Querying For Interactions

ABSTRACT

Advances in Deep Learning and Computer Vision enabled sophisticated information extraction out of images and video frames. As a result there has been recent interest in techniques and algorithms to enable interactive declarative query processing on objects and their associated constraints on video feeds. The emphasis is to make objects, their types and relative locations as the video evolves, first class citizens for query processing purposes.

In this paper, we initiate research to explore declarative style of querying for real time video streams involving objects and their interactions. We seek to efficiently identify frames in a streaming video in which an object is interacting with another in a specific way, such as for example a human kicking a ball. We first propose an algorithm called progressive filters (PF) that deploys a sequence of inexpensive and less accurate models (filters) to detect the presence of the query specified objects on frames. We demonstrate that PF derives a least cost sequence of filters given the current selectivities of query objects and minimizes the per frame cost for object detection. Since selectivities may vary as the video evolves, we present a dynamic statistical test to determine when to trigger re-optimization of the filters. Finally, we present a filtering approach which we call Interaction Sheave (IS) that utilizes learned spatial information about objects and interactions to effectively prune frames that although contain the query objects are unlikely to involve the query specified action between them, thus improves the frame processing rate further.

The techniques we propose constitute a robust approach to process video streams for query specified object interactions achieving very high frame processing rate. We present the results of a thorough experimental evaluation involving real data sets, demonstrating the performance benefits of each of our proposals. In particular we experimentally demonstrate that our techniques can improve query performance substantially while maintaining essentially the same f-score as brute force alternatives.

ACM Reference Format:

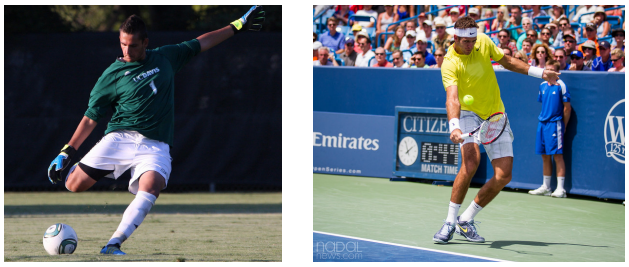
. 2019. Querying For Interactions. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent years, Deep Learning (DL) introduced numerous techniques to comprehend challenging data types such as images, video and unstructured text (among many others) yielding breakthroughs in applications such as image classification, video understanding and natural language processing. Various aspects of DL are still being actively researched. Several algorithms in the areas of object detection in images and video, object classification and object tracking are currently considered state of the art [8, 9, 20, 29]. These algorithms offer the ability to classify objects in image frames, detect object locations in a frame as well as track objects from frame to frame with reasonable accuracy.



(a) (b)

Figure 1: Examples of interactions

By the time one has finished reading this sentence (assuming it takes 3 seconds), as of this writing, the equivalent of 30 hours of video would have been uploaded to youtube alone. The prevalence of video recording devices via inexpensive cameras resulted in an explosion of video content. As a result the development of techniques to efficiently analyze video has become a pressing concern. Video analytics encompasses a very broad set of techniques that depending on the application domain become more specialized. Of particular interest in this paper is video monitoring and surveillance applications in which static cameras record activity in their receptive field. This is the case in applications such as road traffic monitoring and surveillance security. Several recent

works [15–17, 19, 35] focus in these domains and aim to provide declarative style of queries on top of streaming video feeds. Our work complements and enhances this line of research focusing on the efficient execution of an important query primitive, namely capturing *interactions among query objects*. Numerous queries of interest become possible when we can quantify object interactions. For example, automatically detecting frames in which a human holds a gun or a human breaks a window glass would be of great interest in surveillance and security applications. Figure 1 presents examples of interactions between a human and a ball. The corresponding query in SQL for the interaction at the frame of Figure 1(a) would be:

```
SELECT cameraID, frameID,
C1 (F1 (HumanBox1)) AS HumanType1,
C2 (F2 (Ballbox1)) AS BallType1,
FROM (PROCESS inputVideo PRODUCE cameraID, frameID,
HumanBox1 USING HumanDetector, BallBox1 USING BallDetector)
WHERE HumanType1 = human AND BallType1 = SmallBall
AND INTERACTION(HumanType1, BallType1) = KICK
```

The query employs two classifiers (C_i) to detect a *human* and a *ball*, using features F_i extracted from the frame and checks whether the objects, once identified, are related via a KICK interaction. Evidently from a execution perspective it makes sense to invoke the INTERACTION predicate once the operands have been identified on the frame.

In this paper we primarily focus our examples on human and object interactions, to keep our presentation and discussion focused, but our techniques generalize to interactions between general object pairs. We seek to automatically identify frames in a video stream in which query objects interact in a specific way (which is expressed as part of the query). Although one can identify query objects in video frames (e.g., [6, 8]), determining efficiently which frames contain a specific interaction among the objects, necessitates the application of specialized DL models [9, 18, 23, 31, 33, 36] which typically require larger amounts of time per frame to evaluate. The main emphasis of our work is, given a query involving objects and their interaction, propose algorithms to process a video stream and efficiently determine which frames are promising to be part of the query answer, effectively filtering out all irrelevant frames. Thus, we seek to increase the frame processing rate, as frames that are deemed irrelevant are not processed further and are quickly skipped. Even without filtering, application of DL models for object and interaction detection entails false positives/negatives. We will quantify the effect that filtering has into the false positive/negative rate of the techniques.

To realise this goal we present a set of algorithms that when combined, dramatically increase the frame processing rate when executing the query, while maintaining high accuracy. First, it is evident that before we test a frame for a query specified interaction among objects, we should test whether objects of the type specified by the query are present in the frame. Such a test can take place by involving state of the art object classification or detection models [6, 29]. The drawback is that although such models have high accuracy they impose large overheads in terms of processing time per frame [6]. Prior work [17, 35] has utilized cheap inexpensive filters that are trained apriori or on demand to reduce processing overheads at the expense of accuracy; they are typically trained to achieve specific false positive and false negative rates, while relaying any other decision to a high accuracy (but more expensive) model. Such a filter quickly determines whether a frame contains an object of a specific type or not. If the filter is uncertain to make such determination, it invokes a high accuracy filter to process the frame further. The premise of such filters is that a large fraction of frames are not relevant to the query and can be dropped quickly. Moreover, relevant frames can be positively decided by the inexpensive filter, and only tough cases will involve an expensive model. Such inexpensive filters are employed one per object type specified in the query. We observe that one can improve the pruning efficiency of such a filtering approach by deploying a series of inexpensive filters that progressively have higher accuracy (i.e., they are less selective, in the sense that they can drop a larger fraction of their input) and higher cost [1]. Essentially we realise an object detection *operator* as a sequence of filters. Assessing the selectivity for each of the filter predicates, we propose a practical algorithm that derives the optimal set of filter predicates for an object type, to deploy. Thus, we address the *inter-operator scheduling* of filters to minimize detection cost. Assuming the future statistical characteristics of the video stream remain the same, the filtering combination will minimize the cost (alternatively maximize frame processing rate) to process the video stream and determine frames that contain the specific object type.

