Querying For Interactions

Ioannis Xarchakos
University of Toronto
xarchakos@cs.toronto.edu

Nick Koudas
University of Toronto
koudas@cs.toronto.edu

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.

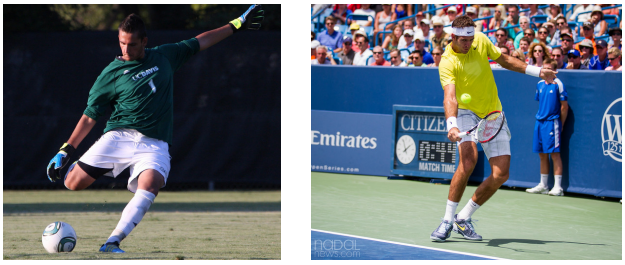
The techniques we propose constitute a robust approach to process video streams for query specified object interactions achieving very high frame processing rate. We present the results of a thorough experimental evaluation involving real data sets, demonstrating the performance benefits of each of our proposals. In particular we experimentally demonstrate that our techniques can improve query performance substantially (up to two orders of magnitude in our experiments) while maintaining essentially the same F1-score as alternatives.

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

PVLDB Reference Format:
. Querying For Interactions. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

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 [29, 14, 12, 45]. 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. According to the Wall Street Journal by 2021 there will be one billion cameras deployed on streets worldwide [21]. Moreover, 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. 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 [25, 23, 24, 52, 28] 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 glass window 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 an 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., [10, 12]), determining efficiently which frames contain a specific interaction among the objects, necessitates the application of specialized DL models [26, 54, 49, 36, 47, 14] 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 [10, 45]. The drawback is that although such models have high accuracy they impose large overheads in terms of processing time per frame [10]. Prior work [25, 52] 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 [22]. 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.

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 [26, 54, 49, 36, 47, 14]. 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

[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.

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 [7] (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 [2] (utilizing learned selectivities).

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 neural 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 to 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 improve frame processing time by orders of magnitude while maintaining high accuracy.

[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.

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 of declarative query processing over video streams [25, 24, 23]. [25] initiates the work on surveillance video query processing. The authors address fast query processing on frame classification queries, by identifying frames that contain specific classes of objects involved in the query. They train deep classifiers to recognize only specific objects, thus being able to utilize smaller and faster networks. They present significantly improved query processing rate with a small loss in accuracy when filtering for specific query objects. 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 maximising frame processing rate. Subsequently [23, 24], present query processing techniques for processing specific type of aggregates over the video stream. They also propose techniques to process aggregation queries across frames over objects with associated spatial constraints. On a related thread [52, 28] present a declarative query processing framework to support queries with spatial constraints among objects on video streams. [52] provides a system demonstration which is incorporating the techniques and concepts introduced in [28]. Other related works [35, 18] 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 [29, 42, 15, 17]. Similarly recent approaches have presented Deep Learning based models that can detect certain types of interactions between objects [26, 54, 49, 36, 47, 14]. 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 [26, 54, 49] frames that are promising to contain the desired type of action.

On a related thread, research in the computer vision community to recognize several types of actions on a video stream is fairly active. From the perspective of data type, research on action recognition can be divided into methods based on color (RGB) data and methods combining color and depth

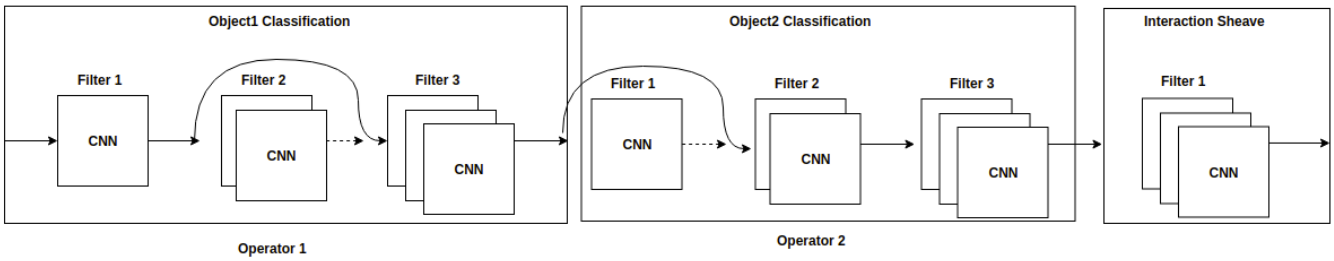


Figure 2: Example of the overall architecture

data (RGBD) [33, 32]. The human action recognition approaches for these data, following the progress of machine learning research, can be categorized as either hand-designed features with machine learning methods [50] or end-to-end deep learning algorithms.

