Scaling Video Analytics on Constrained Edge Nodes

Anonymous Authors[1]

Abstract

As video camera deployments continue to grow, the need to process large volumes of real-time data strains wide-area network infrastructure. When per-camera bandwidth is limited, it is infeasible for applications such as traffic monitoring, pedestrian tracking, and more to offload high-quality video streams to a datacenter. This paper presents FilterForward, a new edge-to-cloud system that enables datacenter-based applications to process content from thousands of cameras by installing lightweight edge filters that back-haul only relevant video frames. FilterForward introduces fast and expressive per-application "microclassifiers" that share computation to simultaneously detect dozens of events using computationally-constrained edge nodes. Only matching events are transmitted to the datacenter. Evaluation on two real-world camera feed datasets shows that FilterForward improves computational efficiency and event detection accuracy for challenging video content while drastically reducing network bandwidth usage.

1 Introduction

The deployment of video cameras in urban areas is ubiquitous: in malls, offices, and homes, and on streets, cars, and people. Almost 100 million networked surveillance cameras were purchased in 2017 (IHS). Machine learning–based analytics on real-time streams collected by these cameras, such as traffic monitoring, customer tracking, and event detection, promise breakthroughs in efficiency and safety. However, tens of thousands of always-on cameras installed in a modern city collectively generate tens of gigabits of data every second, surpassing the capabilities of shared network infrastructure. Wireless and cellular–connected nodes and areas outside of infrastructure-rich metropolitan centers often have significantly less network bandwidth (Google Wireless Internet; ITU/UNESCO Broadband Commission for Sustainable Development, 2017), exacerbating this problem (FCC).

The combination of increasing sensor resolution and deployment proliferation necessitates edge-based filtering that is parsimonious with limited bandwidth. This paper presents FilterForward, a system that offers the benefits of both edge-computing and datacenter-centric approaches to wide-area video processing. Using edge compute resources collocated with the cameras, FilterForward identifies short sequences most relevant to the datacenter applications ("filtering") and offloads only those for further analysis ("forwarding"). In this way, FilterForward supports near real-time processing running in datacenters while limiting the use of low-bandwidth wide-area network links.

FilterForward is designed for scenarios meeting two key assumptions, which hold for some, though certainly not all, applications: First, relevant events are rare. There is, therefore, bandwidth to be saved by transmitting only relevant frames. Second, datacenter applications require sufficiently high-quality frame data in order to best complete their tasks. This precludes solutions such as heavily compressing streams or reducing their spatial (frame dimensions) or temporal (frame frequency) resolutions.

In the FilterForward model, datacenter applications express interest in specific types of visual content (e.g., "send me sequences containing buses"). Each application installs on the edge a set of small neural networks called *microclassifiers (MCs)* that perform binary classification on incoming frames to determine whether an interesting state is occurring. These frame-level results are smoothed to determine the start and end points of an "event" during which the object appears. At runtime, the matched frames, or a subset thereof, are streamed to the cloud. Multiple MCs work in parallel to filter different types of events for many applications. Each MC is trained offline by an application deployer to determine which frames are relevant to that application, using the same training data that would be used to develop a datacenter-based filter.

In contrast to prior video filtering work such as NoScope (Kang et al., 2017), FilterForward is designed to identify events that may occupy arbitrary and small regions of the frame and require fine-grained details; examples might

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the Systems and Machine Learning (SysML) Conference. Do not distribute.

include differentiating pedestrians walking with canes from those without, or spotting specific animals in wide-angle shots.

FilterForward is further designed to scale to multiple, independent detection objectives (i.e., “find any of these five types of fish”). Instead of designing the MCs to operate on raw pixels, FilterForward draws inspiration from the design of modern object detectors and uses a shared base neural network to extract general features from each frame. Each MC reuses the activations from that base network (either by running a convolutional window across an entire layer, or by extracting a spatially-cropped region). This amortizes the expensive task of pixel processing across all of the MCs, allowing FilterForward to scale to tens of concurrent MCs using the CPU power available in a small-form-factor edge node.

For applications meeting the requirements of FilterForward (operating with severe bandwidth constraints and requiring reasonable quality frames), our architecture deliver a greater number of useful frames to the datacenter than either existing filtering approaches, or massively compressing the stream using h.264. Our evaluation using two real-world camera feeds demonstrates that FilterForward is computationally efficient, scaling to X applications on a consumer-grade CPU. Microclassifiers are more accurate than the pixel-based task-specific deep neural net (DNN) filters used in prior work, and they provide a higher frame rate when running 20 or more filters in parallel.

2 Edge-to-Cloud Challenges

The scenarios that motivate this paper include remote IoT-style monitoring, and “smart-city” style deployments of thousands of wide-angle, fixed-view cameras. The scenes these cameras observe contain diverse objects and activities. FilterForward is designed to help overcome three challenges:

2.1 Limited Bandwidth

Each camera’s wide area bandwidth is limited, both by the physical constraints of modern infrastructure and the monetary cost of operating a widespread camera deployment. Previous video analysis platforms (Kang et al., 2017; Zhang et al., 2017) assume that cameras on the edge can stream full video feeds to a datacenter, but we challenge that assumption. Specifically, we consider large-scale deployments where each camera receives a bandwidth allocation of a few hundreds of kilobits per second or less, which is often insufficient to stream video at a high-enough quality to perform accurate analysis. Aggressive compression severely degrades the F1 score of applications running in a datacenter (Pakha et al., 2018). These requirements necessitate edge-based computation to identify which frames to send to the datacenter.

2.2 Scalable Multi-Tenancy

The surveillance video captured by each camera in a large-scale deployment contains many types of content that are interesting to applications. Therefore, any edge-filtering approach must support multiple applications examining a video stream in parallel, but doing so can be computationally expensive. A naïve approach to handling N applications is to run N full DNNs concurrently, however even relatively lightweight DNNs are costly—on modest hardware, MobileNet (Howard et al., 2017) runs at approximately 15 fps on 512×512 pixel RGB input frames while consuming more than a gigabyte of memory. This prevents edge nodes from executing more than a handfull of full DNNs on a real-time video stream. Section 5.2 compares the performance of this approach to FilterForward.

