A Fast Filtering Mechanism to Improve Efficiency of Large-Scale Video Analytics

Chen Zhang, Qiang Cao, *Senior Member, IEEE*, Hong Jiang, Wenhui Zhang, *Student Member, IEEE*, Jingjun Li, and Jie Yao, *Member, IEEE*

Abstract—Surveillance cameras are ubiquitous around us. Emerging full-feature object-detection models can analyze surveillance videos with high accuracy but consume much computation. Directly applying these models for practical scenarios with large-scale cameras is prohibitively expensive. This, however, is wasteful and unnecessary considering that user-defined anomalies occur rarely among these videos. Therefore, we propose FFS-VA, a multi-stage Fast Filtering Mechanism for Video Analytics, to make video analytics much cost-effective. FFS-VA filters out the frames without the user-defined events by two stream-specialized filters and a cheap full-function model, to reduce the number of frames reaching the full-feature model. FFS-VA presents a global feedback-queue approach to balance the processing speeds of different filters in intra-stream and inter-stream processes. FFS-VA designs a dynamic batch technique to achieve a trade-off between throughput and latency. FFS-VA can also efficiently scale to multiple GPUs. We evaluate FFS-VA against the state-of-the-art YOLOv3 under the same hardware and video workloads. The experimental results show that under a 12.88 percent target-object occurrence rate on two GPUs, FFS-VA can support up to 30 concurrent video streams (15× more than YOLOv3) in the online case, and obtain 10× speedup when offline analyzing a stream, with an accuracy loss of less than 2 percent.

Index Terms—Filters, frames, video analytics

1 Introduction

An increasing number of surveillance cameras with low cost and high quality have been deployed in key public areas all over a city (e.g., street corners, shopping malls, and office buildings), to monitor potential accidents as well as record critical clues. In traditional practice, the video surveillance collects live streams to an operational center for further manual observations, which is labor-intensive, error-prone and cost expensive. Automatic video analysis based on object detection has been introduced recently to mitigate human intervention while significantly improving the performance and accuracy. The cases of automatic analysis for surveillance videos can be categorized into two main types: (1) real-time analysis to detect anomalies; and (2) post-facto analysis to look for a certain event retroactively.

Early automatic analysis methods [1] employ support vector machine, but its accuracy and function are limited [2]. 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 [3] first achieves 53.7 percent mean average precision (mAP) on PASCAL VOC 2010 [4]. Afterwards, SSD [5] (74.3 percent mAP, 59 frame per second (FPS)), R-FCN [6] (83.6 percent mAP, 5 FPS), YOLOv2 [7] (76.8 percent mAP, 67 FPS), and YOLOv3 [8] (30 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 full-feature object-detection models are extremely computationally hungry. A powerful GTX Titan X GPU only supports one video stream at 30 FPS with YOLOv3 model. Considering that a typical video surveillance generally deploys hundreds of 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 analytics, a typical anomalous event occurs rarely and when it does occur it appears in a tiny fraction of all frames. For example, even if a serious traffic jam takes place on the main route of a big city, the average blocked time in a day is just less than 5 percent [9]. Therefore, passing all frames over these accurate but weighty models actually wastes considerable computational capability, which is totally unnecessary. The key idea is to fast filter out most frames that are not related to user-defined events while leaving the remaining frames to be accurately analyzed by the final full-feature NN model.

To efficiently employ these highly accurate object detection models on limited computing devices to achieve large-scale high-resolution video analytics, we propose FFS-VA, a fast filtering mechanism, to dramatically reduce the number

- *C. Zhang, Q. Cao, W. Zhang, and J. Li are with the Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System of Ministry of Education, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: {hust_zchen, caoqiang, jingjunli}@hust.edu.cn, singularity_x@outlook.com.*
- *H. Jiang is with the University of Texas at Arlington, Arlington, TX 76019. E-mail: hong.jiang@uta.edu.*
- *J. Yao is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: jackyao@hust.edu.cn.*

Manuscript received 2 Aug. 2019; revised 1 Dec. 2019; accepted 26 Jan. 2020. Date of publication 30 Jan. 2020; date of current version 8 May 2020. (Corresponding author: Qiang Cao.) Recommended for acceptance by Y. Han. Digital Object Identifier no. 10.1109/TC.2020.2970413

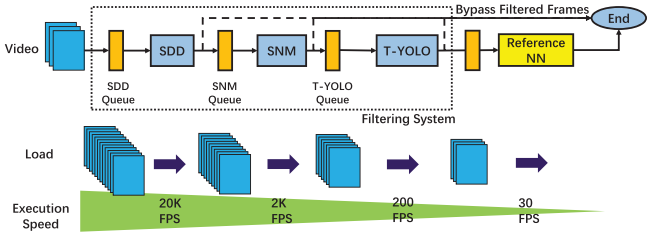


Fig. 1. FFS-VA adds three filters before the Reference NN to filter out frames that are not related to the user-defined events.

of frames actually reaching the full-feature reference model by filtering out vast amounts of frames that do not satisfy the conditions of the user-defined events (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 and each stage is equipped with a specific model to filter out frames with a certain feature. Considering that most surveillance cameras are of a 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 by a cheap full-function Tiny-YOLO-Voc model (T-YOLO), that is shared by multiple streams, to filter out frames whose target objects are fewer than a predefined threshold. Finally, the surviving frames are fed into an ultimate full-feature reference model (Reference NN) for high-accuracy analysis. FFS-VA is designed to run on the mainstream heterogeneous servers with several GPUs and CPUs. Fig. 1 shows the filtering structure of FFS-VA over one video stream.

There are four key challenges in FFS-VA that need to be addressed. (1) For a mainstream server with two GPUs (at least), the workloads should be evenly distributed on CPUs and GPUs of the heterogeneous server to achieve a high performance. (2) Theoretically, the number of frames processing in the several stages within a stream is gradually decreasing due to filtering. Accordingly, the filter at a later stage is also slower than one at an earlier stage in processing speed. Unfortunately, due to the unpredictable video contents the number of frames entering each stage varies significantly over time. How to dynamically balance the video loads among filters within a stream and among multiple streams is a key problem. (3) For the network model executed on the GPU (e.g., SNM), in order to process a frame, the corresponding network model, image data, and the predicted result must be loaded between the CPU memory and the GPU memory. To reduce the overhead of frequent data exchange, a large and static batch size is intuitively better for obtaining high computational efficiency but at the cost of lengthened processing latency. In a multi-stage stream processing system with multiple CNNs, the impact of batch size is considerable. So it is necessary to dynamically trade off at runtime between latency and throughput based on the video contents. (4) If there are multiple GPUs available in the system, all of them can be used to improve the performance of offline analysis and online analysis. To efficiently parallel the GPUs without affecting the order of frames in a stream, it also demands to appropriately assign all tasks to these GPUs at runtime.

FFS-VA uses the next several techniques to solve the above challenges. (1) If just two GPUs are available on a server, all SDDs are executed on the CPU, both SNMs and T-YOLO are executed on a single GPU. The Reference NN runs on another GPU alone. 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 executed concurrently. (2) FFS-VA builds a global feedback approach to orchestrate the processing speeds of all stages based on their respective queue controls. (3) FFS-VA adopts a dynamic batch technique to dynamically determine batch size according to current video contents in a short period of time. (4) If multiple GPUs are configured on a server, a dynamic GPU scheduling method is utilized to allocate stages to GPUs automatically at runtime. And then FFS-VA presents four GPU parallel schemes to make multiple GPUs run efficiently and ensure the frame order during processing a video stream.

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 YOLOv3 model with the same hardware environment consisting of two CPUs and two GPUs, FFS-VA supports up to 15× more concurrent video streams in online video analysis, and obtains 10× speedup in offline video analysis, with a negligible accuracy loss of less than 2 percent. Besides, as the number of GPUs increases, FFS-VA can support real-time detection for more video streams. In addition, we also analyze the performance differences between YOLOv2 model and YOLOv3 model in terms of throughput and accuracy, and show the impact of the improvement of Reference NN model on system performance.