Regardless of data type and computing method, the core aim is to extract robust human action features. Many action features have been proposed for RGB data, such as spatiotemporal volume-based features, spatiotemporal interesting point features, and joint trajectory features. However, factors such as camera movement, occlusion, complex scenes, and the limitations of human detection and pose estimation methods limit the performance of human action representation and recognition based on handcrafted features. Modern methods utilize deep-learning networks for action prediction. [44] proposed a two stream architecture which uses spatial and temporal networks to address the problem. They also present substantial gains by training Convolutional networks using dense optical flow. Current state of the art approaches [4] are using two stream inflated 3D Convolutional networks (I3D). Kong et. al., [27] present a survey of the state of the art in human action recognition and prediction.

Action recognition can be divided into action classification and action detection. Action classification is the analysis of a segmented video containing only a single action that must be classified into a defined action category. Action detection detects the start and end times of each action in the video, locates their position in space, and identifies the action category. In early research, the study of human action recognition focused on the action classification problem. With the development of related research topics, such as machine learning, object detection, and human pose estimation, research on the more challenging human action detection has become popular in recent years. Zhang et. al., [53] present a detailed survey of action recognition techniques in the vision community. Although largely still an active research area, advanced techniques make use of deep network models, action flow [40] as well as spatial information from the frames [31] to assess actions. Current state of the art action recognition research can be broadly categorized into the following four classes of action semantics [1] from low to high: primitive action recognition [5], single-person action recognition [34], interaction recognition [37], and group action recognition [19].

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 [10]. Depending on the model, the number of layers may vary (for example, VGG [45] models have been proposed with 16, 19 or more layers). It has been observed [15] 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 [48, 25] 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 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 is not independent; consider the case, for example, of the camera monitoring pedestrians passing a store front. One filter may stop a frame from further processing (e.g., when its confidence that no human is present 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 a filter) and the associated filter costs (time to process a frame).

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 not 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 $\text{Cost}(f_1, \ldots f_n) = c_1 + \sum_{i=2}^{n} \sigma_{i-1} c_i$. 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 [45] for the case of object recognition). Without loss of generality we assume f_0 is a filter with $c_o = 0$ and $\sigma_0 = 1$. 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 in the solution 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 milli-second and optimization should impose minimal overhead.

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 with the indices of the filters involved in the optimal solution. 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, (as ultimately a frame may propagate in all filters and we have to rely on the most accurate model to make a final determination) consequently its value is zero on the eliminate array. Next, the algorithm decides which filters should be eliminated (lines 14-19). The (expected) cost of the current filter is compared with the (expected) 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 in expectation. Subsequently the solution is extracted.