Alternatively, conventional machine learning techniques for reusing computation provide better scalability, but sacrifice accuracy. Transfer learning accelerates multi-application training and inference by leveraging the observation that DNNs trained for image classification and object recognition identify general features that transfer well to novel, specialized computer vision tasks (Donahue et al., 2014; Yosinski et al., 2014). [1] During inference, DNNs share computation by running one base DNN to completion and extracting its last layer as a feature vector, which is then used by multiple specialized classifiers (one per application) (Pakha et al., 2018). Recent transfer learning approaches (Jiang et al., 2018) allow application-specific DNNs to share different numbers of layers from the base DNN. Unfortunately, these approaches suffer poor accuracy especially for small objects because they retain the original neural network architecture for their retrained layer(s), an approach that is not optimized for small objects.

2.3 Real-World Video Streams

Given most cameras’ wide viewing angle and high mounting position, interesting objects are typically small (e.g., distant pedestrians in busy intersections). Additionally, around-the-clock surviellence video experiences lighting changes and weather phenomenoms (e.g., passing clouds, day-to-night transitions, rain and snow, etc.) that drastically alter the data over time. Therefore, pixel-level or histogram-based

[1] Classical transfer learning of DNNs involves starting with a network trained on an existing large dataset, holding all weights up to a given layer constant, and retraining only the last layer (also known as *fine-tuning* for the task at hand). Fine-tuning more than just the last layer is an instance of *multi-task learning* (Caruana, 1998), which leads to deployment inflexibility because all models must be retrained when applications are added or removed.

detectors are insufficient because large shifts in the content over time lead to false positives—which further exacerbate the aforementioned bandwidth constraint by sending unnecessary video frames to the datacenter—and false negatives—which cause applications to miss events. Filtering based on semantic content is necessary.

Previous systems (Kang et al., 2017) are often evaluated on highly-curated datasets, where video processing is orchestrated to be easier. For example, the video may be cropped so that the object of interest is large (typically occupying the majority of the frame), and edited to exclude challenging time periods, such as nighttime. This constrains the state space that a machine learning model must learn, giving an advantage both in terms of model size and accuracy.

Ultimately, FilterForward addresses these three challenges—limited bandwidth, scalable multi-tenancy, and support for real-world video streams—through computational reuse, novel microclassifier architectures, and content-based filtering.

3 Background and Related Work

Before delving into FilterForward's design, we provide an overview of related video analytics and machine learning work.

Video analytics: Video analytics encompasses various query processing tasks that examine content in video streams. Examples of video analytics include the following: *Image classification* categorizes a whole frame based on its most dominant features (e.g., "This is an image of an intersection."). *Object detection* finds interesting objects that may occupy only a small portion of the view in and cetegorizes them (e.g., "This region is a car and that region is a person."). *Object tracking* aims to label each object across multiple frames (e.g., "This path plots the progress of pedestrian A crossing the road."). These tasks require extensive computation on large amounts of data (e.g., $\tilde{1}.5$ Gbps of uncompressed data per 1080p, 30 fps video stream). Accomplishing video analytics at scale requires abundant compute and storage resources, making it logical to do this processing in the cloud (Kang et al., 2017; Zhang et al., 2017). In contrast to bulk analytics applications running in the cloud, our edge filters economically search the video stream for requested "events" with the explicit goal of reducing the number of frames that must be transmitted and processed later.

Video filtering with event detection: *Filtering* is a basic building block of video analytics. Dropping irrelevant frames is a common techinique to reduce computation and transmission load if analyzing those dropped frames would yield marginal benefit (Kang et al., 2017; Pakha et al., 2018; Wang et al., 2018). NoScope (Kang et al., 2017), for example, detects pixel-level changes versus a reference image or previous frame and executes classifiers on only those frames whose difference exceeds a threshold. NoScope also trains cheap, task-specific CNNs (i.e., a custom "shetland pony" binary classifier instead of fine-tuning an advanced object-detection network such as YOLO9000 (Redmon & Farhadi, 2016)), and only applies an expensive DNN when the confidence of the cheap DNN is below a threshold. However, unlike FilterForward, NoScope assumes that it is possible to stream all of the video to a resource-rich datacenter, so all of their processing is done in the cloud. The basic premise of FilterForward is that this offloading is infeasible. Futhermore, NoScope's cheap binary classifiers are much more expensive than FilterForward's microclassifiers, limiting scalability, as discussed in 5.2. Furthermore, we observe that simple pixel-level DNN-based classifiers are inaccurate on real-world video streams containing small objects, justifying the need for specialized architectures (Section 5.4).

While filtering may take many forms, we focus primarily on filtering as "event detection", wherein each filter discovers contiguous segments of a video stream that contain a particular event, usually the presence of an object (e.g., "video segments containing pedestrians crossing the road"). We focus on event detection because of its role as a preprocessor in advance of other video applications, since detecting a particular type of scene is often the first step in more specialized tasks (e.g., object counting, safety analysis, activity recognition, etc.). FilterForward extends the benefit of video filtering beyond computation reduction to the problem of limited bandwidth, transmitting only video segments with events of interest.

Resource scheduling for video applications: Resource management is crucial for practical video analytics because applications often impose the conflicting goals of maximizing their overall benefit and meeting performance constraints. For instance, VideoStorm (Zhang et al., 2017) adjusts query quality to maximize a combined utility, using efficient scheduling that leverages offline quality and resource profiles. LAVEA (Yi et al., 2017) places tasks across edge nodes and clients (e.g., mobile phones) to minimize latency of video analytics. DeepDecision (Ran et al., 2018) expresses resource scheduling in video processing as a combinatorial optimization problem.

FilterForward shares a similar motivation of balancing quality and throughput for resource-starved edge nodes with constrained network bandwidth. Unlike prior scheduling work that only adjusts general knobs such as video bitrate for performance-quality tradeoffs, FilterForward directly improves the computational efficiency of multiple filters running on the same edge node through computation sharing.

Accelerating DNN inference: Using low-precision numerics (e.g., binary weights and activations (Lin et al., 2017)) can accelerate inference by reducing the DNN model size and simplifying each arithmetic operation. This approach alone is insufficient to provide scalable video processing on FilterForward's target edge platform because using low-precision inference observes visible accuracy degradation when computation reduction exceeds about $4\times$ (Lin et al., 2017; Han et al., 2016), but it is complementary to our work.

