FFS-VA: A Fast Filtering System for Large-scale Video Analytics

Chen Zhang
KLISS, WNLO, HUST
Wuhan, China
hust_zchen@hust.edu.cn

Qiang Cao*
KLISS, WNLO, HUST
Wuhan, China
caoqiang@hust.edu.cn

Hong Jiang
University of Texas at Arlington
Arlington, Texas USA
hong.jiang@uta.edu

Wenhui Zhang
KLISS, WNLO, HUST
Wuhan, China
zhangwenhui@hust.edu.cn

Jingjun Li
KLISS, WNLO, HUST
Wuhan, China
jingjunli@hust.edu.cn

Jie Yao
KLISS, WNLO, HUST
Wuhan, China
jackyao@hust.edu.cn

ABSTRACT

Surveillance video cameras are ubiquitous around us. Full-feature object-detection models such as YOLOv2 can automatically analyze surveillance videos in real-time with high accuracy while consuming huge computational resources. Directly applying these models for practical scenarios with large-scale deployed cameras requires prohibitively expensive computation. This, however, is both wasteful and unnecessary considering the fact that the concerned anomalous events occur rarely among these massive volumes of video streams. Therefore, in this paper, we propose a Fast Filtering System for Video Analytics (FFS-VA), a pipelined multi-stage video analyzing system, to make video analytics much cost-effective. FFS-VA is designed to filter out vast but non-target-object frames by two prepositive stream-specialized filters and a small full-function tiny-YOLO model, to drastically reduce the number of video frames arriving at the full-feature model in the back-end. FFS-VA presents a global feedback-queue mechanism to balance the processing rates of different filters in both intra-stream and inter-stream processes. FFS-VA also designs a dynamic batch technique to achieve an adjustable trade-off between throughput and latency. FFS-VA reasonably distributes all tasks on CPUs and GPUs to fully exploit the underlying hardware resources. We implement a FFS-VA prototype and evaluate FFS-VA against the state-of-the-art YOLOv2 under the same hardware and representative video workloads. The experimental results show that under a 10% target-object occurrence rate on two GPUs, FFS-VA can support up to 30 concurrent video streams (7× more than YOLOv2) in the online case, and obtain 3× speedup when offline analyzing a stream, with an accuracy loss of less than 2%.

*Corresponding author. Key Laboratory of Information Storage System (KLISS), Ministry of Education. Wuhan National Laboratory for Optoelectronics (WNLO), Huazhong University of Science and Technology (HUST).

ICPP 2018, August 13–16, 2018, Eugene, OR, USA

ACM ISBN 978-1-4503-6510-9/18/08…$15.00
https://doi.org/10.1145/3225058.3225103

CCS CONCEPTS

• **Computing methodologies** → **Scene anomaly detection**;

KEYWORDS

Filters, Frames, Video analytics

ACM Reference Format:
Chen Zhang, Qiang Cao, Hong Jiang, Wenhui Zhang, Jingjun Li, Jie Yao. 2018. FFS-VA: A Fast Filtering System for Large-scale Video Analytics. In *ICPP 2018: 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3225058.3225103

1 INTRODUCTION

Increasing numbers of surveillance video cameras with low cost and high quality have been ubiquitously deployed in residential and office buildings, on street corners, and other key public areas all over a city, to monitor potential accidents, and to record critical clues. In traditional practices, the video surveillance collects live streams to an operational center for further manual observations, which is labor-intensive, error-prone and expensive. Automatic video analysis based on object detection using machine learning has been introduced recently into video surveillance with a promise to mitigate human intervention while significantly improving both the overall performance and accuracy. The use cases of automatic analysis for surveillance videos can be categorized into two main types [37]: (1) real-time analysis to detect anomalies or occasional events; and (2) post-facto analysis to look for a certain event or object retroactively.

Early automatic analysis employs support vector machine, but its accuracy and function are limited [40]. Benefitting from the recent development in complex model structures based on neural network (NN), the accuracy and speed of object detection have been significantly improved. In 2014, R-CNN [30] first achieves 53.7% mean average precision (mAP) on PASCAL VOC 2010 [22]. Afterwards, SSD [35](74.3% mAP, 59 frame per second (FPS)), R-FCN [14](83.6% mAP, 5 FPS),and YOLOv2 [16](76.8% mAP, 67 FPS) have been proposed to continuously improve both accuracy and detection speed, making real-time online analysis and high-speed offline analysis for video streams feasible in practical scenarios.

However, these existing object detection full-feature models with high-accuracy are extremely computationally hungry. For example, using YOLOv2, a powerful GPU (e.g., GTX Titan X GPU) only supports up to two concurrent video streams at 30 FPS. Considering that a typical video surveillance generally deploys hundreds of

high-quality cameras, these full-feature models are ill-equipped to perform real-time analysis for large-scale video streams directly, due to their unacceptably high hardware cost.

Fortunately, in large-scale video analysis, a typical anomalous event or object for detection occurs rarely and when it does occur it appears in a tiny fraction of all frames. For example, serious traffic jam takes place for the main routes in a big city, the average blocked time in a day is less than 5% [10]. Therefore, passing all frames over these weighty but accurate models actually wastes considerable computational capability, which is totally unnecessary. The key idea is to fast filter out most non-target-object frames while leaving the remaining frames to be precisely analyzed by the full-feature models.

In order to efficiently and effectively employ these highly accurate object detection models on limited computing devices to achieve real-time detection for large-scale high-definition video streams, we propose FFS-VA, a fast filtering system for video analytics, to dramatically reduce the number of frames actually reaching the full-feature reference model by filtering out vast amounts of non-target-object frames (e.g., frames without target objects or with a number less than a predefined threshold of target objects) to support both fast offline analysis and large-scale online analysis.

FFS-VA is a pipelined multi-stage filtering system designed for efficiently analyzing large-scale videos. Considering that most cameras in surveillance are of fixed viewpoint, we design and train a stream-specialized difference detector (SDD) to remove frames only containing background and a stream-specialized network model (SNM) to identify target-object frames. The remaining frames are further screened to filter out those whose target objects are fewer than a predefined threshold, by passing a small-scale but full-function Tiny-YOLO-Voc model (T-YOLO) that is shared by multiple streams. Finally, the surviving frames are fed into an ultimate full-feature reference model for high-precision analysis. FFS-VA is designed to run on the mainstream heterogeneous servers with several GPUs and CPUs.

To fully exploit the potentials of the underlying hardware with CPUs and GPUs, FFS-VA must address three key challenges. First, the workloads imposed on all models should be evenly distributed on CPUs and GPUs of the heterogeneous server. To this end, all SDDs are executed on the CPU, both SNMs and T-YOLO are executed on a single GPU. The full-feature reference model runs on another GPU alone. In order to achieve a high computational efficiency, adding a queue between any two consecutive stages unlocks all stages from synchronous lock steps, enabling them to be pipelined in an asynchronous manner. Second, the number of frames progressing in the four stages within a stream is gradually and proportionally decreasing due to filtering. The filter at a later stage is much slower than one in an earlier stage in processing speed. The number of frames entering each stage varies significantly due to the unpredictable video contents. To dynamically balance the loads among filters within a stream and among multiple streams, FFS-VA builds a global feedback mechanism to orchestrate the processing speed of all stages based on their respective queue controls. Third, to process each frame, its corresponding network model (e.g., SNM) and image data must be loaded from the CPU memory to the GPU memory. To reduce the data exchange frequency between the CPU and GPU memories and improve computational efficiency, a large and static batch size is intuitively better for obtaining high computational efficiency but at the cost of lengthened processing latency. FFS-VA adopts a dynamic batch mechanism to dynamically determine batch size according to current video contents in a short period of time.

