Dexter: A Fast Search Engine for Video

Aditya Kuroodi, Joseph Ganley, Jiale Hu
Department of Computer Science, University of California, Los Angeles.
{akuroodi, jganley, jialehu}@ucla.edu

Abstract

Video camera deployments continue to grow globally over recent years, with the United States leading the surge with over 70 million installed CCTV cameras as of 2019 [7]. The advent of large-scale camera deployments has led to an influx of data that many video analytics frameworks employ to solve queries such as bounding box detection and object classification. From an application perspective however, the scope of executable queries on many of these systems is limited—often reducing to disjoint object counting or detection tasks (i.e. "How many cars are in my video?").

As the amount of video data increases, users will benefit from more complex queries to take data-driven actions, such as those that can relate object classes together and offer some means to parse temporally-spatially correlated events through a simplified API. To address this need to search over the video space we introduce **Dexter**, a video indexing framework built atop Elasticsearch [3] and object detection framework that provides users with a web interface to run complex queries over any input video. Possible applications include traffic / security surveillance, smart city deployments, and aiding law enforcement.

1 Introduction

Over the past decade video analytics systems have taken off with support for both retrospective and live video feeds. The initial focus of many of these systems revolved around detection accuracy and novel architectures to boost efficiency. Many systems in the space therefore target model compression, split server-edge architectures, and drift detection – all with the goal of enabling more robust video analytics ([4], [8], [5]).

Building atop this established platform, *our work uses the output of object detection frameworks to capture relationships between objects both in time and space*. With Dexter, users can easily and quickly pick out key segments from a larger video based on their query, not unlike a search engine specifically designed for video media. Dexter is model-agnostic, and uses any annotated video file as input. We also offer a default pipeline using DeepSort powered by YOLOv4 trained on the COCO dataset for users who want to provide raw video as input. All of our work is open source and available to those who may benefit from it. [1]

In the following sections we will discuss our system in more detail, starting first with background and motivation (§ 2), then cover system architecture (§ 3), evaluation (§ 4), and finally limitations and future directions (§ 5) before concluding (§ 6).

2 Background and Motivation

A typical traffic camera generates at least 12 hours of usable footage per day. Accumulate this feed for a month, and even a single query would take upwards of 5 hours on a naïve system [4]. With over 500PB of traffic surveillance footage generated per month in U.S. cities [2], it is imperative for systems to provide end users with quick access to only the video data relevant to their application.

We aim to address this problem by developing a video search index with a web interface to perform SQL-like queries over the target video. We extend disjoint object queries (like "Is there a stop sign present?") to capture relations between objects to answer complex queries such as: "Show me scenes where there are 3 cars AND pedestrians near those cars". After a one-time profiling cost to annotate a target video, we show that Dexter can quickly serve multiple, complex queries to the user with sub-second latency while ensuring accurate results via manual checks. Though Dexter was conceived with video surveillance applications in mind (i.e. identifying scenes with people jaywalking, or cars running red lights), its capabilities are bounded only by the specificity of the underlying object detector.

[1] https://github.com/akuroodi/Object-Detection

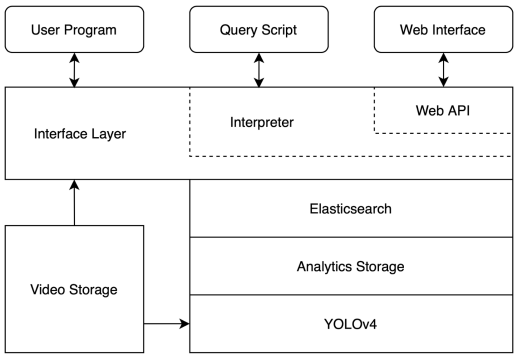

Figure 1: System architecture of Dexter consisting of an object detector (YOLOv4), Elasticsearch, and web interface layer to allow users to run queries on a given video.

