Efficient Inference on Video, In Real-Time and At Scale

by

Samvit Jain

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Joseph Gonzalez, Chair
Professor Trevor Darrell
Professor Ion Stoica

Spring 2019

The thesis of Samvit Jain, titled Efficient Inference on Video, In Real-Time and At Scale, is approved:

Chair	[illegible]	Date	5/2/2019
	[illegible]	Date	5/2/2019
	[illegible]	Date	5/2/2019

University of California, Berkeley

Efficient Inference on Video, In Real-Time and At Scale

Abstract

Efficient Inference on Video, In Real-Time and At Scale

by

Samvit Jain

Master of Science in Computer Science

University of California, Berkeley

Professor Joseph Gonzalez, Chair

Neural network inference has transformed computer vision, enabling the automated detection, tracking, and analysis of objects and activities in visual data at human-level accuracy. Applying these deep models to real applications, however, requires careful attention to performance. Model complexity renders inference too slow, in use-cases that demand high throughput execution (e.g. autonomous perception), and too costly, in use-cases that require operation at the scale of hundreds of video streams (e.g. post-facto video search).

To address these challenges, I pursue two lines of inquiry.

On the inference speed side, I consider the task of accelerating semantic segmentation, a classic image recognition task with applications in autonomous perception, on video by leveraging motion information. Specifically, I explore the use of block motion fields from compressed video (e.g. MPEG-4 / H.264) to warp deep representations, and a new, two-stream network architecture to correct warping error. These techniques seek to amortize the cost of extracting image features, the bottleneck in many neural architectures, over multiple video frames, while preserving, and in some cases, boosting, model accuracy.

On the compute cost side, I investigate how cross-camera person tracking, a video analytics task with applications in security and retail intelligence, can be executed efficiently on multiple video streams. Here I demonstrate how a profile of cross-camera correlations, built offline on historical video data, can be used as a spatial and temporal filter, ruling out cameras and frames unlikely to contain the target identity at inference time. My experiments shows that this filtering, together with a fallback mechanism, can substantially reduce compute cost, as well as improve precision, on video analytics workloads.

This body of work is motivated by a simple statement: *machine learning systems must meet the performance requirements of the applications they enable.* Advances in deep learning applied to vision have unlocked opportunity in robotic navigation, industrial and agricultural monitoring, and retail intelligence, each use case with its own latency, throughput, and cost constraints. This thesis is a step toward solving this constrained optimization problem.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to begin by thanking one of my undergraduate research advisors, Professors Edward Felten. It is not common for an undergraduate student to get the chance to work one-on-one with a senior faculty member with commitments in academia and government, and I am thankful for this opportunity. Working with you, I got my start in research.

I would also like to thank Professor Brian Kernighan, another one of my undergraduate advisors, for being the voice of wisdom on my senior thesis. I am grateful for your kind and unreserved support, continued to this day. Your promptness is unmatched.

I would also like to acknowledge my mentors at Microsoft Research, Yuanchao Shu and Junchen Jiang. Our work together on large-scale video analytics, beginning last summer, forms a critical part of my Master's thesis arc. I appreciate your guidance and support.

Importantly, I would like to thank my Master's research advisor, Joey Gonzalez. This thesis will mark a pause on a close two-year collaboration, spanning multiple projects, many paper submissions, and several key publications. I appreciate your close interest and involvement in all of my Master's work, your creativity and technical insight, and your conscientious, empathetic advising style. It is clear to me now that advising is an art. Your mentorship reflects deliberate attention to this fact. I am grateful for your support.

Next, I would like to thank my sister. Thank you Ananya for always having your door open to share a story or a laugh.

Finally, I would like to thank Mom and Dad. Thank you for always being available for a conversation. Dad, thank you for corresponding with me so closely last winter, when all was new. Mom, thank you for being the voice of reassurance, always. Your selflessness is unbounded. For this and much more, I am grateful.

Chapter 1

Introduction

A large and growing majority of the data streamed on the Internet today is video. In 2016, video constituted 73% of web traffic, more than gaming (1%), file sharing (8%), and basic web data (18%), a figure expected to reach 82% by 2021 [5]. Within video traffic, the most rapidly growing segment is *live video* – video that is streamed as it is captured, to social media users, anomaly detection systems, and other entities, human and non-human. Live video formed only 3% of video traffic in 2016, but its share is projected to triple by 2021 [5].

Underpinning this growth is a plethora of new use cases – journaling and event broadcasting (Facebook Live, Twitter Periscope); visual monitoring for industrial and agricultural oversight, retail intelligence, and public security; and perception-based autonomous systems (Tesla Autopilot). The video these applications accept as input originates from a wide range of sources, including cameras on mobile phones, closed-circuit installations, and vehicles.

Many of the above use cases involve querying or transforming this video to make decisions. We can broadly divide these applications into two categories: perception and analytics. Perception tasks entail high-resolution inference (e.g. object tracking, scene segmentation), executed on single video streams. Analytics tends to involve simpler inference queries (e.g. object search, frame classification), executed on multiple streams or large data stores. Both classes of applications share similar goals: high accuracy, high inference speed, and low compute cost. In general, perception tasks tend to be constrained by inference speed (frames per second), while analytics tasks tend to be constrained by inference cost (dollars per query).

In this thesis, I address both classes of application in detail. In Part I, I investigate the problem of accelerating semantic segmentation, a classic perception task, on video. My focus here is how insights about both data (temporal continuity in video) and models (representation sharing across frames) can be used to achieve better throughput, lower latency, and higher accuracy at inference time. Specifically, I show that multi-frame neural networks that cache image representations, and leverage motion information in video to propagate these representations, can outperform networks that process each video frame in isolation.

In Part II, I examine the opposite end of the spectrum: simpler analytics tasks, executed on *many* concurrent video streams. In particular, I look at how cross-camera person search, the task of tracking a query identity in real-time through a camera network, can be executed more effectively by leveraging historical knowledge of cross-camera traffic patterns. This past data, encoded as a spatio-temporal correlation model indicating the likelihood of particular camera-to-camera trajectories, can be used to filter out cameras and frames unlikely to contain the target identity at inference time. Our experiments in Part II show that such spatio-temporal pruning enables substantially lower compute cost, and higher tracking precision, than blind cross-camera person search.

Together, these two respective lines of work, on high-throughput perception and low-cost video analytics, illustrate the promising applicability of ideas from computer systems to problems in machine learning. With this thesis, I hope to highlight the critical role that research on *performance-oriented computer vision*, one key area where systems and machine learning meet, will play in enabling new applications of machine intelligence.

Part I

Efficient Semantic Segmentation

Chapter 2

Inference on Compressed Video

2.1 Introduction

Semantic segmentation, the task of assigning each pixel in an image to a semantic object class (e.g. “sky”, “vehicle”, “person”), is a problem of long-standing interest in computer vision. Like models for other image recognition tasks (e.g. classification, detection, instance segmentation), semantic segmentation networks have grown drastically in both layer depth and parameter count in recent years, in the effort to segment more complex images, from larger, more realistic datasets, at higher accuracy. As a result, state-of-the-art segmentation networks today require between 0.5 to 3.0 seconds to segment a *single*, high-resolution image (e.g. 2048×1024 pixels) at competitive accuracy [105, 37].

