BlazeIt: An Optimizing Query Engine for Video at Scale

Extended Abstract

Daniel Kang, Peter Bailis, Matei Zaharia
Stanford InfoLab, DAWN Project

ABSTRACT

Recent advances in deep learning allow us to query the increasing volumes of video data for semantic information. However, these techniques are difficult to deploy in practice and are incredibly computationally expensive. To address these limitations, we propose BlazeIt, a system with a query language and optimizer to accelerate complex queries over video. We demonstrate that BlazeIt can answer a wide range of queries and give significant speed improvements, up to 16×.

1 INTRODUCTION

Video is rich with semantic information and is a rapidly expanding source of data at scale: camera traps are widely used to study animal behavior [10], London has over 500,000 CCTVs [1], and autonomous vehicles can generate terabytes of data per day [5]. This growing volume of video can provide answers to queries about the *real world*. A city planner might ask, given a video of an intersection, when did buses arrive at this intersection? An ornithologist might ask, given a video of a bird feeder, what is the average time a bird spends at the feeder? When did red birds feed? An analyst at an autonomous vehicle company might look for clips of crosswalks. We consider these three examples as instantiations of general video analytics.

Recent advancements in deep learning provide powerful primitives for object detection [4] and semantic segmentation [6], which are rapidly improving [9]. However, actually applying these models to answer visual queries requires considerable effort. Consider object detection and entity resolution (the task of identifying objects as the same across several frames or scenes in a video) [3]. To calculate how long a bus stays on average at a stop sign, we could run the following steps over the video: 1) run object detection on every frame, 2) filter the detections for buses, 3) resolve which buses are the same in a scene, 4) return the average stay time. Likewise, the other queries can be answered in an *ad-hoc* fashion, by chaining together computer vision primitives.

Even when these ad-hoc pipelines are constructed, they can be prohibitively expensive to run at scale: state-of-the-art video object detection [12] runs at 5 frames per second (fps) on an NVIDIA P100 GPU (~$4000). Thus, using current technology, computing the average stay time over a week of video would take about 1.5 months of GPU time. As these volumes of videos grow, these analytic capabilities will only become more expensive.

To address the computational and usability challenges of querying video at scale, BlazeIt, a system with a query language and optimizer designed to accelerate queries over video.

BlazeIt's first component is its query language, FrameQL, which provides users with a relational programming interface that allows queries for object occurrence (in space and time) and content (class, metadata). Table 1 shows the schema. FrameQL's schema along with the relational algebra can answer all the example above queries and more.

Field	Type	Description
timestamp	float	Time stamp
mask	(float, float)*	Polygon containing the object of interest, typically a rectangle
class	string	Object class (e.g. bus, car, person)
sceneid	int	Unique identifier for a continuous time segment when the object is visible
globalid	int	Unique global identifier for object (e.g. license plate number)
content	image	The pixel content of the object

Table 1: FrameQL's schema contains spatiotemporal and content information related to objects of interest, as well as metadata (class, identifiers). Each record is associated with a frame, but a frame may have no or many records.

We note that not all fields are possible to compute for every video, e.g. license plate numbers can be used as a global identifier for cars, but identifying birds is difficult for even humans. We allow the user to specify the different components in BlazeIt (e.g. the object detection method, the identification method). For example, an ornithologist may use an object detector that can detect different species of birds, but autonomous vehicle analysis may not need to detect birds at all. Finally, we allow UDFs over the fields and content (e.g. to determine color, center position, etc).

The second component of BlazeIt is an end-to-end query optimizer: instead of simply running object detection to instantiate the rows in an FrameQL scene, BlazeIt leverages a range of techniques including specialization [8] and frame filtering. One key component of BlazeIt is that the rows are *lazily populated*: BlazeIt's query optimizer will create and execute a physical plan that only populates the fields necessary for a given query, which enables a variety of non-trivial optimizations, discussed in Section 3. As a simple example, few buses pass by an intersection at 4AM and a cheap method could be used to filter for when buses appear [8]. Prior work has focused on optimizing basic detection algorithms [11], but we focus on optimizations of end-to-end queries here.