In summary, the key contributions of our work are:

1) We propose a pipelined multi-stage fast filtering mechanism for large-scale video analytics (FFS-VA) in both online and offline scenarios.
2) FFS-VA introduces a global feedback-queue approach to control the processing speeds of all filters in both intra-stream and inter-stream processes. FFS-VA designs a dynamic batch technique with a video-content-based batch-size adjustment to automatically trade off between latency and throughput.
3) FFS-VA adopts a scheduling method to dynamically allocate stages on GPUs. To efficiently perform a stage on multiple GPUs, FFS-VA further designs four GPU parallel schemes under different scenarios.
4) We implement a FFS-VA prototype system to compare it with the state-of-the-art YOLOv3, which indicates that FFS-VA improves the overall throughput by up to 15× 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, batch technique, scalability 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

2.1 Convolutional Neural Network Models

Optical Flow [10], Support Vector Machine (SVM)[1], and Convolutional Neural Network (CNN) [11] can all be used for the recognition of specific targets. The insight of Optical Flow is matching pixels between images using temporal and gradient information. SVM builds linear decision hyperplanes in the feature space to perform classification. The main idea of CNN is to learn the features of targets by dealing with numerous images and updating the weights of artificial neurons. Since CNN has great image recognition capabilities and can recognize the characteristics of the target objects easily and accurately, it has been widely developed in recent years. In what follows, we briefly introduce the 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.

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

2.2 Object Detection

Object detection technology has been widely used to automatically analyze video contents, such as detecting accidents [12] and traffic congestion [13], searching a certain object [14], understanding the flow of vehicles and pedestrians to provide users with the most reasonable traffic planning [15], etc., which greatly reduce the cost of manpower.

In terms of the use of spatial-temporal information on video frames, Nicolas Ballas *et al.* [16] and Lai Jiang *et al.* [17] correct the detection results by using relevant timing and context information, which greatly improve the accuracy of object detection. For static image detection, the accuracy of R-CNN (53.7 percent mAP) [3], fast R-CNN (65.7 percent mAP) [18], faster R-CNN (70.4 percent mAP) [19], and R-FCN (83.6 percent mAP) [6] has been continually improved, but the execution speed of these full-feature object detection techniques remains relatively low and inadequate for real-time detection. Recently, advanced methods such as YOLOv2 (67 FPS) [7], SSD (59 FPS) [5], and YOLOv3 (30 FPS) [8] have been developed to achieve real-time detection, but they are also computationally expensive as a powerful GTX Titan X GPU merely supports the analysis of no more than two concurrent video streams in real time. So in a limited hardware environment, using these advanced models to perform real-time detection for large-scale video streams is a huge challenge.

In fact, there are two methods that are usually used to speed up the inference process of CNN models: compression and specialization. The commonly used compression methods include removing some CONV layers and FC layers [20], reducing the feature size of models [7], and matrix pruning [21]. The general compression methods usually achieve a huge increase in execution speed at the expense of accuracy unless the compression is performed efficiently and effectively. On the other hand, specialization refers to the CNN models are trained by using the dataset in specific conditions or scenes. So the generality requirements for these models are sacrificed, and if they are used in other conditions or scenes the accuracy may drop dramatically. Due to only one specific context is considered, these specialized models can be both accurate and fast. Certainly, both of these two methods are used in this paper.

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. 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 and providing warning for the upcoming risks.

Indeed, the frames without 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

$$TOR = num_{target-object-frames}/num_{all-frames}, \tag{1}$$

where $num_{all-frames}$ is the total number of all frames in a 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 target-object frames that appear in a video slice. TOR is primarily determined by video contents, 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 [22], the target-object occurrence rate in a day is only 8 percent.

Therefore, under the actual low TOR, passing all frames over a full-feature model such as YOLOv3 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 model to further extract interesting information. Therefore, we design three types of preceding filters and a pipelined multi-filter architecture to achieve the aforementioned goal.

Additionally, the fast filtering system should evenly distribute all tasks on CPUs and GPUs, exploiting the maximal potential of the underlying hardware to boost the overall performance under the promised latency and negligible accuracy loss. Besides, If multiple GPU devices are available

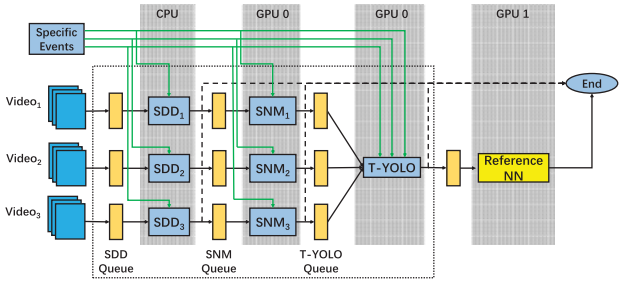


Fig. 2. Architecture of FFS-VA on two GPUs.

in the server, they can be used to improve the performance of online analysis and offline analysis. We will elaborate on these issues in detail next.

3 Design

3.1 Overview of FFS-VA

Fig. 2 illustrates the architectural overview of FFS-VA on a server with two GPU devices, which consists of three types of filters: (1) specialized difference detectors (SDDs), (2) specialized network models (SNMs), and (3) a globally shared object detection model Tiny-YOLO-Voc (T-YOLO).

3.1.1 Main Functions

First, target events, such as the occurrences of cars, persons, etc., as well as their counts, need to be predefined by users. For each given video stream, both SDD and SNM are specialized for it and its predefined event. 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 and 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-accuracy identification and analysis. The details of filters are illustrated in Section 3.3. For clarity and meaningful evaluation, we choose the state-of-the-art full-feature model, YOLOv3, as the Reference NN model for FFS-VA in this paper.

3.1.2 Pipeline Design

The three preceding filters and final Reference NN model form a four-stage pipelined architecture. Each filter is connected by its corresponding input and output queues for reading and forwarding frames. For each video stream, a prefetching thread is specifically created to capture video frames from cameras or disks. 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 preceding filter. For better performance, 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 varied filters, we propose a global feedback-queue approach, as detailed in Section 4.3.1, to coordinate the processing speeds of various filters. In addition, a dynamic batch technique also be introduced to help trade off between throughput and latency automatically at runtime, which is illustrated in Section 4.3.2.

Considering the varied computational complexity of filters and Reference NN, in order to fully exploit the potential of the hardware, SDDs are executed on the CPUs, and SNMs and T-YOLO are executed on a single GPU. The Reference NN model uses another GPU alone. Besides, the runtime scheduling in the system (e.g., feedback queue and dynamic batch) is also controlled by the CPU. In FFS-VA, each filter is associated with an independent thread. Through the parallel and pipelined structure, FFS-VA achieves a high analyzing throughput. Although only two GPUs are used in this part, FFS-VA can conveniently scale the processing capacity to a server with more GPUs or a server cluster, which will be discussed in Section 4.4.

3.2 Formal Description

We provide a theoretical description based on the system with three filters and a Reference NN, which can help understand the key criteria of the filtering mechanism. This description can also be extended to the other system with more filters and various models.