Leveraging these system-level optimizations, FFS-VA is able to efficiently perform both online and offline video analytics on large-scale video streams. Experimental results show that, compared with the state-of-the-art YOLOv2 system with the same hardware environment consisting of two CPUs and two GPUs, FFS-VA supports up to 7× more concurrent video streams in online video analysis, and obtains 3× speedup in offline video analysis, with a negligible accuracy loss of less than 2%.

In summary, the key contributions of our work are:

(1) We propose a fast filtering system for video analytics (FFS-VA) for large-scale video surveillance in both online and offline scenarios.
(2) FFS-VA introduces pipelined multi-stage filters and a global feedback-queue mechanism.
(3) FFS-VA designs a dynamic batch scheme with a video-content-based batch-size adjustment to automatically tradeoff between latency and throughput.
(4) We implement a FFS-VA prototype system to compare FFS-VA with the state-of-the-art YOLOv2, which indicates that FFS-VA improves the overall throughput by up to 7× under the same hardware environment.

The rest of this paper is organized as follows. Section 2 introduces the background of video analysis and key observations that motivate this research. The design and implementation of FFS-VA are presented in Sections 3 and 4 respectively. Section 5 evaluates the performance, sensitivity of key design parameters, and limitations of FFS-VA. Prior studies most relevant to FFS-VA are reviewed in Section 6. Section 7 concludes the paper.

2 BACKGROUND AND MOTIVATION

The deployment of thousands of surveillance video cameras is expected to improve urban management, monitor and prevent potential crimes and accidents, and record clues of anomalous events. Unfortunately, an enormous challenge remains in how to timely identify a few of specific scenes in a sea of surveillance video frames at an acceptable cost, latency, and accuracy. In the early days of the practice, surveillance videos would be live streamed to an operational center for manual analysis, which can be error-prone and very expensive. With the increasing scale of surveillance, automatic analysis based on machine learning has been widely introduced into the video contents analysis.

2.1 Convolutional Neural Network Models

Since Convolutional Neural Network (CNN) [36] has great image recognition capabilities and can recognize the characteristics of the target objects with ease, they are usually used for the detection of specific targets. In what follows, we briefly introduce the design, training, and inference process of typical CNN models for video analysis.

CNN consists of a series of connected layers, including convolutional layer (CONV), fully connected layer (FC), pooling layer (POOL), and so on. The CONV layer is responsible for extracting

local features from high-resolution feature maps. The POOL layer is in charge of organizing the local features from the CONV layer and abstracting them into a low-resolution feature map. The FC layer is used to output the actual prediction based on the outcome of preceding layers. By combining several such layers in a certain order and configuring all layers with appropriate weights, a CNN model is formed. Figure 1 shows a small CNN structure.

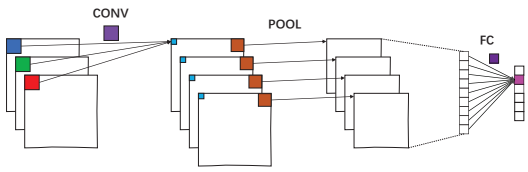


Figure 1: The structure of CNN.

To obtain a specialized CNN model on a video stream, we first label a set of frames containing the predefined target events to form a training dataset. Then, a set of new CNN architectures (configurations of all layers) are designed to inherit from the characteristics of the corresponding successful CNN models, such as AlexNet [2] and VGG-16 [18]. After that, the CNNs can automatically learn the characteristics of the target objects from the training dataset and update their weights by the stochastic gradient descent algorithm [23]. Finally, we determine the best one from these architectures as our CNN model according to accuracy and execution time.

Once the specific CNN model is determined, it can be used as further inferencing on a video stream by passing all frames in the video stream one by one. For each video frame, the CNN gives a predicted probability of whether target events occur in this frame.

2.2 Object Detection

Object detection technology has been widely used to automatically analyze video contents, such as detecting accidents [19] and traffic congestion [21], searching a certain object [11], understanding the flow of vehicles and pedestrians to provide users with the most reasonable traffic planning [8], etc., which greatly reduce the cost of manpower in video analysis.

Since 2012, CNN has been rapidly developed and deployed due to its high predictive accuracy compared with traditional methods. Thanks to these deep-learning models, current object detection has achieved great success in detection speed and accuracy. In terms of the use of spatial-temporal information on video frames, [25] and [20] correct the detection results by using relevant timing and context information, which greatly improves the accuracy of object detection. For static image detection, the accuracy of R-CNN (53.7%) [30], fast R-CNN (65.7%) [31], faster R-CNN (70.4%) [33], and R-FCN (83.6%) [14] has been continually improved, but the execution speed of these object detection techniques remains relatively low and inadequate for real-time detection. Recently, advanced methods such as YOLOv2 (67 FPS) [16] and SSD (59 FPS) [35] have been developed to achieve real-time detection, but they are computationally expensive as a powerful GPU (e.g., GTX Titan X GPU) merely supports the analysis of two concurrent video streams in real time.

2.3 Motivation

As a fundamental requirement for large-scale surveillance video analysis, users expect to know whether their concerned anomalous events occur in a timely manner. A typical anomalous event can be generally described as three questions: 1) what is the target object? 2) how many times the objects will appear? and 3) what has happened? For example, at a crossroad, more cars detected than usual (pre-defined or calculated over statistics) means a traffic jam [29].

The key goal of automatic analysis for large-scale surveillance videos is to identify concerned scenes that contain the target object occurrences and then extract their relevant information according to user requirement [26]. The applicable scenarios for video analysis can be roughly classified into two categories: offline and online. In the offline case, all stored videos need to be processed as fast as possible to capture interesting scenes. In the online case, an analytic system is expected to support more live streams while timely determining any anomaly that the users are concerned about.

Indeed, the frames without the target objects generally are not worth further analyzing and need to be filtered out. The computationally expensive full-feature models should only process the frames with the target object(s). Therefore, we define the target object ratio (*TOR*) as:

$$TOR = num_{target-object-frames}/num_{all-frames} \quad (1)$$

where $num_{all-frames}$ is the total number of all frames in the video stream during a given period of time, and $num_{target-object-frames}$ is the number of frames containing the target objects. *TOR* can help characterize the frequency of the target object frames that appear in a video slice. *TOR* is primarily determined by both video contents and filtering conditions, objectively reflecting the actual utilization of a video analytic system. For large-scale surveillance videos, *TOR* is generally low in long-period videos, which means that the anomalous events are actually infrequent for all monitored videos. According to the analysis results of numerous webcams [41], the target-object occurrence rate in a day is only 8%.