Meanwhile, a new target data format for segmentation has emerged: video. The motivating use cases include both batch applications, where video is segmented in bulk to generate training data for other models (e.g. autonomous control systems), and streaming applications, where high-throughput video segmentation enables interactive analysis of live footage (e.g. at surveillance sites). Video in these contexts consists of long image sequences, shot at high frame rates (e.g. 30 fps) in complex environments (e.g. urban cityscapes) on modern, high-definition cameras. Segmenting individual frames at high accuracy still calls for the use of competitive image models, but their inference cost precludes their naïve deployment on every frame in a raw multi-hour video stream.

A defining characteristic of realistic video is its high level of temporal continuity. Consecutive frames demonstrate significant spatial similarity, which suggests the potential to reuse computation across frames. Building on prior work, we exploit two observations: 1) higher-level features evolve more slowly than raw pixel content in video, and 2) feature computation tends to be much more expensive than task-specific computation across a range of vision tasks (e.g. detection, segmentation) [81, 105]. Accordingly, we divide our semantic segmentation model into a deep *feature network* and a cheap, shallow *task network* [105]. We compute features only on designated keyframes, and propagate them to intermediate frames, by warping the feature maps with frame-to-frame motion estimates. The task net-

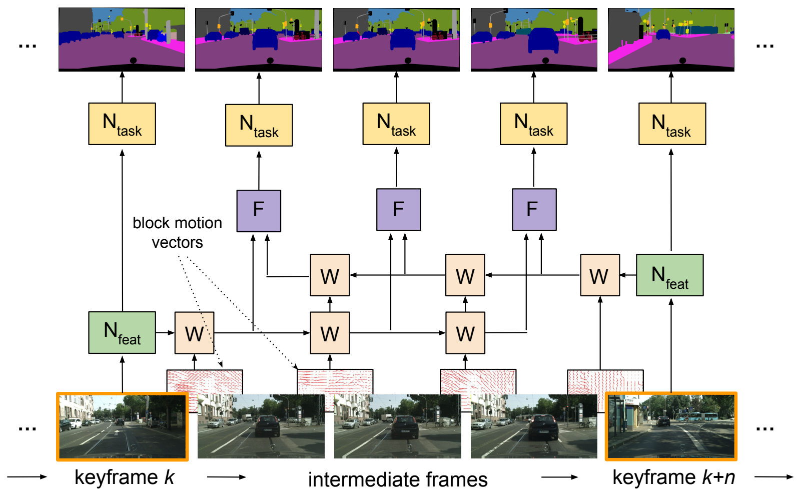


Figure 2.1: **interpolation-BMV** warps and fuses (**interpolates**) the features of enclosing keyframes to generate accurate feature estimates for intermediate frames, using the block motion vectors (**BMVs**) present in compressed (H.264 / HVEC) video.

work is executed on all frames. Given that feature warping and task computation is much cheaper than feature extraction, a key parameter we aim to optimize is the interval between designated keyframes.

Here we make two key contributions. First, noting the high level of data redundancy in video, we successfully utilize an artifact of compressed video, block motion vectors (BMV), to cheaply propagate features from frame to frame. Unlike other motion estimation techniques, which require specialized computation, block motion vectors are freely available in modern video formats, making for a simple, fast design. Second, we propose a novel feature estimation technique that enables the features for a large fraction of video frames to be inferred accurately and efficiently (see Fig. 2.1). In particular, when computing the segmentation for a keyframe, we also precompute the features for the *next* designated keyframe. Features for all subsequent intermediate frames are then computed as a *fusion* of features warped forward from the last visited keyframe, and features warped backward from the incoming keyframe. This procedure implements an *interpolation* of the features of the two closest keyframes. We then combine the two ideas, using block motion vectors to perform the feature warping in feature interpolation. The result is a scheme we call **interpolation-BMV**.

We evaluate our framework on the CamVid and Cityscapes datasets. Our baseline consists of running a competitive segmentation network, DeepLab [18], on every frame, a setup that achieves published accuracy [25], and throughput of 3.6 frames per second (fps) on CamVid and 1.3 fps on Cityscapes. Our improvements come in two phases. First, our use of motion vectors for feature propagation allow us to cut inference time on intermediate frames by 53%, compared to approaches based on optical-flow, such as [105]. Second, our bi-directional feature warping and fusion scheme achieves substantial accuracy improvements, especially at high keyframe intervals. Together, the two techniques allow us to operate at over twice the average inference speed as the fastest prior work, at any target level of accuracy. For example, if we are willing to tolerate no worse than 65 mIoU on our CamVid video stream, we are able to operate at a throughput of 20.1 fps, compared to the 8.0 fps achieved by the forward flow-based propagation from [105]. Overall, even when operating in high accuracy regimes (e.g. within 3% mIoU of the baseline), we are able to accelerate segmentation on video by a factor of 2-6$\times$.

2.2 Related Work

Image Semantic Segmentation

Semantic segmentation is a classical image recognition task in computer vision, originally studied in the context of statistical inference. Historically, the approach of choice was to propagate evidence about pixel class assignments through a probabilistic graphical model [33, 82], a technique that scaled poorly to large images with numerous object classes [57]. In 2014, Long et al. [64] proposed the use of fully convolutional neural networks (FCNs) to segment images, demonstrating significant accuracy gains on several key datasets. Subsequent work embraced the FCN architecture, proposing augmentations such as dilated (atrous) convolutions [96], post-processing CRFs [19], and pyramid spatial pooling [102] to further improve accuracy on large, complex images.

Efficient Video Semantic Segmentation

The recent rise of applications such as autonomous driving, industrial robotics, and automated video surveillance, where agents must perceive and act on the visual world *as it evolves*, has triggered substantial interest in the problem of efficient video semantic segmentation. Shelhamer et al. [81] and Zhu et al. [105] proposed basic feature reuse and optical flow-based feature warping, respectively, to reduce the inference cost of running expensive image segmentation models on video. Recent work explores adaptive feature propagation, partial feature updating, and adaptive keyframe selection as techniques to further optimize the scheduling and execution of optical-flow based warping [106, 60, 93]. In general, these techniques fall short in two respects: (1) optical flow computation remains a computational bottleneck, especially as other network components become cheaper, and (2) forward feature propagation fails to account for other forms of temporal change, besides spatial displacement, such as new scene content (e.g. new objects), perspective changes (e.g. camera pans), and observer movement (e.g. in driving footage). As a result, full frame features must still be recomputed frequently to maintain accuracy, especially in video footage with complex dynamics, fundamentally limiting the attainable speedup.

