

# Picking Interesting Frames in Streaming Video

Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim,  
David G. Andersen, Michael Kaminsky<sup>†</sup>, Subramanya R. Dulloor<sup>†</sup>  
*Carnegie Mellon University; <sup>†</sup>Intel Labs*

## 1 INTRODUCTION

As video camera deployments proliferate in the smart cities of the future [2], software systems are faced with the increasing challenge of determining which segments of data are relevant. For resource-constrained edge nodes, limited network bandwidth back to the datacenter prevents sending entire video streams.

This paper presents a new application-independent *interesting frame (IF) detection algorithm* for identifying relevant frames in streaming video. We envision this IF detector as a preprocessing step in a larger video analytics pipeline where the expensive computation occurs later (similar to the way Bloom filters can guard expensive data structures). Given a target frame rate (or, equivalently, a target bandwidth), the algorithm decides which frames are the most generally interesting and therefore should be processed by downstream applications or forwarded to the datacenter. We decide how “interesting” a frame is based on its semantic difference from other frames. The IF detection algorithm uses a hierarchy of filters to trade off between end-to-end latency and aggressive decimation. The algorithm strives to maximize the semantic diversity of the selected frames. Compared to simply choosing frames at a fixed interval, the IF detector better handles bursty events in the stream.

This algorithm leverages the fact that many (though certainly not all) video analytics applications can tolerate some latency [7]. Many of these applications have latency tolerances of seconds to minutes (E.g., a “real-time” traffic measurement system can run several tens of seconds behind realtime with limited impact to users). The IF detector is therefore safe to accumulate the incoming video in a buffer and periodically run its algorithm over a window of frames.

A defining feature of our algorithm is that it selects exactly the desired number of interesting frames, creating a uniform output framerate from a nonuniform stream of interesting frames. This is in contrast to a more dynamic algorithm that adjusts the number of selected frames based on the content of the video. We designed the IF detector to produce a uniform output framerate to facilitate provisioning compute resources for maximum utilization by preventing bursty frame delivery. We push the configuration of the selectivity to the higher-level applications or the resource control layer, which are free to adjust it dynamically if necessary.

Because the IF detector operates on semantic information, it is suited to a larger range of videos (E.g., streams where the camera moves) than pixel-level difference detectors, such as those used in NoScope [3]. The algorithm does not rely on hand-tuned features, like [4], or require online [9] or offline [4, 8] training. Instead, semantic information is extracted by a pre-trained DNN. Therefore, the IF detector can be applied to larger deployments without the need for specialized tuning for each camera. Our streaming detection algorithm differs from most typical video summarization [4, 5, 8] work in that supporting aggressive decimation is a first class priority, as opposed to forming a representative “story”.

## 2 FINDING INTERESTING FRAMES

**Extracting Semantic Content.** We extract semantic content from incoming frames by evaluating a Deep Neural Network (DNN) and selecting the output from one of the later max-pooling layers as a feature vector. These feature vectors form the basis of the IF detector algorithm. For our experiments, we used the MobileNet [1] architecture, trained on ImageNet [6], and extracted the activations of the “pool6” layer, producing 1024-value feature vectors. Evaluating the DNN is the most computationally intensive phase.

Selecting an appropriate DNN and feature vector representation is key to the IF detector’s generality. We selected MobileNet, a classification network, for our experiments because we found that it was sensitive to content changes while being relatively lightweight. Importantly, the IF detector is agnostic to both the network and the feature vector representation. If the network is not sensitive to the events that an application is interested in, then a different network or feature vector encoding can be used. More advanced methods for computing the feature vectors are the focus of ongoing work.

**Detection Algorithm.** Incoming frames are accumulated in a frame buffer until the buffer is full, at which time all of the frames are passed to the detection algorithm and the buffer is cleared. Consider all of the frames as a field of points in  $N$ -dimensional space, where  $N$  is the length of a feature vector. The location of each frame is given by its feature vector. Let each of these points be a node in a directed, acyclic graph (DAG). Label each node by its frame index. For a node with frame index  $i$ , create a directed edge pointing from node  $i$  to node  $j$  for all nodes  $j$  where  $j > i$ . In other words, each frame has an edge pointing to every future frame. Weight these edges by the distance between frames  $i$  and  $j$  in that  $N$ -dimensional space. This encodes semantic difference information in the structure of the DAG (I.e., dissimilar frames thus have higher-weight edges connecting them) and is the core of the semantic content comparison. For our prototype, we used Euclidean distance as our similarity metric, but other distance metrics (E.g., cosine distance) may be appropriate as well.