2 QUERY LANGUAGE

We introduce FrameQL as a query language for querying videos at the *frame* level. FrameQL's scheme is described in Table 1. Users can query FrameQL using a standard SQL interface and the relational algebra. By providing a simple table-like schema using the relational algebra, we enable users with only familiarity with SQL to query videos, whereas implementing these queries manually would require expertise in deep learning, computer vision, systems building, etc. Thus, by unifying the video queries with the relational algebra, we enable non-specialists to access these queries.

SysML'18, February 2018, Stanford, CA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

We describe how FrameQL can be used to answer a complex query: "show me clips of red birds at least 1 second long in the lower left corner". Here, `color` is a function that operates over the contents of the object and returns a color, and the video is at 30 fps which is why the query is filtered by at least 30 timestamps. Finally, we define the lower left corner to be the bottom 600×600 pixel box. The other queries in the introduction can be answered similarly.

```
SELECT timestamp FROM bird_feeder
WHERE class = 'bird' AND color(content) = 'red'
  AND xmax(mask) < 600
GROUP BY sceneid
HAVING COUNT(timestamp) >= 30
```

3 QUERY OPTIMIZER

Here, we describe BlazeIt's query optimization for FrameQL. In BlazeIt, rows are only instantiated upon request. Fully populating the table (i.e. SELECT *) does not allow for query optimization, but selecting certain fields or ranges can run dramatically faster. For example, an analyst may only require the frames in which a bird appears, but not the bounding boxes (i.e. SELECT timestamp): in this case model specialization [8] can allow several orders of magnitude of speedup, as the query changes from detection to classification. When the user issues a query, BlazeIt first creates a logical plan. Then, BlazeIt estimates the selectivities of various filter and picks the physical plan with the lowest estimated cost.

In BlazeIt, we provide 5 classes of query optimizations: 1) label-based filtering, 2) content-based filtering, 3) temporal filtering, 4) spatial filtering, and 5) predicate reordering. As these filters typically rely on statistical methods, they have some error rate which must be accounted for, e.g. using a cost-based optimizer as in [8]. While some of these optimizations have been previously explored in isolation, we consider larger queries where we can apply other ideas from traditional query optimization [2] as well as take advantage of the nature of video data. BlazeIt, to our knowledge, is the first engine to apply these to video queries using CNNs. Finally, we provide several instantiations of these optimizations, but we note that there are many other such instantiations, which we view as an exciting avenue for future research.

Label-based filtering. In label-based filtering, the video is filtered based on the desired labels. NoScope [8] explores label-based filtering for binary predicates (e.g. presence or absence of a bus). This optimization can provide enormous speedups (up to 6000×) in the form of *model specialization*, in which a cheaper model for the task at hand is trained over a subset of the data and deployed for the rest. While NoScope considers binary predicates, we plan to explore other forms of specialization, including multi-class and counting queries, and we see the selection and training of specialized models in complex queries as an exciting area of future research.

Content-based filtering. In content-based filtering, the video is filtered based on fast, low-level visual features, such as average color. If an analyst were to query for "red birds", we could filter the video to have a certain number of red pixels.

Temporal filtering. In temporal filtering, the video is filtered based on temporal cues. For example, if only clips that are over 3 seconds are desired (e.g. to study the behavior of birds feeding at a feeder, not ones passing by), the video can be sampled at 1.5 seconds. We additionally support basic forms of filtering such as "query the video from 10AM to 11AM".

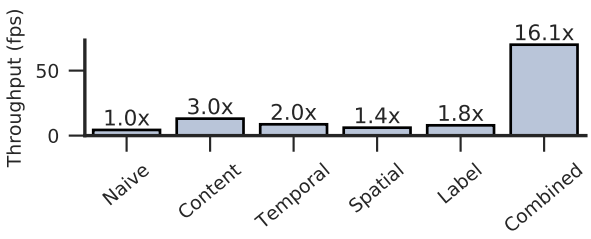

Figure 1: Throughput of BlazeIt's various optimizations.