Therefore, under the actual low *TOR*, passing all frames over a full-feature model such as YOLOv2 is a huge waste of computational resources. This insight motivates us to design an effective and efficient fast filtering system to identify the frames with anomalous events from massive video frames. And then only those identified frames are worth performing over the subsequent full-feature models to further extract interesting information. Therefore, we design two prepositive stream-specialized filters and a global pipelined multi-filter architecture to achieve the aforementioned goal.

Additionally, the mainstream cost-effective servers for automatic video analysis generally consist of dual CPUs and two GPUs [5], which can analyze up to four-way streams using YOLOv2 in real-time. As an important motivation in practice, it is desirable for the same server to handle more streams. Therefore, the fast filtering system should evenly distribute all tasks on CPUs and GPUs and exploit the maximal potential of the underlying hardware to boost the overall performance under the promised latency and negligible accuracy loss.

3 DESIGN

FFS-VA is designed to perform a fast filtering for large-scale video analysis in real-time using commercially available servers. In this section, we first describe the overall structure of FFS-VA, before elaborating on the design of the three filters.

3.1 Overview of FFS-VA

Figure 2 illustrates the architectural overview of FFS-VA, which consists of three cascaded filters of (1) a specialized difference detector (SDD), (2) a specialized network model (SNM), and (3) a globally shared object detection model Tiny-YOLO-Voc (T-YOLO).

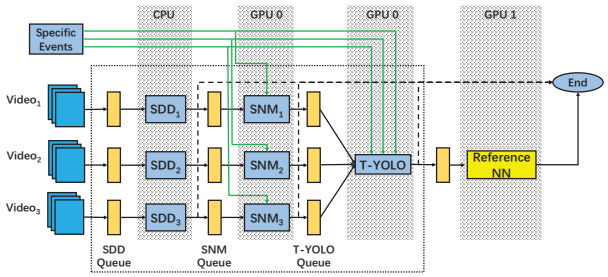


Figure 2: Architecture of FFS-VA. FFS-VA adds three filters before the reference NN to filter out frames that are not related to the user-defined events.

3.1.1 Main Functions. First, the target events to detect, such as the occurrences of cars, persons, etc., as well as their counts, are predefined by users. For each given video stream, both SDD and SNM are specialized for it and its predefined events. The two filters are then followed by a T-YOLO model that is globally shared by all streams. As a result, FFS-VA supports various target events are detected in multiple concurrent video streams.

In the FFS-VA, SDD is dedicated to filtering out background frames. SNM is used to identify the target-object frames to filter out the non-target-object frames. Afterwards, T-YOLO is employed to filter out the frames containing fewer than a threshold number of target objects. Finally, the surviving frames are input to the full-feature reference model for final, high-precision identification and analysis. The details of filters are illustrated in Subsection 3.2. For clarity and meaningful evaluation, we choose the state-of-the-art full-feature model, YOLOv2 [16], as the reference model for FFS-VA in this paper.

3.1.2 Pipelining of Functions. The three prepositive filters and final full-feature model form a four-stage pipelined architecture. Each filter is connected by its corresponding input and output queues for reading and forwarding frames. In fact, all frames of a video stream should pass its dedicated SDD, the very first filter along the pipeline. And then, the number of frames processed by each subsequent filter decreases gradually in proportion to the filtering rate of the prepositive filter. The processing speed of each filter in the pipeline also exhibits gradual decrease accordingly. Note that the input frame rate of each stage is varied with the fluctuation of video contents over time. In order to dynamically balance loads across filters, we propose a global feedback-queue mechanism, as detailed in Subsection 4.3, to coordinate the processing speed of various filters and streams.

In order to fully exploit the potential of hardware while ensuring the execution speed of these models, SDDs are executed on the CPUs, and SNMs and T-YOLO are executed on a single GPU. The powerful full-feature model uses another GPU alone. Besides, in FFS-VA, each prefetching stage and filter are associated with an independent thread. Through the parallel and pipelined structure of multiple threads, FFS-VA achieves a high analyzing throughput. Although only two GPUs are used in our prototype, FFS-VA can conveniently scale the processing capacity to expanding scenarios as a server with more GPUs or a server cluster, which is discussed in Section 4.

3.2 Detailed Design of Filters

The prepositive filters of FFS-VA are three models of small to moderate complexity that are commonly used in a cascaded manner, to classify or identify target objects as needed.

3.2.1 Specialized Difference Detector (SDD). SDD calculates the distance between the reference image and the unlabeled frame to determine whether these two frames are identical. For simplicity, the reference image is usually computed as the average of dozens of background frames. The distance between two video frames can be characterized by Mean Square Error (MSE), Normalized Root Mean Square Error (NRMSE), or Sum of Absolute Differences (SAD). Take MSE as an example, if MSE is larger than the threshold δ_{diff}, an obvious content change is construed to have occurred in the current frame. Otherwise, the frame is considered as a background frame. Note that most surveillance video cameras are deployed in a fixed viewpoint. Hence, the background frames can be safely discarded.

Naturally, the threshold δ_{diff} has a direct and critical affect on the processing efficiency and accuracy. A low δ_{diff} may result in poor filtering efficiency, while a high δ_{diff} can lead to a high error rate. Furthermore, the threshold δ_{diff} may vary greatly in different scenes. For example, a video with a mostly empty sidewalk (a static background) might have a small δ_{diff}. However, a background with changing light color and intensity in the same scene (a dynamic background) results in a larger δ_{diff}. Weather, light intensity, etc. can all contribute to the value of MSE [24], so the filtering efficiency of SDD varies greatly at different scenes.

SDD processes 100*100-pixel images at 100K FPS. In the FFS-VA, the real detection speed of SDD is 300× faster than the reference model YOLOv2, as demonstrated in Section 5.

3.2.2 Specialized Network Model (SNM). Another key filter used in FFS-VA is SNM. SNM utilizes a CNN to detect whether a video frame contains target objects. Generic models can classify and recognize thousands of object classes no matter what the scenes are, but the universality of these methods also leads to huge computational overhead and long execution time. SNM can only identify a class of predefined target objects in the specific video stream, and thus trading off reducing the universality for boosting its speed (70× real-time). In addition, for a fixed-angle camera, the position and the moving trail of the target object in the scenes are relatively fixed. Using SNM for rapid image recognition in this case can ensure the accuracy to be over 95% [7].

In fact, SNM is a three-layer CNN (CONV, CONV, and FC). The design and training process is shown in Subsection 2.1. When a video frame is inferred by SNM, SNM first outputs a predicted probability c of the target object appearing in the frame. If c is below the threshold c_{low}, no target object is considered to be in the frame. If c is higher than c_{high}, the frame is a target-object image. Otherwise, it is unsure whether the frame is target-object or non-target-object.

In our design, a threshold t_{pre} is utilized (between c_{low} and c_{high}) to help SNM distinguish target-object frames and non-target-object frames. SNM puts the target-object frames ($c \geq t_{pre}$) into the T-YOLO queue, and the non-target-object frames ($c < t_{pre}$) are filtered out.