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
```

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. 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.

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

Table 2: Overall Cost array

	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 [22] 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 KS between Σ_t and $\Sigma_{t'}$.

In order to address this, we utilize Cartesian Trees [43, 8] and adapt [9] 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 [43]. 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 [43]. 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 [43] 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 [43, 8].

Thus, by utilizing Cartesian trees we can transform the $O(n \log n)$ traditional KS test into an $O(1)$ operation in our setting, with $O(\log n + n)$ tree maintenance overhead.

5. INTERACTION SHEAVE

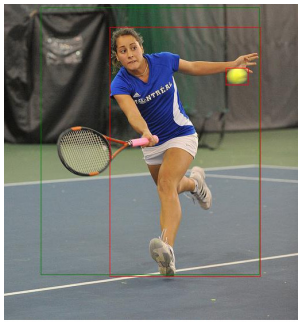
For frames that pass through the filter sequences determined by PF for each object specified by the query, we have confidence that they contain the required objects. The actual objects can be detected precisely on the frame via the application of object detection models [10]. These models will derive 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 [26, 54, 49, 49, 36, 47, 14] employ deep neural networks [10] 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 and 4 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. Thus, given training data of objects and their interactions of specific types with other objects, we can learn a model that *predicts* given a specific object and the interaction type, the spatial area in the frame that the other object (target of the interaction) is located.

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 or other) 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.

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 [41]. 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

(a)



(b)

Figure 3: Examples of interactions

(a)

(b)

Figure 4: Examples of interactions

object τ, following [11] 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 [10] 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.

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	7.9	COCO class

Table 4: Datasets and their characteristics

Filter	Architecture	FPS
1	4 CNN layers, 2 FC	1012
2	7 CNN layers, 2 FC	963
3	9 CNN layers, 2 FC	839
4	11 CNN layers, 2 FC	722
5	14 CNN layers, Dropout, 2 FC	621
6	18 CNN layers, Dropout, 2 FC	550
7	19 CNN layers, Dropout, 2 FC	470
8	25 CNN layers, Dropout, 2 FC	276

Table 5: Filters performance

6. EXPERIMENTAL EVALUATION

We now present the results of a detailed experimental evaluation of the proposed approaches. We utilize five different real data sets namely, **Coral**, **Detrac** [51], **UCSD Pedestrian Dataset** [6], **V-COCO** [16] which is part of COCO dataset and Kinetics [3]. 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 and **Kinetics** a collection of 650,000 video clips that covers 700 human action classes. To be able to control the interactions in the videos, we utilize **V-COCO** and **Kinetics** to generate video sequences using the **Coral** and **UCSD Pedestrian Dataset** injecting interactions from **V-COCO** and **Kinetics**.

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.

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 [46] as a template. We implemented eight 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 [3] presents the processing throughput for each filter in frames per second and its architecture. Additionally every Convolutional layer is using batch normalization [20] and Leaky Relu. A Sigmoid function is used for our predictions. For the **Coral** and **UCSD Pedestrian** data sets we conducted 15 training epochs to achieve good performance. In our environment, 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 stochastic gradient descent (SGD) optimizer with learning rate 10^{-4}, exponential decay of 5×10^{-4}, momentum of 0.9 and stopped training when the performance on the validation set begins to drop. We implement our models on 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.

To train the model predictions for IS we used synchronized SGD [30] 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 [46] on PyTorch library [39]. For the action area prediction, we are using one 1024-d fully connected layer with ReLU [38].

6.1 Filter Configuration