3.2.1 Throughput

In Fig. 2, when one stream runs on the system and V frames are processed, the passing ratios of these frames in the three filters and the Reference NN are r_1, r_2, r_3, and r_4 respectively (their corresponding filtering ratios are $1-r_1$, $1-r_2$, $1-r_3$, and $1-r_4$). In this case, the number of frames outputted by the Reference NN can be calculated as

$$V_{output} = V \cdot r_1 \cdot r_2 \cdot r_3 \cdot r_4. \tag{2}$$

The overall filtering ratio of the system is defined as

$$R_F = (V - V_{output})/V = 1 - r_1 \cdot r_2 \cdot r_3 \cdot r_4. \tag{3}$$

Suppose the detection speeds of three filters and Reference NN are s_1, s_2, s_3, and s_4 respectively. Considering the four stages can be executed concurrently, the total execution time is presented as

$$T = max\left(\frac{V}{s_1}, \frac{V \cdot r_1}{s_2}, \frac{V \cdot r_1 \cdot r_2}{s_3}, \frac{V \cdot r_1 \cdot r_2 \cdot r_3}{s_4}\right). \tag{4}$$

Similarly, the Equations (5) and (6) show the overall filtering ratio and execution time of FFS-VA for the analysis of N video streams. Thus the actual throughput of the system can be considered as the ratio of the total number of input frames to the execution time (Equation (7))

$$R_F = 1 - \frac{\sum_{i=1}^{N} V_{output}^i}{\sum_{i=1}^{N} V^i} = 1 - \frac{\sum_{i=1}^{N} V^i \cdot r_1^i \cdot r_2^i \cdot r_3^i \cdot r_4^i}{\sum_{i=1}^{N} V^i} \tag{5}$$

$$T = max\left(\frac{\sum_{i=1}^{N} V^i}{s_1}, \frac{\sum_{i=1}^{N} V^i \cdot r_1^i}{s_2}, \frac{\sum_{i=1}^{N} V^i \cdot r_1^i \cdot r_2^i}{s_3}, \frac{\sum_{i=1}^{N} V^i \cdot r_1^i \cdot r_2^i \cdot r_3^i}{s_4}\right) \tag{6}$$

$$Xput = \frac{\sum_{i=1}^{N} V^i}{T}. \tag{7}$$

Based on the above definitions, the filtering ratio of FFS-VA is mainly related to the contents of video streams and can be significantly affected by the actual passing ratio at each stage in practice. In addition, except for the filtering ratio, the detection speeds of stages are another factors that affects the actual throughput. Given the video loads show a downward trend due to filtering, the detection speeds of filters and Reference NN should also be gradually decreasing (i.e., $s_1 > s_2 > s_3 > s_4$).

3.2.2 Latency

The video frames outputted by the Reference NN satisfy all the conditions of the user-defined events, and their processing latency is also a key metric. We define the processing latency of a frame as the time interval between it incoming and leaving the system. In Equation (8), it is divided into three parts: $L_{service}$, L_{wait}, and L_{sync}

$$L = L_{service} + L_{wait} + L_{sync}. \tag{8}$$

$L_{service}$ means the sum of processing time on all stages, which is mainly related to hardware configurations, the complexity of models, and batch size. L_{wait} is the queue time of frames, which is affected by resource competition, queue depth, video contents, and so on. And L_{sync} is defined as the sum of synchronous time between multiple GPUs at each stage. L_{sync} is negligible unless one of the stages is assigned to more than one GPU.

In short, $L_{service}$ can be shortened with more powerful computing devices or smaller batch size. Besides, reasonable system configurations and runtime scheduling need to be utilized to reduce L_{wait}. And L_{sync} can be optimized by designing and using appropriate GPU parallel schemes.

3.2.3 Accuracy

In fact, it is possible for a frame to be recognized by the Reference NN but filtered out by its preceding filters, i.e., a false negative. Of course, if a frame unrelated to the user-defined event passes all stages, this is called as a false positive. To quantitatively analyze the accuracy of the system, we define the error rate as

$$ErrorRate = \frac{\sum_{i=1}^{N} FN_i + FP_i}{\sum_{i=1}^{N} V_i}, \tag{9}$$

where FN_i and FP_i refer to the number of false negatives and false positives respectively among V_i frames in a stream i. Then the accuracy is described as

$$Acc = 1 - ErrorRate. \tag{10}$$

Since the last stage of the system is the Reference NN, which serves as the accuracy baseline of the system, and all output frames have to go through this stage, so there are no false positives in output frames in practice.

(a) Background (b) Frame with a car (c) Subtracted frame

Fig. 3. An example of SDD. The subtracted frame highlights the car that entering the scene.

3.2.4 Scalability

If there are idle GPUs available in the server, they can be used to improve the performance of the system. More GPU devices can mitigate resource competition in intra-stream and inter-stream processes as well as can increase the detection speeds of filters and Reference NN (i.e., s_1, s_2, s_3, and s_4). So the system throughput can also be significantly improved. But multiple GPUs also introduce some extra scheduling overheads, such as stage allocations on GPUs, transferring data between CPU memory and GPU memory, and collecting the analysis results synchronously from the GPUs. These overheads lead to that the increase in the throughput is not proportional to the increase in the number of GPUs.

In addition, the scalable approach cannot reduce the latency L since the processing time of all stages $L_{service}$ is not shortened. On the contrary, in order to guarantee the frame order in a stream, a synchronous or reordering process is required when collecting analysis results from multiple GPUs, which introduces a little synchronous time L_{sync}.

More importantly, under different video loads, the optimal stage allocations are varied. For example, when an anomaly occurs in FFS-VA, idle GPUs are more suitable for speeding up the processing of the Reference NN because of its expensive computation. But for the scenes without anomalies, T-YOLO stage is more likely to become the bottleneck of the system and needs to be assigned more GPUs. The differences of stage allocations on GPUs inevitably affect the detection speed and the system throughput even with the same number of GPUs.

3.3 Detailed Design of Filters

3.3.1 Specialized Difference Detector (SDD)

As the first filter of the system, SDD is responsible for foreground segmentation [23] and determines whether a unlabeled frame is a background frame. Commonly used foreground segmentation methods include Averaging, Single Gaussian, Kalman filter, Gaussian Mixture Model, and so on. In FFS-VA, all SDDs use Averaging method because of its fast execution speed (Fig. 3).

In fact, SDD calculates the distance between the reference image (i.e., background image) and the unlabeled frame to determine whether these two images are identical. For simplicity, the reference image is usually computed as the average of dozens of background frames. The distance between two images 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 unlabeled frame. Otherwise, the frame is considered as a background frame. Note that most surveillance cameras are deployed in a fixed viewpoint. Hence, the background frames can be safely discarded.

Naturally, the threshold δ_{diff} has a critical affect on the filtering ratio and accuracy. A low δ_{diff} may result in poor filtering ratio of SDD, 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}. In addition, weather, light intensity, etc. can all contribute to the value of MSE [23], so the filtering ratio of SDD varies greatly at different scenes.

SDD processes 100*100-pixel images at 100K FPS (0.9M multiplication operations and 1.8M addition operations per 30 frames). In the FFS-VA, the detection speed of SDD is 600× faster than the reference model YOLOv3 (7.1G multiplication operations and 34.8G addition operations per 30 frames), as demonstrated in Section 5.

3.3.2 Specialized Network Model (SNM)

Another key filter used in FFS-VA is SNM. SNM utilizes a specialized 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 generality of these methods leads to huge computational overhead and long execution time. On the contrary, SNM can only identify a class of predefined target objects in the specific video stream, and thus trading off reducing the generality for boosting its speed (70× real-time). In addition, for a fixed-angle camera, the position and the moving trail of the target objects in the scene are relatively fixed. In this case, using SNM for rapid image recognition can also ensure the accuracy to be over 95 percent [24].

In fact, SNM is a three-layer CNN (CONV, CONV, and FC). The design and training process is shown in Section 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, threshold t_{pre} is utilized (between c_{low} and c_{high}) at runtime to help SNM distinguish target-object frames and non-target-object frames, which is shown in Section 4.2. 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 (159.7M multiplication operations and 422.8M addition operations per 30 frames). It is 60× faster than the reference model YOLOv3 on the real hardware platform.

3.3.3 Tiny-YOLO-Voc (T-YOLO)

The third filter of FFS-VA is a cheap and compressed object detection network, T-YOLO [25], which is used to filter out the frames whose the number of target objects is less than a certain level. Other object detection networks, such as YOLO, SSD, and R-FCN, due to the slow execution speed caused by more layers and more classes, are not used in our filtering system preferentially. As a compressed model, T-YOLO just consists of 9 CONV layers and 6 POOL layers, and is trained by the VOC dataset with 20 classes. Benefitting from fewer classes and smaller model size, T-YOLO can perform at up to 220 FPS for 416*416 pixel images with just 1.2 GB GPU memory usage (1.3G multiplication operations and 5.1G addition operations per 30 frames).

(a) Car detection (b) Person detection

Fig. 4. Examples of object detections.

