Real-Time Video Analytics: The Killer App for Edge Computing

Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha, Microsoft Research

Video analytics will drive a wide range of applications with great potential to impact society. A geographically distributed architecture of public clouds and edges that extend down to the cameras is the only feasible approach to meeting the strict real-time requirements of large-scale live video analytics.

According to a 2015 report by the Information Handling Services on the installed base for video surveillance equipment, there is a camera installed for every 29 people on the planet, with mature markets having a camera for every 8 people. The report predicts that the number of cameras will grow by 20 percent year over year for the next 5 years. Video analytics from these cameras are used for traffic control, surveillance, and security in both public and private venues, as well as consumer applications including digital assistants for real-time decisions.

We propose that a geographically distributed architecture of public clouds, private clusters, and edges that extend down to the cameras is the only approach that can meet the strict real-time requirements of large-scale video analytics, which must address latency, bandwidth, and provisioning challenges.

First, applications require very low latency when processing video because the output of the analytics is used to interact with humans (such as in augmented reality scenarios) or to actuate some other system (such as traffic lights). Second, high-definition video requires large bandwidth—5 Mbps or even 25 Mbps for 4K video—and streaming a large number of video feeds directly to the cloud might not be feasible. When cameras are connected wirelessly, such as inside a car, the available uplink bandwidth is very limited. Finally, using compute capacities available on the camera itself allows for correspondingly lower provisioning (or usage) in the cloud. This also means that less interesting parts of the video can quickly be filtered out, for example, using motion-detection techniques, which dramatically reduces the bandwidth that needs to be provisioned.

Aside from low latency and efficient bandwidth usage, another major consideration for continuous video analytics is video processing's high compute cost. Because of high data volume, compute demands, and latency requirements, cameras are the most challenging "things" in the Internet of Things. Thus, large-scale video analytics could well be edge computing's "killer app." Tapping into the potential of recent dramatic increases in the capabilities of computer vision algorithms presents an exciting systems challenge.

We propose a real-time video analytics system with low resource costs that produces high-accuracy outputs. Although designed for generic usage, our system will initially focus on traffic planning and safety; this is because traffic accidents are known to be among the top 10 causes of all fatalities. Our video analytics system runs 24/7, processing live camera feeds from traffic intersections in Bellevue, Washington. The system generates directional traffic volumes and raises alerts on anomalous traffic patterns. We are on track to identify dangerous conflict patterns to minimize traffic deaths. We also plan to expand our solution to cities across the US and worldwide.

VIDEO ANALYTICS APPLICATIONS

Video analytics will drive a wide range of applications from surveillance and self-driving cars to personal digital assistants and drone cameras. We explain these applications and highlight their performance requirements. The state of the art is to deploy expensive custom solutions with a hard-coded set of video analytics that typically does not take advantage of the geo-distributed edge infrastructure.

The applications described here have compute requirements ranging from a fraction of a core to several CPU cores—with some requiring a GPU to run complex deep neural networks (DNNs). Video bandwidth requirements vary from hundreds of Kbps to tens of Mbps. Many applications require latency below a second, whereas others require it as low as tens of milliseconds. A geo-distributed architecture is best suited to the low response times and high-bandwidth requirements.

Vision Zero for traffic

Traffic-related accidents are among the top 10 causes of death worldwide. To help reduce these fatalities, many cities have adopted the Vision Zero initiative that aims to eliminate traffic-related deaths (www.visionzeroinitiative.com). Large cities have hundreds or thousands of traffic cameras installed and can analyze video from these cameras for traffic safety and planning.

VIDEO ANALYTICS WILL DRIVE A RANGE OF APPLICATIONS FROM SURVEILLANCE AND SELF-DRIVING CARS TO DIGITAL ASSISTANTS AND DRONE CAMERAS.

Detecting "close calls" between cars, bikers, and pedestrians helps city planners to identify hazardous areas in which preemptive safety measures are needed. For example, detecting areas where people jaywalk can determine where to install crosswalks; detecting roads on which people frequently double-park can help identify areas that need congestion-relief planning; and detecting car, pedestrian, and bike traffic volume is useful in making adjustments to traffic-light controllers.

Self-driving and smart cars

For a self-driving car, a key enabler is the use of video from multiple onboard cameras to make low-latency driving decisions. This involves the ability to fuse multiple video feeds and quickly analyze them. In fact, today's smart cars (which include various sensors to aid the human driver) already use cameras to help make safety decisions about brakes and warning lights, as well as to alert the driver.

Output from traffic camera analytics can also trigger decisions in cars. As we demonstrated at Hannover Messe in 2016,[1] a traffic light–mounted camera can detect hard-to-see pedestrians (for example, those standing between parked cars) and can warn approaching self-driving cars using dedicated short-range communication (DSRC).