We discuss the cut-off thresholds for Progressive filters. Each filter has two cut-off thresholds (upper and lower) to determine how to process a frame. Frames for which the filter has confidence higher than the upper threshold are predicted directly as frames that contain the object of interest. There is no need to continue processing the frame to the next filter of the operator in that case. The frame is passed then to the next operator if one exists. Similarly, frames for which the filter has lower confidence than the lower threshold are predicted as frames that do not include the object of interest and are dropped from further processing. The rest of the frames that do not belong in the first two cases are used as input to the next applicable filter. We define the thresholds based on true positive and true negative rates. We are using a validation set for each data set in order to select the cut-off boundaries. We collect the validation set's frames using random sampling. The upper (lower) threshold for each filter, is selected such that we achieve least 90% true positive rate (80% true negative rate), respectively.

[3] We utilize operators with four filters and six filters for comparison purposes. For operators with four filters we deploy filters 1-4 and for those with six we deploy filters 1-6. In each case in the end we utilize the final (most) accurate filter, VGG-19, to make the final prediction for frames that reach that stage.

6.2 Progressive Filters Accuracy

We now present an evaluation of PF. Figure 7 depicts the execution time of PF on *Coral*, *UCSD Pedestrian* and *Detrac* data set for the case of an operator with 6 filters (Figure 7a) and 4 filters (Figure 7b). 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 the PF algorithm across two different batches, we utilize the (traditional) KS test [22] comparing whether the selectivities of filters have similar statistical properties across the two batches. For example for a batch size of 1000 in Figure 7a 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.

In Figure 7 we observe that the *UCSD pedestrian* data set 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 of the operator with high confidence and more expensive filters are not invoked. *Coral* data set 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 classification model [45] requires 22 seconds on the test data sets (10K frames). For 6 filters, this presents savings of 26% in processing time on average (across the different batch sizes) and 37% for the batch size with the fastest processing time for the *Pedestrian* data set. *Coral* data set outperforms the state of the art model by 24% on average and 29% on the best batch size while on *Detrac* data set, we achieve 26% faster processing time on average and 32% on the best case scenario. Accordingly, for 4 filters we achieve 29%, 32% and 30% on average and 36%, 40% and 35% on the best case scenario for *Pedestrian*, *Coral* and *Detrac* data sets respectively.

Comparing Figures 7a and 7b 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, reveals that when six filters are involved the processing time per frame increases in comparison to using 4 filters. Frames which are harder to predict in the case of an operator with 6 filters, are evaluated from the last filters in the sequence, which are more complex and as a result much slower than the rest of the filters. However more complex filters are much more accurate (as we present in Figure 5). The time difference on the *Pedestrian* data set between 4 and 6 filters is 5% while for the *Coral* data set is 12%. The *Coral* data set is more complex than the *Pedestrian* data set, so, more frames of the former are evaluated by filters with deeper architecture (thus more costly) on average.

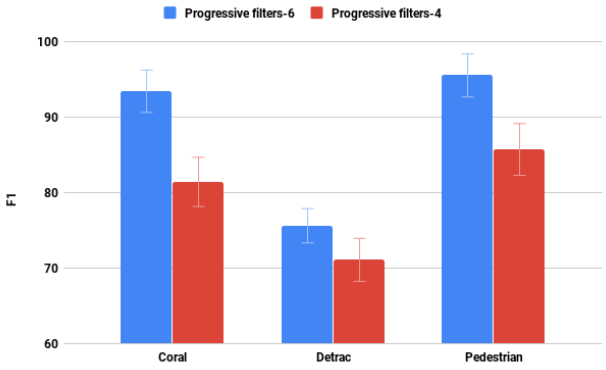


Figure 5: F1 score for 6 and 4 filters

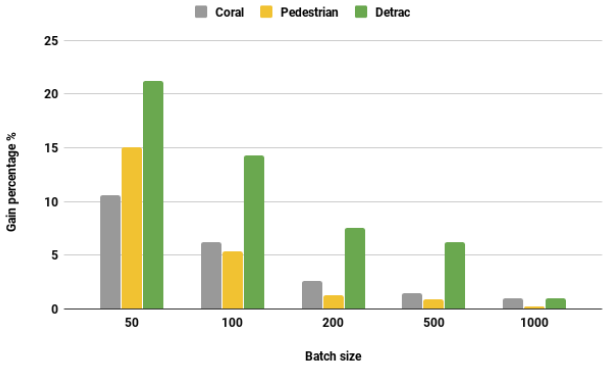
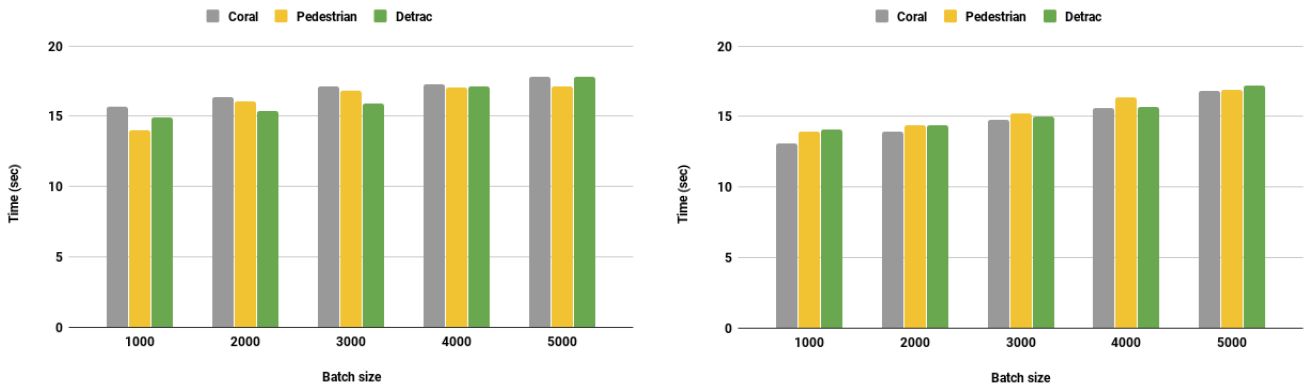


Figure 6: Dynamic KS test Evaluation

In figure 5, we can observe the F1 score of the application of PF for each data set for four and six filters. 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 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. In all cases we can see that the F1 score remains well above 70%. The difference in the average number of objects per frame on Coral and Pedestrian data sets has an impact on the associated F1 score achieved in these datasets. The 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. Such operators 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



(a) Applying 6 filters

(b) Applying 4 filters

Figure 7: Progressive filters performance across data sets

in operators with six filters versus four. Moreover, operators with four filters tend to have higher variance in their F1 score; thus the F1 score computed across different batch sizes exhibit higher variance for operators with four versus six filters. The *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 conduct predictions (in these experiments we are aiming to locate only a single class Bus).