Experiments show that SNM processes 50*50-pixel images at 5K FPS using about 200 KB GPU memory. It is 30× faster than the reference network YOLOv2 on the real hardware platform.

3.2.3 Tiny-YOLO-Voc (T-YOLO). The third filter of FFS-VA is a small object detection network, T-YOLO [44], which is used to filter out the frames where the number of target objects is less than a certain level. T-YOLO consists of 9 CONV layers and 6 POOL layers, and is trained by a VOC dataset with 20 classes. Benefiting from fewer classes and smaller model size, T-YOLO can perform at 220 FPS for 416*416 pixel images with 1.2 GB GPU memory usage.

T-YOLO can recognize multiple target objects in one image. First, T-YOLO divides the input image into a 13*13 grid cells automatically. Each grid cell predicts 5 bounding boxes and confidence scores for these boxes. If the confidence score exceeds the threshold (e.g., 0.2), one target object is considered to appear in the image. By combining the individual grid cell detections of target objects, the total number of target objects appearing in the video frame can be obtained.

As a filter shared among all video streams being processed, T-YOLO needs to traverse each T-YOLO queue of all streams one by one and extract at most num_{t-yolo} video frames from the queue for detection, skipping the stream if its queue is empty.

Here we employ a generic model identifying tens of classes for two main reasons: 1) For different video streams, sharing the same model can reduce the switch overhead of loading different models (e.g., 1.2 GB for T-YOLO). 2) We not only support to detect different target objects for different video streams, but also support the detection of multiple target objects within a video stream, to facilitate the understanding of the scene.

3.3 Accuracy

In video surveillance, users are particularly concerned about missing scenes rather than missing frames. Given a live stream with 30 FPS, in a scene, target objects could continuously appear in a series of frames. Even if just a few legible frame within this set of frames is identified, the scene is in fact correctly identified. This means that the rest of the frames in the frame set pertaining to the scene can be considered redundant or duplicate and filtered out without affecting the detection accuracy of the target scene.

Besides, it is possible for a frame to be recognized by the reference model but filtered out by its preceding filters, i.e., a false negative by the latter. The false negative events can be categorized into two cases. On one case where a merely partial appearance of target objects, i.e., partial scene in which not all target objects or incomplete target object (e.g., head of vehicle) appear, can be detected by the reference model but missed by the T-YOLO model. Nevertheless, its subsequent frames in the same scene can contain entire target objects that the T-YOLO model is able to correctly detect. In this case, the scene is considered correctly detected. On the other hand, if dozens of continuous frames containing the complete target objects are filtered out incorrectly, then the scene is considered lost. The latter case should be avoided as much as possible.

If we slightly relax the filtering condition of a filter (e.g., set the real filtering threshold slightly below the target threshold) and forward a little more frames to the follow-up filters (the latter filter can detect the results of the previous again), the false negative events could be reduced. Therefore, the cascaded structure and relaxed filtering conditions can also prevent excessive filtering errors. To quantitatively analyze the accuracy of FFS-VA, we define the error rate as the number of all false-negative frames divided by the number of all input frames. And we elaborate on these issues in Subsection 5.3.

4 IMPLEMENTATION

4.1 Training SDD and SNM

We apply the model training method mentioned in [7] to train the SDDs and SNMs for each video stream. For each video stream, we first label its video frames by using YOLOv2. These labeled data are divided into two subsets as a training dataset and a test dataset. The former is used to train the SDD and the SNM for each video stream and the latter is used to select a set of suitable thresholds for δ_{diff}, c_{low}, and c_{high} to meet the requirement for accuracy and speed by comparing the predicted results against the real truths.

Different from NoScope [7], which uses specialized filters to query for target-object frames in a single off-line video and only extracts one frame from every 30 frames, FFS-VA designs the specialized filters to filter out the non-target-object video frames from large-scale video streams in both online and offline modes. For each online video stream, FFS-VA is designed to process at a rate of at least 30 FPS. In addition, the main metric of NoScope is throughput, but FFS-VA also pays attention to latency.

Before each filter is executed, raw frames need to be resized to meet the filter's feature size. The resizing times of SDD, SNM, and T-YOLO are 40us, 150us, and 400us respectively.

4.2 Filter Control

4.2.1 FilterDegree. Although the thresholds c_{low} and c_{high} basically determine the prediction probability of a video frame, the values between the two thresholds may also be acceptable. Because different video streams have different c_{low} and c_{high} values, the choice for t_{pre} can vary. So we compute the t_{pre} value as follows:

$$t_{pre} = (c_{high} - c_{low}) * FilterDegree + c_{low} \quad (2)$$

FilterDegree is a parameter set by users, which reflects the aggressiveness of filtering. When $FilterDegree = 1$ ($t_{pre} = c_{high}$), the output frames have a high credibility, but it increases the probability of false negative events. When $FilterDegree = 0$ ($t_{pre} = c_{low}$), more frames pass to the T-YOLO filter in this case, which bring a heavier burden to the T-YOLO filter. Therefore, *FilterDegree* can directly affect filtering efficiency and accuracy of FFS-VA. In our system, the cases $t_{pre} < c_{low}$ and $t_{pre} > c_{high}$ are not considered. This is because values within this range would result in a dramatically increase of error rate (e.g., 2.4%) or a remarkable decrease of throughput (e.g., 60%).

4.2.2 NumberofObjects. *NumberofObjects* reflects the intensity of the target objects in the predefined events. If the number of target objects appearing in a frame is less than *NumberofObjects*, the target object is considered to have a low intensity, which is

outside the scope of the user's interest. Otherwise, it is likely that some unexpected events have occurred.

In fact, T-YOLO can not only monitor the intensity of the target object, achieving purposeful filtering based on user needs, but also catch and correct the false positive results of SNM. For example, if a non-target-object frame is passed by the SNM accidentally, T-YOLO can still filter out the frame by counting the number of target objects, reducing the false positive events of FFS-VA.

4.3 System Optimization

In this section we introduce several methods to optimize throughput and latency of FFS-VA.

4.3.1 Feedback Queue. The primary goal of FFS-VA is to maximize its throughput of processing video frames. For offline video analytics, FFS-VA is expected to handle all frames as quickly as possible at a high throughput. For online scenarios, FFS-VA is expected to support more concurrent live streams. Note that the processing speed of one type of filter is often different from that of another. SDD processes 10× faster than SNM and 100× faster than T-YOLO. Because of the speed imbalance among the different filters and the fluctuation of *TOR* over time for a given video stream, the number of video frames processed by each filter also varies over time. When frames arrive in bursts, their processing filter threads could block.

In order to obtain high throughput at runtime with dynamic workloads, we propose a global feedback-queue mechanism. Due to the limited processing power of a filter, FFS-VA controls the detecting speed of a filter in an earlier stage in the pipeline by detecting the queue depth of the filter at a later stage. For example, when the T-YOLO queue depth exceeds a threshold, the SNM thread automatically slows down or even gets blocked, and stops pushing frames to the T-YOLO queue until the T-YOLO queue is free. Because of the existence of bypass, the filter at an earlier stage can work well even if a filter at a later stage is blocked. As long as the foremost prefetching process can keep at least 30 FPS, the video stream is being analyzed in real-time.