Personal digital assistants

Digital assistants that offer personalized and interactive experiences are an important emerging technology that could dramatically alter our daily activities. These assistants can be incorporated into smart mobile devices or standalone hardware in the home (such as Amazon's Alexa or the so-called personal robot Jibo). They are equipped with cameras and are able to interact through facial recognition, gesture identification, and other activities. Naturally, video processing and analytics have to continuously execute with low latency, either locally on the device or with assistance from the remote cloud.

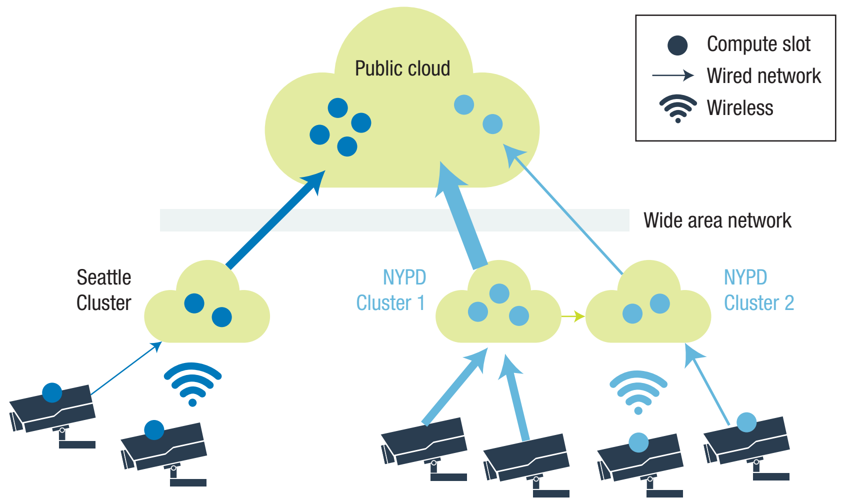


FIGURE 1. Geo-distributed video analytics infrastructure. Each organization deploys their own private cluster of different sizes, and relies on the public cloud for additional capacity. Network links between cameras, private clusters, and the public cloud have diverse bandwidths (represented by the width of the arrow).

Surveillance and security

Enterprise and government surveillance make up one of the largest deployments of cameras—there are millions in the US, the UK, and China. Increasingly, law enforcement officers use body-worn cameras, the feed from which can help identify suspicious license plates or people and alert the officer to call for backup and support.

Of course, cameras are becoming widely adopted in home surveillance too. Companies such as Nest and Ring sell home surveillance cameras that identify motion and people to detect events like package deliveries, break-ins, and stray animals. Analysts predict that drone cameras will also significantly aid in home-surveillance activities.

Augmented reality

In augmented reality (AR) systems, users wear goggles that project additional information into their field of vision, either by projecting holograms (such as in Microsoft's HoloLens) or by recording surrounding views and rendering directly on top of them. To reach its full potential, an AR system needs to detect and track objects around the user, to recognize people and faces, and to perform additional video analytics. However, such analytics cannot be performed directly inside the headset because they require powerful hardware and could overheat. Instead, they could be offloaded to a nearby edge computer (such as a powerful PC in the living room) or to the cloud.

INFRASTRUCTURE DESIGN AND CHARACTERISTICS

To support video analytics across these highly varied applications, infrastructure must be designed to support the following characteristics: geo-distribution, to ensure analytics functionality across cameras, edges, private clusters, and public clouds (not just a central location); multitenancy, to capture and handle many queries per camera as well as queries across multiple cameras; and hardware heterogeneity, to flexibly manage a mix of processing capacities (in CPUs, GPUs, field-programmable gate arrays [FPGAs], and application-specific integrated circuits [ASICs]) and networks.

Many organizations—which include cities, police departments, and retail stores—manage large networks of cameras. Although in some cases all cameras are in a single building, in other cases they are distributed across the city, or perhaps in multiple cities, or even across the entire country. Many of these implementations would benefit from continuous video analytics on all of their live camera feeds (such as counting cars in all intersections for city-wide traffic planning).

Figure 1 illustrates the hierarchical geo-distributed infrastructure for video analytics. The infrastructure includes edge clusters and private clusters with heterogeneous hardware to decode video, detect objects, and perform other video analytics tasks. Edge devices can include the cameras themselves or processing units placed close to the cameras (such as at traffic intersections, in cars, or in buses). Each organization might manage private clusters of varying sizes while also using the processing in public cloud clusters such as Microsoft Azure. The edge, private, and public cloud clusters also differ in the type of hardware available. For example, some clusters (including the cameras) might include GPUs. Other types of hardware, such as FPGAs and custom ASICs,[2] will be in the public clouds.