Motion and Compressed Video

Wu et al. [90] train a network directly on compressed video to improve both accuracy and performance on video action recognition. Zhang et al. [98] replace the optical flow network in the classical two-stream architecture [83] with a "motion vector CNN", but encounter accuracy challenges, which they address with various transfer learning schemes. Unlike these works, our main focus is not efficient training, nor reducing the physical size of input data to strengthen the underlying signal for video-level tasks, such as action recognition. We instead focus on a class of dense prediction tasks, notably semantic segmentation, that involve high-dimensional output (e.g. a class prediction for every pixel in an image) generated on the original uncompressed frames of a video. This means that we must still process each frame in isolation. To the best of our knowledge, we are the first to propose the use of compressed video artifacts to warp deep neural representations, with the goal of drastically improved inference throughput on realistic video.

2.3 System Overview

Network Architecture

We follow the common practice of adapting a competitive image classification model (e.g. ResNet-101) into a fully convolutional network trained on the semantic segmentation task [64, 97, 18]. We identify two logical components in our final model: a *feature network*, which takes as input an image $i \in R^{1\times 3\times h\times w}$ and outputs a representation $f_i \in R^{1\times A\times \frac{h}{16}\times \frac{w}{16}}$, and a *task network*, which given the representation, computes class predictions for each pixel in the image, $p_i \in R^{1\times C\times h\times w}$.

The feature network N_{feat} is obtained by eliminating the final, k-way classification layer in the chosen image classification architecture. The task network N_{task} is built by concatenating four blocks: (1) a feature projection block, which reduces the feature channel dimensionality from A to $\frac{A}{2}$ through a 1×1 convolutional layer and ReLU, (2) a scoring block, a 1×1 convolutional layer which outputs scores for each of the C segmentation classes, (3) an upsampling block, which bilinearly upsamples the score maps to the resolution of the input image, and (4) a prediction block, which converts scores to normalized class probabilities (softmax) and then a final class prediction (argmax) for each pixel.

Block Motion Vectors

MPEG-compressed video consists of two logical components: reference frames, called I-frames, and delta frames, called either P-frames or B-frames. Reference frames are still RGB frames from the video, usually represented as spatially-compressed JPEG images. Delta frames, which introduce temporal compression to video, consist of two subcomponents: block motion vectors and residuals.

Block motion vectors, the artifact of interest in our current work, define a correspondence between pixels in the current frame and pixels in the previous frame. They are generated using *block motion compensation*, a standard procedure in video compression algorithms [77]:

1. Divide the current frame into a non-overlapping grid of 16×16 pixel blocks.
2. For each block in the current frame, determine the "best matching" block in the previous frame. A common matching metric is to minimize mean squared error between the blocks.
3. For each block in the current frame, represent the pixel offset to the best matching block in the previous frame as an (x, y) coordinate pair, or *motion vector*.

The resulting grid of (x, y) offsets forms the *block motion vector map* for the current frame. For a $16M \times 16N$ frame, this map has dimensions $M \times N$. The residuals then consist of the pixel-level difference between the current frame, and the previous frame transformed by the motion vectors.

Feature Propagation

Many cameras compress video by default as a means for efficient storage and transmission. The availability of a free form of motion estimation at inference time, the motion vector maps in MPEG-compressed video, suggests the following scheme for fast video segmentation (see **Algorithm 1**).

Algorithm 1 Feature propagation with block motion vectors (**prop-BMV**)

```
1: input: video frames $\{I_i\}$, motion vectors mv, keyframe interval $n$
2: for frame $I_i$ in $\{I_i\}$ do
3:     if i mod n = 0 then                                ▷ keyframe
4:         $f_i \leftarrow N_{feat}(I_i)$                  ▷ keyframe features
5:         $S_i \leftarrow N_{task}(f_i)$
6:     else                                               ▷ intermediate frame
7:         $f_i \leftarrow \textsc{warp}(f_c, -\text{mv}[i])$   ▷ warp cached features
8:         $S_i \leftarrow N_{task}(f_i)$
9:     end if
10:    $f_c \leftarrow f_i$                                ▷ cache features
11: end for
12: output: frame segmentations $\{S_i\}$
```

Choose a keyframe interval n. On keyframes (every n^{th} frame), execute the feature network N_{feat} to obtain a feature map. Cache these computed features, f_c, and then execute the task network N_{task} to obtain the keyframe segmentation. On intermediate frames, extract the motion vectors mv$[i]$ corresponding to the current frame index. Warp the cached features

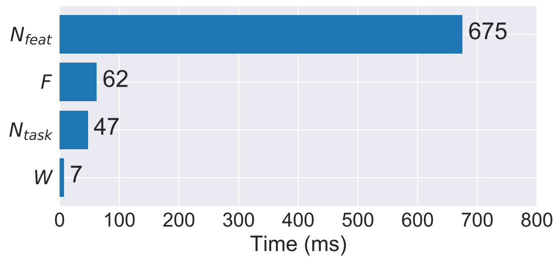


Figure 2.2: A sample runtime breakdown for a ResNet-101 DeepLab network. F is the optical flow network [34]. W is the warp operator. GPU: Tesla K80. Dataset: *Cityscapes*.

f_c one frame forward via bilinear interpolation with $-\text{mv}[i]$. (To warp forward, we apply the negation of the vector map.) Here we employ the differentiable, parameter-free spatial warping operator proposed by [48]. Finally, execute N_{task} on the warped features to obtain the current segmentation.

Inference Runtime Analysis

Feature propagation is effective because it relegates feature extraction, the most expensive network component, to select keyframes. Of the three remaining operations performed on intermediate frames – motion estimation, feature warping, and task execution – motion estimation with optical flow is the most expensive (see Fig. 2.2). By using block motion, we eliminate this remaining bottleneck, accelerating inference times on intermediate frames for a DeepLab segmentation network [18] from 116 ms per frame ($F + W + N_{task}$) to 54 ms per frame ($W + N_{task}$) on the Cityscapes dataset. For keyframe interval n, this translates to a speedup of 53% on $\frac{n-1}{n}$ of the video frames.

Note that for a given keyframe interval n, as we reduce inference time on intermediate frames to zero, we approach a maximum attainable speedup factor of n over a frame-by-frame baseline that runs the full model on every frame. Exceeding this bound, without compromising on accuracy, requires an entirely new approach to feature estimation, the subject of the next section.

Incidentally, we also benchmarked the time required to extract block motion vectors from raw video (i.e. H.264 compression time), and found that `ffmpeg` takes 2.78 seconds to compress 1,000 Cityscapes video frames, or 2.78 ms per frame. In contrast, optical flow computation on a frame pair takes 62 ms (Fig. 2.2). We include this comparison for completeness: since compression is a default behavior on modern cameras, block motion extraction is not a true component of inference time.

Feature Interpolation

Given an input video stream, our goal is to compute the segmentation of every frame as efficiently as possible, while preserving accuracy. In a batch setting, we have access to the entire video, and desire the segmentations for all the frames, as input to another model (e.g. an autonomous control system). In a streaming setting, we have access to frames as they come in, but may be willing to tolerate a small delay of keyframe interval n frames ($\frac{n}{30}$ seconds at 30 fps) before we output a segmentation, if that means we can match the throughput of the video stream and maintain high accuracy.