Edge-based video analytics under bandwidth constraints: Similar to FilterForward, others have approached the challenges of running ML workloads on multitudes of edge-generated video in real time. Both (Pakha et al., 2018) and (Wang et al., 2018) push computation to the edge to determine which frames are "uninteresting" to heavyweight analytics in the cloud.

(Pakha et al., 2018) uses sampling and superposition coding to only send the frames where objects appear, and send those frames at the lowest possible quality. Their iterative algorithm offloads low-quality frames to the cloud first, runs them through the analysis workload, and uses the results as a hint as to whether a frame (or regions of a frame) is relevant and should be considered at higher quality. The authors stress the difference between quality for the sake of human consumption and quality that improves DNN inference accuracy. While the work displays impressive bandwidth savings, the iterative nature of the algorithm along with its requirement on back-and-forth communication between the edge and the cloud mean that the throughput is limited. FilterForward is entirely feed-forward, with no cloud-to-edge communication (except for periodic updates to the microclassifier models), so the throughput is only limited by the processing power of the edge nodes.

(Wang et al., 2018) examines the heavily bandwidth-constrained use case of offloading video in real time from a swarm of autonomous drones using the 4G LTE cellular network. The authors prioritize low end-to-end latency between frame capture and cloud analysis in order to support tight feedback loops with human operators. Similar to FilterForward, this system uses lightweight DNNs (e.g., MobileNet) running on the edge (here, on the drones) to give an early indication of whether a frame is interesting. The system improves accuracy in the face of changing environments using continuously-retrained SVMs that operate on the output from the DNNs' final pooling layers. These SVMs are similar in principle to our microclassifiers. However, since their architecture is constrained and they only use features from the final pooling layer, the SVMs are by design limited in the content they can detect (yet are slightly cheaper) compared to our microclassifiers.

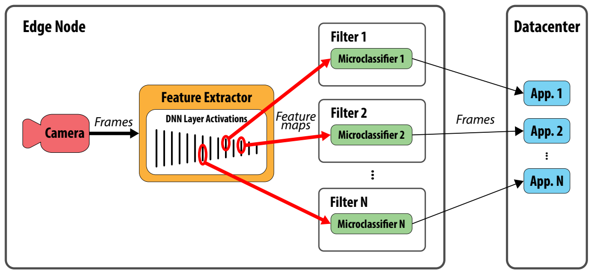


Figure 1. The FilterForward architecture. At the heart of each filter is a microclassifier that detects which frames are relevant to a single application, making the best use of an edge node's bandwidth budget. The resulting set of relevant frames is forwarded to the datacenter. Multiple microclassifiers share feature maps generated by the feature extractor, allowing FilterForward to scale to dozens of filters per machine. Thick lines indicate large volumes of data, whereas thin lines represent decimated streams.

(Wang et al., 2018) also makes the point that degrading a stream's encoding quality in turn measurably lowers DNN inference accuracy. One of the key motivating factors behind FilterForward is the desire to avoid this drop in per-frame accuracy by sending fewer, but higher quality, frames. Prioritizing relevant frames under this constraint is the primary contribution of FilterForward.

Both (Pakha et al., 2018) and (Wang et al., 2018) focus on streams where the camera is moving, whereas FilterForward primarily considers stationary surveillance cameras. Operating on streams with less global motion gives FilterForward an advantage because it is easier to train classifiers for these streams, and the larger proportion of unchanging pixels makes them more compressible. While we only evaulate on static streams, the principles of FilterForward are also applicable to moving streams.

One drawback of these works is that they both rely on dropping *individual frames*, but for the video-centric applications we target, frame dropping is sub-optimal: With h.264 or similar compression, dropping a single frame often saves relatively little bandwidth; event-based approaches are more suited. Finally, of course, this prior work is not optimized for multi-tenant use. FilterForward is designed with system throughput and query scalability as first-class concerns, and can run dozens of concurrent microclassifiers.

4 DESIGN

FilterForward is a novel video analytics architecture that reuses computation to provide high-quality, multi-tenant filtering for bandwidth-constrained edge nodes. The architecture is shown in Figure 1; the relatively more expensive feature extractor's computation is amortized across multiple microclassifiers (MCs).

FilterForward offers applications the flexibility of splitting their work between the edge and the cloud. Purely edge-based approaches constrain applications to the static compute and storage resources of field installations, while datacenter-only deployments necessitate heavily compressing the video for transport. FilterForward takes advantage of high-fidelity data at the edge and makes relevant video sequences available to the cloud.

The rest of this section covers the architecture of FilterForward's feature extractor and microclassifiers, discussing how they address the three challenges described in Section 2.

4.1 Generating Features

To enable computation reuse across multiple filters, FilterForward's microclassifiers take as input the intermediate results from a single reference DNN, which we refer to as the *base DNN*. Later layers of the base DNN typically encode less visual and more semantic information about the image. For example, the output (activations) of the first layer (which is usually a simple convolutional filter such as an edge detector) is still visually recognizable. Later activations are feature maps representing higher-level concepts (e.g., "eye", "fur", "tire", etc.). Convolutional layers reduce the spatial dimensionality of the data. Thus, an image may start as 512x512x3 pixels (three color channels—RGB) and end up as a feature map of varying dimensions (256x256x64, 128x128x128, 32x32x512, etc.). The final layer in an ImageNet-based classifier is typically a 1x1x1000 vector, where the 1000 output floats represent 1000 potential classes for the image.

Selective processing of these feature maps has been used successfully for tasks such as object region proposals, segmentation, and tracking (Ren et al., 2015; Hariharan et al., 2015; Ma et al., 2015; Bertinetto et al., 2016), as well as video action classification (Sharma et al., 2015). As prior work observes (Sharghi et al.; Yeung et al.), the feature maps capture many of the things humans intuitively desire to detect in video, such as the presence and number of objects in a scene, outperforming handcrafted low-level features (Razavian et al., 2014; Yue-Hei Ng et al., 2015; Babenko & Lempitsky, 2015).