The network connecting the cameras, edges, private clusters, and the public cloud is a critical resource. The bandwidth required to support a single camera ranges from hundreds of Kbps for low-resolution video to more than 10 Mbps for multi-megapixel cameras. Cameras are connected using a wired or wireless link (Wi-Fi or LTE) to the edge device or the private cluster. In most cases, the uplink bandwidths between the private clusters and the cloud are not sufficient for streaming a large number of high-resolution videos to the cloud.

Although the infrastructure in Figure 1 is general enough to capture all video analytics deployments, we believe that in practice deployments will vary from having all the analytics in the cloud (for small deployments of cameras), to all the analytics in private clusters (insufficient bandwidths to the cloud or privacy reasons), to a hybrid of edge/private clusters and the cloud (the most widely used setup, perhaps after denaturing the videos to maintain privacy),[3] to even having all analytics on the edge (or camera). Video analytics systems should be designed to function across this wide spectrum of deployments. This is analogous to big data analytics, which originally focused on executing on a single cluster but now executes across geo-distributed clusters.[4]

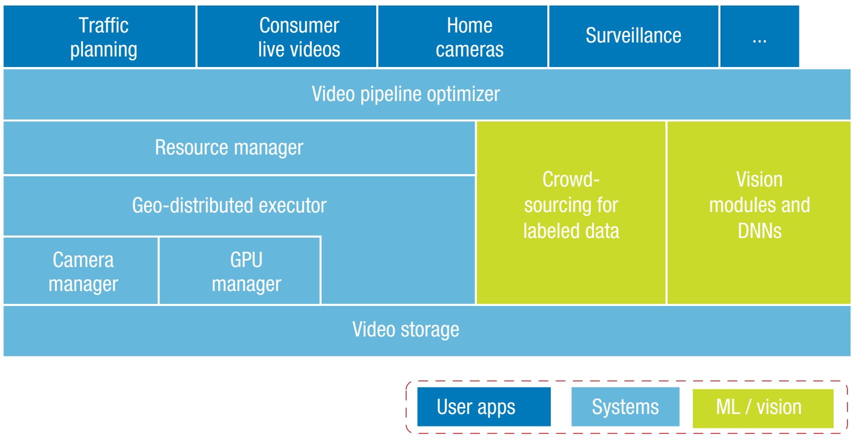


FIGURE 2. Rocket, our video analytics software stack. The video pipeline optimizer converts video queries to pipelines of vision modules invoking deep neural network (DNN) models, along with estimating the resource–accuracy profiles of the pipelines. The pipelines and their corresponding profiles are then passed on to the resource manager, which executes them over the geo-distributed collection of edges, private clusters, and the cloud. Rocket also uses a camera manager and GPU manager to virtualize the pan-tilt-zoom and GPU functionalities across multiple query pipelines. The video storage layer provides interactive querying on stored videos. ML: machine learning.

ROCKET: VIDEO ANALYTICS SOFTWARE STACK

We propose a video analytics software stack called Rocket to handle the varied inputs, as shown in Figure 2.

The *video pipeline optimizer* converts high-level video queries to video-processing pipelines composed of many vision modules; for example, a video decoder, followed by an object detector and then an object tracker. Each module implements predefined interfaces to receive and process events (or data) and then sends its results downstream. The modules also specify their configuration knobs that we modify dynamically as part of resource management. The modules can internally implement one of our application-level optimizations (described below in "Application-level optimizations").

The pipeline optimizer also estimates the query's resource–accuracy profile. Specifically, for the different configurations of knobs and implementations of each vision module in the pipeline, it computes the total resource cost and accuracy of the query. The optimizer invokes crowdsourcing to obtain the labeled data required to compute accuracy. The pipeline and its generated profile are then submitted to the *resource manager*.

The (logically) centralized global resource manager (described below in "Resource–accuracy profiles for scheduling") is responsible for all currently executing query pipelines and their access to all resources—CPU and GPU compute, network, and even pan-tilt-zoom (PTZ) parameters of steerable cameras. Periodically, the resource manager determines the best configuration of each query, and the placement of components across the available clusters, and allocates resources for each vision module. When appropriate, it also merges common modules across pipelines processing the same video stream. The pipelines execute through the *geo-distributed executor*, which runs across all the clusters.