We make two observations. First, all intermediate frames in a video by definition lie between two designated keyframes, which represent bounds on the current scene. New objects that are missed in forward feature propagation schemes are more likely to be captured if both past and incoming keyframes are used. Second, feature fusion techniques are effective at preserving strong signals in any one input feature map, as seen in [31]. This suggests the viability of estimating the features of intermediate frames as the fusion of the features of enclosing keyframes.

Expanding on this idea, we propose the following algorithm (see Fig. 2.1). On any given keyframe, precompute the features for the *next* keyframe. On intermediate frames, warp the previous keyframe's features, $N_{feat}(I_k)$, forward to the current frame I_i using incremental forward motion estimates, $-\text{mv}[k : i]$. Warp the next keyframe's features, $N_{feat}(I_{k+n})$, *backward* to the current frame using incremental backward motion estimates, $\text{mv}[k+n : i]$. Fuse the two feature maps using either a weighted average or learned fusion operator, F. Then execute the task network N_{task} on the fused features. This forms **Algorithm 2**.

To eliminate redundant computation, on keyframes, we *precompute* forward and backward warped feature maps f^f, f^b corresponding to each subsequent intermediate frame, $\{I_{k+1}, ..., I_{k+n-1}\}$. For keyframe interval n, this amounts to $n-1$ forward and $n-1$ backward warped feature maps.

Feature Fusion

We consider several possible fusion operators: max fusion, average fusion, and convolutional fusion [31]. We implement max and average fusion by aligning the input feature maps $f^f, f^b \in R^{1\times C\times h\times w}$ along the channel dimension, and computing a max or average across each pixel in corresponding channels, a parameter-free operation. We implement convolutional fusion by *stacking* the input feature maps along the channel dimension $[f^f, f^b]_C = f^s \in R^{1\times 2C\times h\times w}$, and applying a bank of learned, 1×1 convolutional filters that perform a weighted sum over the channel dimension and reduce the channel dimensionality by a factor of two.

Before applying the fusion operator, we weight the two input feature maps f^f, f^b by scalars α and $1-\alpha$, respectively, that correspond to *feature relevance*, a scheme that works very effectively in practice. For keyframe interval n, and a frame at offsets p and $n-p$ from the previous and next keyframes, respectively, we set $\alpha = \frac{n-p}{n}$ and $1-\alpha = \frac{p}{n}$, thereby penalizing the input features warped *farther* from their keyframe. Thus, when p is small relative to n, we weight the previous keyframe's features more heavily, and vice versa. In summary, the features for intermediate frame I_i are set to: $f_i = F(\frac{n-p}{n}f^f, \frac{p}{n}f^b)$, where $p = i \bmod n$. This scheme is reflected in Algorithm 2.

Algorithm 2 Feature interpolation with block motion vectors (**inter-BMV**)

```
1: input: video frames $\{I_i\}$, motion vectors mv, keyframe interval $n$
2: $W_f, W_b \leftarrow []$                                         ▷ forward, backward warped features
3: for frame $I_i$ in $\{I_i\}$ do
4:     if i mod n == 0 then                                                              ▷ keyframe
5:         $f_i \leftarrow N_{feat}(I_i)$                                        ▷ curr keyframe features
6:         $S_i \leftarrow N_{task}(f_i)$
7:         $f_{i+n} \leftarrow N_{feat}(I_{i+n})$                                 ▷ next keyframe features
8:         $W_f \leftarrow$ PROPAGATE$(f_i, n-1, -\text{mv}[i+1 : i+n])$
9:         $W_b \leftarrow$ PROPAGATE$(f_{i+n}, n-1, \text{mv}[i+n : i+1])$
10:    else                                                                    ▷ intermediate frame
11:        $p \leftarrow$ i mod n                                        ▷ offset from prev keyframe
12:        $f_i \leftarrow F(\frac{n-p}{n} \cdot W_f[\text{p}], \frac{p}{n} \cdot W_b[n - \text{p}])$          ▷ fuse propagated features
13:        $S_i \leftarrow N_{task}(f_i)$
14:    end if
15: end for
16: output: frame segmentations $\{S_i\}$

17: function PROPAGATE(features $f$, steps $n$, warp array $g$)    ▷ warp $f$ for $n$ steps with $g$
18:     $O \leftarrow [f]$
19:     for i = 1 to $n$ do
20:         append$(O,$ WARP$(O[i-1], g[i]))$                                 ▷ warp features one step
21:     end for
22:     return $O$
23: end function
```

2.4 Experiments

Datasets

We train and evaluate our system on CamVid [12] and Cityscapes [22], two popular, large-scale datasets for complex urban scene understanding. CamVid consists of over 10 minutes of footage captured at 30 fps and 960×720 pixels. Cityscapes consists of 30-frame video snippets shot at 17 fps and 2048×1024 pixels. On CamVid, we adopt the standard train-test split of [85]. On Cityscapes, we train on the `train` split and evaluate on the `val` split, following the example of previous work [97, 18, 105]. We use the standard mean intersection-over-union (mIoU) metric to evaluate segmentation accuracy, and measure throughput in frames per second (fps) to evaluate inference performance.

Architecture

For our segmentation network, we adopt a variant of the DeepLab architecture called Deformable DeepLab [25], which employs *deformable convolutions* in the last ResNet block (conv5) to achieve slightly higher accuracy than a standard DeepLab model. DeepLab [18] is widely considered a state-of-the-art architecture for semantic segmentation, and a DeepLab implementation currently ranks first on the PASCAL VOC object segmentation challenge [8]. Our DeepLab model uses ResNet-101 as its feature network, which produces representations $f_i \in R^{1\times 2048\times \frac{h}{16}\times \frac{w}{16}}$. The DeepLab task network outputs predictions $p_i \in R^{1\times C\times h\times w}$, where C is 12 or 20 for CamVid and Cityscapes respectively.

Training

To train our single-frame DeepLab model, we initialize with an ImageNet-trained ResNet-101 model, and learn task-specific weights on the CamVid and Cityscapes `train` sets. To train our video segmentation system, we sample at random a labeled image from the train set, and select a preceding and succeeding frame to serve as the previous and next keyframe, respectively. Since motion estimation with block motion vectors and feature warping are both parameter-free, feature propagation introduces no additional weights. Training feature interpolation with convolutional fusion, however, involves learning weights for the 1×1 convolutional fusion layer, which is applied to stacked feature maps, each with channel dimension 2048. For both schemes, we train with SGD on an AWS EC2 instance with 4 Tesla K80 GPUs for 50 epochs, starting with a learning rate of 10^{-3}.

