Kevin Zhang
(510) 364-8874
kevz@mit.edu

Project Dejavu

2nd July 2016 | https://dejavu.skyi.io

OVERVIEW

This project is my attempt to build a fast video fingerprinting system that is robust to recording artifacts such as glare/saturation, linear transformations, and partial obscuration. One possible application for this system is to allow users to record a short 5 - 10 second clip of some unknown target video and then identify the original content.

Existing research into video fingerprinting primarily focuses on identifying illegal content (DMCA, etc.) where it is safe to assume that the target content and original content are extremely similar by almost any measure. In addition, existing research often assumes the the target and original are of similar length and therefore takes advantage of key frame timings.

Project Dejavu makes neither assumption and is designed to handle extremely noisy content such as recordings of a low-resolution video played on a glossy screen under direct sunlight with a very shaky hand and other similar worst-case scenarios.

GOALS

1. Scalability.
    a. 40 kilobytes for each hour of content
    b. 30 seconds to vectorize an hour of content
    c. 0.2 seconds to identify a clip from an hour of content
    d. Linear time vectorization, logarithmic time recognition.
2. Robustness.
    a. Saturation, brightness, linear-transform invariant.
    b. Works indoors and outdoors, on glossy and matte screens.
    c. > 80% of the time, the top result should be correct
    d. > 90% of the time, one of the top two results should be correct
    e. > 95% of the time, one of the top three results should be correct

MILESTONES

Prototype (v0.0.1)

Evaluate a variety of video vectorization techniques (RGB histograms, HSV histograms, H medians, H ordinals, B ordinals, B binary patterns, etc.) and set up a server (https://dejavu.skyi.io/) to enable remote testing and demoing.

Initial Release (v0.1.0)

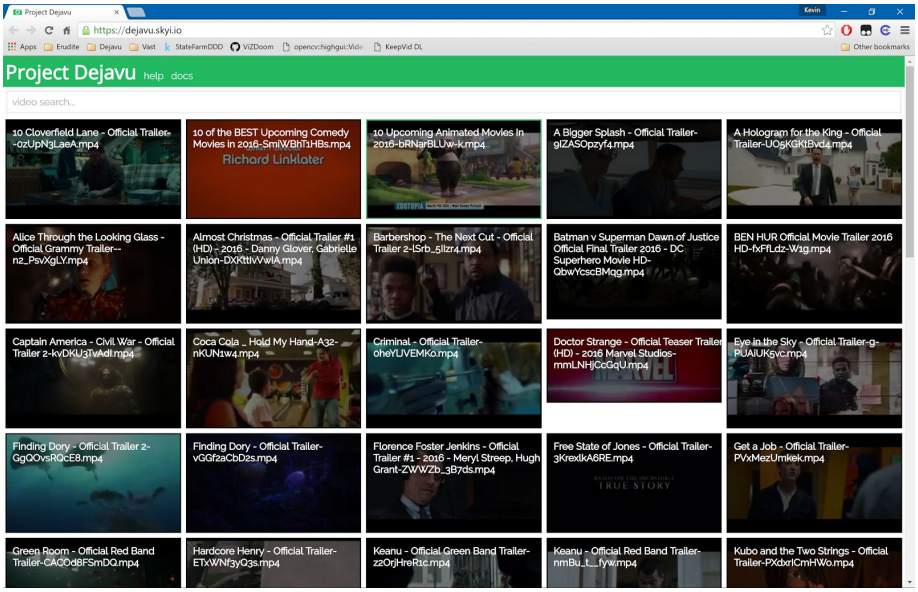

Fig. 1: Video management interface. Please excuse to UI, it's just a demo.

Fig. 2: Video capture mobile web app. You can see my watch in the glare.

DATASETS

The original content data - a set of 63 low-resolution MP4 files that add up to approximately 2.5 hours of content - was extracted from the YouTube playlists listed below. The entire original content can be downloaded using the Google Drive link:

https://drive.google.com/open?id=0B87DjfyJB2c_RXB6a19NUkI5akE
https://www.youtube.com/playlist?list=PLDVdFCWgvHe4mx96dLfVupzXLqXuYnhir
https://www.youtube.com/playlist?list=PLDVdFCWgvHe4jGbLczQINAClpZrHsWxn1
https://www.youtube.com/playlist?list=PLDVdFCWgvHe6mWXSqyKoWxL2NzGEwdo7y
https://www.youtube.com/playlist?list=PLDVdFCWgvHe5LVO6sK1N5MmepBS9saruN

This data set do not belong to me and therefore should not be used for anything other than feature/model evaluation purposes.