We aim to evaluate the utility of the dynamic KS test on the performance of PF. A smaller batch size of frames on which we compute selectivities for filters provides more visibility 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 size the more frequent is the evaluation of the statistical properties of the selectivities using the KS test [22] and as a result we expect, depending on the statistical characteristics, a possible invocation of algorithm PF, in addition to the execution overhead of the KS test in every batch.

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 video streams for varying batch sizes. Figure 6 presents the results. In particular we present for different batch sizes the time required to process the batch size and run the dynamic KS test along with a possible application of algorithm PF to re-optimize the filters, as a percentage of the time to process the batch size and run the traditional KS test [22] 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. Another observation is that the performance savings appear more pronounced on the *difficult* data sets (larger number of objects per frame and larger variance). In these cases re-optimization is warranted frequently and the dynamic nature of the KS test has lower overheads, conducting few tree re-balancing operations and quickly determining that re-optimization is warranted.

6.3 Interaction Sheave Evaluation

We now evaluate the performance of Interaction Sheave (IS). In our experiments we focus on four specific actions, namely human hits tennis ball, human throws baseball, human is surfing, human is skating. Similar performance was obtained for other types of actions as well. We utilize both images with actions as well as video clips with actions in our experiments. The actions of the image data set are acquired from V-COCO data set and the actions of the video data sets from Kinetics [3]. We also gathered additional data that depict both a human and the object of interest (tennis ball, baseball, surfboard, skateboard) but the desired action between them, doesn't take place.

Table 6 presents the performance of the IS algorithm when compared with the approach that passes each frame through an expensive action detection model [26, 54, 49, 49, 36, 47, 14]. We evaluate the F1 score of action images using the V-COCO data set and we use video sequences from Kinetics data set to calculate F1 score for action videos. The tables presents F1 score averages over the four actions types considered for images and video sequences respectively. We also calculate the F1 score on images and video frames in which the desired objects are present but the specific action between the objects is not. These images and videos are acquired from publicly available data sources. In our evaluation we deploy [14] but any other model can be equally applied; we refer to it as Full Model (FM), and depict it along with the associated F1 score for frames containing actions as well as frames containing the desired objects but not the desired actions. We observe that applying the IS filter we can achieve a processing rate of 55 frames per second compared to [14] which achieves a rate of 4 frames per second. The IS filter utilizes fast object detection models [41] 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 when enough confidence exists that they contain the desired action. That is why we can achieve 92% higher frame per second rate. The application of elaborate interaction models [26, 54, 49, 49, 36, 47, 14] on each frame outperforms our algorithm only slightly in terms of F1 score on both images and videos.

Finally, the Interaction Sheave approach outperforms elaborate interaction models on F1 score both on images and

		FM	IS
	Frames per second	4	55
Images	F1 on actions	0.95	0.9
	F1 w/o actions	0.8	0.84
Videos	F1 on actions	0.94	0.88
	F1 w/o actions	0.83	0.87

Table 6: Interaction sheave performance on images and video clips with and without the desired actions being present.

	Images		Videos		
	Time(s)	F1	Time(s)	F1	
Coral (PF+IS)	27.5	0.93	37.4	0.88	
Pedestrian (PF+IS)	25.8	0.95	35.7	0.87	
Coral (PF+FM)	35.5	0.95	45.7	0.89	
Pedestrian (PF+FM)	33.8	0.97	44	0.89	
Full Model	2519	1	2530	1	

Table 7: Hit tennis ball

videos that include the desired objects but not the target action between them. This is because, the action area is predicted and the target object is tested for spatial membership in the target area. That way we can easily recognize frames for which the desired action is not present. Action detection models [26, 54, 49, 49, 36, 47, 14] typically train on positive examples only and that justifies the F1 score difference.

