Simple Content-based Image Retrieve System:
Application for Searching a Screenshot in a Video

Author: NG Yiu Wai
Date: July 2019

Language: **Python**
Environment: **Python 3.6.6, with library OpenCV & NumPy installed**
Core Files: **4 Files**
{extractor.py , searcher.py, _modules/features.py, _modules/searchtree.py}

The aim of this application is to demonstrate basic elements in a Content-based Image Retrieve (CBIR) System. The application consists of two parts:

1) Features Extraction on Video Frames

extractor.py

- Convent video to image frames for further processing by OpenCV
- Dividend each image frame to blocks
- Extract features for each block
  - color-based method which extracts the average intensity of red, green and blue.
  - 3 features are extracted for each block
  - each frame has 3 * (# of block) features
- Reduce the dimension of features by discrete cosine transform
  - each frame has 16 features after reduction
  - features at top are more important
- Save features of all frames in JSON

2) Searching on Screenshot

searcher.py

- Read features of all frames from JSON
- Build octree using the top 3 features
- Save octree in JSON for visualization
- Read image, and extract features
- Search features in octree
- Calculation mean square error between target features and tree's node.
- Return result on screen

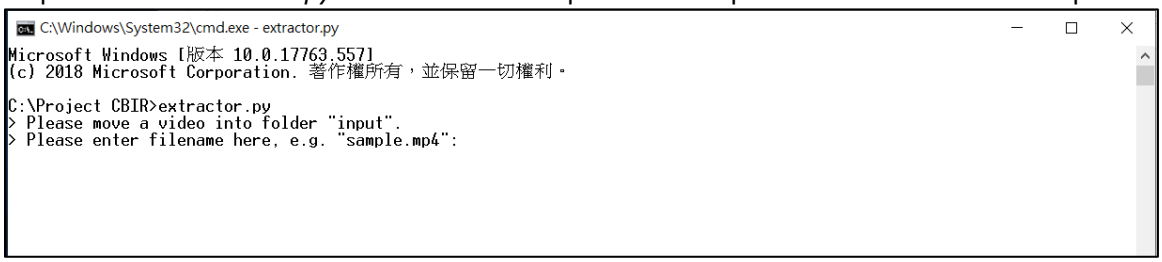
Note: Leaf nodes of search tree is (time, feature vector),
which is equivalent to (file path, file key) in traditional search tree.

This application is modularized which allows to replace the algorithms for

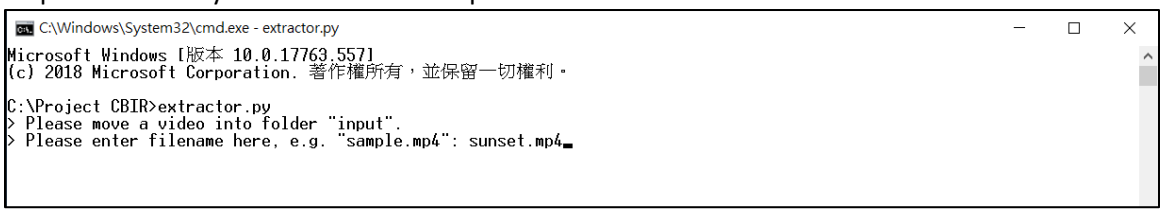
- feature extraction & dimension reduction (***feature.py***)
  [In demo, color-based method & discrete cosine transform are used.]
- search tee & distance function *(**searchtree.py**)*
  [In demo, octree tree & mean square error are used.]

Demo:

Step 1.1: Run *extractor.py* and two folders “input” and “output” will be created in same path.