T-YOLO can recognize multiple target objects in one image (Fig. 4). First, T-YOLO divides the input image into 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, the total number of target objects appearing in the video frame can be obtained.

As the filter is shared among all video streams, T-YOLO needs to traverse each T-YOLO queue of all streams one by one and extracts 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 to identify 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 from CPU memory to GPU memory (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 in the later stages, to facilitate the understanding of the scene.

3.4 About False Negatives

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

Take SNM stage as an example, false negatives in this stage can be categorized into two cases. On one case where a merely partial appearance of target objects (Fig. 5a), i.e., incomplete target object (e.g., head of vehicle) appears, can be identified by the Reference NN but missed by the filters (e.g., SNM and T-YOLO). Nevertheless, its subsequent frames in the same scene can contain entire target objects that SNM and T-YOLO are able to correctly detect (Fig. 5b). In this case, the scene is considered correctly detected. On the

(a) Partial appearance of the object (b) Entire object

Fig. 5. A false negative case but without missing scene.

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. Similarly, in T-YOLO stage, not identifying all target objects can also cause false negatives.

Considering false negatives do not necessarily result in scene loss, so the FN in Equation (9) refers to the number of false negatives that actually cause the scene loss in practice.

In fact, if we slightly relax the filtering condition of a filter (e.g., set the filtering threshold slightly below the target threshold required by the models or anomalous events) and forward a little more frames to the follow-up filters (the latter filter can detect the results of the previous again), the false negatives could be reduced. Therefore, the cascaded structure and relaxed filtering conditions can prevent excessive filtering errors. And we elaborate on these issues in Section 5.3.

4 Implementation

4.1 Training SDD and SNM

We apply the model training method mentioned in NoScope [24] to train specialized models (i.e., SDDs and SNMs) for each video stream. For a video stream, we first extract a representative fraction of the video and label its video frames by using YOLOv3. These labeled data are divided into two subsets as a training dataset and a test dataset. The former is used to train a SDD and a 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 execution speed by comparing the predicted results against the real truths.

Different from NoScope, which uses specialized models 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, the raw frames need to be resized to meet the filter's default feature size. The resizing times of SDD, SNM, and T-YOLO are about 40, 150, and 400 us respectively. Accordingly, the raw frames always exist in the system unless these frames are filtered by any one filter.

4.2 Filter Control

4.2.1 FilterDegree

Although the two thresholds in SNM, c_{low} and c_{high}, basically determine the prediction 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:

$$t_{pre} = (c_{high} - c_{low}) \times FilterDegree + c_{low}. \tag{11}$$

FilterDegree is a parameter set by users, which reflects the aggressiveness of filtering in SNM stage. When $FilterDegree = 1$ ($t_{pre} = c_{high}$), the output frames have a high credibility, but it increases the probability of false negatives. 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 ratio of SNM 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 percent) or a remarkable decrease of throughput (e.g., 60 percent).

4.2.2 NumberofObjects

NumberofObjects refers to the filtering threshold in T-YOLO stage, which controls the filtering conditions of the T-YOLO model. Normally, *NumberofObjects* is required to be equal to the intensity of the target objects in the predefined events (i.e., target threshold). But *NumberofObjects* may be less than the target threshold in the case of relaxing filtering condition.

In the running process of the T-YOLO filter, 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 cannot only monitor the intensity of the target object, achieving purposeful filtering based on user needs, but also catch and correct the false positives 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 positives of FFS-VA.

4.3 System Optimization

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

4.3.1 Feedback Queue

Note that the processing speed of one type of filter is often different from that of another. SDD processes about 10× faster than SNM and 100× faster than T-YOLO. Because of the fluctuation of video contents over time, the number of video frames processed by each filter also varies over time. When frames arrive in bursts, their processing filter threads are able to compete for hardware resources with each other seriously and even block.

Sine resource competition between different filters exists in both intra-stream and inter-stream processes, it is necessary to balance the execution speeds of these filters at runtime. Therefore, we propose a global feedback-queue approach. First, FFS-VA controls the detecting speed of a filter at 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 depth is blow the threshold. As long as the system can keep at least 30 FPS for each video stream, the stream is being analyzed in real time. In addition, for some video streams, the number of target-object frames varies greatly over time. To balance loads among video streams, T-YOLO filter extracts a different number of frames from different queues in one cycle. For the video streams with a long T-YOLO queue depth, FFS-VA extracts up to 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.

The setting of the queue depth thresholds is important. Too small an threshold may reduce the detection speeds of filters (e.g., s_2), even the system throughput $Xput$, due to the filters at later stages cannot get enough frames (e.g., a batch) from their queues at a time. On the contrary, too large an threshold will increase feasible overloads and queue time L_{wait} since too many frames are cached in the queue. Unless otherwise stated, we initially and empirically determine 2, 10, 10 and 10 as the queue depth thresholds of SDD queues, SNM queues, T-YOLO queues, and the Reference NN queue respectively.

4.3.2 Dynamic Batch

Since each video stream has its own SNM, it causes the GPU to load these models frequently when processing frames coming from different video streams. The data exchange overhead significantly lowers the overall computational efficiency. Intuitively, feeding a batch of frames to the GPU and processing them at a time can greatly reduce the amount of model and image data loading. For example, when the batch size is 30, the loading frequency is reduced by 30×.

Feedback-queue indeed achieves 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. First, if the number of frames in the queue is less than the batch size, in order to compose a batch of data, waiting for the remaining frames can introduce some delay (i.e., L_{wait}). In addition, even if enough frames exist in the queue, the batch processing itself can also introduce additional latency (i.e., $L_{service}$), especially for a large batch size. All of these will result in a long latency for the system.

In order to solve these problems, we propose a dynamic batch technique based on the feedback-queue approach. If there are enough video frames in the SNM queue, to get a high throughput, SNM pops out a batch of (*BatchSize*, e.g., 30) images from the queue for SNM prediction. Otherwise, the frames are popped from the SNM queue until the queue is empty to guarantee a low system latency. Note that although batch processing is used in SNM, when we pop (or push) frames from (or to) a queue, the chronological order of video frames needs to be strictly guaranteed. We elaborate the reasons in Section 4.4. In fact, dynamic batch can also be applicable to other CNN models for a better trade-off between throughput and latency. Experimental results show that compared with using the feedback-queue approach alone, the dynamic batch technique reduces the average latency by 27 percent while lowering the throughput by only 6 percent, ensuring better real-time performance.

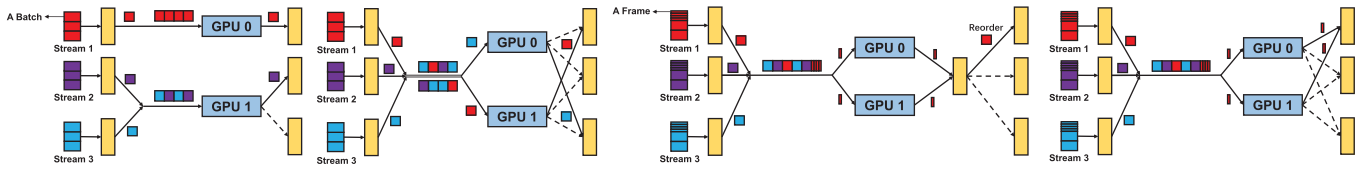
4.4 Scalability

We previously described the design of FFS-VA based on the typical system with two GPUs. Indeed, FFS-VA also can scale to the server with more GPUs, to provide larger processing capacity for both offline and online analysis. It introduces a new challenge how to reasonably and dynamically allocate streams and their relevant stages across multiple GPUs with workload variation, to ensure the high efficiency of hardware.

As aforementioned, the models in the latter two stages are highly consumptive in computation (e.g., T-YOLO at 200 FPS and Reference NN at 30 FPS per GPU). When $TORs$ of the streams are in a low level, only a few of frames can arrive at T-YOLO and then Reference NN. In this case, two working GPUs are enough. Nevertheless, when target events occur in one or more streams immediately, the loads can overload these two GPUs. The extra idle GPUs are gradually activated to share the heavier-load stages.