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. To be able to control the types as well as volume of interactions we have on a video stream, we devise data sets utilizing the *Coral* and *Pedestrian* data sets. We do so, by injecting frames containing the suitable interactions at various time instances in the *Coral* and *Pedestrian* video sequences. The injected frames are acquired from V-COCO and the Kinetics data sets, thus experimenting both with single frames depicting an interaction (from V-COCO) as well as video sequences with specific interactions (from Kinetics data set). In particular we inject frame/video sequences with the interactions of hit tennis ball (example shown in Figure 3(b)), throw baseball (example shown in Figure 3(a)), human/skateboard (example shown in Figure 4(b)) and human/surfboard (example shown in Figure 4(a)) interactions. Tables 7, 8, 9 and 10 present our query results for the case when the injected frame/video sequences constitute 1% of the total video sequence (i.e., of the original size of Coral or Pedestrian data sets). Out of this 1% of frames, approximately 20% contain negative examples (i.e., the required interaction objects are present but the corresponding action is not). For query evaluation we first deploy an operator with six filters to detect ball, skateboard, surfboard object as applicable to each interaction case (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 are successfully evaluated by both filter operators (PF) detecting the suitable objects and pass the IS filter, we evaluate the full interaction model (in our experiments we used [14]).

In the tables we demonstrate both query time and accuracy (F1 score). In order to also isolate the impact of the IS filter into query evaluation we present two query alternatives. The first (labeled PF+IS) applies the PF filter first both all query targets (a sequence of six filters per object mentioned in the query) followed by the IS filter (and subsequently by a interaction model to qualify all frames that pass all filters). The second option (labelled PF+FM) applies the PF filters but excludes the IS filter, directly testing all frames that pass the PF filters with an interaction model (we use FM to refer to the interaction model, in our case [14]). Finally we evaluate everything against the baseline, *full model*, of applying no filters, but utilizing only an interaction model and passing every frame in the video sequences through this model (labeled FM in the table).

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* in comparison with full model (FM) approach (compering PF+IS to FM approaches). Moreover, it is evident that the IS filter has a significant contribution to query evaluation, offering query performance benefits of almost two times faster, when it is enabled (compering PF+IS to PF+FM in the tables). We can achieve higher than two times faster query performance when the percentage of frames with negative examples is higher than 20%. The FM approach utilizes Mask R-CNN [17] and [13] at each frame. Although this offers advantages in accuracy it is fairly heavyweight. At the same time, we also observe that all queries attain highly competitive F1 scores compared to the FM approach, which attests that the overall methodology is highly accurate. In summary our results indicate that with a very small loss in accuracy (consistently above .85 in our experiments and typically above 0.9 in F1 score) we can achieve up to two order of magnitude query speedup. Thus, our framework offers an interesting trade-off in cases where query performance is a major concern.

With our next experiment we aim to evaluate the performance and accuracy trends as the percentage of action frames/video frames in the video sequence increase. We utilize the Coral data set and we vary the percentage of video action frames injected in the underlying video utilizing the hit tennis ball action video sequences from the Kinetic data set. Figures 8 and 9 present the time and F1 score of PF+IS, PF+FM and Full model (FM) as the total number of frames in the action video sequences injected in the video stream increase. Figure 8, presents the time of each respective approach, as the percentage of action video sequences increase from 1% to 10% of the total size of the original Coral data set. Similarly in each case 20% of the video sequence frames contain negative examples (both human and tennis ball are present but the action is not). In the Figure it is evident that time increases in proportion of the action frames injected as more action frames are detected by the PF+IS and PF+FM approaches, while the performance of FM remains unchanged as expected. We also observe that the time performance gains of PF+IS in comparison to PF+FM become higher when more action frames are used. Namely, at 5% action frames PF+FM is at least two times slower than PF+IS while at 7% and 10% is almost 3 times slower. This is mainly due to the ability of the IS filter to effectively discard negative frames. Figure 9 presents the associated

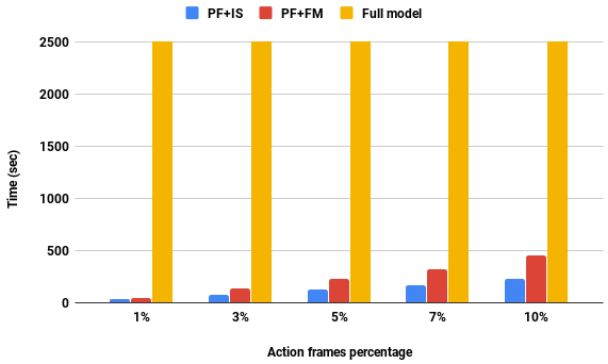


Figure 8: Time Performance for Different Percentages of Action Frames. At each percentage point 80% of the frames are positive and 20% negative (i.e., the objects are present but the requested action is not).