Since one cannot expect the distribution of objects on video frames over time to remain the same, we propose an algorithm to incrementally assess when the statistical properties of the underlying object distribution change. When such properties change, we trigger re-optimization deriving a new optimal filter sequence to deploy. The basic idea behind our proposal is to treat the selectivity of filters as a statistical population and employ a dynamic version of the popular Kolmogorov-Smirnof test [14]. That way we monitor the underlying distribution of objects relevant to the query between two time epochs and trigger re-optimization when the statistical properties change.

[1] Cost is determined by the number of layers in the deep network the frame has to pass through in order to make its prediction.

The goal of these two developments is to prune or positively determine effectively all frames irrelevant/relevant to the query and maintain such effectiveness over time. In order to address our ultimate objective, each frame that contains the objects of interest to the query has to be processed by a DL model that determines the *type* of interaction between objects [9, 18, 23, 31, 33, 36]. These models determine if the frame is relevant to the query output. Such object interaction models impose their own overheads per frame however. Current state of the art models, require between a quarter to half a second per frame. To improve the frame processing rate even further, we develop a *filtering mechanism for object interactions*. The basic observation is that when objects interact (e.g., human kicking ball) the interaction typically takes place at a *specific spatial region* of the frame involving the first object (i.e., the human). For each type of interaction, we propose a model to identify such regions. The frame is relayed to an expensive object interaction model for further processing, only when the second object (e.g., ball) is located within the spatial region of the action, relative to the first object (i.e., human). Thus, by processing the objects spatially we obtain a filtering mechanism for their interaction. Our final proposal combines these three techniques delivering an effective framework to process object interaction queries, efficiently improving the frame processing rate in a video stream dramatically.

The overall approach is shown in Figure 2. There are two operators each consisting of a number of filters. The application of *PF* for operator one determined that filters 1 and 3 are in use currently. Similarly for operator two, filters 2,3 are in use. Frames that successfully pass the operators with a positive determination [2] that they contain the objects of interest, are tested by the Interaction Sheave. If they pass that filter they are subsequently tested by an object interaction model. We emphasize that our focus in this work is how to effectively order filters within an object detection operator. Typically a query will check for presence of more than one object (as in the case of the sample SQL query presented). Scheduling across operators (intra-operator scheduling) can be accomplished using standard practise [3] (ordering operators by the inverse of their selectivities) if selectivities of each operator are known (for example, we expect more humans to appear in frames than balls) or by utilizing approximation results for operator scheduling [1] (utilizing learned selectivities).

[2]For the first operator a frame may get a positive determination by filter 1 and in that case there is no need for further processing by this operator. Filter 3 (the highest accuracy model in this simple case) will be involved if Filter 1 is unable to make a positive or negative determination for the frame.

In this paper we focus on the problem of enabling object interaction queries over video streams and make the following contributions:

- We present an algorithm called *Progressive Filters (PF)* that given a set of filters, for a specific object type, of increasing cost (number of DL layers) utilizes their selectivity to derive the optimal sequence with which the filters should be applied to minimize the total cost of processing the video stream. We analyze the effectiveness of this algorithms in terms of maximizing the frame processing rate for typical queries.
- Since we expect that on a video stream, the statistical properties for objects of a specific type will vary over time, we adapt a dynamic version of the Kolmogorov-Smirnoff test that can trigger algorithm PF when the statistical properties of the filter selectivities change. Effectively we are adjusting the filter sequence, in anticipation of stable (predictable) object statistics, until the next re-optimization. We experimentally demonstrate that such a strategy can provide great savings in temporally maintaining a high frame processing rate, compared with alternate approaches.
- We propose a filtering mechanism for object interaction queries, which we refer to as *Interaction Sheave (IS)*. IS is capable to spatially filter the location of objects on frames and drop frames that although contain objects relevant to the query, are not promising to encompass the suitable interaction among the query objects.
- We present the results of a thorough performance evaluation utilizing real benchmark data sets and demonstrate the effectiveness of each technique in isolation as well as evaluate the total effectiveness of combining all proposals in our prototype evaluation. Our results indicate that our proposals can increase frame processing rate substantially while maintaining high accuracy.

This paper is organized as follows: In Section 2 we review related work. Section 3 presents algorithm *PF*, followed by Section 4 in which we discuss dynamic approaches to statistical population tests and in particular Kolmogorov-Smirnoff that allows us to assess when *PF* is required to execute on the video stream. Section 5 presents the Interaction Sheave filter to effectively refine the execution of expensive interaction detection models and improve frame processing rate further. Section 6 presents our experimental evaluation and Section 7 concludes the paper discussing avenues to future work in this area.

2 RELATED WORK

Recognizing the importance of query processing on streaming video, several recent works focus on different aspects

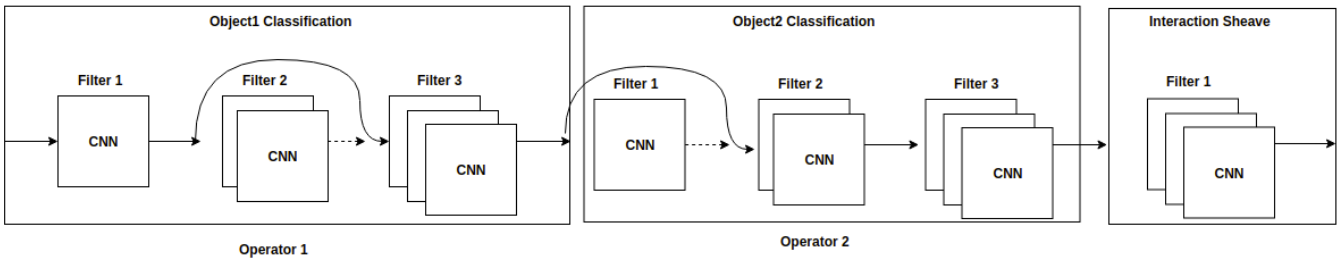


Figure 2: Example of the overall architecture

of declarative query processing over video streams [15–17]. [17] presents a first approach to filter frames via inexpensive filters for query processing purposes. Our work extends the approach to multiple (progressive) filters and proposes an optimal algorithm to identify which filters are most beneficial at a given instant for maximing frame processing rate. Subsequently [15, 16], present query processing techniques for processing specific type of aggregates over the video stream. On a related thread [19, 35] present a declarative query processing framework to support queries with spatial constraints among objects on video streams. They also propose techniques to process aggregation queries across frames over objects with associated spatial constraints. Other related works [13, 22] present system approaches to query processing across video streams and are concerned with scalability aspects.

In the vision community, several Deep Learning approaches for object detection and classification have been proposed with impressive accuracy [10, 12, 20, 27]. Similarly recent approaches have presented Deep Learning based models that can detect certain types of interactions between objects [9, 18, 23, 31, 33, 33, 36]. However such models are rather heavyweight and require every frame to be evaluated against them, imposing unnecessary overhead for frames that are not promising to contain the interactions. For this reason our approach is to deploy our proposed Interaction Sheave (IS) as a lightweight filter to remove irrelevant frames and only route to expensive action detection models [18, 33, 36] frames that are promising to contain the desired type of action.