Spatial filtering. In spatial filtering, only regions of interest (ROIs) of the scene are considered. For example, a street may have cars parked on the side but the analyst may only be interested in vehicles in transit. The ROI is specified by the user and can be used in smaller models for faster inference, and activity outside the ROI can be ignored, which increases the selectivity of other filters.

Predicate reordering. In predicate reordering, different predicates in the query can be reordered based on the selectivity, error rate, and cost of the filters. For example, in the query "show me clips of red birds that are present for more than 3 seconds", the filters of "bird", "red", and "more than 3 seconds" can be ordered in several ways, which affects the runtime. The selectivities and runtime costs can be estimated using a held-out dataset and optimized using a cost-based optimizer [8].

4 PRELIMINARY RESULTS

We manually implemented the query in Section 2 using a state-of-the-art video object detector [12] (FGFA) and OpenCV. Then, we implemented each optimization in isolation and combined them. Experiments were done on a server with an NVIDIA P100 GPU and an Intel Xeon E5-2690 CPU. As is standard, we ignored the time to load the video. Video from a bird-feeder was collected for 3 days in December with 9 hours of video per day, to prevent over-fitting. A separate day was used for training labels, estimating selectivity, and inference. We only time the cost of inference. The naive baseline was computed by running FGFA over every frame of the video and using Seq-NMS [7] to identify birds across frames.

The results are shown in Figure 1. As we can see, by performing end-to-end optimization, we can achieve a speedup of up to 16×. The combined speedup is higher than the product of the individual speedups as the selectivity of "bird" and "red" is higher than product of the individual selectivities. Predicate reordering is only used in the combined setting.

5 CONCLUSION

Video volumes and our ability to query these videos through deep learning continue to grow. However, when naively applied, these methods are prohibitively expensive to run at scale, and the need for computation will only increase. In response, we introduce FrameQL for complex video queries and a query optimizer for executing FrameQL queries. We demonstrate several classes of optimizations and show how they can combine to achieve 16× speedups.

ACKNOWLEDGEMENTS

This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Google, Intel, Microsoft, Teradata, and VMware—as well as DARPA under No. FA8750-17-2-0095 (D3M), industrial gifts and support from Toyota Research Institute, Juniper Networks, Keysight Technologies, Hitachi, Facebook, Northrop Grumman, NetApp, and the NSF under grants DGE-1656518 and CNS-1651570.

REFERENCES

[1] 2015. CCTV: Too many cameras useless, warns surveillance watchdog Tony Porter. (2015). http://www.bbc.com/news/uk-30978995

[2] Morton M. Astrahan, Mike W. Blasgen, Donald D. Chamberlin, Kapali P. Eswaran, Jim N Gray, Patricia P. Griffiths, W Frank King, Raymond A. Lorie, Paul R. McJones, James W. Mehl, et al. 1976. System R: Relational approach to database management. *ACM Transactions on Database Systems (TODS)* 1, 2 (1976), 97–137.

[3] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 5.

[4] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. 2010. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence* 32, 9 (2010), 1627–1645.

[5] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. 2013. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)* (2013).

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.

[7] Wei Han, Pooya Khorrami, Tom Le Paine, Prajit Ramachandran, Mohammad Babaeizadeh, Honghui Shi, Jianan Li, Shuicheng Yan, and Thomas S Huang. 2016. Seq-nms for video object detection. *arXiv preprint arXiv:1602.08465* (2016).

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

[9] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

[10] J Marcus Rowcliffe, Roland Kays, Bart Kranstauber, Chris Carbone, and Patrick A Jansen. 2014. Quantifying levels of animal activity using camera trap data. *Methods in Ecology and Evolution* 5, 11 (2014), 1170–1179.

[11] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. 2017. Fast video classification via adaptive cascading of deep models. *CPVR* (2017).

[12] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. 2017. Flow-Guided Feature Aggregation for Video Object Detection. *ICVV* (2017).