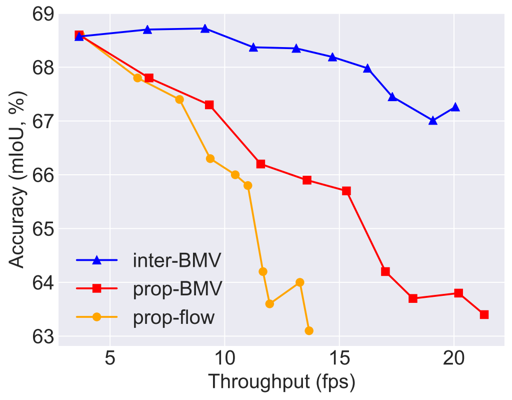
Results

Baseline

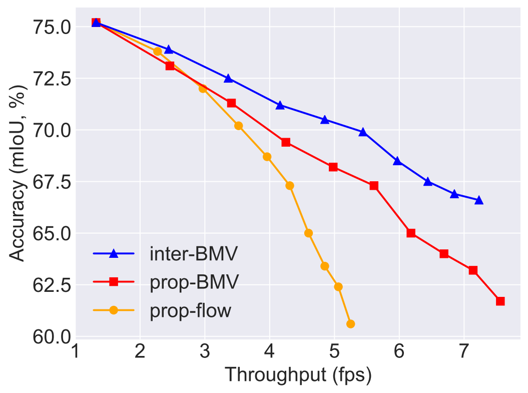
For our accuracy and performance baseline, we evaluate our full DeepLab model on every labeled frame in the CamVid and Cityscapes `test` splits. Our baseline achieves an accuracy of 68.6 mIoU on CamVid, at a throughput of 3.7 fps. On Cityscapes, the baseline model achieves 75.2 mIoU, matching published results for Deformable DeepLab [25], at 1.3 fps.

Propagation and Interpolation

In this section, we evaluate our two main contributions: 1) feature propagation with block motion vectors (**prop-BMV**), and 2) feature interpolation, our new feature estimation scheme, implemented with block motion vectors (**inter-BMV**). We compare to the closest available existing work on the problem, a feature propagation scheme based on optical flow [105] (**prop-flow**). We evaluate by comparing accuracy-runtime curves for the three approaches on CamVid and Cityscapes (see Fig. 2.3). These curves are generated by plotting accuracy against throughput at each keyframe interval in Tables 2.4 and 2.5 (see Appendix, Sec. 2.6), which contain comprehensive results.

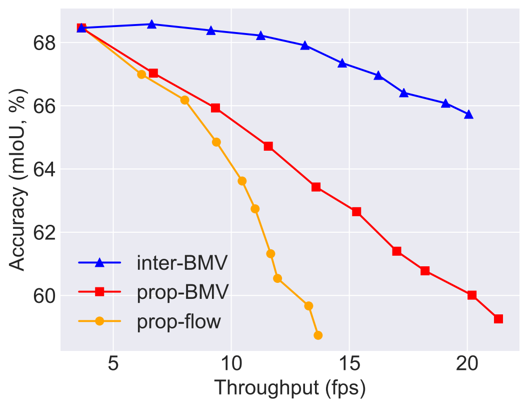


(a) **CamVid**. Data from Table 2.4.

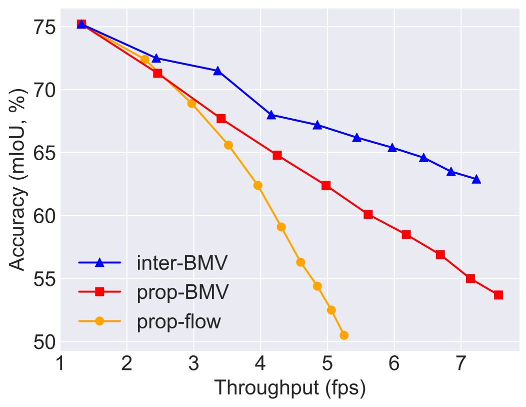


(b) **Cityscapes**. Data from Table 2.5

Figure 2.3: Accuracy (**avg.**) vs. throughput for all schemes on CamVid and Cityscapes.



(a) **CamVid**. Data from Table 2.4.



(b) **Cityscapes**. Data from Table 2.5.

Figure 2.4: Accuracy (**min.**) vs. throughput for all schemes on CamVid and Cityscapes.

First, we note that block motion-based feature propagation (**prop-BMV**) *outperforms* optical flow-based propagation (prop-flow) at all but the lowest throughputs. While motion vectors are slightly less accurate than optical flow in general, by cutting inference times by 53% on intermediate frames (Sec. 2.3), prop-BMV enables operation at much lower keyframe intervals than optical flow to achieve the same inference rates. This results in a much more favorable accuracy-throughput curve.

Second, we find that our feature interpolation scheme (**inter-BMV**) *strictly outperforms* both feature propagation schemes. At every keyframe interval, inter-BMV is more accurate than prop-flow and prop-BMV; moreover, it operates at similar throughput to prop-BMV. This translates to a consistent advantage over prop-BMV, and an even larger advantage over prop-flow (see Fig. 2.3). On CamVid, inter-BMV actually registers a small *accuracy gain* over the baseline at keyframe intervals 2 and 3, utilizing multi-frame context to improve on the accuracy of the single-frame DeepLab model.

Metrics. We also distinguish between two metrics: the standard *average* accuracy, results for which are plotted in Fig. 2.3, and *minimum* accuracy, which is a measure of the lowest frame-level accuracy an approach entails, i.e. accuracy on frames farthest away from keyframes. Minimum accuracy (see Fig. 2.4) is the appropriate metric to consider when we wish to segment a video as efficiently as possible, while ensuring that all frame segmentations meet some threshold level of accuracy. As an example, at an accuracy target of 65 mIoU, feature interpolation enables operation at 20.1 fps on CamVid (see Fig. 2.4a). This is 2.5× faster than achievable inference speeds with feature propagation alone, using either optical flow (8.0 fps) or block motion vectors (9.3 fps). In general, feature interpolation achieves over twice the throughput as [105] on CamVid and Cityscapes, at any target accuracy.

Table 2.1: Comparing **inter-BMV** to various related work on CamVid.

Scheme	Accuracy (mIoU, %)	Throughput (fps)	Model
GRFP [71]	66.1	–	D8+GRFP (best)
DFF [105]	67.4	8.0	KI=3
LinkNet [17]	68.3	–	LinkNet (best)
inter-BMV	68.7	9.1	KI=3
DDSC [10]	70.9	–	Single scale (best)

Table 2.2: Comparing **inter-BMV** to various related work on Cityscapes.