3 PROGRESSIVE FILTERS

DL models for object classification/detection typically consist of a number of convolutional layers followed by a number of fully connected layers responsible for the final prediction [6]. Depending on the model, the number of layers may vary (for example, VGG [29] models have been proposed with 16 and 19 layers). It has been observed [10] that each layer is responsible for automatically learning features that are important for predictions. Typically early layers learn high level object features that become progressively specialized as we move to later layers. A DL model requires that a frame is passed through all layers to produce the prediction. Each time a frame passes a layer, this imposes a processing overhead, thus processing cost of a DL model is directly related to the number of layers it involves. Deeper models (consisting of more layers) are capable of delivering higher accuracy at the expense of processing cost. One way to improve the frame processing rate for a video stream is to train models that consist of fewer layers [17, 32] thus trading accuracy for speed. Such cheaper DL models are widely referred to as *filters* since they are capable of either dropping frames from further processing, when the filter is confident that the frame does not contain the object of interest or declaring object presence if their confidence is high. Such filters are typically optimized for specific true positive and true negative rates, passing all frames for which the filter is uncertain to the heavyweight model.

Since higher layers on a DL model learn progressively more specialized features of an object, deploying a sequence of filters of increasing complexity may offer benefits. For example when searching for a human object, the first filter will seek to infer generic object presence (i.e., the frame is not empty), following by a model that infers human shape, followed by a model inferring human features such as hair, eyes etc. The application of these filters are not independent; consider the case, for example, of the camera monitoring pedestrians passing a store front. Although one filter can stop a frame from further processing, if its confidence is high, under a setting that progressively applies all filters to a frame, if the confidence of intermediate filters is not high, all filters will have to be evaluated before a decision is made by the final (expensive) model. This of course is sub-optimal as in such a case the intermediate filters are not effective and their (inconclusive) application increases the overall processing cost. In these cases, it would make sense to skip intermediate filters, since their application is not beneficial. It is evident that the decision on which filters to progressively apply depends on the selectivity of the filters

(fraction of frames that pass the filter) and the associated filter costs.

Let $f_1, \ldots f_n$ be a sequence of filters with associated costs $c1, \ldots c_n$, with $c_1 < c_2 < \ldots < c_n$. The costs of each filter are fixed and depend on the number of layers each filter encompasses. Let R be the frame sequence and r a specific frame. We designate as σ_i the fraction of frames $r \in R$ that are not classified conclusively by filter f_i and pass through the i-th filter for further evaluation. Clearly $\sigma_1 > \sigma_2 \ldots > \sigma_n$, namely higher cost filters are able to classify conclusively more frames. For example in a busy intersection with cars and occasional pedestrians, a set of filters f_1, f_2, f_3, f_4 (determining, object presence, object shape, human features such as hair and finally applying a fully featured VGG object detection model) have associated selectivities 0.9, 0.5, 0.03, 0.01. This means that 90% of the frames are not classified at this stage as having or not a human based on object presence, 50% of them are not classified conclusively at this stage based on object shape, 3% of them no classified conclusively based on hair features and 1% of them invoke an expensive and accurate model to determine the presence of a human in the remaining frames.

Assuming n filters, $f_1 \ldots f_n$ participate to detect an object type on a video stream, and assuming all frames are relayed exactly through the same sequence of n filters in a progressive manner (f_1 followed by f_2 upto f_n), the per frame cost to process the stream would be Cost($f_1, \ldots f_n$) = $c_1 + \sum_{i=2}^{n} \sigma_{i-1} c_i$. Given the filters f_i and associated selectivities σ_i we are interested to produce the optimal set of filters to be applied that minimizes the per frame processing cost. Assume f_n is the most expensive model that yield the highest accuracy but requires the highest frame processing time (e.g., VGG-19 [29] for the case of object recognition). This filter will always be invoked as we wish to determine the exact answer in the end. Without loss of generality we assume f_0 is a filter with $c_o = 0$ and $\sigma_0 = 1$. Assume v_i if true if filter f_i is evaluated. The optimal subset of filters to utilize in order to minimize the per frame processing cost, can be computed as follows.

Let F be the sequence of filters $f_1, \ldots f_n$. Let $F_s = f_s \ldots f_n$. Let $Cost_{k,s}$ denote the cost of the optimal solution including filters starting with f_s assuming the filter applied previously was f_k (that is filters between f_k and f_s have not been evaluated). Some of the filters in F_s will be included in the optimal solution and some may not. For the case of f_s there are two cases. If it is included it will incur a cost in F_s of $\sigma_k c_s + Cost_{s,s+1}$, that is the cost to evaluate f_s plus the cost to optimally complete the sequence. If it is not included, the cost in F_s will be $Cost_{k,s+1}$ (since f_s is not part of the solution). Thus, let $Cost_{k,s}$ be the optimal cost by filters in F_s assuming $k < s$, the s-th filter was evaluated and $\nexists j$ such that $k < j < s$ having the j-th filter evaluated. We have that:

$$Cost_{k,s} = \begin{cases} \sigma_k c_s & \text{if } s = n \\ min\{\sigma_k c_s + Cost_{s,s+1}, Cost_{k,s+1}\} & \text{otherwise} \end{cases}$$

Evaluating the equation above for all values of k and s with $k < s$ provides the desired solution. This is an $O(n^2)$ algorithm, but given that n (the number of filters) is a small number (typically in the ranges of 3 to 14) the computation is efficient. We wish to process frames at a fraction of a millisecond and optimization should impose minimal overhead.

Algorithm 1: Progressive Filters algorithm

```
1 def progressiveFilters(N, S, C):
2 lastFilter = N - 1
3 if N == 1:
4    filtersCost = C[lastFilter]
5    predicates = [lastFilter]
6 else:
8    for i in range (1, lastFilter):
9       OverallCost[i][i+1] = 0
10      eliminate[i][i+1] = 0
11      OverallCost[i][lastFilter] = \
12      S[i]*C[lastFilter]
13      eliminate[i][lastFilter] = 0;
14   for j in range (lastFilter-1,0,-1):
15       for k in range(1, j+1):
16          cost1 = S[k]*C[j] + \
            OverallCost[j][j+1]
17          cost2 = OverallCost[k][j+1]
18          OverallCost[k][j] = \
            min(cost1, cost2)
19          eliminate[k][j] = \
            cost1 > cost2?1:0
20   finalCost = OverallCost[0][1]
21 predicates = list()
22 index = 1
23 for i in range (1, lastFilter):
24    if not eliminate[index][i]:
25       predicates.append(i)
26    index = i
27 return predicates
```

Algorithm *PF* is shown in Algorithm 1. It's inputs consist of variables N, S and C which are the number of available filters, selectivity array and cost array of each filter respectively. The algorithm outputs a predicate array. The algorithm uses selectivity and cost of each filter to determine the overall cost per filter. First, variables OverallCost (the cost of the connection between some filter with the last) and eliminate (a binary array which represents the eliminated filters of the

final solution) are initialized. The last filter always appears on the final solution, consequently its value is zero on the eliminate array. Next, the algorithm decides which filters should be eliminated (lines 14-19). The cost of the current filter is compared with the cost of the filters preceding it. The current filter is eliminated from the final solution if its cost is higher than the preceding filter's cost. Subsequently the solution is extracted.