CPU, network, and memory resources are allocated to modules

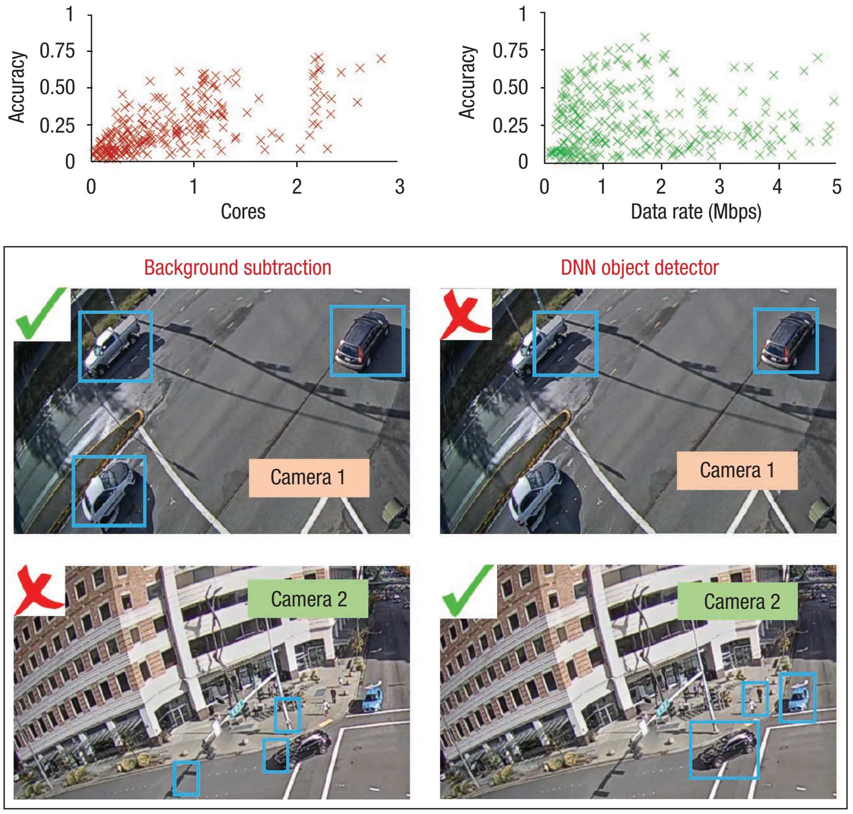


FIGURE 3. Resource–accuracy relationship. (a and b) Sample resource–accuracy plot of the canonical object tracker processing a video with an incoming rate of 30 frames/second—note the wide range in accuracy as well as the CPU demand and data rates (representing network demands). (c and d) The impact of using background subtraction versus a DNN object detector for object detection on different cameras.

using standard OS mechanisms, but custom mechanisms are used for GPUs and steerable cameras. Execution of DNNs on GPUs is made dramatically more efficient by running a *DNN execution service* on each machine that handles all DNN requests on the GPU (described below in "Efficient DNNs execution on GPUs"). In addition, a camera manager adjusts resolution, frame rate, and quality of the video in each camera, and for those that can be steered, the manager virtualizes across multiple query pipelines accessing the camera (described later in the "Virtualizing steerable cameras" section).

Video storage is another key component of our stack, and we intend to build upon ideas from prior work such as GigaSight[5] for faster cataloging and retrieval of videos.

CORE TECHNOLOGIES

Rocket's important technical features are designed to perform real-time, highly accurate, low-cost video analytics execution.

Resource–accuracy profiles for scheduling

Video analytics can have very high resource demands. Tracking objects in video is a core primitive for many scenarios, but even the best tracker in the Visual Object Tracking (VOT) Challenge of 2015 processed only 1 frame per second on an 8-core machine, whereas most cameras stream data at 30 frames per second. Some of the most accurate DNNs for object recognition (another core primitive) require 30 Gflops to process a single frame.[6] To reduce high processing costs and data transfers, we propose *approximation* as a first-order abstraction in our system—we carefully adjust the configuration of each query and the amount of allocated resources to control the query output accuracy.

A key property of video analytics is the resource–accuracy relationship. Vision algorithms typically contain various parameters, including frame resolution, frame sampling, and internal algorithmic parameters. In general, processing each frame in the video stream (without sampling) at the highest resolution results in the highest accuracy, but it also has the highest resource demand. Other parameters can include the specific DNN model to execute. Different parameter configurations determine the accuracy of the produced output as well as associated resource demand.

We abstracted all the application-specific parameters into the resource–accuracy profile. Figure 3 shows a sample resource–accuracy profile of the canonical object tracker processing a video with an incoming rate of 30 frames per second. We profiled

300 parameter configurations (such as frame sampling, resolution, and implementation choices) and compared it to the ground truth of the tracks obtained using crowdsourcing. Note the vast spread in accuracy as well as the CPU demand and data rates (which represent network demands). For each of the configurations, if the allocated resource is less than the demand, the analytics cannot keep up with the video stream's incoming rate.