Operating on feature maps from a single DNN provides the microclassifiers with competitive accuracy and an order of magnitude lower computation load than operating directly on raw pixels (Section 5.2). For our experiments, we use the full-size MobileNet (Howard et al., 2017) architecture trained on ImageNet (Russakovsky et al., 2015) as the base DNN. This network offers a balance between accuracy and computational demand that is appropriate for constrained edge nodes. One important difference from a typical use of MobileNet is that FilterForward uses full-resolution frames as input, instead of resizing to 224x224 pixels (or smaller). The type of video that FilterForward is designed to process (wide-angle surveillance video) contains many small details that are interesting to applications but difficult to detect if the video is too aggressively down-sampled. Using a large input imposes a significant computation overhead, as the work done by each layer increases. However, by operating on more pixels, small content such as distant pedestrians, model-specific details in automobiles, and faces are much more visible.

Evaluating the DNN is the most computationally intensive phase of FilterForward, but its results are reused by all parts of the system, effectively amortizing the up-front, per-frame overhead across filters. Each microclassifier can pull features from a different layer of the base DNN, as well as crop the features to focus on a certain portion of the frame. Feature cropping not only reduces computation, but increases accuracy by forcing the microclassifier to only consider that region when making its prediction.

Ultimately, the features generated by the base DNN, and the flexible manner in which we use them, underpin FilterForward's scalability and accuracy achievements.

4.2 Finding Relevant Frames Using Microclassifiers

4.2.1 Microclassifiers From Afar

Microclassifiers are single-frame, lightweight binary classification neural networks that take as input the features extracted by the base DNN (Section 4.1) and output the probability that a frame is relevant to a particular application. An application developer chooses a microclassifier architecture (we present several possibilities below) and trains it offline so that it can detect the application's desired content. To deploy a MC, the developer supplies the network weights and architecture along with a specification of which layer activations (and optionally, a crop of these) to use as input. Ideally, a microclassifier will identify all of the frames that an application needs to process in the cloud (the redundancy inherent in video provides a safety margin for false negatives), while rejecting a large fraction of unimportant frames (false positives, which pollute the bandwidth with irrelevant data). Dropping irrelevant frames is crucial to limiting bandwidth use (Section2.1).

By design, microclassifiers are cheap enough to run on every frame, enabling dozens to run concurrently on each edge node (Section 2.2). Their resource use is static and predictable, so the system designer can guarantee that they operate within edge node compute constraints. A single edge node can run several microclassifiers on a single, content-rich stream, or a handful of microclassifiers on several streams simultaneously. By analyzing high resolution video at the edge, FilterForward can achieves better accuracy than approaches which attempt to relay all of the frames to the datacenter under the bandwidth constraints (Section 5.2.1).

4.2.2 Microclassifier Architectures

The architecture of a microclassifier must be specialized to the wide-angle, surveillance-style video that FilterForward processes. As discussed in Section 2.3, running off-the-shelf classifiers and detectors on this type of video is inaccurate because the objects of interest are small. We propose three custom architectures, shown in Figure 2, that solve these challenges in different ways. An important feature of these designs is that they operate on different layers of the base DNN and optionally crop their input features. The microclassifiers operate on whichever granularity of features is most appropriate for their task while "ignoring" regions of the frame that are uninteresting. Both of these capabilities are critical to achieving high accuracy on real-world data (evaluated in Section 5.4) and differentiate FilterForward from existing systems like NoScope (Kang et al., 2017).

Full-Frame Object Detector (Figure 2a): Modeled after sliding window-style object detectors (such as SSD and Faster R-CNN), the full-frame object detector microclassifier applies a small binary classification DNN at each location in a CNN layer feature map and then aggregates the detections to make a global prediction. This is achieved by using multiple layers of 1×1 convolutions and then applying a `max` operator over the grid of logits (signifying looking for ≥ 1 objects). Spatial scoping could be accomplished by selectively zeroing the grid of logits prior to response aggregation. This model is specifically for pattern matching queries, with an implicit assumption of translational invariance (i.e., the model runs the same template matcher everywhere, treating every location the same).

Localized Binary Classifier (Figure 2b): Localized binary classification microclassifiers are lightweight CNNs that use a fixed rectangular crop of an internal CNN layer as input. The crop is selected ahead of time by the user. The ability to select a static subregion is a natural option for a fixed camera feed, enabling the model to perform spatially scoped classification at reduced computational cost. A potential extension could see these classifiers operate off of a region-of-interest selected at runtime.

Windowed Localized Binary Classifier (Figure 2c): This model is a simple extension of the localized binary classification microclassifier that incorporates nearby temporal context to perform better per-frame classification. The user specifies a fixed temporal window of size K. Given the convolutional features in a symmetric K-sized window at time t (i.e., features from time steps $[t - K/2, t + K/2]$), the windowed localized microclassifier first applies a 1×1 convolution on each frame's features in the given window. It then depthwise concatenates the resulting features and applies a CNN on the resulting output to predict whether the event was occurring at time t. This setup allows the model to pick up on motion cues in the scene, which helps achieve higher accuracies on the tasks where objects are constantly moving. In addition, the initial 1×1 convolution is only computed once and its output is shared by subsequent windows; this requires low overall memory footprint since first step significantly reduces the size of the input feature map.

4.2.3 Choosing Microclassifier Inputs

Choosing which layer to use as each microclassifier's input is critical to its accuracy. The layers of a CNN feature hierarchy offer a trade-off between spatial localization and semantic information. Too late a layer may not be able to observe small details (because they have been subsumed by global semantic classification). Too early a layer could be computationally expensive due to the large size of early layer activations and the amount of processing still required to transform them into a classification result. Given that a key feature of microclassifiers is their flexibility to draw from any internal activations in a reference CNN, how can we "train" a microclassifier such that it does so effectively?

As a baseline, we hand-select a layer and crop region based upon two simple heuristics: For the layer depth, we tried to match the typical size of the object class we were detecting. (e.g., to detect pedestrians in an input video in which the average human height was 40 pixels, we would choose the first layer at which a roughly 20-50:1 spatial reduction had been effected). We chose the crop layer based upon the region of interest for the application, such as the crosswalks in the Jackson Hole pedestrian task (Section 5).

In the future, we plan to experiment with sampling from non-contiguous regions and across multiple layers. We also plan to further automate the process of choosing which layer activations and specific crops to use. Our preliminary experiments with choosing random crops were encouraging, but not much simpler than the guided heuristics above, so we do not report on them further.