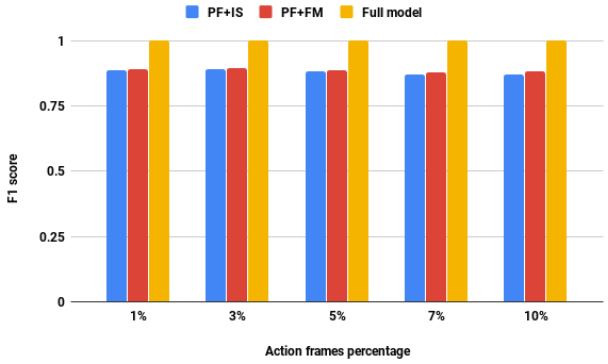


Figure 9: F1 Performance for Different Percentages of Action Frames. At each percentage point 80% of the frames are positive and 20% negative (i.e., the objects are present but the requested action is not).

F1 score. It is evident that F1 score remains largely unchanged across the different approaches as expected. As the fraction of actions in the underlying video stream increases, the filters are successful to detect the suitable proportion of actions and keep the F1 score the same (modulo negligible statistical variations). Results for the remaining data sets and action types are consistent with those presented in Figures 8 and 9 and are omitted for brevity.

7. CONCLUSIONS

We have considered the problem of efficiently executing queries on video streams involving certain types of interactions between object types detected in video frames. We presented an algorithm called *PF* that utilizes multiple filters to facilitate fast detection of objects in video streams. The algorithm can adjust the number of filters utilized by an operator adjusting to stream object selectivity. We also presented a dynamic version of the KS test to facilitate the choice to apply algorithm *PF* in a dynamic fashion. Finally we presented a filter called IS that can efficiently prune frames that are not promising in containing the desired interaction specified in the query thus increasing the frame processing rate further.

Via a detailed experimental study utilizing real data sets, we demonstrated that our proposals when combined in a query execution setting can deliver up to two orders of magnitude performance advantage with a very small loss in F1 accuracy. We believe that the set of techniques presented herein constitute important building blocks in the design of a video query processor.

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. In addition it is important to study a dynamic framework to determine, given a query the optimal number of filters to instantiate for each operator. Although currently, we statically instantiate the number of filters, it is possible to extend our work and develop dynamic approaches that adjust the number of filters per operator as the underlying properties of the video stream change.

	Images		Videos	
	Time(s)	F1	Time(s)	F1
Coral (PF+IS)	21.9	0.94	37.4	0.85
Pedestrian (PF+IS)	20.3	0.95	35.7	0.86
Coral (PF+FM)	29.2	0.96	45.7	0.87
Pedestrian (PF+FM)	27.5	0.98	44	0.89
Full Model	2513	1	2530	1

Table 8: Throw baseball

	Images		Videos	
	Time(s)	F1	Time(s)	F1
Coral (PF+IS)	42.4	0.92	37.4	0.93
Pedestrian (PF+IS)	40.7	0.93	35.7	0.94
Coral (PF+FM)	67.7	0.94	45.7	0.95
Pedestrian (PF+FM)	66	0.96	44	0.95
Full Model	2537	1	2530	1

Table 9: Surfing surfboard

8. REFERENCES

[1] J. Aggarwal and M. Ryoo. Human activity analysis: A review. *ACM Comput. Surv.*, 43(3):16:1–16:43, Apr. 2011.

[2] B. Babcock, S. Babu, R. Motwani, and M. Datar. 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, pages 253–264, New York, NY, USA, 2003. ACM.

	Images		Videos	
	Time(s)	F1	Time(s)	F1
Coral (PF+IS)	38	0.93	37.4	0.93
Pedestrian (PF+IS)	36.3	0.95	35.7	0.95
Coral (PF+FM)	47.2	0.94	45.7	0.94
Pedestrian (PF+FM)	45.5	0.95	44	0.96
Full Model	2532	1	2530	1

Table 10: Skating skateboard

[3] J. Carreira, E. Noland, C. Hillier, and A. Zisserman. A short note on the kinetics-700 human action dataset. *ArXiv*, 07 2019.

[4] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. pages 4724–4733, 07 2017.

[5] B. K. Chakraborty, D. Sarma, M. K. Bhuyan, and K. F. MacDorman. Review of constraints on vision-based gesture recognition for human–computer interaction. *IET Computer Vision*, 12(1):3–15, 2018.

[6] A. B. Chan and N. Vasconcelos. Modeling, clustering, and segmenting video with mixtures of dynamic textures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):909–926, May 2008.

[7] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. *ACM Trans. Database Syst.*, 24(2):177–228, June 1999.

[8] E. D. Demaine, G. M. Landau, and O. Weimann. On cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014. A preliminary version of this paper appeared in ICALP [15].

[9] D. M. dos Reis, P. Flach, S. Matwin, and G. Batista. 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, pages 1545–1554, New York, NY, USA, 2016. ACM.

[10] R. B. Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1440–1448, 2015.

[11] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[12] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. 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*, pages 580–587, 2014.

[13] G. Gkioxari, R. Girshick, P. Dollár, and K. He. Detecting and recognizing human-object interactions. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 04 2017.