Filter	Selectivity	Time Cost
Filter1	1	0.1
Filter2	0.6	0.5
Filter3	0.2	1

Table 1: Progressive Filters input example

Table 1 presents a possible input to Algorithm 1.

i/j	1	2	3
1	0	0	1
2	0	0	0.6
3	0	0	0

Table 2: Overall Cost array

The initialization of OverallCost is depicted in Table 2; *eliminate* is a same sized array initialized with zero values. The outcome of executing lines 14-19 is presented on Table 3. Since cost1 of both filter1 and filter2 is higher than cost2 the only filter in the final result is filter3.

	j=2, i=1	j=1, i=1
cost1	1.1	1.1
cost2	1	1

Table 3: Filters' cost

Algorithm *PF* computes the optimal number and ordering of filters to apply, deciding object presence, minimizing per frame cost, given a set of known selectivities for each filter. The selectivities for each filter are expected to change as the video stream evolves and the statistics of object presence in frame change. Thus, it is imperative to be able to detect such change and determine when re-optimization is warranted. We address this in the next section.

4 TRIGGERING RE-OPTIMIZATION

Let $\sigma_1, \ldots \sigma_n$ be the selectivities of filters $f_1, \ldots f_n$. The selectivity of a filter is the fraction of video frames that pass this filter (i.e., the filter cannot conclusively decide if the frame contains or not a specific object type). The application of algorithm *PF* may employ a subset of filters. For the filters employed we can maintain their selectivities up to date by observing the result of each frame tested by the filter. For the filters that are not part of the optimal solution, we periodically (every few seconds) route a frame from the input sequence through them and obtain a selectivity estimate as well.

When the video stream starts, we do not possess selectivity estimates for the filters. We route all frames through all filters obtaining the selectivity for each filter for a time interval t. Let $\Sigma_t = \{\sigma_1^t, \ldots, \sigma_n^t\}$ (with $\sigma_1^t > \ldots > \sigma_n^t$) be the resulting selectivities. We execute *PF* utilizing the selectivities in Σ_t, and obtain a sequence of filters to apply. Let $\Sigma'_t = \{\sigma_1^{t'}, \ldots \sigma_n^{t'}\}$ (with $\sigma_1^{t'} > \ldots > \sigma_n^{t'}$) the selectivity of each filter for a time interval t'. We need a test to determine whether given, Σ_t and $\Sigma_{t'}$, invocation of algorithm *PF* is warranted. An approach is to view Σ_t and $\Sigma_{t'}$ as samples from an unknown distribution and test at a significance level α whether to accept or reject the null hypothesis that the observations in Σ_t and $\Sigma_{t'}$ originate from the same distribution. We can use the Kolmogorov-Smirnov (KS) test to verify the hypothesis. According to this test we can reject the null hypothesis at level α if the inequality $D > c(\alpha)\sqrt{(\frac{2}{n})}$ is satisfied. $c(\alpha)$ is obtained from known tables [14] and $|\Sigma_t| = |\Sigma_{t'}| = n$. D is the KS statistic defined as:

$$D = sup_x |F_{\Sigma_t}(x) - F_{\Sigma_{t'}}(x)|$$

and

$$F_A(x) = \frac{1}{|A|} \sum_{j \in A, j \leq x} 1$$

D can be computed as

$$D = max_{x \in \Sigma_t \cup \Sigma_{t'}} |F_{\Sigma_t}(x) - F_{\Sigma_{t'}}(x)|$$

and requires $O(n \log n)$ time to compute.

One could determine the selectivities for a new time intervals t'', $\Sigma_{t''} = \{\sigma_1^{t''}, \ldots, \sigma_n^{t''}\}$ and apply the KS test between $\Sigma_{t'}$ and $\Sigma_{t''}$ at the cost of $O(n \log n)$ to continuously determine if re-optimization is warranted. We assume a sliding window model of time and device a dynamic KS-test. Assume that Σ_t and $\Sigma_{t'}$ have been computed for time intervals t and t' (typically the lengths of these intervals will be the same, say 10 seconds). For an interval t'' we start computing the selectivities $\Sigma_{t''}$. This means that for filters in the solution of *PF* we compute the selectivities for frames presented to them at the start of t'' and for filters not participating in the solution, we periodically route a frame from the input through them to obtain selectivity estimates. Every $\lfloor \frac{t''}{n} \rfloor$ seconds, for $1 \leq j \leq n$ we remove σ_j^t from Σ_t, we remove $\sigma_j^{t'}$ from $\Sigma_{t'}$ and insert it into Σ_t and we insert $\sigma_j^{t''}$ into $\Sigma_{t'}$. That way we adjust the selectivities of sets Σ_t and $\Sigma_{t'}$. We cycle

through $1 \leq j \leq n$ as we expect the lower the value of j the more frames from the input starting at time interval t'' the corresponding filter f_j will be presented with, when obtaining the selectivity estimate during $\lfloor \frac{t''}{n} \rfloor$ seconds. When we reach the end of the time interval t'' we set all selectivity estimates in $\Sigma_{t''}$ to zero and start the estimation again. We seek an efficient way to conduct the KS-d between Σ_t and $\Sigma_{t'}$.

In order to address this, we utilize Cartesian Trees [4, 28] and adapt [5] in our setting. More specifically, define $Diff(x) = F_{\Sigma_t} - F_{\Sigma_{t'}}$, then

$$D = \frac{1}{n} max(max_{x \in \Sigma_t \cup \Sigma_{t'}} Diff(x)), -(min_{x \in \Sigma_t \cup \Sigma_{t'}} Diff(x))$$

Cartesian Trees organize all values $v_i \in \Sigma_t \cup \Sigma_{t'}$ in an ordered fashion (namely $\forall i, v_i < v_{i+1}$ and also store for each v_i, $Diff(v_i)$ and a random priority value. For a list of pairs of unique v_i's and unique priorities, there is only one possible Cartesian tree, namely the in-order traversal is determined by the keys and the *ranking* of the priorities. If priorities are chosen at random, the Cartesian Tree, adopts a property of randomized binary search trees and grows in height logarithmically. Consequently inserting/removing requires logarithmic time to the total number of elements.