For some videos, the number of target-object frames varies greatly over time. In this case, a sudden increase in the proportion of target objects can significantly impact the T-YOLO queue depth and detection speed. Therefore, it is necessary to balance loads among video streams. In order to do this, for the video streams with a long T-YOLO queue depth, FFS-VA extracts no more than num_{t-yolo} frames from the queue per cycle. Otherwise, a fewer number of frames are processed by the T-YOLO filter in this cycle.

In the offline case, FFS-VA adaptively adjusts queue depth of each filter to obtain the highest throughput for a stream. In the online case, when the execution speed of T-YOLO is lower than a certain level (e.g.,140 FPS) for a period of time (e.g.,5s), it means this FFS-VA instance has spare ability to serve extra streams. Consequently, a new stream can be considered to add into the instance. In contrast, when any queue of T-YOLO or SNM is longer than its predefined threshold, it means that the FFS-VA instance overloads. The corresponding video stream is re-forwarded to another FFS-VA instance with spare capacity immediately.

The setting of the queue depth thresholds is important. Too small an threshold may reduce the throughput while too large an threshold will increase feasible overloads and latency. Unless otherwise stated, we initially and empirically determine 2, 10, and 2 as the queue depth thresholds of the SDD queues, SNM queues, and T-YOLO queues respectively.

4.3.2 Dynamic Batching. Since each video stream has its own SNM, it results in loading these models frequently when processing frames coming from different video streams. Even if a GPU has a strong parallel processing power, loading the network model for every frame significantly lowers the overall computational efficiency. To overcome this problem, we propose to process a batch of frames fed into a SNM at a time, which can greatly reduce the amount of model loading. For example, when the batch size is 30, the frequency of model loads is reduced by 30×.

Feedback-queue can achieve a higher throughput by static batch. However, a triggering strategy based on a fixed number of input frames per batch can lead to unnecessary delay if the number of frames being fed is less than the batch size. Besides, since we set a limit on the queue depth threshold, when the number of video frames fed from the SNM queue reaches the queue depth, the next frame requests from SNM are also delayed automatically. For example, when the batch size is greater than the queue depth threshold, video frames have to wait in the SNM. In addition, the batching operation itself can also introduce additional latency, especially for a large batch size.

In order to solve these problems, we propose a dynamic batch method based on a feedback-queue mechanism. If there are enough video frames in the SNM queue, SNM pops out a batch of (*Batch-Size*) image from the queue for SNM prediction. Otherwise, the frames are popped from the SNM queue until the queue is empty. Experimental results show that compared with using the feedback-queue mechanism alone, the dynamic batch mechanism reduces the average latency by 50% while lowering the throughput by only 16%, ensuring better real-time performance.

Note: Although we use two GPUs as a representation in the design, tasks of SNM or T-YOLO can be reasonably distributed across multiple GPUs to increase the overall performance in a single FFS-VA instance. Moreover, FFS-VA can effectively scale to multi-GPU or multi-server platforms by deploying multiple instances to support large scale video analysis. And load balancing can also be achieved by the re-forwarding of video streams between FFS-VAs instances. More details about FFS-VAs running on a server cluster platform will be elaborated in the future.

5 EVALUATION

5.1 Experimental Setup

Table 1: Information of Evaluation Videos.

Video Name	Resolution	Object	FPS	*TOR*
Coral	1280*720	Person	30 FPS	50%
Jackson	600*400	Car	30 FPS	8%

We use two real-world public videos, *Jackson* [43] and *Coral* [42], as our evaluation workloads. Their relevant information is summarized in Table 1. Each video contains about 10 million video frames in the time span of one day. We extract typical non-overlapping video clips from each video file to simulate multiple video streams. *Jackson* describes the scenes of various vehicles (e.g., car, bus, truck, etc.) traveling at a crossroad. *Coral* describes the scenes of people

watching colorful fish in an aquarium. Even for the same video, the number of target objects varies greatly with time. Hence, we can analyze the impact of scenes with different *TOR* values on the filtering performance. For fairness, the feature sizes of YOLOv2 used in both FFS-VA and the baseline are similar to be set as 416*416.

Hardware Platform We perform our experiments on a platform with two NVIDIA Geforce GTX1080 GPUs, dual Intel Xeon E5-2683 v3 CPU, and 128 GB DRAM. The multi-core CPUs provide good support for multi-threaded evaluation workload.

First, we explore the throughput of a video stream in offline situation and the maximum number of video streams a system can support in online situation. Second, we analyze the effect and sensitivity of filtering thresholds in FFS-VA to the overall system accuracy and filtering efficiency. Finally, we study the impact of the dynamic batch mechanism on the throughput and latency. For the sake of simplicity, we select 5000 consecutive frames from each video stream to perform the inference tasks.

5.2 System Performance

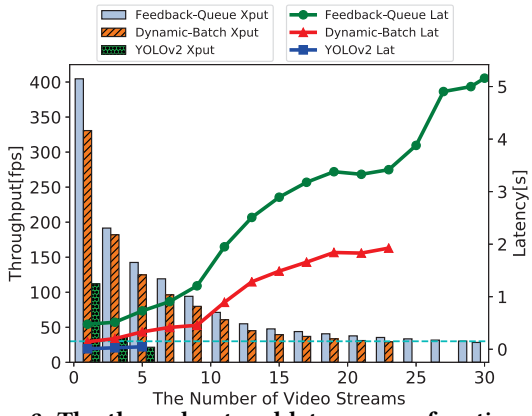


Figure 3: The throughput and latency as a function of the number of video streams with a *TOR* value of 0.103. Cases failing to meet the real-time requirements (throughput < 30 FPS) are not shown in the figure.

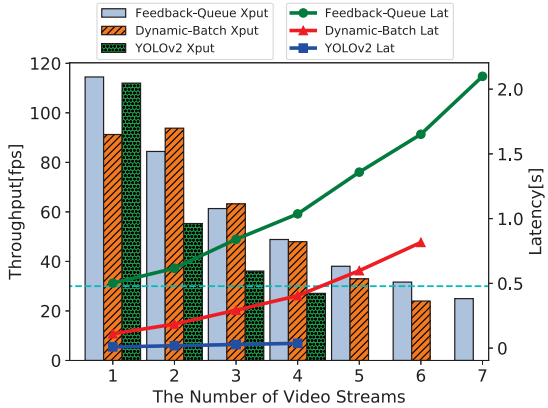
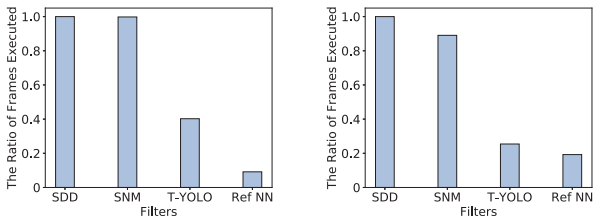
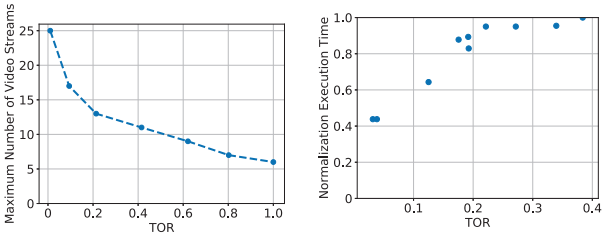


Figure 4: The throughput and latency as a function of the number of video streams with a *TOR* value of 1.000. Cases failing to meet the real-time requirements (throughput < 30 FPS) are not shown in the figure.