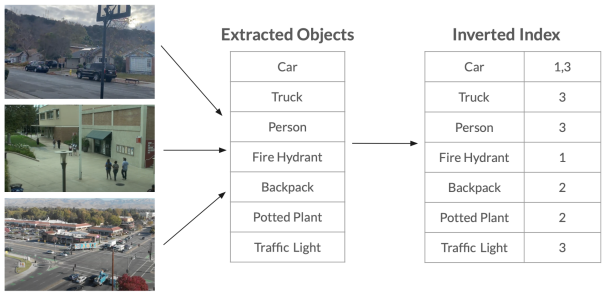

Figure 2: Example inverted index built from still frames from various target videos. The indexing enables low-latency queries even over large video files.

3 System Architecture

The architecture of Dexter consists of a SOTA object detector (§ 3.1), analytics storage, Elasticsearch (§ 3.2), interface layer and web interface (§ 3.3). External video storage provides video to be analyzed and queried by the system (Figure 1). The video analysis stage runs offline while Elasticsearch indexes the analysis to support low latency queries.

3.1 Offline Object Detection

Underpinning Dexter at the lowest level is the YOLOv4 object detector that works in concert with the DeepSORT object tracker. The YOLO family of object detectors are SOTA, and known for their high accuracy while requiring only a single pass over a set of images [1]. DeepSORT employs a Kalman Filter to efficiently and accurately track bounding boxes after they've been marked by YOLO [9]. Our default pipeline runs DeepSORT with YOLO to label all frames for a given target video. For each detected object we record coordinates for the bounding box along with the detector's confidence level in identifying the proper object class. We consider all frames for optimal accuracy, but discuss limitations and future improvements of this approach in § 5.1. The detector outputs are piped into plain text files and fed into the Elasticsearch engine after pre-processing.

3.2 Elasticsearch Query System

Elasticsearch is an open-source search engine software system which can index over textual, numerical, geospatial, and unstructured data. There are three main stages in the Elasticsearch query system. The first stage is an offline indexing system. In this stage, we take annotation data for each frame and use it to build data structures which are easily scalable and easily searchable. Specifically, the indexing stage builds an inverted index, which maps object class names to frame numbers. This mapping allows us to easily access frames which could be relevant and avoid looking at frames which are certain to be irrelevant. A diagram showing how the inverted index is built is shown below in Figure 2.

The next two stages are the search stage and the ranking stage. The search stage is fairly straightforward. In this stage we leverage the inverted index structure to extract only the frames which contain the objects we are looking for. For example, if the query is searching for cars, we can return the frames which are referenced by the key for car in the inverted index. In the case of a multiple object query, we simply perform a JOIN over the two sets of frames. The last stage is the ranking stage. This stage is the most delicate of all the stages. In the ranking stage we have to rank the frames returned by the search stage in the order of most relevant to least relevant. To do this, we have to use various signals provided by the frame data such as object detection confidence scores, object bounding boxes, object counts, etc. Our solution is to use the object confidence scores for most queries. For example, when ranking two frames, both of which contain a car, the frame in which the object detector was most confident is ranked higher. This is a fairly naive solution and has a lot of room for improvement which is discussed in § 5.2.

3.3 Web User Interface

The interface layer of Dexter exposes SQL-like queries to users while hiding the complexity of configuring and running Elasticsearch. A web user interface further reduces the effort involved in querying the video by providing query selector options (Figure 3) and video selection previews. Dexter's web interface allows users to dispatch with installing libraries, writing SQL script, and manipulating video.

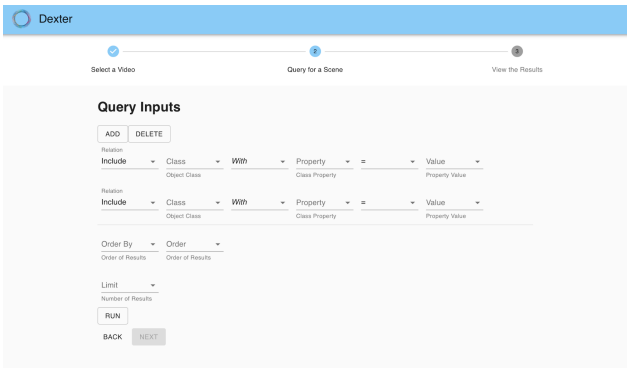

Figure 3: A screenshot showing Dexter's web interface, which is designed with simplicity in mind and allows users to query video even without a programming background.