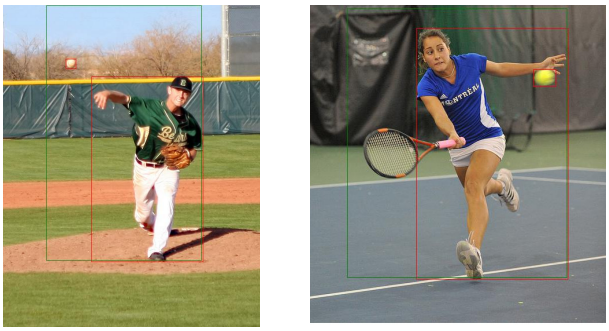
Upon insertion of a new value v_j, Cartesian tree properties, guarantee that $v_{j-1} < v_j < v_{j+1}$. Due to this property, $Diff(v_i) = Diff(v_{j-1}) + k$, where $k = 1$ if $v_j \in \Sigma_t$ and $k = -1$ if $v_j \in \Sigma_{t'}$. As a result of the insertion of v_j, all $Diff(v_i)$ with $i < j$ do not change and all $Diff(v_i)$ with $i > j$ are increased by k. The removal of a value v_j decreases all $Diff(v_i)$ for which $i > j$ by k. Based on the standard properties of Cartesian trees, we can insert/delete v_j's from the tree as well as add/subtract a value from all $Diff(v_i)$ with $i > j$ in $O(\log n + n)$.

The tree also allows to compute the maximum and minimum values of $Diff(v_i)$ in $O(1)$ utilizing summary information stored in each node. Since priorities are fixed, predecessor/successor relationships among nodes is preserved under sub-tree merge and split [28] as required by the Cartesian Tree re-balancing algorithms under inserts/deletes. This allows us to enhance each node p in the tree with summary information regarding the $Diff()$ values in the sub-tree rooted at p and efficiently compute minimum and maximum values for each sub-tree.

5 INTERACTION SHEAVE

For frames that pass through the filter sequences determined by *PF* for each object specified by the query, we have confidence that they contain the required objects. The actual objects can be detected precisely on the frame via the application of object detection models [6]. These models will derive a bounding box that encloses the location of each object as specified by the query on the frame. We then test the frame whether it relates the objects via the query specified interaction. State of the art object interaction models [9, 18, 23, 31, 33, 33, 36] employ deep neural networks [6] and typically require around 250-500 milli-seconds per frame, imposing a significant overhead to frame processing rate.

We present a technique to assess whether a frame is likely to contain an interaction of the desired type, thus effectively processing through expensive interaction detection models, only frames for which we have enough confidence they contain the suitable interaction. The underlying idea of our approach is that for most interactions, depending on the object types, the spatial area of the frame in which the interaction takes place is typically known or can be predicted. Consider Figure 3 depicting examples of interactions. For the case of *human throwing a ball* the spatial interaction between the object typically takes place at the area of the frame close to the hands of the human. Similarly for the case of *human hitting a ball*. Thus for human interactions with various objects, the target object (e.g., ball) is typically located in a spatial region that is conditioned on the location of the human object. The same observation carries over to interactions between different object types. These spatial areas don't have to be defined apriori but can be easily learned from data examples. As a result, when the objects specified by the query are detected in the frame, one can check, given interaction a specified by the query, whether the target object is located in the predicted region for a, given the detected human object. If this is the case, we can process the frame further with more advanced object interaction models. If not, we can filter out the frame from further processing.



(a) (b)

Figure 3: Examples of interactions

We determine spatial regions for specific interactions by learning them from data directly. More specifically, given an object prediction o (e.g., a human) and the spatial coordinates of the object on the frame (typically a bounding box box_o represented as (x_c^o, y_c^o, w^o, h^o), where (x_c, y_c) are the center coordinates, h the height and w the width), we determine

for a given action type a the mean μ_o^a of a Gaussian density. This is the density over the possible locations of objects τ interacting with o via a. To determine the bounding boxes of objects on a frame we deploy fast techniques for object detection [26]. Given the bounding box box_o of object o and the bounding box $box_\tau = (x_c^\tau, y_c^\tau, w^\tau, h^\tau)$ of object τ, following [7] we compute $t_{o|\tau} = (t_x, t_y, t_w, t_h)$ as:

$$t_x = (x_c^\tau - x_c^o)/w^o \qquad t_y = (y_c^\tau - y_x^o)/h^o$$
$$t_w = \log w^o/w^\tau \qquad t_h = \log h^o/h^\tau$$

We then score the two bounding boxes based on the estimated Gaussian density, namely:

$$\frac{1}{\sqrt{(2\pi\sigma^2)}} \exp \frac{(||t_{o|\tau} - \mu_o^a||)}{2\sigma^2}$$

That way we can rank how distant is the target object τ from the mean of the density of the action around object o and decide if we will proceed with passing the frame to a heavyweight action detection model. Following common practise we set the threshold to two standard deviations from the mean.

We learn to predict μ_o^a given examples of frames that contain objects o_i of a specific class interacting via a with other objects τ_i with classes as specified by the query. We train a deep network minimizing smooth L_1 loss [6] between box_o and $t_{o|\tau}$.

$$L_{loc}(box_o, t_{o|\tau}) = \sum_i smooth_{L_1}(box_{o_i} - t_{o|\tau_i})$$

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

During training we are provided with ground truth objects and their location boxes box_o in frames depicting action a with other objects τ with bounding box box_τ. We determine σ as a hyper-parameter during experiments. We are using synchronized SGD [21] with momentum (0.9) on 2 GPUs and the batch size of each GPU is 4 images, so the total batch size consists of 8 images. We are using learning rate of 0.0001, we set momentum at 0.9 and weight decay at 0.0001. The available data for each action are split on train, test and validation data. We are using 80% of the data for training, 10% for testing and 10% for validation. Our implementation is based on VGG-16 [30] on PyTorch library [25]. For the action area prediction, we are using one 1024-d fully connected layer with ReLU [24].

6 EXPERIMENTAL EVALUATION

We now present the results of a detailed experimental evaluation of the proposed approaches. We utilize four different real data sets namely, **Coral**, **Detrac** [34], **UCSD Pedestrian Dataset** [2] and **V-COCO** [11] which is part of COCO dataset. Of these data sets, **Coral**, **UCSD Pedestrian Dataset** are video sequences shot from a single fixed-angle camera, while **Detrac** is a traffic data set composed of 100 distinct fixed angle video sequences shot at different locations. We utilize these data sets for evaluating algorithm *PF*. **V-COCO** dataset is a collection of 10346 frame sequences that includes 23 distinct human actions. To be able to control the interactions in the videos, we utilize **V-COCO** to generate video sequences using the **Coral** and **UCSD Pedestrian Dataset** injecting interactions from **V-COCO**. In order to maintain the consistency of our models, we annotate the four data sets using Mask R-CNN. The **Coral** data set is an 80 hour fixed-angle video sequence shot at an aquarium. Similarly, **UCSD Pedestrian Dataset** is a 40 hour fixed-angle sequence shot at University of California San Diego (UCSD). Finally, **Detrac** consists of 10 hours of fixed-angle traffic videos shot at various locations in China.