4.3 Detecting Events Using Microclassifiers

A microclassifier assigns per-frame classifications, which FilterForward then smooths into event detections. First, each MC's results for N consecutive frames are accumulated into a window. Then, to smooth spurious positives, we apply K-Voting to this window, treating the middle frame as a detection if at least K of the frames in the window are positive detections. We set $N = 5$ and $K = 2$. The resulting smoothed per-frame labels are fed into an event detector that considers each contiguous segment of positively-classifier frames as a unique event. Each event is assigned a unique ID, which is stored in each frame's metadata. When frames arrive in the datacenter, this ID is used to reestablish the event boundaries.

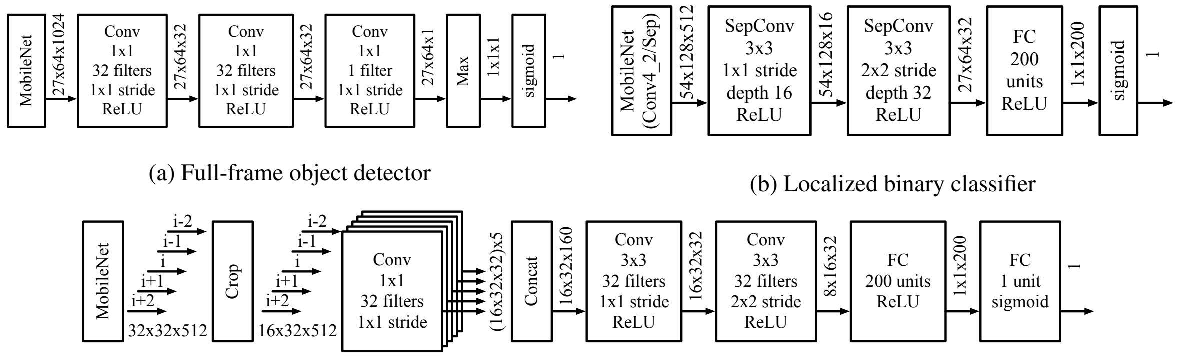

MobileNet
27x64x1024
Conv 1x1 32 filters 1x1 stride ReLU
27x64x32
Conv 1x1 32 filters 1x1 stride ReLU
27x64x32
Conv 1x1 1 filter 1x1 stride ReLU
27x64x1
Max
1x1x1
sigmoid
1
(a) Full-frame object detector
MobileNet (Conv4_2/Sep)
54x128x512
SepConv 3x3 1x1 stride depth 16 ReLU
54x128x16
SepConv 3x3 2x2 stride depth 32 ReLU
27x64x32
FC 200 units ReLU
1x1x200
sigmoid
1
(b) Localized binary classifier
MobileNet
i-2
i-1
i
i+1
i+2
32x32x512
Crop
16x32x512
Conv 1x1 32 filters 1x1 stride
(16x32x32)x5
Concat
16x32x160
Conv 3x3 32 filters 1x1 stride ReLU
16x32x32
Conv 3x3 32 filters 2x2 stride ReLU
8x16x32
FC 200 units ReLU
1x1x200
FC 1 unit sigmoid
1
(c) Windowed, localized binary classifier

Figure 2. Layer details for the three proposed microclassifier architectures. The details for the full-frame object detector and localized binary classifier are for the case where features generated from 2048 x 850 images. The details for the windowed, localized binary classifier shown in this diagram is for features generated from 512 x 512 images, due to lack of computational resources/time to train this model on features generated from 2048 x 850 images.

Most classification metrics operate on a per-frame basis; because our work is event-centric, we adopt a simplified metric from (Lee et al., 2018) that defines a modified recall metric for events that span time. The resulting metric weighs two success measures: *Existence* rewards detecting at least one frame from an event, and *Overlap* rewards detecting an increasing fraction of the frames from an event:

$$Existence_i = \begin{cases} 1 & \text{if any frame predicted in event } i \\ 0 & \text{otherwise} \end{cases}$$

$$Overlap_i = \sum_j \frac{|Intersect(R_i, P_j)|}{|R_i|}$$

where R_i and P_j are the ground-truth event ranges and predicted event ranges, respectively.

The final event recall metric is: $\alpha \times Existence + \beta \times Overlap$. We choose $\alpha = 0.9$ and $\beta = 0.1$ in order to place a greater importance on detecting a single frame in each event. In real-time streaming event detection in a city surveillance setting, we believe that not missing a single event is more important than processing all frames in an event. We evaluate microclassifier accuracy using this metric in Section 5.4.

For precision, we retain the standard definition: Precision is the fraction of predicted frames that are true positives (i.e., $\frac{\text{\# detected true positives}}{\text{all detected frames}}$. For FilterForward, this corresponds to a measure of how well the edge node's bandwidth is used: A precision of 1 means that all bandwidth is used to send "true positive" frames. We combine precision with our modified definition of event recall to calculate an event F1-score, which is the basis for our evaluation.

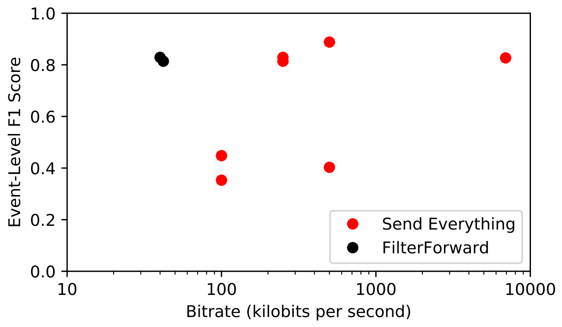
5 Evaluation

FilterForward's goal is to maximize filtering accuracy while limiting bandwidth use and scaling to multiple applications. We show that FilterForward achieves a high frame rate on commodity hardware by sharing computation between the microclassifiers while maintaining high event-level accuracy on two event detection tasks.