Now, the task of finding the  $k$  most interesting frames reduces to the problem of finding the longest  $k$ -node path in this DAG. Intuitively, the longest path consists of the frames that are maximally different, and therefore the most interesting. Our IF detection algorithm adds the last interesting frame from the previous buffer to the graph and uses it as the source of the path, so the algorithm solves for the longest  $(k + 1)$ -node path starting with the last interesting frame. Importantly, the frame at which the path ends is not fixed.

Since the graph is a DAG, this algorithm is computationally feasible. For a buffer size of  $B$  frames, the algorithmic complexity of our dynamic programming solution for finding the longest path consisting of  $k$  nodes is  $O(kB^2)$ , with a space complexity of  $O(B^2 + kB)$ . Our implementation is a modification of the Floyd–Warshall algorithm for finding the all-pairs shortest paths in a graph.

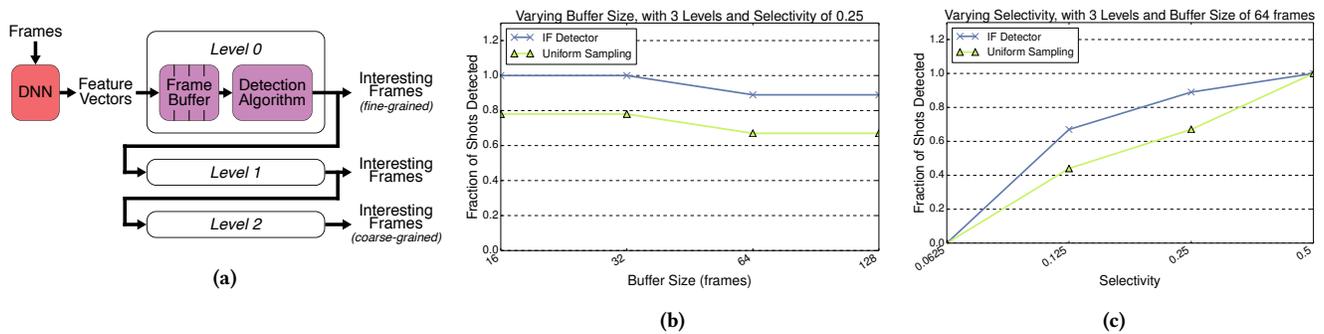


Figure 1: System architecture and detection accuracy when varying the buffer size and selectivity.

Although this algorithm succeeds in its requirement of achieving exactly the desired reduction factor, the characteristics of the results depend on the buffer size. The algorithm selects exactly the desired number of interesting frames from each buffer. If the content of the video stream is such that one buffer contains no truly interesting frames and the next buffer contains only interesting frames, then the algorithm would output first  $k$  boring frames and then  $k$  interesting frames. Therefore, using a large buffer size causes the IF detection algorithm to select more globally-interesting frames, whereas a small buffer results in very localized results. The optimal size depends on the “stability” of the video stream. However, using a very large buffer has significant latency and memory overheads due to the polynomial complexity of the longest path algorithm.

**Hierarchical Architecture.** To mitigate the algorithm’s overheads and support aggressive decimation, we build a hierarchy of IF detectors, each with a smaller buffer than when using a single, monolithic IF detector. When the buffer at level  $i$  reaches capacity, the IF detection algorithm executes and places the detected frames in the buffer for level  $i + 1$ . Assuming that each level uses the same selectivity  $d$  and buffer size  $B$ , an  $h$ -level hierarchy can accomplish an effective selectivity of  $d^h$  while only storing  $B \times h$  frames. The total CPU cost of all levels of the IF detection hierarchy will scale as  $O(hkB^2)$ . Increasing  $h$  has a linear effect on CPU usage, which is preferable to the quadratic effect of scaling the buffer size. Figure 1a shows a three-level hierarchy.