Dataset	Train Size	Test Size	Obj/Frame	std	Classes
Coral	52000	10000	8.7	5.1	Person
Pedestrian	26845	10000	2.2	2.1	Person
Detrac	55020	10000	15.8	9.8	car (92%) bus (6%) Truck (2%)
V-COCO	5400	4946	8.8	7.95	COCO classes

Table 4: Datasets and their characteristics

To evaluate our model, we partition the video sequences to create a training, validation, and test sets for each data set. The **Detrac** data set contains 60 and 40 different sequences for training and testing respectively. **UCSD Pedestrian Dataset** contains 171 sequences in total. For the purposes of our experiments, we combine the train and test set of the original data set and partition the ordered frames into train, validation, and test set with equal ratios between sequences. Table 4 presents a description and key characteristics of each data set.

With our experiments we seek to quantify the accuracy of using multiple filters in algorithm *PF* compared to baselines. We also evaluated the impact of the dynamic statistical test for re-optimization in the total execution time. Finally we evaluate the impact of our proposed *Interaction Sheave* (IS) to filter irrelevant frames. In all cases we evaluate both performance but also accuracy of all techniques compared to ground truth.

Progressive filters utilize VGG architecture [] as a template. We implemented six filters with different number of layers for prediction. Since each filter has different complexity the time required to pass a frame through the filter varies. Table 5 presents each filter's processing throughput in frames

Filters	Architecture	FPS
1	4 Convolutional layers, 2 fully-connected	1012
2	9 Convolutional layers, 2 fully-connected	839
3	14 Convolutional layers, Dropout, 2 fully-connected	621
4	19 Convolutional layers, Dropout, 2 fully-connected	470
5	22 Convolutional layers, Dropout, 2 fully-connected	333
6	25 Convolutional layers, Dropout, 2 fully-connected	276

Table 5: Filters performance

per seconds and its architecture. Additionally every Convolutional layer is using batch normalization [] and Leaky Relu. A Sigmoid function is used for our predictions. For the **Coral** and **UCSD Pedestrian** data sets 15 epochs of training were required to achieve optimal performance. In our environment, the training for the 6 filters takes approximately 7 hours for **Coral** and 4 hours for **UCSD Pedestrian**. The more complex **Detrac** dataset with three classes requires 20 epochs to converge. For Progressive filters in all four data sets, we use SGD optimizer with learning rate 10^{-4}, exponential decay of 5×10^{-4}, momentum of 0.9 and stop training when the performance on the validation set begins to drop. We implement our models with the PyTorch framework and perform all experiments on two Nvidia Titan XP GPUs using an HP desktop with an Intel Xeon Processor E5-2650 v4, and 128GB of memory.

6.1 Filter Configuration

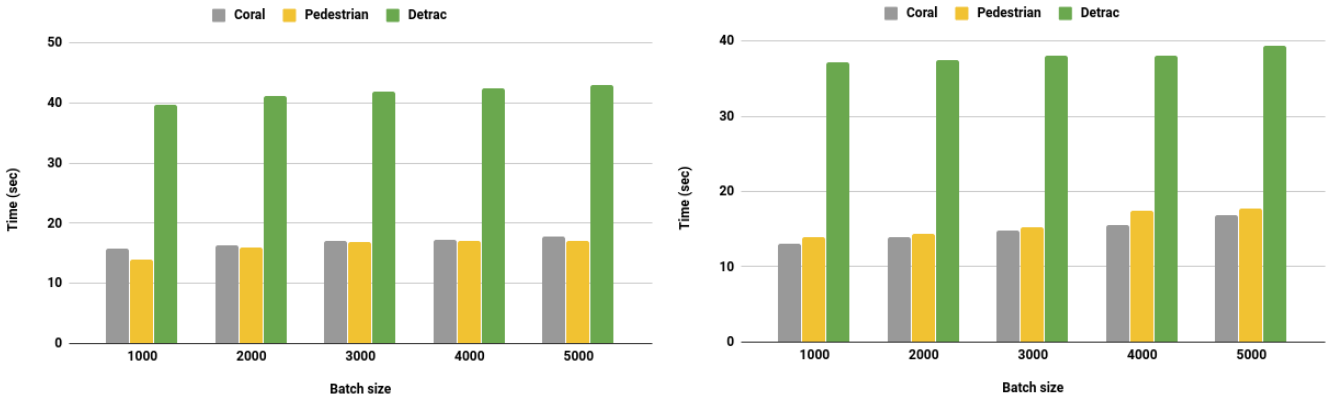
We discuss the cut-off thresholds for Progressive filters. Each filter has two cut-off thresholds. Frames for which the filter has confidence higher than the upper threshold are predicted directly as frames that contain the object of interest and subsequently become the input of the next operator if one exists. Similarly, frames for which the filter has lower confidence than the lower cut-off boundary are predicted as frames that do not include the object of interest and they are dropped from further processing. The rest of the frames that do not belong in the first two cases are used as input on the next applicable filter. We define the thresholds based on true positive and true negative rate. Both for the upper and the lower thresholds for each filter, are selected such that we have at least 90% true positive rate and 80% true negative rate, respectively.

6.2 Progressive Filters Accuracy

We now present an evaluation of *PF*. Figure 4 depicts the execution time of *PF* on *Coral*, *UCSD Pedestrian* and *Detrac* data set for the case of 6 filters (Figure 4a) and 4 filters (Figure 4b. We present the time to execute algorithm *PF* over each video sequence varying the number of frames (batch size) we consider when assessing the execution of *PF*. For this experiment, to decide when to invoke algorithm *PF* across two different batches we utilize the traditional KS test [] comparing whether the selectivities of filters have similar statistical properties across the two batches. For example for a batch size of 1000 in Figure 4a we will accumulate statistics over 1000 frames to estimate selectivity and evaluate the KS test on the selectivities of the filters between two adjacent batches of size 1000 frames. Re-optimization via algorithm *PF* is triggered based on the outcome of the KS test. The time reported is the total time to run the video sequence over all frames, accumulate statistics and assess the KS test along with the time to re-optimize if required.