Scheme	Accuracy (mIoU, %)	Throughput (fps)	Model
Clockwork [81]	64.4	–	Alternating (best)
DRN [97]	70.9	–	DRN-C-42 (best)
DeepLab-v3 [18]	71.4	–	DL-101 (best)
DFF [105]	72.0	3.0	KI=3
inter-BMV	72.5	3.4	KI=3
RefineNet [62]	73.6	–	RN-101 (best)

Baseline. We also compare to our frame-by-frame DeepLab baseline, which offers low throughput but high average accuracy. As Figures 2.3a and 2.3b indicate, even at average accuracies above 68 mIoU on CamVid and 70 mIoU on Cityscapes, figures competitive with contemporary single-frame models (see Table 2.1 and Table 2.2), feature interpolation offers

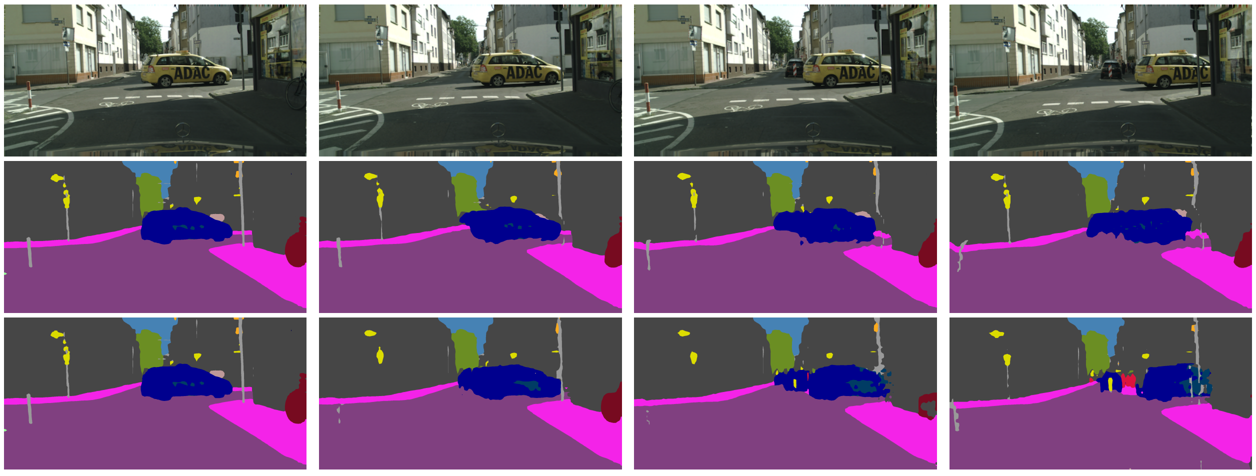

(a) k (b) k+2 (c) k+4 (d) k+6

Figure 2.5: Example segmentations at keyframe interval 7. Column $k + i$ corresponds to outputs i frames past the selected keyframe k. **First row:** input frames. **Second row:** prop-flow [105]. **Third row:** inter-BMV (us). Note that, by $k + 6$, prop-flow has significant warped the moving car, obscuring the people, vehicle, and street sign in the background (image center), while these entities remain clearly visible with interpolation, which exploits full scene context. Dataset: *Cityscapes*.

speedups of 4.5× and 4.2×, respectively, over the baseline. Notably, at key interval 3, interpolation obtains a 2.5× speedup over the baseline on CamVid, at slightly *higher than baseline accuracy* (see Fig. 2.3a).

Delay. Recall that feature interpolation introduces a delay of keyframe interval n frames, which corresponds to $\frac{n}{30}$ seconds at 30 fps. For example, at $n = 3$, inter-BMV introduces a delay of $\frac{3}{30}$ seconds, or 100 ms. To put this in context, prop-flow [105] takes 125 ms to segment a frame at key interval 3, and inter-BMV takes 110 ms. Thus, by lagging by less than 1 segmentation, we are able to segment 2.5× more frames per hour than the frame-by-frame model (9.1 fps vs. 3.6 fps). This is a suitable tradeoff in almost all batch settings (e.g. training data generation, post-hoc video analysis), and in many interactive applications (e.g. video anomaly detection, film editing).

Fig. 2.5 depicts a **qualitative comparison** of interpolation and prop-flow [105].

Feature Fusion

In this second set of experiments, we evaluate the accuracy gain achieved by feature fusion, in order to isolate the contribution of fusion to the success of our feature interpolation scheme. As Table 2.3 demonstrates, utilizing any fusion strategy, whether max, average, or conv fusion, results in higher accuracy than using either input feature map alone. This holds true even when one feature map is significantly stronger than the other (rows 2-4), and for both short and long distances to the keyframes. This observed additive effect suggests that feature fusion is highly effective at capturing signal that appears in only one input feature map, and in merging spatial information across time.

Table 2.3: An evaluation of feature fusion. We report final accuracies for various keyframe placements. `forward` and `backward` refer to the input feature maps. Dataset: *Cityscapes.*

Distance to keyframe(s)	`forward` mIoU	`backward` mIoU	`max` fusion mIoU	`avg` fusion mIoU	`conv` fusion mIoU
1	71.8	69.9	72.6	72.8	72.6
2	67.8	62.4	68.2	68.5	68.2
3	64.9	59.8	66.3	66.7	66.4
4	62.4	57.3	64.5	65.0	64.7

2.5 Conclusions

We develop **interpolation-BMV**, a novel segmentation scheme that combines the use of block motion vectors for feature warping, bi-directional propagation to capture scene context, and feature fusion to produce accurate frame segmentations at high throughput. We evaluate on the CamVid and Cityscapes datasets, and demonstrate significant speedups across a range of accuracy levels, compared to both a strong single-frame baseline and prior work. Our methods are general, and represent an important advance in the effort to operate image models efficiently on video.

2.6 Appendix

This section includes full tabular results for the CamVid and Cityscapes datasets (Table 2.4 and Table 2.5).

Table 2.4: Accuracy and throughput on **CamVid** for three schemes: (1) feature propagation with optical flow [105] (**prop-flow**), (2) feature propagation with block motion vectors (**prop-BMV**), and (3) feature interpolation with block motion vectors (**inter-BMV**).

		keyframe interval									
Metric	Scheme	1	2	3	4	5	6	7	8	9	10
mIoU, avg	prop-flow	68.6	67.8	67.4	66.3	66.0	65.8	64.2	63.6	64.0	63.1
(%)	prop-BMV	68.6	67.8	67.3	66.2	65.9	65.7	64.2	63.7	63.8	63.4
	inter-BMV	**68.6**	**68.7**	**68.7**	**68.4**	**68.4**	**68.2**	**68.0**	**67.5**	**67.0**	**67.3**
mIoU, min	prop-flow	68.5	67.0	66.2	64.9	63.6	62.7	61.3	60.5	59.7	58.7
(%)	prop-BMV	68.5	67.0	65.9	64.7	63.4	62.7	61.4	60.8	60.0	59.3
	inter-BMV	**68.5**	**68.6**	**68.4**	**68.2**	**67.9**	**67.4**	**67.0**	**66.4**	**66.1**	**65.7**
throughput	prop-flow	3.6	6.2	8.0	9.4	10.5	11.0	11.7	12.0	13.3	13.7
(fps)	prop-BMV	3.6	6.7	9.3	11.6	13.6	15.3	17.0	18.2	20.2	21.3
	inter-BMV	3.6	6.6	9.1	11.3	13.1	14.7	16.2	17.3	19.1	20.1

Table 2.5: Accuracy and throughput on **Cityscapes** for the three schemes: prop-flow [105], prop-BMV, and inter-BMV.