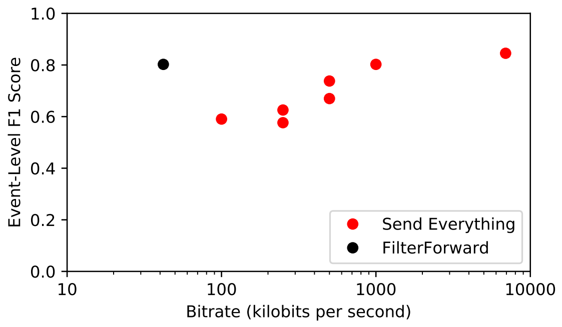
5.1 Datasets

Because our target for FilterForward is large real-world deployments, we evaluate using two datasets (Figure 4) captured with cameras and surveying types of scenes that are representative of our intended use cases. The first dataset consists of video captured from a traffic camera deployment in Jackson Hole, Wyoming (the *Jackson* dataset). We collected two six-hour videos from two consecutive days between 10AM and 4PM. Then, we annotated the twelve hours of data with labels for two events: (1) when pedestrians appear in the crosswalks (the *Pedestrian* task); and (2) when red pickup trucks appear in the left street area (the *Red Truck* task). These tasks allow us to demonstrate the spatial selectivity of our microclassifiers in a way that is hopefully relevant to future traffic monitoring applications. For example, combined with a simple traffic light detector, one could craft composite queries to detect near-misses on unprotected left turns.

Because the encoding quality of the Jackson dataset (obtained from Youtube) is relatively low and lacks sufficent detail to reliably distinguish, e.g., clothing or other finer details, we collected a second dataset from high-quality cameras of our own. It consists of two three-hour videos from a camera overlooking a city street (the *Roadway* dataset), captured back-to-back during the middle of the day. We then labeled segments during which the camera observed

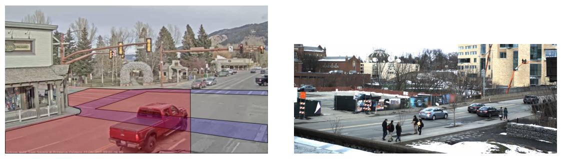


(a) Full-frame binary classifier, with FilterForward operating on a 250 kbps stream.



(b) Localized binary classifier, with FilterForward operating on a 1000 kbps stream.

Figure 3. Achieved bitrate use on the *Roadway* dataset for two strategies for offloading data: (1) compressing the video using h.264 and sending all frames; and (2) FilterForward, where only relevant sequences are sent (also h.264 encoded). By dropping irrelevant frames, FilterForward uses bandwidth more efficiently and achieves a higher event-level F1 score despite using less bandwidth. We evaluate our microclassifiers as proxies for datacenter applications, running full training on each experimental bitrate. To pick the bitrate at which to run FilterForward, we (i.e., the application developer) select the bitrate at which the event-level F1 score's improvement tappers off (250 kbps for (a) and 1000 kbps for (b)).



(a) Left: The *Jackson* dataset; crosswalk and street regions for the *Pedestrian* and *Red Truck* tasks, respectively. Right: The *Roadway* dataset.

Dataset	Jackson		Roadway
Resolution	512x512		2048x850
Frame rate	15		15
# Frames	648,000		324,000
Task	Pedestrian	Red Truck	People with red
# Event Frames	95,238	8,872	71,296
# Events	506	139	326

(b) Dataset details

Figure 4. Video datasets used for evaluation.

pedestrians wearing red articles of clothing or carrying red parcels (the *People with red* task). We found throughout our experiments that the coarseness of the Jackson video made it difficult to train classifiers to observe smaller objects or fine details, and so, in general, we focus on the Roadway dataset because of its higher-fidelity images.

All performance experiments are conducted on a desktop computer with a quad-core Intel i7-6700K CPU and 32 GB of RAM. We will make the datasets available online once not bound by anonymity requirements.

5.2 End-to-End Performance

On an H.264 encoded, 1920×1080 video, including decoding and recording to disk, FilterForward is capable of processing its full-resolution input at 5 fps with a single query, or roughly 3 fps with 10 concurrent queries.

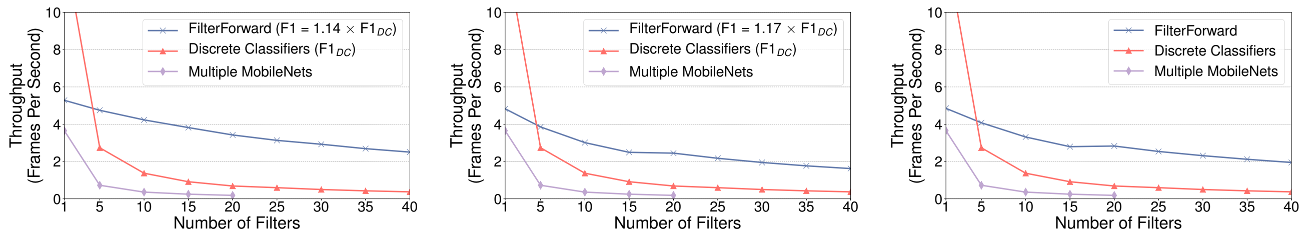
Figure 6 shows the breakdown of execution time for our three proposed microclassifier architectures. With few queries, the feature extraction DNN execution time dominates, but the total execution cost remains modest even with dozens of concurrent queries. (We expect that real-world deployments would use either GPU or specialized hardware acceleration; our results here are using full 32 bit versions of Mobilenet).

An alternative approach is to train discrete, from-the-pixels classifiers specialized to a specific class (and camera view), as used in NoScope (Kang et al., 2017). A single discrete classifier is faster than the base DNN used in FilterForward (Figure 7), but also achieves lower accuracy. Because the discrete classifiers do not share computation, once there are four or more concurrent queries, FilterForward's shared design both achieves higher accuracy and lower cost.