ALGORITHMS

Vectorization (B-ordinals, H-medians)

Brightness ordinals - start by splitting the frame using a 3x3 grid and computing the average brightness for each of the 9 cells. Assign a ordinal to each of the cells according to the relative brightness - if stored optimally, this should take less than 18.5 bits. This ordinal hash is surprisingly robust and can be used to greatly reduce the search space. Supposing a random distribution and perfect camera, this hash would reduce the search space 1 / 362,880 - after taking noisy capture (blurs, glare, etc.) into account, the actual reduction is closer to 1 / 1,000 but this is still surprisingly effective.

Hue medians - using the same 9 cells, compute the median hue for each component. The median is superior to the mean because there is a high likelihood that there will be outliers - experimentation shows that the median is more robust to recording artifacts. The hue value can be stuffed into 8 bins without significant loss in accuracy - this means it can be stored in 27 bits. When measuring the distance between hue values, take care to have the value wrap around - assuming 0 indexing, the distance between 7 and 0 is 1, the distance between 6 and 0 is 2, and so on due to the shape of the HSV color space.

46 bits of data per frame translates to roughly 20 kilobytes of storage for each hour of video sampled at 1 frame per second. Based on initial observations, this data is not easily compressible as there is very little redundancy - brightness and hue are orthonormal in the color space and the large time difference would make it difficult to take advantage of similarities between neighboring frames.

Recognition (Linear scan, ANN, FFT)

Currently, the B-ordinal is used to index into a subset of the original time series search space before the H-median distance is applied using a linear scan to fine tune the results. This operation scales linearly once the B-ordinal index space is saturated - although initially there are 362,880 possible values, noisy data needs to be taken into account and an index table is used to transform similar values into the same bin, resulting in roughly 1000 values, many of which will be unused since only a subset of these spaces are occupied by natural images.

One option for improving the time complexity is to use approximate nearest neighbor search, but that is difficult to implement due to the non-Euclidean hue dimension. The wrap-around nature of the hue value makes classic time series techniques such as the Fourier transform tricky to implement correctly - it appears that creating an extra copy of the data shifted by 4 (half of the maximum hue value) is an decent way of handling the wrap-around, but I remain sceptical.

DEMO

Check out https://dejavu.skyi.io to see the demo in action. Just open one of the videos on your computer / TV and scan it using the web app. The mobile app only runs in Chrome, but it does work on Android devices. It should successfully recognize whatever video is playing in less than a second - if it takes longer, that's probably because my extraordinarily cheap 512mb RAM server is busy processing other requests.

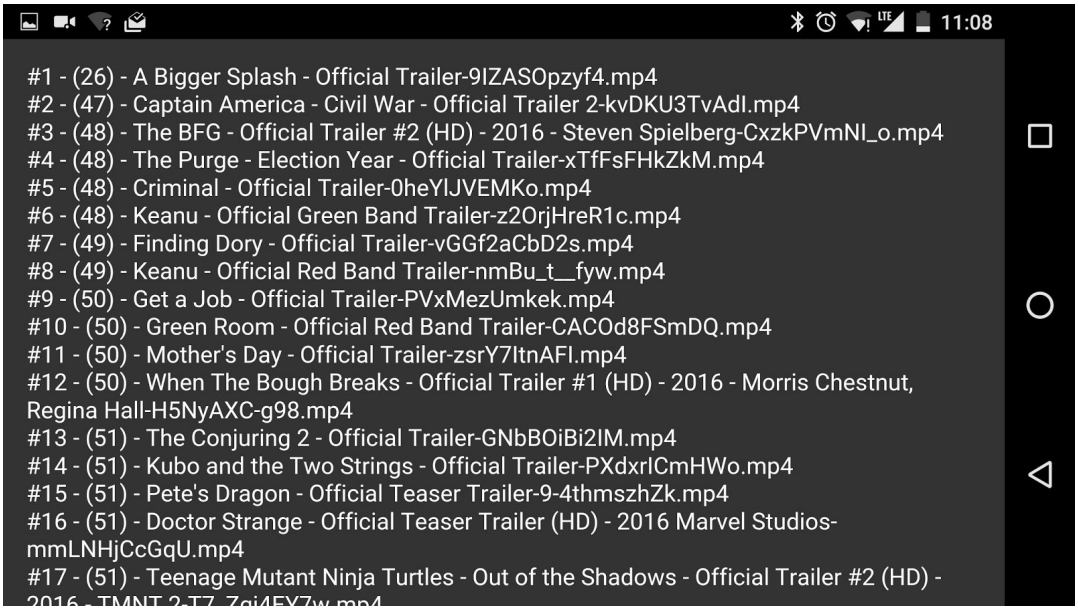

Fig. 3: Video capture app results. The first result (diff score 26) is correct.