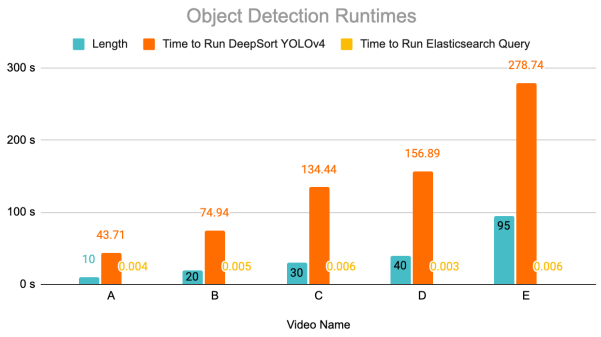

Figure 4: Overhead comparison of the different components of Dexter, showing that the bulk of runtime comes from the object detector and does not scale well as video length increases. Query time–what the user sees–however is negligible in comparison and we consistently achieve sub-second latency regardless of video size.

$$DCG@k = \sum_{i=1}^{k} \frac{s(i)}{\log_2(i+1)}$$

Figure 5: DCG equation

4 Evaluation

There are two primary areas in which we want to evaluate the Dexter system. The first area in which we want to measure performance is the object detection pipeline. In particular, we want to ensure that the object detection model provides accurate object labels and bounding boxes for each object in a given frame. Additionally, even though the object detection model is run offline, we still want to ensure that this pipeline can maintain a reasonable throughput. For this reason, we choose to run the YOLOv4 model, which achieves state-of-the-art average precision of 43.5 percent while maintaining a throughput of more than 60 frames per second on a Tesla V100 GPU [1]. While we did not have access to a Tesla V100 GPU, we did test the YOLOv4 model on Tesla K80 GPU and achieved reasonable throughput on our various test video datasets, as shown in Figure 4 below. Nevertheless, this stage of the application could prove to be a major bottleneck, so we hope to optimize this pipeline in the future. We discuss this further in § 5.1.

We also want to evaluate the efficiency and accuracy of the Elasticsearch query system. In particular, we want to ensure two things: the overall latency of the queries stayed very low even as the size of the dataset increased, and the accuracy of the ranking function was high. To evaluate the latency, we ran a simple query over 5 different video datasets ranging in length from 10 seconds to 95 seconds. The queries simply checked for frames which contained a particular object, such as a car or a person. As shown in the figure above, the latencies over each of the video datasets was incredibly small (in the range of 3 to 6 milliseconds) and also stayed fairly constant despite the growing dataset size. This result is very encouraging, although we still need to test the system on larger datasets over a distributed cluster environment.

To test the accuracy of the query system, we employed a metric commonly used in the search engine field. The metric, discounted cumulative gain (DCG), measures the accuracy of ranking systems by penalizing a result when highly relevant documents are ranked too low. The formula for DCG is shown above. In this formula, k represents the length of the ranking and s(i) represents the relevance score for document i. We measured the DCG for the results of three different queries: a multiple object query, an object count query, and a distance ranking query. Table 1 below shows the results. The top result is the average DCG for all three queries. The middle result is the ideal discounted cumulative gain (IDCG) which is the score of a perfect ranking. The bottom result is the normalized discounted cumulative gain (nDCG), which is simply the computed DCG divided by the IDCG. While this result is encouraging, it offers plenty of room for optimization, which we discuss in § 5.2.

