EVA: An Efficient System for Exploratory Video Analysis

Ziqiang Feng
Carnegie Mellon University
zf@cs.cmu.edu

Junjue Wang
Carnegie Mellon University
junjuew@cs.cmu.edu

Jan Harkes
Carnegie Mellon University
jaharkes@cs.cmu.edu

Padmanabhan Pillai
Intel Labs
padmanabhan.s.pillai@intel.com

Mahadev Satyanarayanan
Carnegie Mellon University
satya@cs.cmu.edu

1 INTRODUCTION

An ever-increasing number of high-resolution, continuously-on camera systems are being deployed around the world today. Videos captured by nearly-ubiquitous cameras can provide great value for retrospective video analysis tasks such as crime investigations and scientific research. These tasks are often both *interactive* and *exploratory* in nature, meaning that the best query to find the desired results is unknown in the beginning, but is itself discovered and refined in an iterative process of exploring the data. For example, suppose a police investigator is searching for a suspect in a large collection of recent surveillance footage. Initially, she has very little information, so starts with just a HOG-based face recognition query. As positive results roll in, she finds a particular car to be associated with the incident. Using the found footage, she trains a Faster R-CNN [5] car instance detector and reruns the search with both face recognition and car detection. The new query may find additional scenes of interest that were previously missed, e.g., due to obstructed or profile views of the face. Furthermore, the car instance detector may be retrained as more footage of the car is collected, improving its accuracy for the next iteration of search.

In this paper, we present EVA: an efficient system for Exploratory Video Analysis. EVA aims to provide a *usable* and *efficient* end-to-end system for domain experts with little machine learning training. EVA lets the user rapidly search not only the *data space*, but also the *query space*. The latter involves iteratively trying queries to find better parameters and features to use. Such features may be explicit ones, like color histograms, HOG or SIFT features, or can be implicit ones defined by a neural network. EVA provides a GUI for specifying queries through examples, setting parameters, and for refining pre-trained DNN models based on collected results.

EVA is intended for interactive use, and must return results quickly, saving the domain expert's precious time and attention. However, it is generally infeasible to build an index of the video data to handle queries in EVA. One cannot anticipate all potential queries and index all possible features, or continually rebuild indices in the face of ever-changing and rapidly emerging analysis algorithms. The video cameras are always on, collecting new data that would not have had time to be indexed. As a result, EVA will have to execute video analysis quickly and on demand.

In a sense, EVA is a rapid-prototyping system for iteratively improving and running effective video queries. It complements other emerging video monitoring systems such as NoScope [4] and VideoStorm [8], which try to optimize one or a set of long-lived queries. These assume the right queries are known for a given task. EVA serves as a means of finding the right queries to run.

2 CHARACTERISTICS OF EVA QUERIES

EVA queries have the following unique properties, that greatly influence the architecture and operation of our system:

(1) EVA queries apply over extended durations of captured video, so our system is optimized for scanning through a large amount of video data. In contrast, prediction serving systems like TensorFlow Serving [2] and Clipper [1] are optimized for evaluating a prediction model on a single data point.
(2) It is common to abort and restart query execution in EVA, as the user interactively refines the query in response to partial results, unlike traditional big data systems like Spark [7] where queries typically run to completion. This characteristic motivates our choice of fine-grained caching over traditional per-query result caching (Section 4).
(3) EVA queries may need to use newly-developed vision algorithms. Our flexible system allows use of new analytics built on arbitrary frameworks, languages, and libraries. Efficient support for and convenient use of them motivate our filter abstraction and a narrow waist protocol (Section 3).
(4) Although the query space is unbounded, EVA queries will likely exhibit temporal similarity. Due to iterative exploration and refinement of queries, the system will see sequences of semantically-correlated queries, using similar features, parameters, or algorithms. We exploit this characteristic judiciously to improve EVA's performance.

3 SYSTEM ARCHITECTURE

EVA has two main components: an *atomizer* that splits video into items and a *query processing engine* that executes filters on items.

3.1 Atomizer

The atomizer scans through long video streams and emits small data units called *items*, which are then processed in the query execution engine. The granularity of an item depends on the query, e.g., individual frames for object detection, or short overlapping video clips for activity inference. Queries can also set the *scope* of the atomizer, selecting which video data is used, e.g., by time, location, etc. Unlike stream analytics, where each item is processed once, iterative refinement of EVA queries may involve revisiting items, so the atomizer aggressively caches generated items.

3.2 Query Processing Engine

The query processing engine evaluates the query independently on each item from the atomizer, exploiting the available data parallelism. A query consists of an acyclic graph of *filters*. Each filter,

Query	Filter 1	Filter 2 (varied)	Intention
Q1	SIFT extraction	SIFT matching	Object detection
Q2	MobileNet	SVM	Image classification

Table 1: Experiment query templates

which can be an arbitrary, user-supplied program, implements an image processing or analytics primitive. A filter communicates to the system and other filters through a narrow waist protocol, based on setting key-value pairs called *attributes* that can be attached to items.

