where N is the number of
%              frames in the video and M is the number of features
%      'par': an array with parameters
%             par(1) = number of stds for the change threshold (default 1)
%             par(2) = minimum segment length (# frames, default 6)
%             par(3) = block size for smoothing (# frames, default 4)
%
%   Output -----
%      'frame_index': an array with the indices of the selected keyframes
%
% [1] % Doherty, A. R.; Byrne, D.; Smeaton, A. F.; Jones, G. J. F. &
% Hughes, M. Investigating keyframe selection methods in the novel
% domain of passively captured visual lifelogs, Proceedings of the 2008
% International Conference on Content-based Image and Video Retrieval
% CIVR, 2008, 259-268.
% [2] Kuncheva L. I., P. Yousefi & J. Almeida, Comparing keyframe
% summaries of egocentric videos: Closest-to-centroid baseline,
% Proceedings of The Seventh International Conference on Image
% Processing Theory, Tools and Applications (IPTA 2017), 2017,
% Montreal, Canada.
%=========================================================================
```

4.3 Example video

We take as an example the video "Playing ball" (`playing_ball.mp4`) from the SumMe project: `https://paperswithcode.com/dataset/summe`. A dog and a crow play with a ping-pong ball and the camera moves around following the crow. There are no well-defined segments as such; the game goes continually throughout. Therefore, the uniform summary can be just as good the segmentation-based one.

(a) Uniform (9 frames)

(b) Segmentation-based

Figure 5: Results from the two summarisation methods on the video "Playing ball".

`FoxsVideoToolboxTester_KeyframeSummary` applies the two summarising methods (uniform and segmentation) to the chosen video. The two summaries are shown in Figure 5.

5 Auxiliary functions

- Plot a cell grid with short text in the middle of each cell

```
function fox_plot_grid(r,c,t,axesHandle,fontName)
%=========================================================================
%fox_plot_grid Plot a cell grid with short text in the middle of each
%cell
%    fox_plot_grid(r,c,t,axesHandle,fontName)
%
%    Input -----
%       'r': number of rows
%       'c': number of columns
%       't': an r-by-c cell array with text in the cells (optional)
%            (No provision is made to centre long text within the cell.)
%       'axesHandle': handle of the axes to plot the grid (optional)
%       'fontName': string (optional)
%
%    Example 1:
%       fox_plot_grid(5,4)
%
%    Example 2:
%       t = {'sun','rain','hale';'dry','cloudy','hurricane'};
%       fox_plot_grid(2,3,t)
%
%    Example 3:
%       t = {'sun','rain','hale';'dry','cloudy','hurricane'};
%       figure, fox_plot_grid(2,3,t,[],'Calibri')
%=========================================================================
```

- Visualise a summary as a montage, with an option to put a rim of different colour for each image.

```
function fox_montage(f,s,c,w)
%=========================================================================
%fox_montage Plots a customised montage of images with a given border
%   fox_montage(f,s,c,w)
%
%   Input -----
%      'f': cell array with n images of the same size
%      's': size [rows,columns] of the montage (optional)
%      'c': colour of the outside border of each image (optional)
%      'w': with of the border, proportion of the smaller dimension of
%           the image (optional, default = 0.08)
%      If 'c' is not specified, MATLAB 'montage' function is used
%      (with no gaps between the images). If 'c' is of size (1,3),
%      this is taken to be the colour of all the borders. The values
%      must be between 0 and 1. If c is of size (n,3), each image will
%      have a border with colour specified in the corresponding row
%      of 'c'.
%=========================================================================
```

- Store all the frames of a video into a folder. If the folder does not exist, it will be created.

```
function fox_store_frames(fn,odir)
%=========================================================================
%fox_store_frames Store the frames in the video in file 'fn' in folder
%'odir'.
%   fox_store_frames(fn,odir)
%
%   Input -----
%      'fn': video filename
%      'odir': folder name without the divider '/' or '\'
%=========================================================================
```

- Use frame numbers to retrieve the images from the original video.

```
function frames = fox_retrieve_frames(fn,index)
%=========================================================================
%fox_retrieve_frames Returns the frames with numbers in array 'index' by
% reading the original video.
%   frames = fox_retrieve_frames(fn,index)
%
%   Input -----
%      'fn': video filename (full path)
%      'index': number of the frames to be retrieved
%
%   Output -----
%      'frames': a cell array with the retreieved images in the order
%                they are given in 'index'
%=========================================================================
```

References

[1] L. I. Kuncheva, P. Yousefi, I. A. D. Gunn, On the evaluation of video keyframe summaries using user ground truth, arXiv:1712.06899 (2017).
URL `https://lucykuncheva.co.uk/papers/lkpyigArXiv17.pdf`

[2] I. A. D. Gunn, L. I. Kuncheva, P. Yousefi, Bipartite graph matching for keyframe summary evaluation, arXiv:1712.06914 (2017).
URL https://lucykuncheva.co.uk/papers/iglkpyArXiv17.pdf

[3] S. E. F. De Avila, A. P. B. Lopes, A. Da Luz, A. De Albuquerque Araújo, VSUMM: A mechanism designed to produce static video summaries and a novel evaluation method, Pattern Recognition Letters 32 (1) (2011) 56–68.

[4] A. R. Doherty, D. Byrne, A. F. Smeaton, G. J. F. Jones, M. Hughes, Investigating keyframe selection methods in the novel domain of passively captured visual lifelogs, in: Proceedings of the 2008 International Conference on Content-based Image and Video Retrieval CIVR, 2008, pp. 259–268.

[5] L. I. Kuncheva, P. Yousefi, J. Almeida, Comparing keyframe summaries of egocentric videos: Closest-to-centroid baseline, in: Proceedings of the 7th International Conference on Image Processing Theory, Tools and Applications (IPTA 2017), Montreal, Canada, 2017.
URL https://lucykuncheva.co.uk/papers/lkpyjaIPTA17.pdf