Specifically, FFS-VA adopts a utilization-based scheduling method. It measures the GPU utilizations of the T-YOLO stage and Reference NN stage every three seconds. When the average utilization of a GPU (or GPUs) performing T-YOLO stage in five minutes exceeds a threshold (e.g., 75 percent), FFS-VA activates an idle GPU to join this stage. When the average utilization is lower than 20 percent, one of the working GPUs will be deactivated. It is similar for the stage of Reference NN. To ensure normal operations of the system, FFS-VA guarantees that at least one GPU runs on each of these two stages. We design two interfaces *ActivateGPUDevice()* and *InactivateGPUDevice()* to be responsible for the allocation and release of the two models on GPUs at runtime. The specific operations are implemented by traversing each layer of the NN model as well as calling CUDA interfaces *cudaMalloc()*, *cudaMemcpy*, and *cudaFree()*.

When multiple GPUs share the same stage, FFS-VA needs to further determine how to assign frames of streams to the GPUs. Notice that the chronological order of frames in a stream should be strictly kept after these frames passing a stage, thereby guaranteeing the functional correctness of FFS-VA. To achieve this goal, we present four parallel schemes according to the scheduling granularity (stream, batch in a stream, or frame in a batch) and the parallel mode of GPUs (asynchronous or synchronous) as SMa, BMa, FMa, and FMs (Fig. 6). *SMa* means that each stream (stream granularity) is fixedly allocated to a GPU. In this case, each GPU traverses its own involving streams and extracts a batch of frames for processing. *BMa* means that each GPU can serve all streams but merely alternatively and exclusively extracts a batch of frames (batch granularity) in a stream at a time. *FMa* means that all GPUs work for a batch of frames (frame granularity) in a stream simultaneously in a frame-level asynchronous manner. The specific number of processing the frames depends on the computing power of the GPUs. However, the processed frames need to be reordered in a buffer to maintain their chronological order. *FMs* means that all GPUs work for a batch of frames (frame granularity) in a stream simultaneously but in a frame-level synchronous way. These frames are orderly allocated to all GPUs in turn, and each GPU gets one video frame at a time.



(a) SMa (b) BMa (c) FMa (d) FMs

Fig. 6. Four parallel schemes are presented according to the scheduling granularity and the parallel mode.

These four schemes have their own advantageous scenarios. FMa and FMs are advantageous in the cases with small number of streams since all GPUs can serve one stream simultaneously. SMa has a better latency due to a batch of frames is serially processed without inter-frame synchronization and reordering process. BMa does well in heterogeneous environments. FFS-VA can well support these four parallel schemes. The experimental performance analysis for these four schemes will be further explained in Section 5.5.

5 Evaluation

5.1 Experimental Setup

We use two real-world public videos, *Jackson* and *Coral*, as our evaluation workloads. *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. Their relevant information is summarized in Table 1. Each video contains about one 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. 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 YOLOv3 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 four NVIDIA Geforce GTX 1080 GPUs, two GTX Titan X 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 system performance of FFS-VA in online and offline situation. Second, we analyze the sensitivity of filtering thresholds in FFS-VA to the overall system accuracy and filtering ratio. And then, we study the impact of batch techniques on the throughput and latency. Finally, the performance of FFS-VA deployed into a multi-GPU environment is demonstrated. For the sake of simplicity, we select 5,000 consecutive frames from each video stream to perform the inference tasks.

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%

5.2 System Performance

Fig. 7 shows the average throughput and latency as a function of the number of video streams with 0.129 TOR on two GTX 1080 GPUs.

Offline Analysis. With a single video stream, which represents the offline analysis performance, the maximum throughput that FFS-VA can support is 592 FPS, which is 10× and 6× that supported by YOLOv3 and YOLOv2 respectively. Compared with YOLOv3 the total execution time of FFS-VA is reduced by 90.9 percent. 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-resolution video files.

Online Analysis. Experiments show that our system can support up to 31 video streams for real-time detection, which is 15 × more than what YOLOv3 can support. Although the throughput of YOLOv2 is about 2× than YOLOv3 in the offline situation, both of them can only support real-time analysis of no more than 3 concurrent video streams. Besides, dynamic batch technique has a 27 percent lower latency than using feedback-queue approach alone in most cases, but at the cost of 6 percent reduction in throughput caused by the small batch size. Moreover, it is necessary to note that although FFS-VA has a latency of several seconds, these delays are insignificant and tolerable in some applications, such as intelligent video surveillance [26].

For video streams with 1.000 TOR as an extreme case, Fig. 8 shows that FFS-VA can only support 5 video streams in real time. This is because 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 addition, the offline detection throughput has also dropped noticeably in our experimental platform, and the overall execution time is only 67 percent shorter than the YOLOv3. This is because, on the same hardware platform, only T-YOLO performs efficient filtering on one GPU and another GPU is used to perform YOLOv3, while the baseline YOLOv3 can perform on both two GPUs.

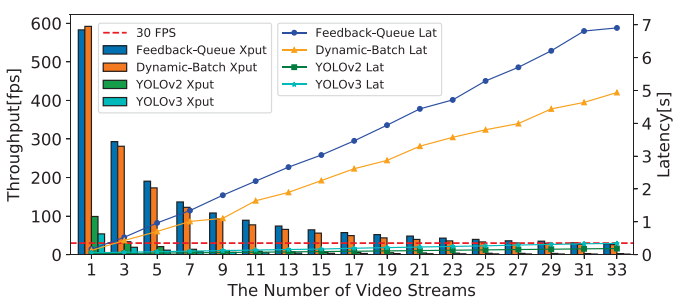


Fig. 7. The average throughput and latency as a function of the number of video streams with a TOR value of 0.129.

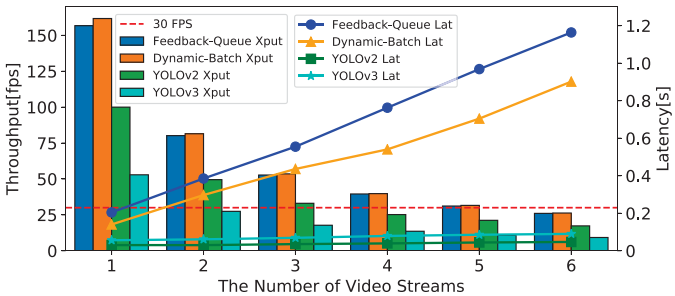


Fig. 8. The average throughput and latency as a function of the number of video streams with a TOR value of 1.000.

Fig. 9 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 video contents change in the daytime. The filtering ratio of SNM is largely related to the TOR. And T-YOLO can all work well in any case. It is worth noting that different time periods, weather, occlusion, video contents, illumination, etc., may all affect the performance of the three filters, and we only show a small part of them here.

5.3 Sensitivity of Key Thresholds

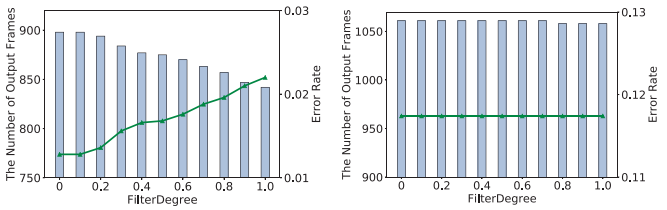
Next, we use offline video streams to examine how two key thresholds, *FilterDegree* and *NumberofObjects*, impact the filtering ratio and accuracy of FFS-VA. To verify the accuracy, all the filtered frames by FFS-VA are completely detected by the reference model YOLOv3.

5.3.1 FilterDegree

Fig. 10a illustrates the effect of the *FilterDegree* threshold on the number of output frames and the filtering error rate for the car detection (TOR=0.186). As the threshold increases, more frames whose prediction probability c is between c_{low} and c_{high} are filtered out, so the error rate also rises accordingly. Fig. 10b shows the results of person detection (TOR=1.000). The adjustment of the *FilterDegree* value has little effect on the filtering ratio and error rate 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.