All data and results are communicated using `get-attribute` and `set-attribute` calls. For example, a JPEG decoding filter may get the raw item data using the special null key. It then sets an attribute containing the decoded RGB array using the key 'rgb'. A computer vision filter can then get the 'rgb' attribute, execute a face detector, and create an attribute containing the list of bounding boxes of faces with the key 'faces'. This mechanism is general enough for inter-filter communications, is flexible, and can be readily adapted to work in diverse languages, libraries, and frameworks used to implement query filters.

Finally, a `set-score` call allows a filter to specify a floating point "score" to an item. This score is then thresholded by the system to decide which items to ultimately present to the user.

4 OPTIMIZATIONS

4.1 Filter Container

EVA encapsulates all filters in Docker containers, efficiently supporting arbitrary run-time environments and libraries, while providing execution isolation. Filter containers are given access to multi-core CPUs as well as specialized hardware (e.g., GPUs). To reduce overheads, EVA reuses running containers whenever possible, e.g., when the same filter is used in multiple queries. To facilitate filter development, we implement Docker base images of different OSes with the logic needed to interface with EVA. They include a TCP server that waits for connections from the query processing engine and launches a filter process in the container. They also support the narrow, high-level attribute API outlined above.

4.2 Fine-grained Caching

EVA uses a per-item, per-filter approach to caching, as opposed to coarse-grained, per-query result caching. This allows partial results to be reused in subsequent queries using some of the same filters. When a filter is evaluated against an item, EVA records the hash digests of all attributes accessed or written by the filter. These hash digests, along with the score, are kept in a Redis database, indexed by the tuple of filter id and item id. EVA also stores a mapping from the hash digest of an output attribute to its actual value in Redis.

For future queries, as EVA traverses the filter graph, it retrieves database entries indexed by filter id and item id. It then tries to validate those entries. A cache entry is deemed valid if and only if all hash digests of its input attributes match the hash digests of the output attributes of another validated entry, or a newly-executed filter. If a valid entry is found, cached results are used; otherwise, the filter is re-executed. This approach avoids duplicated execution, ensures correctness of cached results, and minimizes recomputations of hash digests. It is a form of just-in-time indexing as described in [6] that helps reduce computation when subsequent queries are modified, but still share some filters, a very common occurrence in the iterative EVA usage model.

Query	No cache	Refined query	Identical query
Q1	23	69	1535
Q2	172	793	1924

Table 2: End-to-end throughput (frames per second).

4.3 Adaptive Batching

We are currently extending EVA to support adaptive batching of items in order to exploit the efficient batch processing implementations in many deep learning frameworks such as TensorFlow. We omit the details here due to space limit.

5 PRELIMINARY EXPERIMENTS

We perform some initial experiments to demonstrate that (1) EVA supports a variety of video analysis queries; and (2) our approach to caching can improve end-to-end search speed. Using a small dataset containing 10 videos, we execute queries based on the templates in Table 1. Each query consists of two main filters. Q1 detects object of interest by first running a filter to extract SIFT key points from the frames, and then running a second filter to compare the SIFT features with those of the desired object. Frames with sufficiently close matches are presented as query results. Q2 classifies each frame by first applying a MobileNet-v1[3] DNN filter to extract a 1024-dimensional feature vector, and then feeding this vector to an SVM filter to classify the frame. To test effects of caching, we repeat the queries, keeping the first filter fixed, and modifying the parameters of the second filter, akin to refining the query.

Table 2 reports EVA's end-to-end throughput (processed frames per second) for both query templates. The "No cache" column reports throughput for the initial execution of a query, when no cached results are available. "Refined query" corresponds to a refined query with a modified filter 2, where the cached attributes of filter 1 can be reused. "Identical query" corresponds to rerunning the exact same query, where whole-query result caching is achieved by reusing results for all filters.

When running a refined query, with partial results reuse, both query templates see substantial improvements of 3X (Q1) and nearly 5X (Q2). Q2 shows such great improvement due to the very high cost of extracting MobileNet features, relative to just running SVM on cached values. When cached results are reused for both filters in the queries, both query types see more than an order of magnitude increase in throughput.

6 CONCLUSIONS

We have introduced the concept of exploratory video analytics, and explained how it varies from traditional analytics. Based on the unique characteristics of this style of data search, we have designed and built an efficient system to allow users to iteratively explore a video dataset with incrementally refined queries. We have optimized our system to the EVA usage model, and have demonstrated how our approach to fine-grained caching can help accelerate a sequence of iteratively refined queries.

REFERENCES

[1] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System.. In *NSDI*. 613–627.

[2] Google. 2017. TensorFlow Serving. https://www.tensorflow.org/serving/. (2017).

[3] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[4] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1586–1597.

[5] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems (NIPS)*.

[6] Mahadev Satyanarayanan, Phillip B Gibbons, Lily Mummert, Padmanabhan Pillai, Pieter Simoens, and Rahul Sukthankar. 2017. Cloudlet-based Just-in-Time indexing of IoT video. In *Global IoT summit*. 1–8.

[7] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.

[8] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance.. In *NSDI*. 377–392.