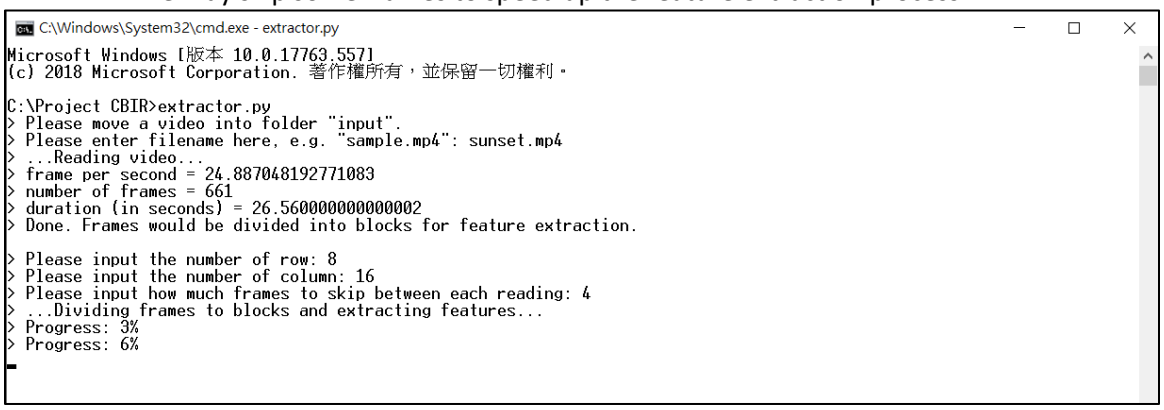


Step 1.2: Put any video into folder “input”. Enter file name of video here.

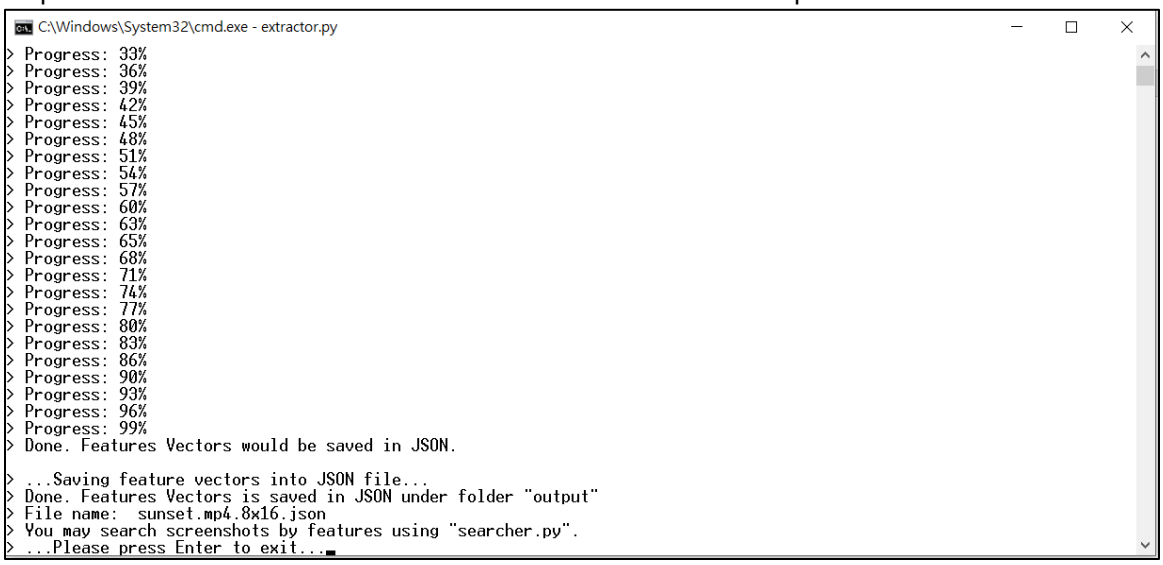


*In this demo, the videos are from **pexel** which allows free to use.*
License: https://www.pexels.com/photo-license/

Step 1.3: Video is transformed to frames.
[Important! The *script is not optimized which consumes a lot of computer resources.]*
Each frame will be divided into blocks for features extraction.
Enter the rows and columns to decide the number of blocks.
We may skip some frames to speed up the feature extraction process.



Step 1.4: Features are extracted and saved in JSON in folder “output”

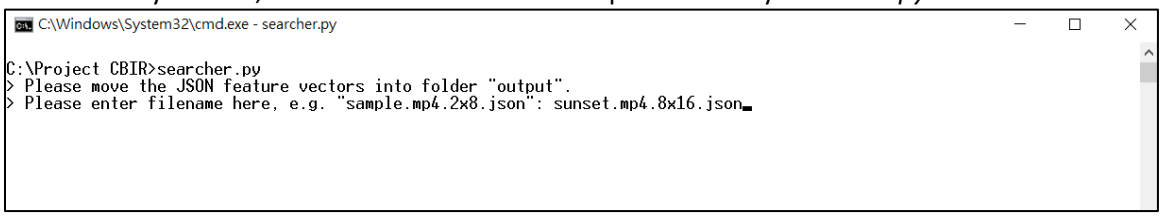


Example of JSON Feature vectors:

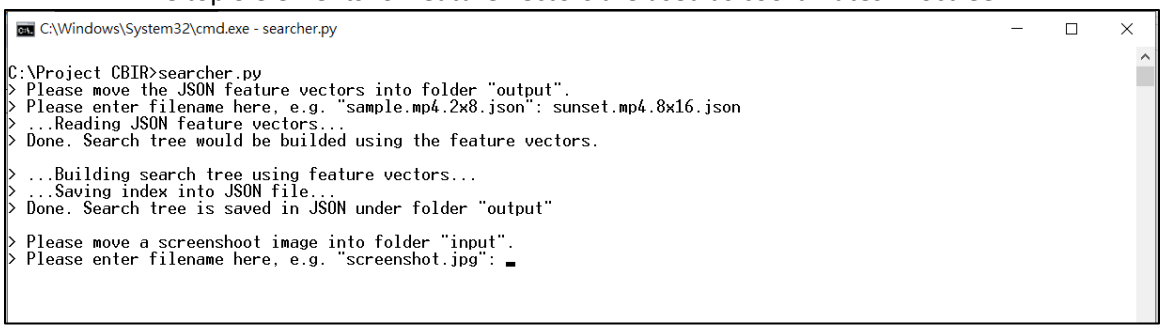

```
[
[0.0, [2301, -387, 81, -31, 73, 36, -40, -37, -30, -14, 16, -24, 1, -25, 17, -24]],
[0.20090771558245085, [2309, -393, 69, -20, 63, 52, -44, -36, -32, -17, 15, -20, 3, -17, 18, -19]],
[0.4018154311649017, [2307, -399, 51, -7, 51, 62, -46, -38, -35, -20, 12, -16, 3, -8, 17, -13]],
[0.6027231467473525, [2312, -401, 28, 11, 36, 72, -50, -38, -36, -22, 11, -9, 1, 0, 14, -10]],
[0.8036308623298034, [2315, -400, 2, 27, 19, 78, -55, -35, -37, -20, 7, 0, -1, 7, 9, -4]],
[1.0045385779122542, [2324, -395, -25, 42, 5, 77, -55, -33, -31, -20, 8, 1, 0, 10, 5, -1]],
[1.205446293494705, [2303, -391, -45, 47, 1, 70, -48, -39, -16, -24, 15, -3, 6, 1, 7, -6]],
[1.406354009077156, [2303, -385, -66, 54, 1, 62, -37, -46, -1, -28, 21, -11, 16, -6, 11, -12]],
[1.6072617246596068, [2315, -376, -83, 60, 5, 53, -28, -55, 5, -27, 21, -12, 20, -9, 13, -16]],
[1.8081694402420576, [2319, -361, -99, 61, 14, 42, -18, -68, 11, -27, 18, -8, 17, -2, 10, -14]],
[2.0090771558245084, [2320, -351, -114, 58, 23, 29, -8, -80, 11, -26, 12, -1, 14, 7, 7, -11]],
[2.2099848714069594, [2318, -339, -129, 55, 32, 20, -1, -86, 5, -19, 0, 11, 8, 14, 6, -11]],
[2.41089258698941, [2298, -324, -143, 54, 30, 19, -3, -84, 0, -15, -7, 13, 1, 18, 2, -4]],
[2.611800302571861, [2291, -307, -151, 50, 31, 18, -4, -78, -7, -6, -14, 16, -2, 18, 2, -8]],
[2.812708018154312, [2285, -297, -154, 38, 40, 16, -6, -71, -15, 3, -22, 15, 0, 10, 12, -16]],
[3.0136157337367626, [2291, -291, -156, 28, 47, 16, -6, -69, -15, 11, -21, 8, 10, 0, 19, -15]],
[3.2145234493192136, [2297, -280, -156, 18, 54, 14, -9, -66, -13, 14, -16, 3, 16, -3, 25, -17]],
[3.415431164901664, [2326, -270, -161, 7, 60, 15, -15, -62, -9, 14, -7, 0, 18, -1, 27, -16]],
[3.6163388804841152, [2339, -263, -161, -6, 66, 15, -19, -59, -4, 10, 2, -4, 21, 2, 25, -8]],
[3.8172465960665662, [2360, -256, -165, -14, 66, 16, -20, -60, 0, 9, 4, -2, 17, 7, 21, 2]],
[4.018154311649017, [2381, -252, -167, -17, 60, 23, -21, -59, 0, 11, 4, 4, 13, 12, 18, 11]],
[4.219062027231468, [2385, -249, -169, -23, 56, 27, -20, -59, 0, 13, 7, 6, 14, 13, 17, 14]],
[4.419969742813919, [2378, -240, -166, -28, 48, 28, -18, -56, -3, 20, 11, 5, 15, 15, 16, 15]]
]
```


- Feature vectors are in format *(time, 16-dimension reduced feature vector)*

Step 2.1: Run *searcher.py* and enter file name of JSON feature vectors
By default, the JSON file is saved in "output" folder by *extractor.py*.