This hierarchical design is also attractive because it enables the IF detection algorithm to consider frames from extremely large segments of the video stream. Whereas lower levels of the hierarchy find interesting frames at a very fine granularity, the higher levels operate at a very coarse granularity. When the highest level in the hierarchy executes detection, it has a visibility of  $(1/d)^{h-1} \times B$  frames. As a concrete example, a four-level hierarchy with a buffer size of 1000 and a selectivity of 0.1 would choose the 100 most interesting frames from almost 9.3 hours (at 30 fps) of video while storing only 4,000 frames in memory. The non-hierarchical version would need to buffer 1,000,000 frames to achieve the same visibility. Comparisons show that the hierarchical design selects similar frames to a single, monolithic buffer.

Furthermore, applications are not restricted to using interesting frames from only the highest level in the hierarchy. By pulling from

all intermediate IF detectors simultaneously, applications have a choice of prioritizing latency or achieving an aggressive selectivity.

### 3 PRELIMINARY EVALUATION

**Performance.** The IF detector’s running time depends primarily on the length of the frame buffer, but the algorithm runs only once each time the buffer fills. For typical buffer sizes—fewer than 1000 frames—the IF detector’s processing time is small. (Larger buffers are not required with the hierarchical design.) For example, running the detection algorithm on a buffer of 1000 frames takes, per frame, approximately 16% of the time to evaluate the full DNN.

**Accuracy.** To evaluate the quality of the frames that our algorithm selects, we manually annotated the events in several minutes of video from a stationary camera overlooking a railway. We consider an event boundary to be when a train enters or leaves the frame, so the ground truth events look like “train”, “no train”, “train”, etc. Our metric is the fraction of these events from which at least one frame is selected, and for this preliminary evaluation we compare against frame filtering through uniform sampling. However, not all of the IF detector levels fill for every configuration: aggressive selectivities or large buffer sizes might require hundreds of thousands to millions of frames for the buffers to fill and trigger a detection. To determine the results of the uniform sampling algorithm, we treated the last frame that traversed the IF detector as the endpoint of the experiment, then selected from that region the same number of frames as the IF detector using a uniform stride.

Figures 1b and 1c demonstrate that for various buffer sizes and selectivities, the IF detector consistently achieves superior shot coverage compared to uniform sampling. These experiments use a three-level hierarchy. The IF detector is sensitive to the large semantic difference that is a train entering or leaving because those events trigger a larger disturbance in the feature vectors than most of the ambient motion in the scene. In Figure 1b, the reason that the ratio of shots detected decreases as the buffer size increases is that other action in the scene (E.g., cloud activity) causes a large change in the feature vectors. With large buffers, the IF detector focuses on the other changes, instead of the presence of a train. This can be remedied by using a DNN or feature vector representation that is more in tune with the specific application, or by using smaller buffers, thereby forcing the detected frames to be more evenly temporally distributed.

## REFERENCES

- [1] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017). <http://arxiv.org/abs/1704.04861>
- [2] IHS 2017. Top Video Surveillance Trends for 2017. <https://cdn.ihs.com/www/pdf/TEC-Video-Surveillance-Trends.pdf>. (2017).
- [3] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Deep CNN-Based Queries over Video Streams at Scale. *PVLDB* 10, 11 (2017), 1586–1597.
- [4] Yong Jae Lee, Joydeep Ghosh, and Kristen Grauman. 2012. Discovering Important People and Objects for Egocentric Video Summarization. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- [5] Zheng Lu and Kristen Grauman. 2013. Story-Driven Summarization for Egocentric Video. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '13)*.
- [6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.
- [7] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.
- [8] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. 2016. Video Summarization with Long Short-Term Memory. In *2016 European Conference on Computer Vision (ECCV '16)*.
- [9] Bin Zhao, Li Fei-Fei, and Eric P. Xing. 2011. Online Detection of Unusual Events in Videos via Dynamic Sparse Coding. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.