(a) Car detection (*TOR*=0.435) **(b) Person detection (*TOR*=0.259)**

Figure 5: The ratio of frames executed in each filter. The ecution speed of the four filters is about 20K FPS, 2K FPS, 200 FPS, and 56 FPS respectively.



(a) The maximum scalability **(b) The load balance**

Figure 6: The maximum scalability as a function of *TOR* and the load balance.

Figure 3 shows the throughput and latency as a function of the number of video streams at a low *TOR*.

Offline analysis. With a single video stream, which represents the offline analysis performance, the maximum throughput FFS-VA can support is 404 FPS, which is 3× that supported by YOLOv2. Compared with YOLOv2 the total execution time is reduced by 72.3%. In addition, for a 55 GB video file, the entire system uses less than 8 GB CPU memory, which implies greatly increased support capacity for long-time high-definition video files.

Online analysis. Experiments show that our system can support up to 30 video streams for real-time detection, which is 7 × more than what YOLOv2 can support. In addition, the dynamic batch mechanism has a 50% lower latency than using the feedback-queue mechanism alone in all cases, but at the cost of 20% reduction in the number of supported video streams. Compared with the YOLOv2, although FFS-VA has a latency of several seconds, these delays are insignificant and tolerable in normal applications, especially in intelligent video surveillance [1].

For video streams with 1.000 *TOR* as an extreme case, Figure 4 shows that SDDs and SNMs filter out fewer video frames and most of the frames are still fed to the T-YOLO for filtering, limiting the amount of increase in the overall throughput. In this case, FFS-VA can only support 5-6 video streams in real time. The offline detection throughput, implied by the single-stream performance, has also dropped noticeably in our experimental platform, and the overall execution time is close to the YOLOv2. This is because, on the same hardware platform, we use a GPU to perform YOLOv2 and another GPU is used to perform inefficient filtering, while the baseline YOLOv2 can perform on both GPUs.

Figure 5 presents the ratio of frames executed in each filter with different *TOR* during the day. SDD filters out few frames due to frequent movement and scene changes in the daytime. The filtering

efficiency of SNM is largely related to *TOR*. And T-YOLO can all work well in any case. It is worth noting that different time periods, weather, video contents, illumination, etc., may all affect the filter's performance of each stage, and we only show a small part of them because of the space restrictions.

In order to better understand the impact of *TOR* on the performance of FFS-VA, we extract a set of video clips with different *TOR* values and use FFS-VA to analyze them. Figure 6a shows that the maximum number of video streams supported by FFS-VA increases as *TOR* decreases. In this case, the execution speed of offline detection also accelerates with a decreasing *TOR*. Figure 6b shows the execution time of video streams, normalized to that of the longest execution time, with an even *TOR* distribution between 0 and 40%. Except the very low *TOR*, there is not much difference between these execution times. This shows that load balancing is well performed.

5.3 Sensitivity of Key Thresholds

Next, we examine how two key thresholds in FFS-VA, *FilterDegree* and *NumberofObjects*, impact the filtering efficiency and accuracy of FFS-VA. To verify the accuracy of FFS-VA, all the filtered frames by FFS-VA are completely detected by the reference model YOLOv2.

5.3.1 FilterDegree. Figure 7a illustrates the effect of the *FilterDegree* threshold on the number of output frames and the filtering error rate for the detection of cars. As the threshold increases, more frames whose prediction probability c is between c_{low} and c_{high} are filtered out.

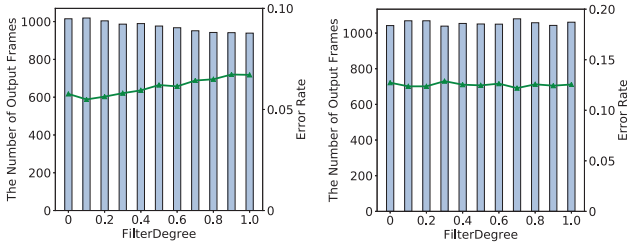
Figure 7b shows the results of person detection. The adjustment of the *FilterDegree* value has little effect on the filtering efficiency in this case. This is because during the entire observed time period, the aquarium is in the tourist peak and all frames contain many persons, which prevents SNM from filtering out any video frame.

5.3.2 NumberofObjects. As shown in Figure 8a, for car detection, as *NumberofObjects* increases, the number of output frames decreases significantly (about 80%). This is because, in this video, the size of a car is relatively large in a scene that can only contain no more than three target objects. Figure 8b illustrates the case of person detection. The number of output frames gradually decreases with the increase of *NumberofObjects*.

Table 2: Statistics of error frames in 5000 consecutive video frames.

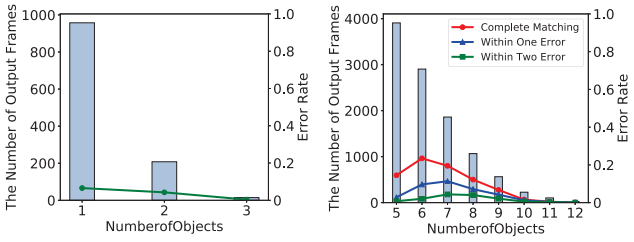
Error Frame	Number of Frames
An isolated single error frame	3
2-3 isolated-continuous error frames	5
Continuously-error frames less than 30	73
Continuously-error frames more than 30	140

5.3.3 Accuracy analysis. To better understand the error rate, we first analyzed the false-negative frames for car detection with a *TOR* value of 0.25. Table 2 illustrates the statistics of these error frames. The cases of an isolated single-error frame and 2-3 error frames do not affect the correct identification of the scene. In addition, a series of consecutive error frames whose size is less than a certain level (e.g., 30 frames) is usually caused by a distinguish criterion for partial-appearance of target object between T-YOLO and YOLOv2. In most cases where there are many consecutive error frames, it



(a) Car detection (TOR=0.197) **(b) Person detection (TOR=1.000)**

Figure 7: The throughput and error rate as a function of *FilterDegree*.



(a) Car detection (TOR = 0.197) **(b) Person detection (TOR = 1.000)**

Figure 8: Number of output frames and error rate as a function of *NumberofObjects*. In Figure 7(b), the number of filtered video frames approaches 0 as *NumberofObjects* further increases beyond 12.

is because a single partially appeared vehicle is waiting for traffic lights. By analyzing these images, we observe that only about 50 frames out of a total of 5000 frames are those with actual scene losses.