Step 2.2: Octree is built using the feature vectors. The tree is saved in "output" folder.
The top 3 elements for feature vectors are used as coordinates in octree.

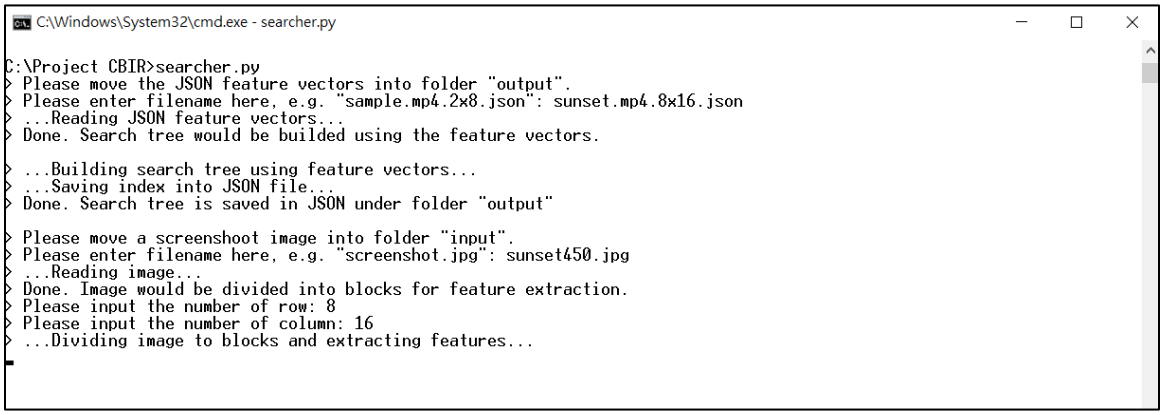


Example of JSON tree:

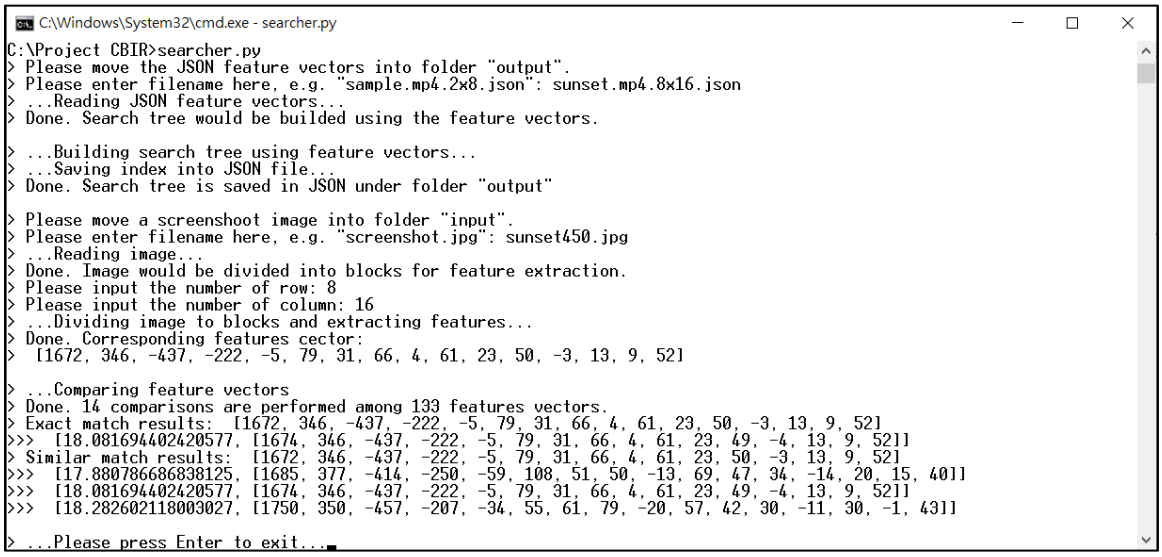

```
[
    [false, 97, -401, -472, 2772, 702, 81, 8],
    [
        [true, 97, -401, -472, 1434.5, 150.5, -195.5, 11]
    ],
    [
        [false,97, -401, -195.5, 1434.5, 150.5, 81, 8],
        [
            [true, 97, -401, -195.5, 765.75, -125.25, -57.25, 11]
        ],
        [
            [true, 97, -401, -57.25, 765.75, -125.25, 81, 11]
        ],
        [
            [true, 97, -125.25, -195.5, 765.75, 150.5, -57.25, 11],
            [23.7071104387292, [393, 77, -60, -34, -25, 11, 15, 23, 1, 11, 2, 14, 1, 9, 1, 13]],
            [23.50620272314675, [460, 93, -75, -49, -29, 14, 21, 26, 1, 10, 2, 17, 1, 12, 0, 16]],
```