		keyframe interval									
Metric	Scheme	1	2	3	4	5	6	7	8	9	10
mIoU, avg	prop-flow	75.2	73.8	72.0	70.2	68.7	67.3	65.0	63.4	62.4	60.6
(%)	prop-BMV	75.2	73.1	71.3	69.4	68.2	67.3	65.0	64.0	63.2	61.7
	inter-BMV	**75.2**	**73.9**	**72.5**	**71.2**	**70.5**	**69.9**	**68.5**	**67.5**	**66.9**	**66.6**
mIoU, min	prop-flow	75.2	72.4	68.9	65.6	62.4	59.1	56.3	54.4	52.5	50.5
(%)	prop-BMV	75.2	71.3	67.7	64.8	62.4	60.1	58.5	56.9	55.0	53.7
	inter-BMV	**75.2**	**72.5**	**71.5**	**68.0**	**67.2**	**66.2**	**65.4**	**64.6**	**63.5**	**62.9**
throughput	prop-flow	1.3	2.3	3.0	3.5	4.0	4.3	4.6	4.9	5.1	5.3
(fps)	prop-BMV	1.3	2.5	3.4	4.3	5.0	5.6	6.2	6.7	7.1	7.6
	inter-BMV	1.3	2.4	3.4	4.2	4.9	5.4	6.0	6.4	6.9	7.2

Chapter 3

Accel: Corrective Fusion

3.1 Introduction

Semantic segmentation is an intensive computer vision task that involves generating class predictions for each pixel in an image, where classes range from foreground objects such as "person" and "vehicle" to background entities such as "building" and "sky". When applied to frames in high resolution video, this task becomes yet more expensive, as the high spatial dimensionality of the output is further scaled by the video's temporal frame rate (e.g. 30 frames per second). By treating video as a collection of uncorrelated still images, contemporary approaches to semantic video segmentation incur this full computational cost, achieving inference throughput of less than 1.5 frames per second (fps) on a 30 fps video feed [18, 25, 97]. Moreover, by ignoring temporal context, frame-by-frame approaches fail to realize the potential for improved accuracy offered by the availability of nearby frames.

Prior work has proposed feature reuse and feature warping as means to reduce computation on video. In particular, exploiting the observation that higher-level representations evolve more slowly than raw pixels in a video [81], these approaches relegate feature extraction, the most expensive component of most video recognition architectures [105], to select keyframes, and project these features forward via naïve copying or warping based on optical flow. While feature warping does enable some speedup [105], its efficacy is constrained by video dynamics. Fast scene evolution necessitates frequent feature re-computation, and feature warping in videos with a moving observer (e.g. driving footage), where the entire scene moves relative to the camera, introduces significant warping error. Warping error, moreover, compounds with repeated application of the warping operator.

Our proposed system, Accel (Fig. 3.1), addresses the challenges of efficient video segmentation by combining the predictions of a *reference branch*, which maintains an incrementally warped representation of the last keyframe, with the predictions of an *update branch*, which processes the current frame, in a convolutional fusion step. Importantly, this *update branch* has the ability to serve two purposes: 1) correction and 2) anchoring. When a cheap, shallow update network is used (e.g. ResNet-18), the warped keyframe features form the more accu-

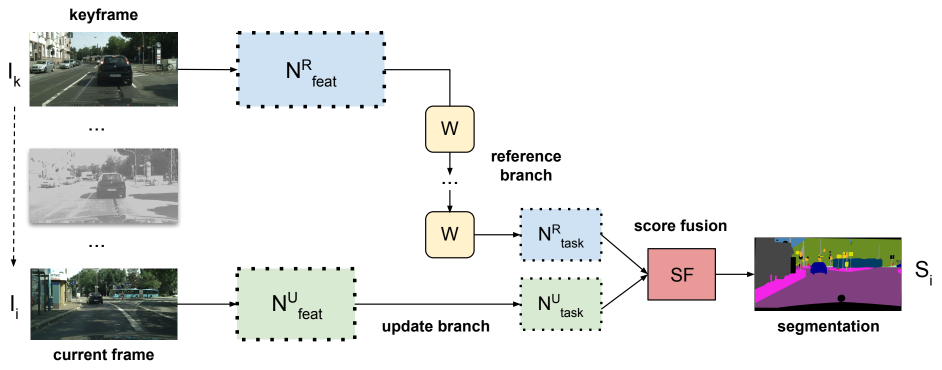


Figure 3.1: Accel is a fast, high-accuracy, end-to-end trainable video recognition system that combines two network branches: 1) a *reference branch* that computes a score map on high-detail features warped from the last visited keyframe, and 2) a cheap *update branch* that corrects this prediction based on features of adjustable quality (e.g. ResNet-18 to ResNet-101) computed on the current frame.

rate input to the fusion operator, and the update branch *corrects* warping-related error with information from the current frame. When an expensive, deep update network is used (e.g. ResNet-101), the update branch *anchors* the network on the features of the current frame, which is the higher accuracy input, while the reference branch augments the prediction with context from preceding frames. These two modes of operation represent two extremes on the highly competitive accuracy-throughput trade-off curve Accel unlocks.

We evaluate Accel on Cityscapes and CamVid, the largest available video segmentation datasets [12, 38, 22], and demonstrate a full range of accuracy-inference speed modalities. Our reference network, which we operate on keyframes, is an implementation of the DeepLab segmentation architecture [18] based on ResNet-101. Our chosen update networks range from the fast ResNet-18 (in Accel-18) to the accurate ResNet-101 (in Accel-101). On the high throughput side, the cheapest version of Accel, Accel-18, is both faster and more accurate than the closest comparable DeepLab model. On the high accuracy side, Accel-101 is more accurate than the best available single-frame model, DeepLab-101. As a set, the ensemble of Accel models achieve significantly higher accuracy than previous work on the problem at every keyframe interval. Taken togther, these results form a new state-of-the-art on the task of efficient semantic video segmentation.

3.2 Related Work

Image Semantic Segmentation

Semantic video segmentation is a recent offshoot of the study of semantic *image* segmentation, a problem of long-standing interest in computer vision. The classical approach to image segmentation was to propagate information about pixel assignments through a graphical model [33, 82, 41], a costly technique that scaled poorly to complex image datasets [57]. Most recent research follows the lead of Long et al. in the use of fully convolutional networks (FCNs) to segment images [64]. Recent work has augmented the FCN model with explicit encoder-decoder architectures [9, 62], dilated convolutions [96, 97], and post-processing CRFs [19, 18], achieving higher accuracy on larger, more realistic datasets [12, 28, 22].

