Vooble: A Visual Search Engine

Carlo C. del Mundo
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98125
cdel@cs.washington.edu

Abstract

We detail the design and implementation of Vooble, a content-based video search engine that takes as input a short video clip and outputs other videos of similar visual content. We leveraged Dense Trajectories for feature extraction, quantize the features using a bag of visual words model, and applied the k-nearest neighbor algorithm using Euclidean distance as a metric for similarity. We scraped approximately 20,000 videos from Vine, a mobile application that allows users to upload short 6-second videos. Results indicate that feature extraction dominates execution time comprising 94.1% of overall execution time for the full end-to-end pipeline (when including offline and online portions). Furthermore, storage of features is identified as a core problem as storing the features consumes three orders of magnitude more storage space than the actual video (.mp4) itself. Sample query results of the end-to-end system can be viewed at [http://homes.cs.washington.edu/~cdel/vooble/landing.php](http://homes.cs.washington.edu/~cdel/vooble/landing.php) (use Safari v8.0.6+).

1 Introduction & Related Work

To index and search all of the videos in the Internet requires massive resources. Using commodity parts and existing libraries, a conservative estimate for a system requires 32 petabytes to store feature data, 480,000 CPU cores to build and update the index, and 1000 servers to service 1 million concurrent clients at a latency on the order of seconds. Such requirements in both power consumption and performance rival those of the top supercomputers [1].

Notable content-based systems include Google Image Search [2], VideoQ [3], and VisualSEEk [4]. Image-based search systems has been widely studied with neural-network based feature extraction methods performing best in terms of retrieval accuracy. On the other hand, video-based search systems are poorly understood in terms of both retrieval and system efficiency.

The following project work was completed during CSE 576 and is summarized as follows.

1. **System.** We designed a full end-to-end content-based video search system using Vine videos. The user inputs a 6-second video to the system. The system outputs the top-16 most similar results to the query.

2. **Dataset.** We scraped 10,000 Vine videos using the Twitter Streaming API.

Sample query results can be found in [http://homes.cs.washington.edu/~cdel/vooble/landing.php](http://homes.cs.washington.edu/~cdel/vooble/landing.php) Use Safari v8.0.6+ as videos do not render on Chrome.

2 Framework (Background)

Figure[1] demonstrates our end-to-end system pipeline. The user inputs a video to the system, and the system performs the similarity search calculations to find other videos of similar visual content. The
dataset consists of approximately 20,000 6-second Vine videos scraped under the Twitter Streaming API. Many Vine videos contain abrupt transitions which affect the efficacy of the search algorithm. Therefore, we leveraged an existing shot detection algorithm to segment each video into continuous scenes [5]. Our resulting dataset consisted of 80,000 scenes of varying duration (e.g., from 0.5 to 6 seconds).

2.1 Offline Pipeline

**Feature Extraction (Figure 1a).** We leverage Dense Trajectories [6] to extract features for each scene. Dense Trajectories extracts shape, appearance, and motion to generate a descriptor for each detected interest point. The number of feature vectors vary for each input scene and is dependent on (1) the duration the interest point is tracked (i.e., 15 frames), (2) the length of the video, and (3) the number of interest points. The descriptor consists of HOG, HOF, MBH, and relative point coordinates concatenated and normalized into a 426-dimensional feature vector representation.

**Bag of Words (Figure 1b).** We apply a Bag of Words (BoW) k-means clustering model to quantize each video into a histogram-based representation. BOW sidesteps the issue of different video frame lengths as it quantizes the number of resulting dimensions to the user-selected parameter, k, or the number of clusters. We chose k = 100 and trained the codebook with 877,936 426-dimensional feature vectors. These feature vectors were selected via random sampling; we chose 16 feature vectors per scene in a corpus of over 80,000 scenes.

Once the codebook is trained and generated, we finish the offline pipeline by generating histogram representations for each scene in the database. Each scene generates a single histogram representation (shown as a green box) and are used as the candidate vectors for matching in the k-nearest neighbor step.