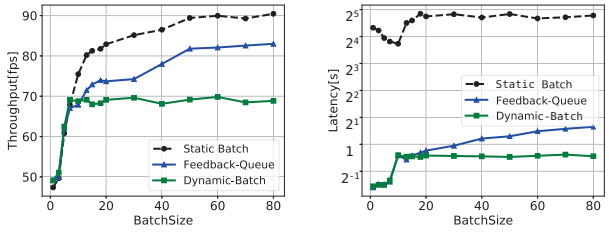
In addition, as shown in Figure 8, the error rate is relatively high. This is due to for the detection of small and dense targets, such as persons in the crowd, T-YOLO generally identifies fewer target objects than YOLOv2, resulting in a high error rate. In this case, Figure 8b shows that if one or two object misjudgment can be tolerated by relaxing the filtering threshold, the error rate will be greatly reduced (80.7% and 94.8% respectively). Even if relaxing filtering conditions may have a little impact on the filtering efficiency of the system (about 12.6% and 22.2% respectively), it is worthwhile to ensure the overall accuracy of the system. Therefore, it exists a trade-off between accuracy and filtering efficiency.

In short, the experiments show that thanks to the relaxed filtering conditions and the cascaded structure, the actual cases of missing scenes is less than 2% in our system, which is arguably negligible.

5.4 Batch Mechanism

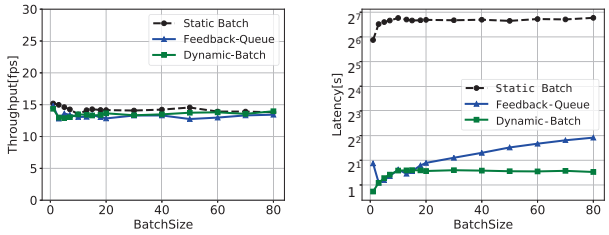
We analyze the impacts of the static batch, feedback-queue, and dynamic batch mechanisms on overall throughput and latency on 10 video streams.

5.4.1 Throughput of FFS-VA. Figure 9a shows the throughput of the static batch, feedback-queue, and the dynamic batch mechanisms respectively with 0.203 *TOR*. When *BatchSize* is low, these three methods can process a full batch of video frames at a time from the SNM queue. While SNM processes the current batch of video frames, SDD quickly pushes enough video frames to the SNM queue to form the next batch of video frames, thus having little



(a) Throughput (b) Latency

Figure 9: Throughput and latency under different batch mechanisms with *TOR* 0.203.



(a) Throughput (b) Latency

Figure 10: Throughput and latency under different batch mechanisms with *TOR* 0.980.

impact on the throughput. When *BatchSize* is high, for the static batch mechanism, the throughput can still continue to increase with sufficient data provided by SNM queues. For the feedback-queue mechanism, the wait for video frames increases the execution time, resulting in a slight decrease in throughput (about 8%). For the dynamic batch mechanism, the smaller batch size leads to a decrease in computation efficiency, resulting in a lower throughput.

Figure 10a shows the experimental results with 0.980 *TOR*. In this case, most of the frames are eventually executed by T-YOLO no matter what the *BatchSize* value is. Therefore, *BatchSize* has little effect on the throughput in this case.

5.4.2 Average Latency. Figure 9b and 10b show that when *BatchSize* is small, the difference in average latency between the feedback-queue and the dynamic batch mechanisms is very small. As *BatchSize* increases, more video frames need to wait a period of time in the feedback-queue because of the fixed batch size. For the dynamic batch mechanism, since the batch size can be adjusted automatically according to video contents, the average latency is basically unchanged. Due to the same queue management mechanism, for video streams with 0.980 *TOR*, average latency has a similar trend.

In summary, for videos with a low *TOR* value, the feedback-queue mechanism has a greater throughput, and the dynamic batch mechanism has a smaller average latency. For videos with a high *TOR* value, there is not much difference in throughput between the feedback-queue and dynamic batch mechanisms, but the dynamic batch mechanism has a lower average latency and should be considered first.

5.5 Limitations and Remedies

While the FFS-VA filtering system is shown to be highly effective in real-time object detection for large-scale video streams, there remain some limitations.

Target Object Rate Sensitivity. In practice, a sudden increase in *TORs* in video streams can lead to poor filtering efficiency, even if the probability of multiple videos having their *TORs* increase simultaneously is extremely low. If necessary, we can temporarily store these video frames in the storage system, to be processed later. For some latency-sensitive scenes, it is necessary to allocate more GPUs to provision for peak-load periods.

Error Rate. The reason for the cases of relative high error rates is the performance difference between T-YOLO and the reference model YOLOv2. Deep compression [6] (e.g., pruning, sparsity constraint) can transform a larger but more accurate NN model to a tiny model without compromising the accuracy of the prediction, resulting in a 3× throughput improvement[32]. Therefore, we can replace T-YOLO with a high-precision mode that was deeply compressed to obtain a low error rate.

Scene Switch. We train SDD and SNM models for each fixed-angle camera and specific target object, so the changes of video scene may affect the detection accuracy. If the scene change in the video is periodic (e.g., alternating between day and night), the training data just needs to include representative frames under all conditions. However, when the scene changes dramatically or the function and position of the camera have changed, the previous specialized models will no longer work. If there are no saved models in the past that can match the current environment, a new network model needs to be trained according to the new scene, which takes about one hour.

Single Target Object. In this paper, we assume that there is only one user-interested target object for each video stream. If multiple target objects exist in a video stream, the structure of the specialized network model only needs to be changed to support the identification of all the target objects in the video.

6 RELATED WORK

Prior studies relevant to FFS-VA can be classified into three main categories, model cascades, object detection and video monitoring.

Model Cascades. Cascade is defined as a sequence of classifiers to improve inference speed. Paul Viola et al. [27] proposed the first cascade, the Viola-Jones detector, which cascades traditional image processing features. The sub-windows which are not rejected by an initial classifier are processed by a sequence of classifiers. If any classifier rejects the sub-window, no further processing is performed. Recent work has concentrated on learning cascades. [39] achieves an optimal trade-off between accuracy and speed by learning a complexity aware cascade. [9] configured a CNN cascade for real-world face detection to accurately differentiate faces from the backgrounds. [38] proposed a cascaded regression approach for facial point detection to make more accurate predictions. In our filtering system, we use a cascade to filter out video frames that we do not care about, not specific to the features. Besides, FFS-VA focuses on improving the processing throughput of the system rather than the effectiveness of a single model.

Object Detection. SPPnet [17] and Fast R-CNN [31] have achieved a very high accuracy for the image object detection, by using region proposal object detection methods. OverFeat [28] and YOLO [15] have achieved a high detection speed by skipping the proposal step altogether, and predicting bounding boxes and confidences for

multiple categories directly. Our goal is to increase the throughput of the overall system using these models and some other models for filtering in practice.

Video Monitoring. Video monitoring involves many tasks [34], including vehicle tracking [4], object detection [13] and so on. Each task has been tailored for a specific system (e.g., vehicle counting [12], license plate detection [6], cars or pedestrians tracking [3]). And the main target objects are car and pedestrian. Our filtering system focuses on video analysis, and several objects (e.g., dogs, cats) can be detected in FFS-VA to facilitate the understanding of the scene if needed. Besides, more event-related details (e.g., detail behavior analysis) can be fine-grainedly detected by the back-end network model instead of just trajectory analysis.