5.2.1 *Bandwidth Savings*

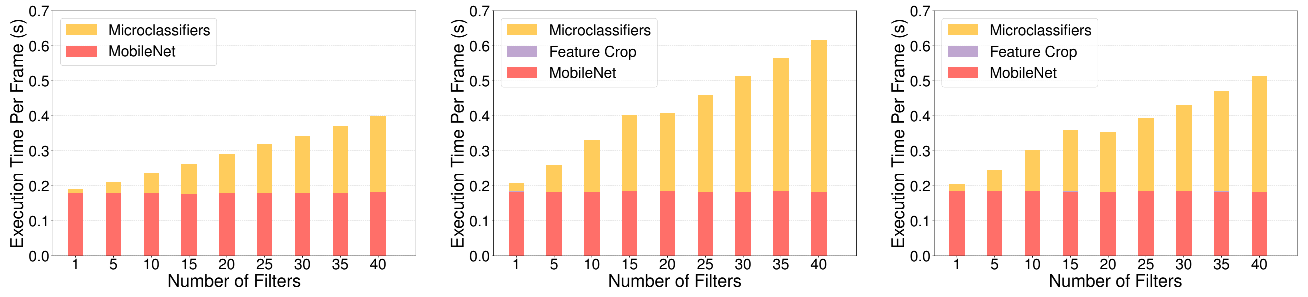
When FilterForward identifies an event, it streams an h.264-encoded sequence of the event's frames back to the datacenter, at whatever bitrate was determined necessary for the application. Figure 3 shows the bandwidth and event-level F1 scores (which, as the reader will recall, combine precision and recall, but using precision modified to give slight extra weight to detecting any frames during an event). Using either the crop-based "localized classifier" or the full-frame binary classifier, FilterForward is able to reduce the network bandwidth by nearly an order of magnitude compared to sending the equivalent full video stream back to the datacenter. Compared to alternatives that compress the video more highly, FilterForward provides substantially better accuracy.

In the remainder of the evaluation, we dig into the performance and accuracy of each of FilterForward's components



(a) Full-frame binary classifier (b) Localized binary classifier (c) Windowed, localized binary classifier

Figure 5. Throughput of FilterForward for our three microclassifier architectures. FilterForward achieve superior throughput as the number of filters increases because more filters means more effective amortization of the cost of the base DNN.



(a) Full-frame binary classifier (b) Localized binary classifier (c) Windowed, localized binary classifier

Figure 6. Execution time breakdown of the main components in FilterForward for our three microclassifier architectures. The full-frame binary classifier MC does not contain a "Feature Crop" stage because it operates on a full feature map. For (b) and (c), the "Feature Crop" is so fast that it is not visible. FilterForward pays the upfront cost of evaluating the base DNN (MobileNet), but reaps the resulting benefit of each additional microclassifier being relatively cheap.

to explore *why* it is able to provide these savings, and to help understand where it may or may not generalize.

5.3 Feature Generation

The first step in the FilterForward processing pipeline uses an existing DNN to create feature maps for each frame. After evaluating a set of modern networks, we selected MobileNet (Howard et al., 2017), for its balance of maturity, accuracy and inference performance. (This was a choice made at the outset of this research; picking a good base network is, of course, a moving target, and we do not view the selection of a specific base as particularly important—- improved base networks, in fact, improve FilterForward's performance relative to discrete classifiers.)

We run the DNNs using a version of the Caffe deep learning framework (caffe) that has been optimized for Intel CPUs and uses the Intel Math Kernel Library for Deep Neural Networks (Intel MKL-DNN) (intel-mkl-dnn). When performing *only* DNN inference, our hardware can achieve approximately 5 fps. We record the full frame rate video to disk so that applications can request full-rate video frames from the edge nodes' local storage upon a match.

Generating feature maps through DNN inference is the single most expensive component of FilterForward, but we view this cost as a less important aspect of our system: this area is rapidly evolving, and both the CPU requirements and accuracy of DNNs are improving. (For example, the MobileNet architecture was published in April 2017 and has already been surpassed.) All of the inference peformed in FilterForward could be accelerated by using acceleration frameworks that quantize and/or sparsify networks, such as Nvidia's TensorRT, and future platforms may be able to draw from a growing slate of accelerators for deep learning inference, such as Google's TPUs (Jouppi et al., 2017), NVidia's Tensor Core-containing platforms, Apple's Neural Engine (Apple, 2017), Movidius's Neural Compute Stick (intel-movidius), and Microsoft's Project Brainwave (microsoft-project-brainwave). We therefore believe that low-cost 30 or 60 fps inference will be feasible on edge camera systems in the near future, and we have designed the rest of FilterForward under this assumption.

5.4 Microclassifier Accuracy and Utility

The purpose of the microclassifier evaluation is twofold. First, we show that microclassifier models are reasonably accurate over a number of tasks. Second, we compare the microclassifiers to CNNs operating on raw pixels and demonstrate that microclassifiers scale more effectively to large numbers of applications.

We evaluate the frame-by-frame accuracy of our microclassifiers on the tasks described above using the following

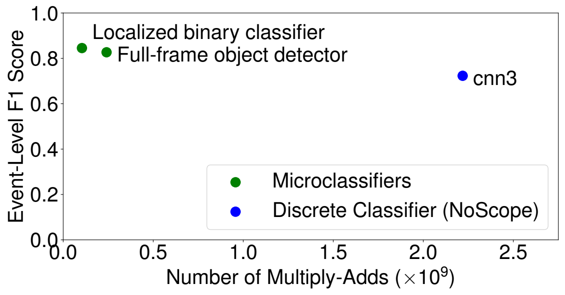


Figure 7. Number of operations versus event-level F1 score for various cheap classifier architectures. Models are trained on the *Roadway* dataset (2048x850 pixel input resolution). The microclassifiers are very cheap, but rely on features extracted from a base DNN (e.g., MobileNet). With sufficiently large numbers of microclassifiers, this upfront cost is amortized away.

training-test splits. In the *Jackson* dataset, we sampled the first and second 6 hour segments at 15 fps to obtain 300,000 training and test frames respectively. Similarly, for the *Roadway* dataset, we sample the two adjacent 3 hour segments at 15 fps to obtain 162,000 training and test frames respectively. These splits ensure that only past information is used for training.

To illustrate the microclassifier's better scaling, we compare the accuracy and computational cost of microclassifiers and discrete classifiers on the *People with Red* task. We count the number of multiply-adds within each model and use the F_1 score—the harmonic mean of precision and recall—as an accuracy metric that captures class imbalance. We constructed discrete classifiers having between roughly 100 million and 2.5 billion multiply-adds. They varied in the number of convolutional layers (2-4), number of kernels (16-64), stride length (1-3), number of pooling layers (0-2), and type of convolution (standard or separable). Kernel sizes are fixed to 3. The best-peforming (highest accuracy without unnecessary extra cost) discrete classifier we devised is "cnn3" in Figure 7. In this figure, a small 2-layer CNN microclassifier with a few hundred million multiply-adds achieves comparable (actually, better) F_1 score to a much larger 2.25B multiply-add discrete classifier (*cnn3*). Using these two models as representative workloads and factoring in the cost of running MobileNet (18 billion multiply-adds on full resolution images), we find that, the microclassifier approach wins out with more than 8 queries in theory, and empirically performs better for deployments running 4 or more concurrent queries (the FMA count and the runtime are not identical; in practice, the mobilenets ops are somewhat cheaper, perhaps due to much of the cost happening in larger convolutions).