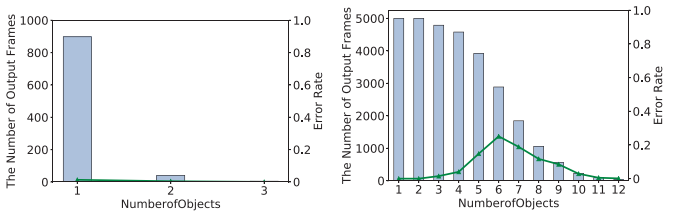
5.3.2 NumberofObjects

In Fig. 11a, for car detection, as *NumberofObjects* increases, the number of output frames decreases significantly (about 80 percent). 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. Fig. 11b illustrates the case of person detection. The number of output frames gradually decreases with the increase of *NumberofObjects*.



(a) Car detection (TOR=0.186) (b) Person detection (TOR=1.000)

Fig. 10. The throughput and error rate as a function of *FilterDegree*.



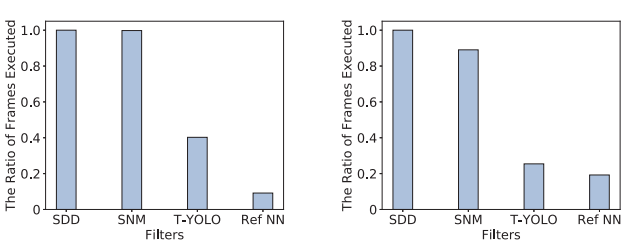
(a) Car detection ($TOR = 0.186$) (b) Person detection ($TOR = 1.000$)

Fig. 11. Number of output frames and error rate as a function of *NumberofObjects*.

5.3.3 Accuracy Analysis

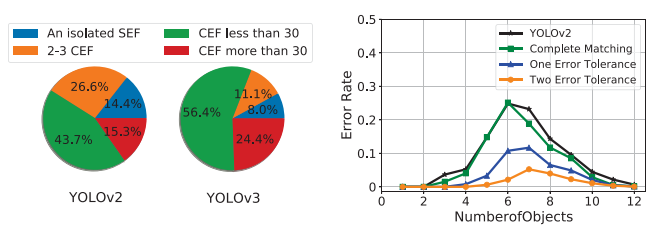
To better understand the error rate, we first analyzed the false-negative frames for car detection with a TOR value of 0.186. Fig. 12a illustrates the statistics of these error frames. The cases of one isolated single-error frame (SEF) and 2-3 continuously-error frames (CEF) do not affect the correct 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 YOLOv3. In this case where there are many consecutive error frames, it is because a single partially appeared vehicle is waiting for traffic lights. By analyzing these images, we observe that about 50 frames out of a total of 5,000 frames are those with actual scene losses ($< 2\%$). In addition, by comparing the statistics of error frames based on YOLOv2 and YOLOv3, we find that the proportion of one isolated SEF and 2-3 CEF in YOLOv3 is smaller, and YOLOv3 has a stronger and more stable recognition ability than YOLOv2.

In addition, as shown in Fig. 11b, 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



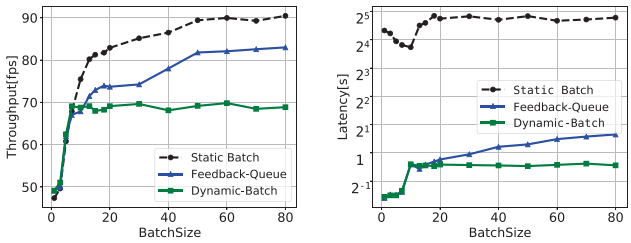
(a) Car detection (TOR=0.435) (b) Person detection (TOR=0.259)

Fig. 9. The ratio of frames executed in each filter. The processing speeds of the four stages at runtime are about 20K FPS, 2K FPS, 200 FPS, and 30 FPS, respectively.



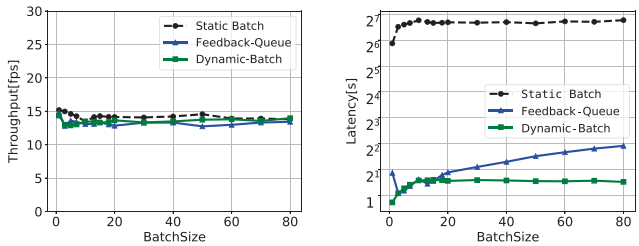
(a) YOLOv2 vs. YOLOv3 (b) Relaxing Filtering Conditions

Fig. 12. (a) Statistics of error frames in 5000 consecutive video frames. (b) The error rate as a function of NumberofObjects by relaxing filtering conditions.



(a) Throughput (b) Latency

Fig. 13. Throughput and latency under different batch techniques with TOR 0.203.



(a) Throughput (b) Latency

Fig. 14. Throughput and latency under different batch techniques with TOR 0.980.

identifies fewer target objects than YOLOv3, resulting in a high error rate. In this case, Fig. 12b shows that if one or two object misjudgment can be tolerated by relaxing the filtering threshold, the error rate will be greatly reduced (57.2 and 91.5 percent respectively). Even if relaxing filtering conditions may have a little impact on the filtering ratio of the system (about 12.6 and 22.2 percent respectively), it is worthwhile to ensure the overall accuracy of the system ($<$ 2%). Therefore, it exists a trade-off between accuracy and filtering ratio. Moreover, in the process of accuracy analysis for filtered frames, the error rate of using YOLOv2 as Reference NN is slightly higher than that of YOLOv3 in terms of complete matching. This is because YOLOv2 occasionally misses a few target objects during the detection process so that more frames are filtered incorrectly.

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

5.4 Batch Technique

We analyze the impacts of the static batch without feedback-queue, feedback-queue, and dynamic batch techniques on average throughput and latency over 10 video streams.

5.4.1 Throughput of FFS-VA

Fig. 13a shows the average throughput of the static batch, feedback-queue, and the dynamic batch techniques 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 impact on the throughput. When *BatchSize* is high, for the static batch technique, the throughput can still continue to increase with sufficient data provided by SNM queues. For the feedback-queue approach, the wait for a batch of frames increases the execution time, resulting in a slight decrease in throughput (about 8 percent). For the dynamic batch technique, the smaller batch size further leads to a decrease in computation efficiency.

Fig. 14a shows the experimental results with 0.980 TOR. In this case, most of the frames are eventually executed by T-YOLO model no matter what the *BatchSize* value is. Therefore, *BatchSize* has little effect on the throughput.

5.4.2 Average Latency

Fig. 13a shows that when *BatchSize* is small, the difference in average latency between the feedback-queue and the dynamic batch techniques 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 technique, since the batch size can be adjusted automatically according to video contents, the average latency is basically unchanged. Due to the same queue management method, for video streams with 0.980 TOR, the average latency has a similar trend in Fig. 14b.

In summary, for videos with a low TOR value, the feedback-queue approach has a greater throughput, and the dynamic batch technique 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 techniques, but the dynamic batch technique has a lower average latency and should be considered first.

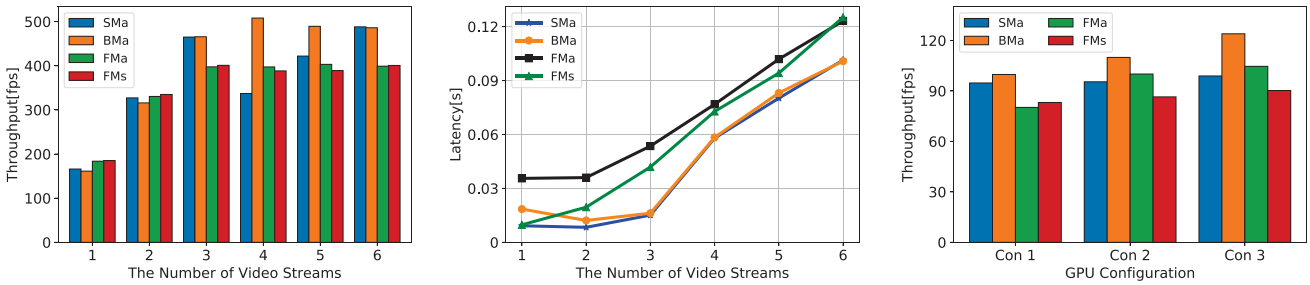
5.5 Scalability

We take T-YOLO stage as an example to evaluate the throughput and latency of four GPU parallel schemes on a server with three homogenous GPU devices. Then, for heterogeneous GPU devices, the performance of these four parallel schemes on the Reference NN stage is demonstrated. Finally, we show the scale performance of system as a function of TOR in online and offline cases.