7 CONCLUSION

In real life, there are a lot of camera resources to be explored. And NN makes it easy to extract semantic information from these videos. However, its computational efficiency is low. Besides, the huge number of video frames in the large-scale video streams also poses a great challenge to NN. In response, we propose a filtering system that equips a SDD model, a SNM model, and a global T-YOLO model for each video stream, filtering out the video frames that the users are not concerned about in the video, and reducing the number of video frames that need to be detected by the full-feature model. The experimental results show that our filtering system provides 3-7× scalability improvement over the state-of-the-art YOLOv2. In addition, the reference model in this paper is an object detection network, but FFS-VA can also be applied to other fields, such as face recognition, by configuring other appropriate reference models.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. This research is partly sponsored by the US NSF under Grant No. CCF-1704504 and CCF-1629625, Nanjing Superstack Co.

REFERENCES

[1] Antonio C. Nazare Jr et al. 2016. A scalable and flexible framework for smart video surveillance. *Computer Vision and Image Understanding* 144, C (2016), 258–275.
[2] Alex Krizhevsky et al. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of NIPS*. 1097–1105.
[3] Boris Babenko et al. 2011. Robust Object Tracking with Online Multiple Instance Learning. *IEEE Transactions on Pattern Analysis Machine Intelligence* 33, 8 (2011), 1619–32.
[4] Bin Tian et al. 2011. Video processing techniques for traffic flow monitoring: A survey. In *Proceedings of ITSC*. 1103–1108.
[5] Carlos Graca et al. 2017. Hybrid multi-GPU computing: accelerated kernels for segmentation and object detection with medical image processing applications. *Journal of Real-Time Image Processing* 13, 1 (2017), 1–18.
[6] Christos Nikolaos Anagnostopoulos et al. 2008. License Plate Recognition From Still Images and Video Sequences: A Survey. *IEEE Transactions on Intelligent Transportation Systems* 9, 3 (2008), 377–391.
[7] Daniel Kang et al. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1586–1597.
[8] Guoliang Mo et al. 2010. Vehicles detection in Traffic Flow. In *Proceedings of ICNC*. 751–754.
[9] Haoxiang Li et al. 2015. A convolutional neural network cascade for face detection. In *Proceedings of CVPR*. 5325–5334.
[10] Haizhong Wang et al. 2010. Calculation of traffic flow breakdown probability to optimize link throughput. *Applied Mathematical Modelling* 34, 11 (2010), 3376–3389.
[11] Hui Wei et al. 2017. *Efficient graph-based search for object detection.* Elsevier Science Inc. 395–414 pages.
[12] Haoyu Zhang et al. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *Proceedings of USENIX NSDI*. 377–392.
[13] Jong Bae Kim et al. 2003. Efficient region-based motion segmentation for a video monitoring system. *Pattern Recognition Letters* 24, 1-3 (2003), 113–128.
[14] Jifeng Dai et al. 2016. R-FCN: Object Detection via Region-based Fully Convolutional Networks. In *Proceedings of NIPS*. 379–387.
[15] Joseph Redmon et al. 2015. You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of CVPR*. 779–788.
[16] Joseph Redmon et al. 2016. YOLO9000: Better, Faster, Stronger. (2016), 6517–6525.
[17] Kaiming He et al. 2015. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis Machine Intelligence* 37, 9 (2015), 1904–1916.
[18] Karen Simonyan et al. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *Computer Science* (2014).
[19] Lei Huang et al. 2017. Detection of abnormal traffic video images based on high-dimensional fuzzy geometry. *Automatic Control and Computer Sciences* 51, 3 (2017), 149–158.
[20] Lai Jiang et al. 2017. Predicting Video Saliency with Object-to-Motion CNN and Two-layer Convolutional LSTM. *CoRR* (2017).
[21] Li Li et al. 2008. A Traffic Congestion Estimation Approach from Video Using Time-Spatial Imagery. In *Proceedings of ICINIS*. 465–469.
[22] Mark Everingham et al. 2010. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision* 88, 2 (2010), 303–338.
[23] Mengdi Wang et al. 2014. Stochastic compositional gradient descent: algorithms for minimizing compositions of expected-value functions. *Mathematical Programming* (2014), 1–31.
[24] Norbert Buch et al. 2011. A Review of Computer Vision Techniques for the Analysis of Urban Traffic. *IEEE Transactions on Intelligent Transportation Systems* 12, 3 (2011), 920–939.
[25] Nicolas Ballas et al. 2015. Delving Deeper into Convolutional Networks for Learning Video Representations. (2015), 5644–5651.
[26] Papert et al. 1966. The Summer Vision Project. *Memo AIM-100, MIT AI Lab* (1966).
[27] Paul A. Viola et al. 2001. Rapid Object Detection using a Boosted Cascade of Simple Features. In *Proceedings of CVPR*. 511–518.
[28] Pierre Sermanet et al. 2013. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *CoRR* (2013).
[29] Qi Wang et al. 2018. Locality Constraint Distance Metric Learning for Traffic Congestion Detection. *Pattern Recognition* (2018), 272–281.
[30] Ross B. Girshick et al. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of CVPR*. 580–587.
[31] Ross B. Girshick et al. 2015. Fast R-CNN. (2015), 1440–1448.
[32] Song Han et al. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proceedings of ISCA*. 243–254.
[33] Shaoqing Ren et al. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis Machine Intelligence* 39, 6 (2017), 1137–1149.
[34] V. Kastrinaki et al. 2003. A survey of video processing techniques for traffic applications. *Image Vision Computing* 21, 4 (2003), 359–381.
[35] Wei Liu et al. 2015. SSD: Single Shot MultiBox Detector. In *Proceedings of ECCV*. 21–37.
[36] Wenpeng Yin et al. 2017. Comparative Study of CNN and RNN for Natural Language Processing. *arXiv preprint arXiv:1702.01923* (2017).
[37] Yao Lu et al. 2016. Optasia:A Relational Platform for Efficient Large-Scale Video Analytics. In *ACM Symposium on Cloud Computing*. 57–70.
[38] Yi Sun et al. 2013. Deep Convolutional Network Cascade for Facial Point Detection. In *Proceedings of CVPR*. 3476–3483.
[39] Zhaowei Cai et al. 2015. Learning Complexity-Aware Cascades for Deep Pedestrian Detection. (2015), 3361–3369.
[40] Zhicheng Wang et al. 2017. A Novel Fire Detection Approach Based on CNN-SVM Using Tensorflow. In *Proceedings of ICIC*. 682–693.
[41] Jackson. 2017. town-square-southwest. https://www.seejh.com/webcams/jacksonhole/jackson/town-square-southwest. (2017).
[42] Noscope-data. 2017. coral-reef-long.mp4. https://storage.googleapis.com/noscope-data/videos/coral-reef-long.mp4. (2017).
[43] Noscope-data. 2017. jackson-town-square.mp4. https://storage.googleapis.com/noscope-data/videos/jackson-town-square.mp4. (2017).
[44] Ameema Zainab. 2017. Real-time Object Detection. https://blog.mindorks.com/detection-on-android-using-tensorflow-a3f6fe423349. (2017).