Generating the resource–accuracy profile is challenging. Unlike SQL queries, there are no well-known analytical models to capture resource–accuracy relationships as they often depend on the specific camera view. Figure 3b shows how different object detection implementations can change efficacy in different cameras. Likewise, reducing video resolution might not reduce the accuracy of a license plate reader if the camera is zoomed in enough, but it would otherwise impact accuracy. Therefore, we need to generate the profile using a labeled dataset of ground truths for each video query pipeline for each camera. However, exhaustively running through all the configurations can be prohibitive. For instance, generating the profile for a license plate query consumed 20 CPU days.

To efficiently generate the profile, Rocket's resource optimizer[7] selectively explores only the promising configurations and avoids wasting CPU cycles exploring the accuracy of configurations that have really low accuracy or high resource demand. In profiling queries that are a pipeline of vision modules, it caches the intermediate results of modules and avoids re-running those modules. Then, to avoid an explosion of cache space, it profiles configurations with overlapping modules (or "prefixes") together, thus saving on both CPU cycles as well as caching.

The choice of configurations is fundamental to resource management because it dictates the resource demands. In a multitenant cluster with many video queries, our approach is to schedule for accuracy rather than taking the classic approach to schedule for fairness—as was adopted by modern cluster schedulers like Apache Yarn. Rocket's resource optimizer schedules thousands of video queries according to their resource–accuracy profiles as well as their tolerance to delay in processing to produce their results—that is, the time between arrival and processing of the frames.[7] It uses model predictive control and preferentially allocates resources to those queries that provide higher accuracy for the same quantum of allocated resources. Our system also decides on the placement of the modules in the pipeline and whether to execute each at the camera, private cluster, or cloud. This involves considering capacities of multiple resources (compute, network) at all locations.

Evaluation using real-world video analytics on live camera streams (HD streams) from traffic intersections and indoor surveillance cameras shows that our system

- achieves 80 percent better average query accuracy than the competing fair scheduler, and
- consumes 3.5-fold fewer CPU cycles for profile generation.

We previously published a detailed evaluation of Rocket's query optimizer and resource manager.[7]

TO REDUCE HIGH PROCESSING COSTS AND DATA TRANSFERS, WE PROPOSE APPROXIMATION AS A FIRST-ORDER ABSTRACTION IN OUR SYSTEM.

Efficient DNNs execution on GPUs

For many vision tasks, executing DNNs on GPUs is vital. The simplest model for this is an application that sets up and executes the corresponding operations on the GPU by using memory allocation and matrix computation functions provided by a library. Each invocation of the library leads to one or more distinct instances of computational kernels being lined up for execution by the GPU runtime. This simple approach, however, is inadequate in the common setting when several applications try to perform possibly distinct DNN computations on incoming video.

There are several possible sources for this complication. First, loading weight matrices corresponding to DNNs into GPU memory is a heavyweight operation, and the models might be large enough—ranging from 10 to 100 Mbytes—so that they cannot all be preloaded into memory. Therefore, it is important to consider when to load or evict individual models. Second, it is significantly more efficient to execute models in batch mode—meaning that the same model is executed in parallel on several inputs—than it is to present a series of heterogeneous models for execution to the GPU. Many applications share the models and inputs they execute, so it can be beneficial to combine models across multiple applications to reduce resource demand. Finally, some models are too large to execute on the edge (for example, the VGG16 model requires 30 Gflops), so they are best executed in the cloud, if latency permits. We implement the above optimizations in a DNN execution service, which runs on each machine with a GPU. All DNN requests run against this service, which reconciles these complexities across multiple applications.

Approximate scheduling relies on the fact that for each DNN, we can apply a set of optimizations to produce DNN variants with differing accuracy and resource use. The machine-learning community has proposed several optimizations, including replacing matrices with smaller factors,[8] limiting the number of bits used to represent model parameters,[9] and reducing the architectural complexity of models in systematic ways.[10] In addition, we have proposed new optimizations such as specialization—replacing large, slow models useful for classifying many classes (thousands of objects) with much smaller, faster models that handle just the classes observed at runtime (less than 10 objects). Such optimizations often reduce resource requirements significantly: the factorized VGG16 model consumes 4.9-fold fewer cycles at 0.5 percent accuracy loss.

[CAMERAS CAN BE ELECTRONICALLY STEERABLE (WITH PTZ CAPABILITIES) AND HAVE TO SUPPORT MULTIPLE APPLICATION QUERIES SIMULTANEOUSLY.]

Our system gathers these optimizations into a single "optimizing compiler" for DNNs.[6] It selects the appropriate DNN variant and its placement to optimize the accuracy, latency, and cost of the whole query pipeline (not just the DNN). We believe similar design considerations apply to emerging accelerators beyond the GPU such as FPGAs and custom ASICs.[2] We have omitted details to save space, but please refer to our previous work for a full description.[6]

Virtualizing steerable cameras

Cameras can be electronically steerable (with PTZ capabilities) and have to support multiple application queries simultaneously—like Amber Alert scanning based on license plate recognition and traffic-volume monitoring. The primary challenge in supporting multiple such application queries concurrently is that view and image requirements can differ. Allowing queries to directly steer the cameras inevitably leads to conflicts. If the license plate–recognition system zooms in to get a better view of the text, the traffic volume monitor loses the larger picture and cannot count all the cars in the intersection.