From Figure 4 we can observe that the *UCSD pedestrian* Dataset requires the lowest amount of time to process with *PF*, followed by *Coral* and *Detrac*. The behaviour is consistent across different batch sizes. The reason behind this, lies in the number of objects per frame and associated variance of objects per frame for each data set. *UCSD Pedestrian* data set consists of a single class (human) and small number of objects per frame with low variance, and presents a relatively easier detection problem for filters. Thus many frames are classified accurately in the first filters in the sequence with high confidence and more expensive filters are not involved. *Coral* Dataset presents a more challenging detection problem with a single class (human) and larger average number of objects per frame (and variance) followed by *Detrac*. We can observe that the time required to process each video sequence increases with the "complexity" (larger average number of objects per frame and associated variance) of each video stream. Also *Detrac* has three classes of objects (in this experiment we are only searching for class Bus) and that increases the level of detail for the filters involved in the classification. For this reason is the most challenging to process. For comparison purposes, running the entire *Coral*, *UCSD Pedestrian* and *Detrac* data sets over a state of the art object detection model [] requires X,Y,Z time respectively. This presents savings of XXX, YYY and ZZZ. **to fill in** Comparing Figures 4a and 4b we observe that as the number of filters increases, the time required to process the sequences increases as well. A closer inspection of the filters that are selected when a larger number of filters is involved, reveals that **need an explanation for this not clear why**

In figure **??**, we can observe the F1 score of the application of *PF* for each data set for four (Figure **??**) and six filters (Figure 5). In the figures we present an F1 of 100% for comparison (labelled Full Model, to denote the F1 of a single expensive but accurate model that has ideal precision and recall). **we can remove the red bar contains no info, just let the graphs go up to 100, combine both in a single graphs**. The F1 score is not expected to vary across different batch sizes (only minor variation due to small differences in the cut-off values of each filter as each batch may utilize



(a) Applying 6 filters

(b) Applying 4 filters

Figure 4: Progressive filters performance across data sets

different filter combinations). Thus, we calculate the F1 score of each data set as the average F1 across different batch sizes. We also report the variance in the F1 score for each data set. *Detrac* data set has the lowest F1 score in comparison to *Coral* and *Pedestrian* data sets. Since *Detrac* has the highest average number of objects per frame, it is harder to do accurate predictions (we are searching for class Bus in our experiments). We observe the importance in the F1 score, of the number of objects per frame on *Coral* and *Pedestrian* data sets. *Pedestrian* data set has 5% less objects per frame than *Coral*, as we can observe on Table 4. This difference results on 2.1% higher F1 score for six filters and 4.3% for operators with four filters. Furthermore, operators with six filters depict higher F1 score. Operators with six filters, include more complex and deeper networks that can attain better accuracy during classification resulting in a higher F1 score. We observe at least 10% higher F1 score in operators with six filters versus four. Moreover, operators with four filters tend to have higher variance in their f1 score; that means the F1 score computed across different batch sizes exhibit higher variability for operators with four versus six filters.

We next evaluate the utility of dynamic KS test on the performance of *PF*. The smaller the batch size of the frames on which we compute selectivities of filters the more visibility we get on the underlying statistical properties of the object distribution of the frames. So a small batch is desirable. At the same time the smaller the batch the more frequent is the evaluation of the statistical properties of the selectivities using the KS test and as a result we expect a more frequent invocation of algorithm *PF* both of which (KS test and the *PF* algorithm) have their run time overheads. In the next experiment we compare the impact of running the dynamic version of the KS test of section 4 to the run time of *PF* over

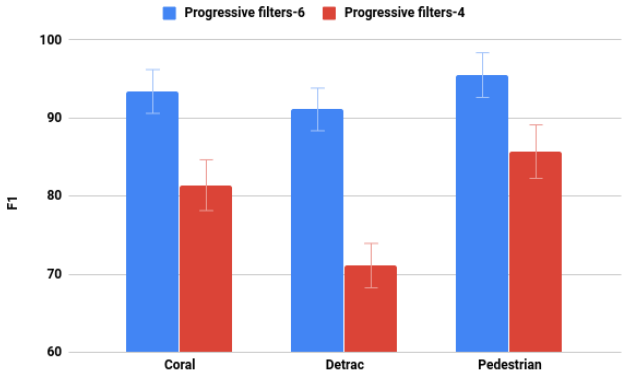


Figure 5: Applying 6 filter

video streams for varying batch sizes. Figure 6 **we need percentages in the figure** presents the results. In particular we present for batches sizes the time required to process the batch size and run the dynamic KS test along with possible application of algorithm *PF* to re-optimize the filters, as a percentage of the time to process the batch size and run the KS test along with the possible application of algorithm *PF*. It is evident that across data sets, the smaller the batch size the larger the time savings of applying the dynamic KS test. Thus the dynamic KS test enables us to monitor much smaller batch sizes and its performance advantages are evident the smaller the batch size is.

6.3 Interaction Sheave Evaluation

We now evaluate the Interaction Sheave (IS) performance. We focus on three specific actions, namely human hits tennis ball, human throws baseball and human talks on the phone.

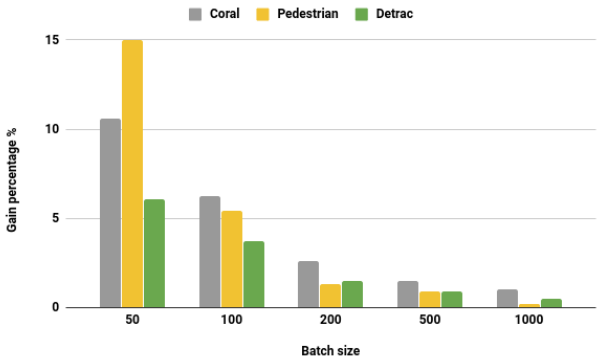


Figure 6: Dynamic KS test Evaluation

	Full model	Interaction Sheave
Frames per second	4	55
F1 on actions	0.952	0.937
F1 w/o actions	0.803	0.86

Table 6: Interaction sheave performance

The actions are acquired from V-COCO dataset. We also gathered additional data that have both a human and the object of interest (tennis ball, baseball and phone) but an action between them, didn't exist. Table 6 presents the performance of the IS algorithm when compared with the approach that passes each frame through an expensive action detection model [] (labelled Full Model in the table), along with the associated F1 score for frames containing actions as well as frames without actions. We observe that applying the IS filter we can achieve a processing rate of 55 frames per second compared to applying [] on each frame which achieves 4 a rate of 4 frames per second. The IS filter utilizes fast object detection models [] and predicts the area of the action, testing for the presence of the target of the action in the predicted area. For this reason, it can filter out irrelevant frames and pass frames to a more expensive model for which enough confidence exists that they contain the desired action. That is why we can achieve 92% higher frame per second rate. The application of [] on each frame outperforms our algorithm only slightly in terms of F1 score. Finally IS outperforms [] on F1 score on images that include human and object but no action between them. This is because, the action area is predicted and the target object is tested for membership in the target area. That way we can easily recognize frames for which the desired action is not present. Action detection models [] typically train on positive examples only and that justifies in F1 score difference.

	Coral	Pedestrian	Full model
Time (sec)	15.76	14.07	2520
F1 score	0.933	0.953	1

Table 7: Hit tennisball action

	Coral	Pedestrian	Full model
Time (sec)	15.73	14.05	2470
F1 score	0.938	0.954	1

Table 8: Throw baseball action

6.4 Query Results

We now present an evaluation of the effectiveness of all the techniques presented so far when processing a video stream for specific interaction queries. We devise data sets utilizing the *Coral* and *Pedestrian* data sets injecting frames containing the suitable interactions. In particular we inject frame sequences with the hit tennis ball and throw baseball interactions. Tables 7 and 8 present the query results. For query evaluation we first deploy an operator with six filters to detect the suitable object (since we expect less frames will have the suitable object such as tennis ball and baseball), next operators with six filters to detect humans and finally the interaction sheave filter. Finally if frames successfully are evaluated by both operators detecting the suitable objects and pass the IS filter, we evaluate the full interaction model []. As is evident from the tables, the performance advantages offered by the filters proposed are very large. In particular we observe a query speedup of two orders of magnitude. At the same time we also observe that queries attain highly competitive F1 scores with attests that the overall approach is highly accurate.