2.2 Online Pipeline

Once the user specifies an input video to the system, feature vectors are extracted using Dense Trajectories and is quantized using the aforementioned BoW model. The query histogram (shown in purple) is then passed through the k-nearest neighbor (KNN) algorithm (Figure 1c). The KNN algorithm performs the similarity search by finding the closest neighbor of a query vector in a 100-dimensional space. KNN performs pairwise Euclidean distances and the top-k candidate neighbors that minimize the Euclidean distance is returned. Finally, for each candidate histogram vector, a reverse lookup is performed to return the video filename. The system returns the URLs of similar videos in the database for the input query video.
3 Results & Discussion

3.1 Performance Characterization

Figure 2 demonstrates a basic performance characterizations of each algorithm used to implement content-based video searching. Each measurement was normalized to running on a 48-core machine with 512 GB of RAM. Feature Extraction (Figure 1a) took the longest averaging approximately 5 minutes per 6-second video (or 12 days for 80,000 scenes). Codebook Training (Figure 1b) using 100 clusters for 877,396 426-dimensional feature vectors took approximately 0.5 days. Transforming the feature vectors into histograms in Histogram Extraction took approximately 0.25 days, and finally, the kNN search took less than 10 minutes for a corpus of 80,000 scenes. Table 1 shows the breakdown of each software algorithm in terms of CPU or I/O time. Most of the execution time is from Feature Extraction and generally is dominated by CPU operations. Table 2 shows the disk usage for video files (.mp4s) and video features. There exists a gap of at least three orders-of-magnitude of a difference to store the feature vectors versus the underlying video media!

3.2 Sample Retrieval Results

Figure 3 shows a sample query using the Voogle search system (see: http://homes.cs.washington.edu/~cdel/voogle/search.php?fname=587689178940465154-0001.mp4 to view the videos). The query input is of a woman in the foreground and a dancing man in the background. The system is functionally correct because the first most similar result found was a duplicate of the original query video. Results #2 and #3 show a panda with a similar dancing motion to the man in the query video.
Table 2: File Size Statistics of Prototype

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Size of Video Files (.mp4)</td>
<td>16 GB</td>
</tr>
<tr>
<td>Disk Size of Features (.bin)</td>
<td>4.1 TB</td>
</tr>
<tr>
<td>Average .mp4 Size</td>
<td>185 KB</td>
</tr>
<tr>
<td>Average Feature File Size</td>
<td>90 MB</td>
</tr>
</tbody>
</table>

4 Conclusion

We presented a prototype content-based video search system that can retrieve duplicates and similar content using the Dense Trajectories feature extractor. To improve upon this work, we propose the following next action steps.

**Improving the Codebook Training in the Bag of Words Model.** We briefly considered varying the codebook size (e.g., number of clusters), but due to computational limitations, we decided to limit the codebook size to 100 clusters because it was fast to compute and because it is low enough dimensionality that the nearest neighbor search can be executed within minutes (instead of hours for 1000s of dimensions). Increasing the codebook size could potentially decrease the average distortion (the sum squared error for each data point to its nearest cluster). Minimizing the distortion can theoretically improve the search retrieval of the system.

**Using a larger video dataset than the initial 20,000 videos and using a labeled dataset.** Initially, we used Vine videos because of its relatively short durations (thus shorter feature extraction times). In the end, it doesn’t matter if the videos are 6-seconds or 60-minutes in length — segmenting these videos into much smaller continuous scenes can be performed at any arbitrary duration.

Vine videos, unfortunately, introduce significant noise. Many clips contain abrupt transitions, and because the Vine videos are short to begin with, the interest points are not tracked for a significant amount of time causing a degradation in retrieval efficiency.

Ideally, the dataset would also be labeled with ground-truth knowledge on what constitutes exact content-based video matching. Unfortunately, it’s difficult and expensive to acquire a human-annotated labeled video dataset, and even if they do exist, we suspect the sample size of these videos are fairly small.

**Testing the Quantization Idea.** Our original project proposal focused on binary quantization, but due to unforeseen hiccups in the system, we were not able to implement this idea. The next action steps would binarize the histogram vectors and create a precision/call curve to determine if Hamming distance calculations is a good approximate metric for the Euclidean distance. We plan on testing binary quantization as part of our research project.

References