Rocket's camera manager virtualizes the camera hardware, thereby breaking the one-to-one binding between the camera and an application query. The query binds itself to a virtual instance of the camera and specifies its view requirements—for example, the orientation, resolution, and zoom. Our system is designed to provide each application query with the most recent view that meets its requirements. The virtual camera abstraction makes steering changes transparent to applications. The insight that led to this design came from observations that temporarily steering the camera away can be masked by replaying the image from the previous view, and that the impact of incorrect representation of the actual scene is minimized when no significant change is expected in the view during that time. To know when to steer, our software learns the movement patterns in its view and predicts when motion or change is likely to occur in each of the views it is managing. Traffic systems, for example, exhibit regular, constricted motions, which make it possible to learn movement patterns in the scene. The camera can then be quickly moved to the different views needed to support different vision applications. Experiments with live camera feeds have shown that in a typical traffic intersection, our system captures up to 80 percent more events of interest in a wide scene, compared to a system that allows application queries to control the cameras directly.[11]

Application-level optimizations

To further reduce resource demand and tolerate high latencies of executing expensive DNN operations in the cloud, we developed additional application-level techniques.

Intelligent frame selection. As mentioned earlier, the resource demands of certain vision algorithms can make running them on every frame prohibitively expensive. Fortunately, most video streams have temporal redundancy among frames, making it possible to sub-sample them without losing accuracy. We developed a suite of content-aware and application-aware techniques to sample only a few key frames for processing without compromising the application's accuracy.[12]

The core nugget is to use easily computable "shallow-vision" features to decide whether a frame is worthy of "deep-vision" analysis. We use frame-differencing, which is extremely cheap, to discard frames that have not changed significantly compared to the previously processed frame. We also compute cheap quality metrics such as blurriness and lighting to decide if the frame is of sufficient quality to process. Application-level requirements can also help us discard more frames. For instance, the output of an object classifier DNN might remain the same even when the visual content changes significantly. Application-specific sampling both enhances and complements application-agnostic sampling.

Intelligent feed selection. When a single camera's coverage is limited because of its position, it is reasonable to deploy multiple cameras at different locations and positions to cover the same physical space. The challenge is, for a given region, more cameras generate more video streams, which require more network bandwidth and computing resources in the cloud. Smart cameras—that is, cameras with computational abilities or that can access edge nodes—reduce the network and cloud computing demands while improving the accuracy of the video analytics system.

Our system processes the incoming video stream at each of the smart cameras and/or edge nodes to detect objects of interest.[13] Only the most important frames from the different video streams are then uploaded to the cloud for further processing. To quantify a frame's importance, we define a new metric: *objects per second* (OPS).