Throughput. In Fig. 15a, since all GPUs can serve the same video stream at a time, FMa and FMs work well when there is a few video streams in FFS-VA. As the number of video streams increases, BMa achieves the best performance. This is because, in BMa, multiple GPUs only need to alternatively process a batch of frames of streams, without synchronizing and reordering process. For SMa, due to uneven distribution of streams on the GPUs, only when the number of video streams is an integer multiple of the number of GPUs, FFS-VA can achieve the load balance and the peak performance.

Latency. In Fig. 15b, due to without inter-frame synchronization (i.e., smaller L_{sync}), SMa and BMa exhibit a short latency. Switching video streams over multiple GPUs makes the latency of BMa slightly higher than the SMa. Besides, the extra reordering process also significantly increases the latency of FMa.

Heterogeneous GPU Environment. Fig. 15c shows the throughput of the four parallel schemes on the Reference NN stage over three GPU configurations: (1) three GTX 1080 GPUs (Con1); (2) two GTX 1080 GPUs and one GTX Titan X GPU (Con2); and (3) one GTX 1080 GPU and two GTX Titan X GPUs (Con3). Since GTX Titan X GPU is more powerful than GTX 1080 GPU, the throughput of Con3 is higher than Con2 and Con1 in all cases. However, due to



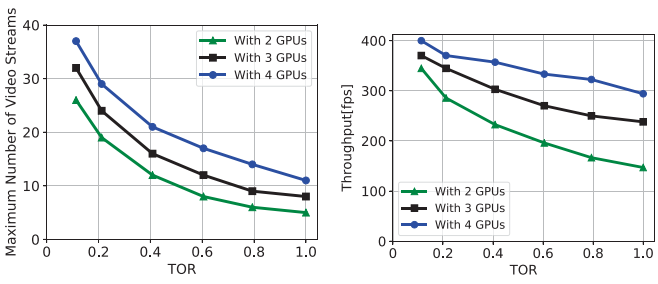
(a) Throughput (b) Latency (c) Heterogeneity

Fig. 15. The performance of four GPU parallel schemes.

frequent inter-frame synchronization and inter-stream synchronization, the slowest GPU (or GPUs) limits the overall throughput of the system. Therefore, improving GPU configurations have little impact on the performance of FMs and SMa. In contrast, the throughput of BMa is improved considerably as the GPU capacity improves.

Scale Performance. Based on the previous analysis, TOR has a pivotal impact on the maximum number of video streams supported by FFS-VA. Therefore, we discuss the impact of the number of GPUs on the overall performance under different $TORs$ with BMa. In fact, we assign two, three, and four GTX 1080 GPUs to FFS-VA respectively in the three experiments. Fig. 16a shows that the maximum number of video streams supported by FFS-VA increases as TOR decreases in the online case. Moreover, for a system configured with four GPUs, the overall performance has been improved by up to 122 and 45 percent than the system configured with two GPUs and three GPUs respectively. In addition, we also measure the impact of the number of GPUs on the latency. The results show that the latency remains essentially the same regardless of the number of GPUs. This is because, on the one hand, additional GPUs increase the detection speeds of filters but do not reduce processing time (i.e., $L_{service}$). On the other hand, the synchronous time L_{sync} of BMa is too small compared with the processing time $L_{service}$ and the queue time L_{wait}.

For offline scenarios, in Fig. 16b, as TOR increases, the throughput also shows a downward trend. When TOR is small in a video slice, few target-object frames exist, and the throughput is mainly determined by the detection speed of SNM. As a result, adding GPUs to T-YOLO stage and Reference NN stage has little effect on the throughput. With TOR increasing, growing frames arrive at the latter two stages. Compared with the server with two GPUs, deploying three and four GPUs can improve the throughput by 62 and 100 percent respectively. However, even with utilization-based scheduling method to allocate stages dynamically at runtime, using extra GPUs does not bring the same amount of performance gains. Due to the existence of scheduling overheads, the throughput improvement from adding the second GPU is about 20 percent lower than adding the first GPU. With more additional GPUs, new performance bottlenecks are gradually emerging.



(a) Online (b) Offline

Fig. 16. The scale performance as a function of TOR in online and offline cases.

5.6 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 $TORs$ increase in video streams can lead to poor filtering ratio, 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 a storage system, to be processed later. For some latency-sensitive scenes, it is necessary to use more GPUs or a server cluster to provision for peak-load periods.

Error Rate. The reason for the cases of relative high error rate is the performance difference between T-YOLO and the reference model YOLOv3. Deep compression [27] (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 [28]. Therefore, we can replace T-YOLO with a high-accuracy 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 six main categories, model cascades, object detection, video monitoring, model specialization, stream processing, and video analysis.

Model Cascades. Cascade is defined as a sequence of classifiers to improve inference speed. Paul Viola *et al.* [29] 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. [30] achieves an optimal trade-off between accuracy and speed by learning a complexity aware cascade. [31] configured a CNN cascade for real-world face detection to accurately differentiate faces from the backgrounds. [32] 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 [33] and Fast R-CNN [18] have achieved a very high accuracy for the image object detection, by using region proposal object detection methods. OverFeat [34] and YOLO [35] 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, including vehicle tracking [36], object detection [37] and so on. Each task has been tailored for a specific system (e.g., vehicle counting [38], license plate detection [27], cars or pedestrians tracking [39]). 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.

Model Specialization. Compared with generic models, specialized models can guarantee a high throughput and accuracy by sacrificing generality in the same hardware environment. Both NoScope [24] and Foucs [40] use this technique in cheap CNN to help query for target-object frames in a off-line video faster. In contrast, FFS-VA pays more attention to the analysis of more real-time video streams through model specialization. In addition, FFS-VA also analyzes the impact of video content fluctuation (e.g., *TOR*) on real-time system performance instead of just getting the target-object frames.

Stream Processing. The general stream processing challenges, such as distributed execution, fault tolerance, and real-time performance, have received widespread attention in various stream processing system [41]. In contrast, FFS-VA focuses on supporting more video streams on the same hardware device for high-accuracy fine-grained analysis by utilizing CNN models, which is orthogonal to these advanced works.

Video Analysis. In terms of information retrieval on videos, there are several techniques widely used to analyze video contents, such as semantic video search [42], spatio-temporal information-based video retrieval [43], and shot boundary detection [44]. The main goal of FFS-VA is to build a filtering system for video analysis. Benefit from low coupling structure and strict order guarantee for video frames, these advanced video analysis methods can be integrated into one stage of our system conveniently if necessary.

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 neural network. 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 stream, 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 5-15× scalability improvement over the state-of-the-art YOLOv3. 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.

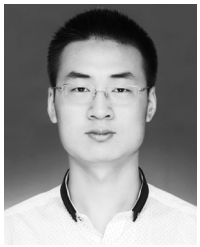
Acknowledgments

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. This work was supported in part by National key research and development program of China (Grant No. 2018YFA0701804), Creative Research Group Project of NSFC No.61821003, NSFC No. 61872156, Fundamental Research Funds for the Central Universities No. 2018KFYXKJC037, the US National Science Foundation under Grant No.CCF-1704504 and No.CCF-1629625, and Alibaba Group through Alibaba Innovative Research (AIR) Program.