- The example is a node with 3 levels.
- Header of a node is a list (with length = 8) representing *[*
  isLeaf: Bool,
  xLowerBound: int, yLowerBound: int, zLowerBound: int,
  xUpperBound: int, yUpperBound: int, zUpperBound: int,
  maxChild: int]
- If it is a root or intermediate node, the Boolean is *FALSE*. The maximum number of children equals to 8. All children are nodes.
- If it is a leaf node, the Boolean is *TRUE*. All children are feature vectors or empty. Given than it is an octree, some leaf nodes may have no children.
- Searching is performed using [x, y, z] coordination, or in other words, the top 3 features. It serves as the lower-bound of kNN search.
- The retrieved data is time, which means file path for a video.

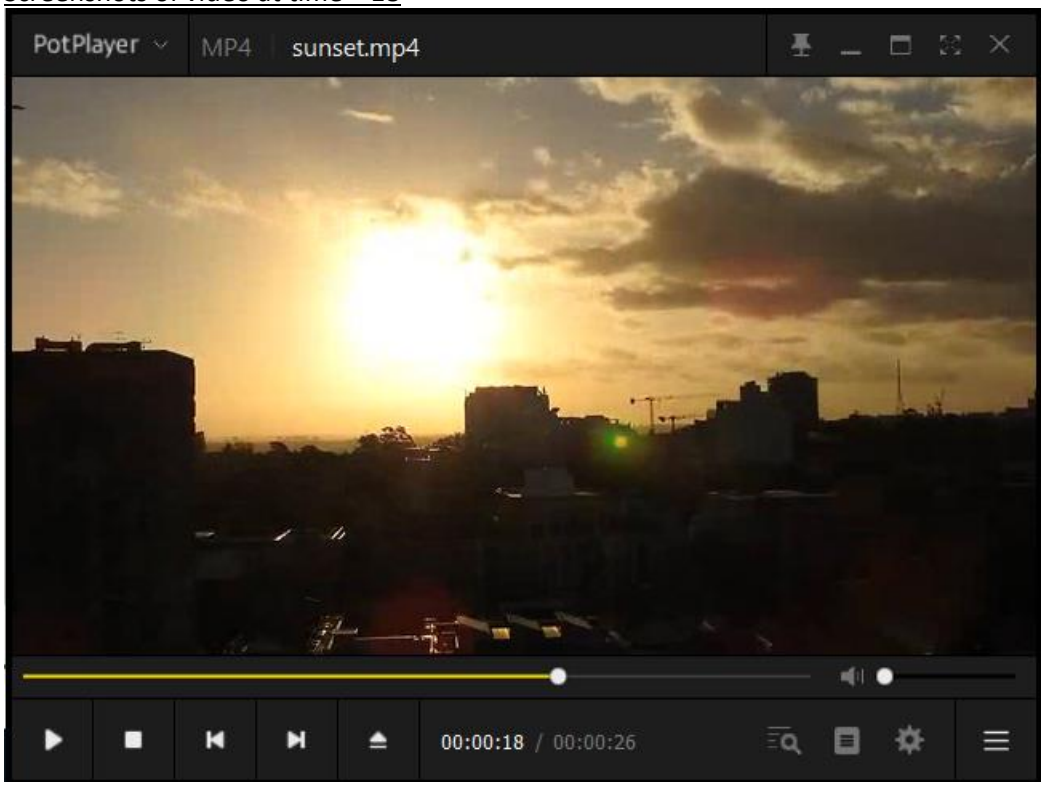
Step 2.3: Put a screenshot image into folder “input”.
Enter the file name to extract image’s features.
Also enter number of rows and columns for feature extraction.



Step 2.4: Searching result is shown on screen.
For below example, 14 comparisons are performed to find matched result.
The screenshot could be found at around *time = 18*.



Screenshots of video at time = 18



Screenshot input for searching

----------END----------