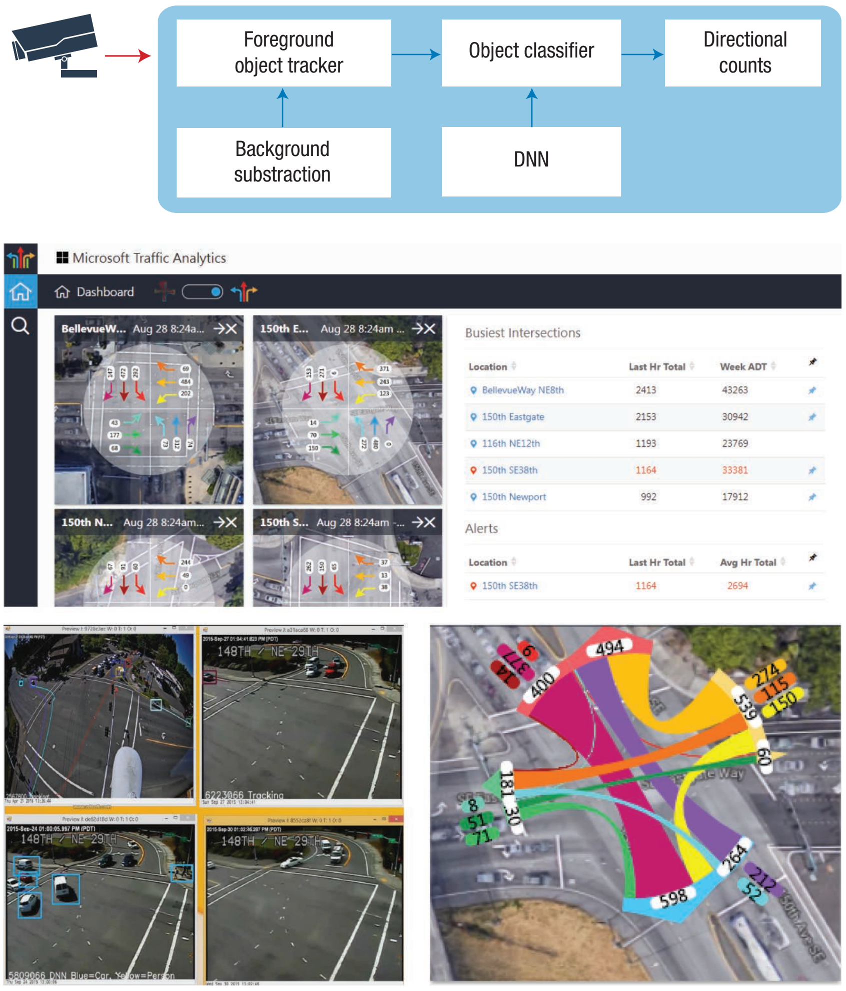
Each frame in each video stream has some number of objects. Our system's goal is to use the minimum amount of bandwidth while maximizing the number of query-specified objects (of interest in the scene) delivered to the cloud. A smart traffic-scheduling algorithm uploads frames from only the cameras that have video frames containing the greatest number of relevant objects to the user's query. Our content-aware uploading strategy suppresses a large fraction of unrelated video data, and in the process reduces network utilization and increases the utility of the video analytics system, because more cameras can now be supported. With this strategy in place—and depending on the amount of activity in the region—we have demonstrated that for a fixed bandwidth, our analytics system can support a coverage area that is 5- and 200-fold larger than one that simply streams video from the camera to the cloud. Looking at it another way, for a fixed area of coverage and bandwidth, our system outperforms the default equal throughput (per camera) allocation strategy by delivering up to 25 percent more objects relevant to a user's query.[13]

Delay-tolerance. Real-time computer vision applications such as augmented reality can suffer from high frame-processing delays. DNNs can take multiple frame-times[6] to produce a result; if processing is done over the network, the latencies could be even higher. By the time the application receives the results, they might already be stale, and detected objects could have moved to a different location or be out of frame.

We developed a new technique to hide such delays and provide a good user experience.[12] Our approach uses local object tracking to project the stale results (object locations) obtained from a processed frame to the current frame. To make tracking effective—because objects' positions can change significantly—we maintain an active cache that stores all subsequent frames from the frame that is selected for deep-vision processing. When the processed frame result comes back, we run tracking from the processed frame, through the cached frames, to catch up with the current frame. When newer objects appear in the processed frame, we start tracking them thereon.

TRAFFIC VIDEO ANALYTICS

Our traffic analytics solutions based on the Rocket software stack (shown in Figure 2) are being actively deployed. Since December 2016, a multimodal object counter has run 24/7 using live traffic cameras in Bellevue, Washington, to help the city understand and


Foreground object tracker
Object classifier
Directional counts
Background substraction
DNN
Microsoft Traffic Analytics
Dashboard
BellevueW... Aug 28 8:24a...
150th E... Aug 28 8:24am ...
150th N... Aug 28 8:24am...
150th S... Aug 28 8:24am -...
148TH / NE 29TH

Busiest Intersections

Location	Last Hr Total	Week ADT
BellevueWay NE8th	2413	43263
150th Eastgate	2153	30942
116th NE12th	1193	23769
150th SE38th	1164	33381
150th Newport	992	17912

Alerts

Location	Last Hr Total	Avg Hr Total
150th SE38th	1164	2694

FIGURE 4. Production deployment details of traffic analytics in Bellevue, Washington, based on live traffic intersection camera streams. The figure shows a dashboard of multiple camera feeds being analyzed simultaneously, the video analytics pipeline, and charts of directional volumes.

track volumes of cars, pedestrians, and bikes. The system raises alerts on anomalous traffic patterns used by traffic control operators.

The counters implement the interfaces specified above. They are run on a small distributed cluster of machines and provide directional counts at the intersections being monitored. Output from these counters will feed actuation systems that control traffic light durations. We use wide-angle cameras to cover the whole intersection with a single camera. Each server is equipped with the NVIDIA GTX 1080 GPU to run our custom DNN to recognize cars, pedestrians, and bicycles.

Figure 4 shows the counter's video analytics pipeline as well as a sample of the output of directional counts over time. Using crowdsourcing-based ground truths, we see that our counts are more than 95 percent accurate. Using the techniques described above for picking resource-efficient configurations, each counter pipeline consumes less than 15 percent of a quad-core server.