7 CONCLUSIONS

We have presented a series of filters to estimate the presence of humans in a frame, the presence of objects in a frame as well as to estimate the target area of a specific action. These filters were evaluated for F1 score and we experimentally demonstrated using real video data sets that they attain good F1 score for classification and action target estimation purposes. When applied in query scenarios over video streams, we demonstrated that they achieve dramatic speedups by several orders of magnitude.

This work opens numerous avenues for further study. Declarative query languages and query processors for video streams is largely an open research area. Study of additional query types involving spatial and temporal predicates is a natural extension. Finally extension of the filters for crowd counting and estimation scenarios is also necessary.

REFERENCES

[1] Brian Babcock, Shivnath Babu, Rajeev Motwani, and Mayur Datar. 2003. Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*. ACM, New York, NY, USA, 253–264. https://doi.org/10.1145/872757.872789

[2] A. B. Chan and N. Vasconcelos. 2008. Modeling, Clustering, and Segmenting Video with Mixtures of Dynamic Textures. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 5 (May 2008), 909–926. https://doi.org/10.1109/TPAMI.2007.70738

[3] Surajit Chaudhuri and Kyuseok Shim. 1999. Optimization of Queries with User-defined Predicates. *ACM Trans. Database Syst.* 24, 2 (June 1999), 177–228. https://doi.org/10.1145/320248.320249

[4] Erik D. Demaine, Gad M. Landau, and Oren Weimann. 2014. On Cartesian Trees and Range Minimum Queries. *Algorithmica* 68, 3 (2014), 610–625. https://doi.org/10.1007/s00453-012-9683-x A preliminary version of this paper appeared in ICALP [15].

[5] Denis Moreira dos Reis, Peter Flach, Stan Matwin, and Gustavo Batista. 2016. Fast Unsupervised Online Drift Detection Using Incremental Kolmogorov-Smirnov Test. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 1545–1554. https://doi.org/10.1145/2939672.2939836

[6] Ross B. Girshick. 2015. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. 1440–1448. https://doi.org/10.1109/ICCV.2015.169

[7] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2013. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR* abs/1311.2524 (2013). arXiv:1311.2524 http://arxiv.org/abs/1311.2524

[8] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. 580–587. https://doi.org/10.1109/CVPR.2014.81

[9] Georgia Gkioxari, Ross B. Girshick, Piotr Dollár, and Kaiming He. 2017. Detecting and Recognizing Human-Object Interactions. *CoRR* abs/1704.07333 (2017). arXiv:1704.07333 http://arxiv.org/abs/1704.07333

[10] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org/

[11] Saurabh Gupta and Jitendra Malik. 2015. Visual Semantic Role Labeling. (05 2015).

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. 2017. Mask R-CNN. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. 2980–2988. https://doi.org/10.1109/ICCV.2017.322

[13] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. 2018. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 269–286. https://www.usenix.org/conference/osdi18/presentation/hsieh

[14] Frank J. Massey Jr. 1951. The Kolmogorov-Smirnov Test for Goodness of Fit. *J. Amer. Statist. Assoc.* 46, 253 (1951), 68–78. https://doi.org/10.1080/01621459.1951.10500769 arXiv:https://www.tandfonline.com/doi/pdf/10.1080/01621459.1951.10500769

[15] D. Kang, P. Bailis, and M. Zaharia. [n. d.]. BlazeIT: Fast Exploratory Video Queries Using Neural Networks. In *https://arxiv.org/abs/1805.01046*.

[16] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Challenges and Opportunities in DNN-Based Video Analytics: A Demonstration of the BlazeIt Video Query Engine. In *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. http://cidrdb.org/cidr2019/papers/p141-kang-cidr19.pdf

[17] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1586–1597. https://doi.org/10.14778/3137628.3137664

[18] Alexander Kolesnikov, Christoph H. Lampert, and Vittorio Ferrari. 2018. Detecting Visual Relationships Using Box Attention. *CoRR* abs/1807.02136 (2018). arXiv:1807.02136 http://arxiv.org/abs/1807.02136

[19] Nick Koudas, Raymond Li, and Ioannis Xarchakos. 2020. Video Monitoring Queries. In *Proceedings of IEEE ICDE*.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (May 2017), 84–90. https://doi.org/10.1145/3065386

[21] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* 1, 4 (Dec 1989), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

[22] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, New York, NY, USA, 1493–1508. https://doi.org/10.1145/3183713.3183751

[23] Chih-Yao Ma, Asim Kadav, Iain Melvin, Zsolt Kira, Ghassan AlRegib, and Hans Peter Graf. 2017. Grounded Objects and Interactions for Video Captioning. *CoRR* abs/1711.06354 (2017). arXiv:1711.06354 http://arxiv.org/abs/1711.06354

[24] Vinod Nair and Geoffrey Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair. *Proceedings of ICML* 27, 807–814.

[25] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[26] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: Better, Faster, Stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. 6517–6525. https://doi.org/10.1109/CVPR.2017.690

[27] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *CoRR* abs/1804.02767 (2018). arXiv:1804.02767 http://arxiv.org/abs/1804.02767

[28] R. Seidel and C. R. Aragon. 1996. Randomized search trees. *Algorithmica* 16, 4 (01 Oct 1996), 464–497. https://doi.org/10.1007/BF01940876

[29] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556 http://arxiv.org/abs/1409.1556

[30] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556* (09 2014).

[31] Alexandros Stergiou and Ronald Poppe. 2018. Understanding human-human interactions: a survey. *CoRR* abs/1808.00022 (2018). arXiv:1808.00022 http://arxiv.org/abs/1808.00022

[32] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2017. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks. *CoRR* abs/1709.01686 (2017). arXiv:1709.01686 http://arxiv.org/abs/1709.01686

[33] Oytun Ulutan, Swati Rallapalli, Mudhakar Srivatsa, and B. S. Manjunath. 2018. Actor Conditioned Attention Maps for Video Action

Detection. *CoRR* abs/1812.11631 (2018). arXiv:1812.11631 http://arxiv.org/abs/1812.11631

[34] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. 2015. DETRAC: A New Benchmark and Protocol for Multi-Object Tracking. *CoRR* abs/1511.04136 (2015). arXiv:1511.04136 http://arxiv.org/abs/1511.04136

[35] Ioannis Xarchakos and Nick Koudas. 2019. SVQ: Streaming Video Queries. In *Proceedings of ACM SIGMOD, Demo Track*.

[36] Yubo Zhang, Pavel Tokmakov, Cordelia Schmid, and Martial Hebert. 2018. A Structured Model For Action Detection. *CoRR* abs/1812.03544 (2018).