6 Conclusion

Scaling wide-area live video analytics poses a challenge for bandwidth-limited, compute-constrained camera deployments. FilterForward is designed to connect cameras at the edge with video analysis applications in the datacenter when the edge nodes face bandwidth constraints due to, e.g., physical topology or cost. To reduce bandwidth use, FilterForward filters events at the edge and only forwards to the datacenter sequences of frames that are relevant to analysis applications. For scalable and high-accuracy filtering with limited computation resources at the edge, FilterForward shares computation between multiple concurrent applications (or queries) by deriving per-query microclassifiers from a shared base DNN—but unlike traditional transfer learning approaches, the microclassifiers can draw from arbitrary interior activations in the base DNN. We show empirically that extracting and transmitting high-quality, compressed, short video segments of events can save up to an order of magnitude of the bandwidth that would be needed to stream the equivalent-quality full video, and that further dropping the transmitted bitrate would be deleterious for the accuracy of the in-datacenter application. Although there are many more challenges in fully realizing the vision of wide-area video analytics at scale, we believe that these mechanisms for enabling edge-to-cloud hybrid video analytics represents a useful advance for this emerging environment.

References

Apple. The future is here: iPhone X. `https://www.apple.com/newsroom/2017/09/the-future-is-here-iphone-x/`, 2017.

Babenko, A. and Lempitsky, V. Aggregating local deep features for image retrieval. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision*, pp. 850–865. Springer, 2016.

caffe. Caffe. `http://caffe.berkeleyvision.org/`, 2017.

Caruana, R. Multitask learning. In *Learning to learn*, pp. 95–133. Springer, 1998.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pp. 647–655, 2014.

FCC. 2016 BROADBAND PROGRESS REPORT. 2016.

Google Wireless Internet. Google's Excellent Plan To Bring Wireless Internet To Developing Countries. https://www.forbes.com/sites/timworstall/2013/05/25/googles-excellent-plan-to-bring-wireless-internet-to-developing-countries/, 2013.

Han, S., Mao, H., and Dally, W. J. Deep Compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proc. ICLR 2016*, 2016.

Hariharan, B., Arbeláez, P., Girshick, R., and Malik, J. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 447–456, 2015.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

IHS. Top Video Surveillance Trends for 2017. https://cdn.ihs.com/www/pdf/TEC-Video-Surveillance-Trends.pdf, 2017.

intel-mkl-dnn. Intel Math Kernel Library for Deep Neural Networks. https://01.org/mkl-dnn, 2018.

intel-movidius. Intel Movidius Neural Compute Stick. https://developer.movidius.com/, 2018.

ITU/UNESCO Broadband Commission for Sustainable Development. The State of Broadband: Broadband catalyzing sustainable development. 2017.

Jiang, A., Wong, D. L.-K., Canel, C., Misra, I., Kaminsky, M., Kozuch, M., Pillai, P., Andersen, D. G., and Ganger, G. R. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., et al. In-datacenter performance analysis of a tensor processing unit. *CoRR*, abs/1704.04760, 2017. URL http://arxiv.org/abs/1704.04760.

Kang, D., Emmons, J., Abuzaid, F., Bailis, P., and Zaharia, M. Noscope: Optimizing deep cnn-based queries over video streams at scale. *PVLDB*, 10(11):1586–1597, 2017.

Lee, T. J., Gottschlich, J., Tatbul, N., Metcalf, E., and Zdonik, S. Precision and recall for range-based anomaly detection. In *Proc. SysML Conference*, Stanford, CA, February 2018.

Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, 2017.

Ma, C., Huang, J.-B., Yang, X., and Yang, M.-H. Hierarchical convolutional features for visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3074–3082, 2015.

microsoft-project-brainwave. Microsoft unveils Project Brainwave for real-time AI. https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/, 2018.

Pakha, C., Chowdhery, A., and Jiang, J. Reinventing video streaming for distributed vision analytics. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, Boston, MA, 2018. USENIX Association. URL https://www.usenix.org/conference/hotcloud18/presentation/pakha.

Ran, X., Chen, H., Zhu, X., Liu, Z., and Chen, J. DeepDecision: A mobile deep learning framework for edge video analytics. In *Proc. INFOCOM*, 2018.

Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. Cnn features off-the-shelf: An astounding baseline for recognition. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, CVPRW '14, 2014.

Redmon, J. and Farhadi, A. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. URL http://arxiv.org/abs/1612.08242.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

Sharghi, A., Laurel, J. S., and Gong, B. Query-focused video summarization: Dataset, evaluation, and A memory network based approach. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, CVPR 2017.

Sharma, S., Kiros, R., and Salakhutdinov, R. Action recognition using visual attention. *arXiv preprint arXiv:1511.04119*, 2015.

Wang, J., Feng, Z., Chen, Z., George, S., Bala, M., Pillai, P., Yang, S.-W., and Satyanarayanan, M. Bandwidth-efficient live video analytics for drones via edge computing. In *Proceedings of the Third ACM/IEEE Symposium on Edge Computing (SEC 2018)*, Bellevue, WA, 2018.

Yeung, S., Russakovsky, O., Mori, G., and Fei-Fei, L. End-to-end learning of action detection from frame glimpses in videos. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, CVPR 2016.

Yi, S., Hao, Z., Zhang, Q., Zhang, Q., Shi, W., and Li, Q. LAVEA: Latency-aware video analytics on edge computing platform. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pp. 3320–3328, 2014.

Yue-Hei Ng, J., Yang, F., and Davis, L. S. Exploiting local features from deep networks for image retrieval. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2015.

Zhang, H., Ananthanarayanan, G., Bodik, P., Philipose, M., Bahl, P., and Freedman, M. J. Live video analytics at scale with approximation and delay-tolerance. In *Proc. 14th USENIX NSDI*, Boston, MA, March 2017.