We are in the process of engaging with additional cities for a larger rollout of our traffic analytics system, and plan to extend the system for other uses in the future. ◼

REFERENCES

1. S. Ray, "From Airplane Engines to Street Lights, Transportation Is Becoming More Intelligent," blog, 2 May 2016; blogs.microsoft.com/transform/feature/from-airplane-engines-to-street-lights-transportation-is-becoming-more-intelligent.
2. N. Jouppi, "Google Supercharges Machine Learning Tasks with TPU Custom Chip," blog, 18 May 2016; cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-task-with-custom-chip.html.
3. J. Wang et al., "A Scalable and Privacy-Aware IoT Service for Live Video Analytics," *Proc. ACM Multimedia Systems Conf.* (MMSys 17), 2017; doi:10.1145/3083187.3083192.
4. Q. Pu et al., "Low Latency Geo-Distributed Data Analytics," *Proc. ACM Conf. Special Interest Group Data Communication* (ACM SIGCOMM 15), 2015, pp. 321–324.
5. P. Simoens et al., "Scalable Crowd-Sourcing of Video from Mobile Devices," *Proc. 11th Ann. Int'l Conf.*

Mobile Systems, Applications, and Services (MobiSys 13), 2013, pp. 139–152.
6. S. Han et al., “MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints,” *Proc. 11th Ann. Int’l Conf. Mobile Systems Applications Services* (MobiSys 16), 2016, pp. 139–152.
7. H. Zhang et al., “Live Video Analytics at Scale with Approximation and Delay-Tolerance,” *Proc. USENIX Symp.* Networked Systems Design Implementation (NSDI 17), 2017, pp. 377–392.
8. Y. Kim et al., “Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications,” *Proc. Int’l Conf. Learning Representations* (ICLR 16), 2016; arxiv.org/pdf/1511.06530.pdf.
9. I. Hubara et al., “Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations,” arXiv preprint, 2016; arXiv:1609.07061v1.
10. Y. Guo, A. Yao, and Y. Chen, “Dynamic Network Surgery for Efficient DNNs,” *Proc. 30th Conf. Neural Information Processing Systems* (NIPS 16), 2016; papers.nips.cc/paper/6165-dynamic-network-surgery-for-efficient-dnns.pdf.
11. S. Jain et al., “Panoptes: Servicing Multiple Applications Simultaneously Using Steerable Cameras,” *Proc. ACM/IEEE Int’l Conf. Information Processing in Sensor Networks* (IPSN 17), 2017, pp. 119–130.
12. T. Chen et al., “Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices,” *Proc. ACM Conf. Embedded Networked Sensor Systems* (SenSys 15), 2015, pp. 26–29.
13. T. Zhang et al., “The Design and Implementation of a Wireless Video Surveillance System,” *Proc. ACM Ann. Int’l Conf. Mobile Computing Networking* (MobiCom 15), 2015, pp. 426-438.

ABOUT THE AUTHORS

GANESH ANANTHANARAYANAN is a researcher in the Mobility and Networking Group at Microsoft Research. His research interests include large-scale systems and networking. Ananthanarayanan received a PhD in computer science from the University of California, Berkeley. Contact him at ga@microsoft.com.

PARAMVIR BAHL is a distinguished scientist in the Mobility and Networking Group at Microsoft Research. His research interests include mobile computing, wireless systems, cloud services, and datacenter networking and management. Bahl received a PhD in computer science from the University of Massachusetts at Amherst. Contact him at bahl@microsoft.com.

PETER BODÍK is a researcher in the Mobility and Networking Group at Microsoft Research. His research interests include distributed systems and networking. Bodík received a PhD in computer science from the University of California, Berkeley. Contact him at peterb@microsoft.com.

KRISHNA CHINTALAPUDI is a researcher in the Mobility and Networking Group at Microsoft Research. His research interests include wireless networking systems and mobile computing. Chintalapudi received a PhD in computer science from the University of Southern California. Contact him at krchinta@microsoft.com.

MATTHAI PHILIPOSE is a researcher in the Mobility and Networking Group at Microsoft Research. His research interests include machine perception and learning under resource constraints. Philipose received a PhD in computer science from the University of Washington. Contact him at matthaip@microsoft.com.

LENIN RAVINDRANATH is a researcher in the Mobility and Networking Group at Microsoft Research. His research interests include all aspects of mobile computing and networked systems. Ravindranath received a PhD in computer science from the Massachusetts Institute of Technology. Contact him at lenin@microsoft.com.

SUDIPTA SINHA is a researcher in the Interactive Media Group at Microsoft Research. His research interests include computer vision, robotics, and computer graphics. Sinha received a PhD in computer science from the University of North Carolina at Chapel Hill. Contact him at sudipta.sinha@microsoft.com.