References

[1] Y. Lin *et al.*, "Large-scale image classification: Fast feature extraction and SVM training," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2011, pp. 1689–1696.
[2] Z. Wang, Z. Wang, H. Zhang, and X. Guo, "A novel fire detection approach based on CNN-SVM using tensorflow," in *Proc. Int. Conf. Intell. Comput.*, 2017, pp. 682–693.
[3] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 580–587.
[4] M. Everingham, L. J. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman,"The pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010.
[5] W. Liu *et al.*, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.*, 2015, pp. 21–37.
[6] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 379–387.
[7] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 6517–6525.

[8] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767
[9] H. Wang, K. Rudy, J. Li, and D. Ni, "Calculation of traffic flow breakdown probability to optimize link throughput," *Appl. Math. Modelling*, vol. 34, no. 11, pp. 3376–3389, 2010.
[10] A. OttlikandH.-H. Nagel, "Initialization of model-based vehicle tracking in video sequences of inner-city intersections," *Int. J. Comput. Vis.*, vol. 80, no. 2, pp. 211–225, 2008.
[11] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of CNN and RNN for natural language processing," *CoRR*, vol. abs/1702.01923, 2017. [Online]. Available: http://arxiv.org/abs/1702.01923
[12] L. Huang, Z. Li, andB. Wang, "Detection of abnormal traffic video images based on high-dimensional fuzzy geometry," *Autom. Control Comput. Sci.*, vol. 51, no. 3, pp. 149–158, 2017.
[13] L. Li, L. Chen, X. Huang, and J. Huang, "A traffic congestion estimation approach from video using time-spatial imagery," in *Proc. 1st Int. Conf. Intell. Netw. Intell. Syst.*, 2008, pp. 465–469.
[14] H. Wei, C. Yang, and Q. Yu, *Efficient Graph-Based Search for Object Detection*. Amsterdam, The Netherlands: Elsevier, 2017.
[15] G. Mo and S. Zhang, "Vehicles detection in traffic flow," in *Proc. Int. Conf. Netw. Comput.*, 2010, pp. 751–754.
[16] N. Ballas, L. Yao, C. Pal, and A. C. Courville, "Delving deeper into convolutional networks for learning video representations," in *Proc. 4th Int. Conf. Learn. Representations*, Y. Bengio and Y. LeCun, Eds. 2016. [Online]. Available: http://arxiv.org/abs/1511.06432
[17] L. Jiang, M. Xu, and Z. Wang, "Predicting video saliency with object-to-motion CNN and two-layer convolutional LSTM," *CoRR*, vol. abs/1709.06316, 2017. [Online]. Available: http://arxiv.org/abs/1709.06316
[18] R. B. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1440–1448.
[19] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 91–99.
[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Representations*, Y. Bengio and Y. LeCun, Eds. 2015. [Online]. Available: http://arxiv.org/abs/1409.1556
[21] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[22] Jackson, "Town-square-southwest," 2017. [Online]. Available: https://www.seejh.com/webcams/jacksonhole/jackson/town-square-southwest
[23] N. E. Buch, S. A. Velastin, and J. Orwell, "A review of computer vision techniques for the analysis of urban traffic," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 3, pp. 920–939, Sep. 2011.
[24] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "NoScope: Optimizing neural network queries over video at scale," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1586–1597, 2017.
[25] A. Zainab, "Real-time object detection," 2017. [Online]. Available: https://blog.mindorks.com/detection-on-android-using-tensorflow-a3f6fe423349
[26] A. C. Nazare Jr, and W. R. Schwartz, "A scalable and flexible framework for smart video surveillance," *Comput. Vis. Image Understanding*, vol. 144, no. C, pp. 258–275, 2016.
[27] C. Anagnostopoulos, I. Anagnostopoulos, I. D. Psoroulas, V. Loumos, and E. Kayafas, "License plate recognition from still images and video sequences: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 9, no. 3, pp. 377–391, Sep. 2008.
[28] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. Annu. Int. Symp. Comput. Archit.*, 2016, pp. 243–254.
[29] P. A. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2001, pp. 511–518.
[30] Z. Cai, M. J. Saberian, and N. Vasconcelos, "Learning complexity-aware cascades for deep pedestrian detection," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 3361–3369.
[31] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A convolutional neural network cascade for face detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 5325–5334.
[32] Y. Sun, X. Wang, and X. Tang, "Deep convolutional network cascade for facial point detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 3476–3483.
[33] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 346–361.
[34] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated recognition, localization and detection using convolutional networks," in *Proc. 2nd Int. Conf. Learn. Representations*, Y. Bengio and Y. LeCun, Eds. 2014. [Online]. Available: http://arxiv.org/abs/1312.6229
[35] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 779–788.
[36] B. Tian, Q. Yao, Y. Gu, K. Wang, and Y. Li, "Video processing techniques for traffic flow monitoring: A survey," in *Proc. Int. IEEE Conf. Intell. Transp. Syst.*, 2011, pp. 1103–1108.
[37] J. B. KimandH. J. Kim, "Efficient region-based motion segmentation for a video monitoring system," *Pattern Recognit. Lett.*, vol. 24, no. 1–3, pp. 113–128, 2003.
[38] H. Zhang, G. Ananthanarayanan, P. Bodík, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2017, pp. 377–392.
[39] B. Babenko, M. Yang, and S. J. Belongie, "Robust object tracking with online multiple instance learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 8, pp. 1619–1632, Aug. 2011.
[40] K. Hsieh *et al.*, "Focus: Querying large video datasets with low latency and low cost," in *Proc. USENIX Conf. Operating Syst. Des. Implementation*, 2018, pp. 269–286.
[41] H. Zhang, G. Ananthanarayanan, P. Bodík, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2017, pp. 377–392.
[42] H. H. Kim and Y. H. Kim, "Semantic video search using tagsonomies," in *Proc. 73rd ASIS&T Annu. Meet. Navigating Streams Inf. Ecosyst.*, 2010, pp. 1–2.
[43] S. Verstockt, O. Janssens, S. V. Hoecke, and R. V. de Walle, "Spatio-temporal video retrieval by animated sketching," in *Proc. Int. Conf. Comput. Vis. Theory Appl.*, 2013, pp. 723–728.
[44] S. H. Abdulhussain, A. R. Ramli, M. I. Saripan, B. M. Mahmmod, S. A. R. Al-Haddad, and W. A. Jassim, "Methods and challenges in shot boundary detection: A review," *Entropy*, vol. 20, no. 4, 2018, Art. no. 214.



Chen Zhang received the BS degree in electromagnetic wave propagation and antenna from Xidian University, Xi'an, China, in 2016. He is currently working toward the PhD degree in the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China. His research interests include video analysis and neural network.

Qiang Cao (Senior Member, IEEE) received the BS degree in applied physics from Nanjing University, Nanjing, China, in 1997, and the MS degree in computer technology and the PhD degree in computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2000 and 2003, respectively. He is currently a full professor of the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology. His research interests include computer architecture, large scale storage systems, and performance evaluation. He is a senior member of the China Computer Federation (CCF) and a member of the ACM.

Hong Jiang received the BSc degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1982, the MASc degree in computer engineering from the University of Toronto, Toronto, Canada, in 1987, and the PhD degree in computer science from the Texas A&M University, College Station, Texas, in 1991. He is currently a chair and Wendell H. Nedderman endowed professor with Department of Computer Science and Engineering, University of Texas at Arlington. His present research interests include computer architecture, computer storage systems and parallel I/O, high performance computing, big data computing, cloud computing, and performance evaluation.

Wenhui Zhang (Student Member, IEEE) received the BS degree in mathematics from Wuhan University, Wuhan, China, in 2011. He is currently working toward the PhD degree at the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China. His research interests include erasure codes, storage systems, and parallel algorithms. He is a student member of the ACM.

Jingjun Li received the BS degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2016. He is currently working toward the master's degree at the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology. His main research interests include computer architecture, machine learning, and large scale key-value storage systems.

Jie Yao (Member, IEEE) received the BS degree in computer science and technology and the MS, and PhD degrees in computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2001, 2004, and 2009, respectively. He is currently a lecturer with the Huazhong University of Science and Technology. His research interests include computer architecture, large scale storage systems, and performance evaluation. He is a member of the China Computer Federation (CCF) and ACM.