5 Limitations and Future Directions

5.1 Object Detector

One key limitation of Dexter is that its application scope is *limited by the scope of the object detector*. YOLO for example is pre-trained with the COCO dataset that consists of 80 common classes of objects. Thus, Dexter can only index these 80 types of objects and any request out of scope is impossible. Broadening the range of an object detector is no trivial task, but a potential direction in increasing scope could be to cascade multiple object detectors that are trained on separate, more niche subsets. A thorough investigation

***DCG@10* = 15.498**
***IDCG@10* = 29.966**
***nDCG@10* = 0.517**

Table 1: From top to bottom, the table displays raw discounted cumulative gain (DCG), ideal DCG, and normalized DCG of Dexter's Elasticsearch backend.

would be required to see if Dexter performs better with several smaller, specific models or with a single generic model like YOLO. Additional considerations for training time of separate, lightweight models compared to one general-purpose model must also be considered.

Detectors also suffer from enormously high computation costs, indicated by the fact that over 90 percent of Dexter's runtime comes from the object detector. We show in § 3 that YOLO runtimes are 4x the length of the video file, elucidating that running any object detector (all CNN-based) frame-by-frame is not scalable. To address this scalability challenge, we propose selectively running the detector on a subset of frames in the video. One approach would be to run YOLO on just a single frame per second, based on the observation that video contents do not change rapidly on a sub-second interval. Though this naive approach would lead to a near 30x reduction in object detector runtime, we would need to ensure that system accuracy remains competitive despite frame skipping.

5.2 Search Optimization

The area with the most room for optimization is the ranking stage of the Elasticsearch query system. This stage tries to determine which of the relevant frames are the most relevant to the original query. In many ways, this is a subjective determination which makes it difficult to easily determine an optimal solution. It is possible, however, to view this problem as a standard machine learning problem. Each frame in the set of relevant frames has various signals: confidence scores, bounding boxes, object counts, etc. and each of these frames must be mapped to a particular relevance score. When viewed as a machine learning problem, it is possible to envision a regression model or even a deep learning model which can optimize the ranking function to produce high DCG scores.

5.3 Query Extension

The query condition for an object can be further extended to include properties like the dominating color of the object, and the OrderBy operation can be extended to allow a richer set of ranking parameters like confidence level of a specific criteria. In addition, future work also includes supporting GroupBy operations to enable more effective queries that involve large numbers of results that share some similarity. One approach could be to adopt BlazeIt's FrameQL language, that behaves similarly to SQL and is optimized for limit and aggregation queries [6]. However, this would inhibit everyday users from being able to interact with the videos which is not our intent. Despite this interface difference, many optimizations and lessons from BlazeIt can be folded into Dexter in future work.

6 Conclusion

Facing the tremendous amount of video being continuously generated, Dexter provides an easy-to-use query interface that allows users to quickly retrieve scenes of interest even with complex object relations. Dexter separates stage of object detection and stage of indexing and query to achieve low latency in query time, leverages the power of Elasticsearch to enable complex query, thus adapting the system to wider use cases. Dexter's utility is largely orthogonal to other advancements in the video analytics space such as increasing model accuracy, efficiently processing video, and widening the scope of detected object classes. With promising results from preliminary experiments, we hope Dexter can greatly increase the value that can be extracted from all kinds of video feeds, and provide insights in further improving video query systems.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[2] Cisco. Cisco annual internet report white paper. Technical report, 09 2020.

[3] ElasticSearch Co. Open source search, 2018.

[4] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 269–286, Carlsbad, CA, October 2018. USENIX Association.

[5] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable

adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 253–266, New York, NY, USA, 2018. Association for Computing Machinery.

[6] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: Fast exploratory video queries using neural networks. *CoRR*, abs/1805.01046, 2018.

[7] CBS News. Video Surveillance in U.S. on par with China, year = 2019, url = https://www.cbsnews.com/news/the-u-s-uses-surveillance-cameras-just-as-much-as-china/: :text=Byurldate = 2019-12-10.

[8] Abhijit Suprem, Joy Arulraj, Calton Pu, and Joao Ferreira. Odin: Automated drift detection and recovery in video analytics. *Proc. VLDB Endow.*, 13(12):2453–2465, July 2020.

[9] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric, 2017.