[14] G. Gkioxari, R. B. Girshick, P. Dollár, and K. He. Detecting and recognizing human-object interactions. *CoRR*, abs/1704.07333, 2017.

[15] I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[16] S. Gupta and J. Malik. Visual semantic role labeling. *arXiv preprint arXiv:1505.04474*, 05 2015.

[17] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2980–2988, 2017.

[18] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. 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, Oct. 2018. USENIX Association.

[19] M. S. Ibrahim, S. Muralidharan, Z. Deng, A. Vahdat, and G. Mori. Hierarchical deep temporal models for group activity recognition. *CoRR*, abs/1607.02643, 2016.

[20] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[21] W. S. Journal. https://www.wsj.com/articles/a-billion-surveillance-cameras-forecast-to-be-watching-within-two-years-11575565402?mod=searchresultspage=1pos=6. 2019.

[22] F. J. M. Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[23] D. Kang, P. Bailis, and M. Zaharia. Blazeit: Fast exploratory video queries using neural networks. In *https://arxiv.org/abs/1805.01046*, 2018.

[24] D. Kang, P. Bailis, and M. Zaharia. 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*, 2019.

[25] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. Noscope: Optimizing neural network queries over video at scale. *Proc. VLDB Endow.*, 10(11):1586–1597, Aug. 2017.

[26] A. Kolesnikov, C. H. Lampert, and V. Ferrari. Detecting visual relationships using box attention. *CoRR*, abs/1807.02136, 2018.

[27] Y. Kong and Y. Fu. Human action recognition and prediction: A survey, 2018.

[28] N. Koudas, R. Li, and I. Xarchakos. Video monitoring queries. In *Proceedings of IEEE ICDE*, 2020.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[30] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Dec 1989.

[31] C. Li, Q. Zhong, D. Xie, and S. Pu. Collaborative spatio-temporal feature learning for video action recognition. *CoRR*, abs/1903.01197, 2019.

[32] A. Liu, Y. Su, W. Nie, and M. Kankanhalli. Hierarchical clustering multi-task learning for joint human action grouping and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(1):102–114, Jan 2017.

[33] A. Liu, N. Xu, W. Nie, Y. Su, Y. Wong, and M. Kankanhalli. Benchmarking a multimodal and multiview and interactive dataset for human action recognition. *IEEE Transactions on Cybernetics*, 47(7):1781–1794, July 2017.

[34] L. Lo Presti and M. La Cascia. 3d skeleton-based human action classification. *Pattern Recogn.*, 53(C):130–147, May 2016.

[35] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri. Accelerating machine learning inference with probabilistic predicates. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1493–1508, New York, NY, USA, 2018. ACM.

[36] C. Ma, A. Kadav, I. Melvin, Z. Kira, G. AlRegib, and H. P. Graf. Grounded objects and interactions for video captioning. *CoRR*, abs/1711.06354, 2017.

[37] M. Meng, H. Drira, and J. Boonaert. Distances evolution analysis for online and off-line human object interaction recognition. *Image Vision Comput.*, 70:32–45, 2018.

[38] V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, volume 27, pages 807–814, 06 2010.

[39] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[40] A. J. Piergiovanni and M. S. Ryoo. Representation flow for action recognition. *CoRR*, abs/1810.01455, 2018.

[41] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6517–6525, 2017.

[42] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[43] R. Seidel and C. R. Aragon. Randomized search trees. *Algorithmica*, 16(4):464–497, Oct 1996.

[44] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 568–576, Cambridge, MA, USA, 2014. MIT Press.

[45] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[46] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[47] A. Stergiou and R. Poppe. Understanding human-human interactions: a survey. *CoRR*, abs/1808.00022, 2018.

[48] S. Teerapittayanon, B. McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. *CoRR*, abs/1709.01686, 2017.

[49] O. Ulutan, S. Rallapalli, M. Srivatsa, and B. S. Manjunath. Actor conditioned attention maps for video action detection. *CoRR*, abs/1812.11631, 2018.

[50] H. Wang, M. M. Ullah, A. Kläser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. In *BMVC*, 2009.

[51] L. Wen, D. Du, Z. Cai, Z. Lei, M. Chang, H. Qi, J. Lim, M. Yang, and S. Lyu. DETRAC: A new benchmark and protocol for multi-object tracking. *CoRR*, abs/1511.04136, 2015.

[52] I. Xarchakos and N. Koudas. Svq: Streaming video queries. In *Proceedings of ACM SIGMOD, Demo Track*, 2019.

[53] H. Zhang, Y. Zhang, B. Zhong, Q. Lei, L. Yang, J. Du, and D. Chen. A comprehensive survey of vision-based human action recognition methods. *Sensors*, 19(5):1005, 2019.

[54] Y. Zhang, P. Tokmakov, C. Schmid, and M. Hebert. A structured model for action detection. *CoRR*, abs/1812.03544, 2018.