Video Semantic Segmentation

Unlike video object segmentation, where a vast literature exists on using motion and temporal cues to track and segment objects across frames [74, 36, 70, 87], the video *semantic* segmentation task, which calls for a pixel-level labeling of the entire frame, is less studied. The rise of applications in autonomous control and interactive video analysis, however, have sparked significant interest in the problem of *efficient video semantic segmentation*. Recent papers have proposed selective re-execution of feature extraction layers [81], optical flow-based feature warping [105], and LSTM-based, fixed-budget keyframe selection policies [66] as means to achieve speedup over frame-by-frame approaches. Of the three, the optical flow-based approach [105] is the strongest contender, achieving greater cost savings and higher accuracy than both the first approach, which naïvely copies features, and the third, which is *offline* and has yet to demonstrate strong quantitative results. Despite its relative strength, however, flow-based warping [105] introduces compounding error in the intermediate representation, and fails to incorporate other forms of temporal change (e.g. new objects, occlusions). As a result, significant accuracy degradation is observed at moderate to high keyframe intervals, restricting its achievable speedup.

To address these problems, new work has proposed adaptive feature propagation, partial feature updating, and adaptive keyframe selection as schemes to optimally schedule and propagate computation on video [106, 60, 93]. These techniques have the drawback of complexity, requiring the network to learn auxiliary representations to decide: (1) whether to recompute features for a region or frame, and (2) how to propagate features in a spatially-variant manner. Moreover, they do not fundamentally address the problem of mounting warping error, instead optimizing the *operation* of [105]. In contrast, in Accel, we resolve the challenges by proposing a simple network augmentation: a second branch that cheaply processes each video frame, and corrects accumulated temporal error in the reference representation.

Network Fusion

Feature and network fusion have been extensively explored in other contexts. A body of work, beginning with [83] and extending to [31, 30, 32], studies spatial and temporal two-stream fusion for video action recognition. In the two-stream model, softmax scores of two network branches, one which operates on single RGB frames (spatial stream) and another on multi-frame optical flow fields (temporal stream), are fused to discern actions from still video frames. Variants of this approach have been subsequently applied to video classification [56, 91] and video object segmentation [50, 86], among other tasks. Unlike spatio-temporal fusion, which attempts to jointly deduce scene structure from RGB frames and motion for video-level tasks, the Accel fusion network uses keyframe context and optical flow as a means to conserve computation and boost accuracy in intensive *frame* and *pixel-level prediction tasks*, such as segmentation. In Accel, both branches process representations of single frames, and motion (optical flow) is used implicitly in the model to update a latent reference representation. Together, these design choices make Accel robust and configurable. The fact that network components are independent, with clear interfaces, allows the entire system to be operated at multiple performance modalities, via choice of update network (e.g. ResNet-x), motion input (e.g. optical flow, H.264 block motion [49]), and keyframe interval.

3.3 Approach

Problem Statement

Given a video I composed of frames $\{I_1, I_2, ...I_T\}$, we wish to compute the segmentation of each frame: $\{S_1, S_2, ...S_T\}$. We have at our disposal a single-frame segmentation network N that can segment any still frame in the video: $N(I_i) = S_i$. This network is accurate, but slow. Since N only takes single images as input, it cannot exploit the temporal continuity of video; the best we can do is to run N on every frame $I_i \in I$ sequentially.

Instead, we would like to develop a *video* segmentation network N' that takes as input a frame I_i, and potentially additional context (e.g. nearby frames, features, or segmentations), and renders S'_i. Our goals are two-fold: (1) $\{S'_i\}$ should be *at least as accurate* as $\{S_i\}$, and (2) running $N'(\{I_i\})$ should be *faster* than running $N(\{I_i\})$.

Operation Model

Our base single-frame semantic segmentation architecture N consists of three functional components: (1) a feature subnetwork N_{feat} that takes as input an RGB image $I_i \in R^{1\times3\times h\times w}$ and returns an intermediate representation $f_i \in R^{1\times2048\times\frac{h}{16}\times\frac{w}{16}}$, (2) a task subnetwork N_{task} that takes as input the intermediate representation f_i and returns a semantic segmentation score map $s_i \in R^{1\times C\times h\times w}$, where C is the number of labeled classes in the dataset, and (3) an output block P that converts s_i to normalized probabilities $p_i \in [0,1]^{1\times C\times h\times w}$ and then segmentations $S_i \in R^{1\times1\times h\times w}$.

This division follows a common pattern in image and video recognition architectures [105]. The feature network, N_{feat}, is largely identical across different recognition tasks (object detection, instance segmentation, semantic segmentation), and is obtained by discarding the final k-way classification layer in a standard image classification network (e.g. ResNet-101), and decreasing the stride length in the first block of the conv5 layer from 2 to 1 to obtain higher-resolution feature maps (spatial dimension $\frac{h}{16}\times\frac{w}{16}$ instead of $\frac{h}{32}\times\frac{w}{32}$). The task network, N_{task}, for semantic segmentation includes three blocks: (1) a feature projection block, which consists of a 1×1 convolutional layer, plus a non-linear activation (ReLU), and reduces the feature channel dimension from 2048 to 1024, (2) a scoring layer, which consists of a single 1×1 convolutional layer, and further reduces the channel dimension from 1024 to the C semantic classes, and (3) an upsampling block, which consists of a transposed convolutional layer and a cropping layer, and upsamples the predicted scores from $\frac{h}{16} \times \frac{w}{16}$ to the spatial dimensionality of the input image, $h \times w$. Finally, output block P consists of a softmax layer, followed by an argmax layer.

Exploiting the observation that features can be reused across frames to reduce computation [81, 105], we now adopt the following operation model on video. N_{feat}, which is deep and expensive, is executed only on select, designated *keyframes*. Keyframes are selected at regular intervals, starting with the first frame in the video. The extracted keyframe features f_i are warped to subsequent frames using a computed optical flow field, O. N_{task}, which is shallow and cheap, is executed on every frame. Since computing optical flow $O(I_i, I_j)$ on pairs of frames, and warping features with the flow field $W(f_i, O)) \rightarrow \hat{f}_j$, is much cheaper than computing $N_{feat}(I_j)$ [105], this scheme saves significant computation on *intermediate frames*, which form the vast majority of video frames.

Accel

In Accel, we introduce a lightweight feature network, N^U_{feat}, on intermediate frames to update score predictions based on the warped keyframe features, with information from the current frame. On keyframes, we execute our original feature network, now denoted as the *reference* feature network, N^R_{feat}. In our system, we use ResNet-101 as N^R_{feat}, and a range of models, from ResNet-18 to ResNet-101, as N^U_{feat}, depending on specific accuracy-performance goals. In this section, we discuss a forward pass through this new architecture, Accel (see Fig. 3.2).

On keyframes, denoted by index k, we execute the full reference network $P(N^R_{task}(N^R_{feat}(I_k)))$ to yield a segmentation S_k. We cache the intermediate output, $N^R_{feat}(I_k)$, as features f^c.

On intermediate frames i, we compute scores s^R_i and s^U_i along both a *reference branch* and an *update branch*, respectively. On the reference branch, we warp f^c from the previous frame I_{i-1} to the current frame I_i, and then execute N^R_{task}. As our warping operation W, we spatially transform our cached features f^c with a bilinear interpolation of the optical flow field $O(I_{i-1}, I_i)$, as in [105]. On the update branch, we run the full update network N^U. Symbolically, the two branches can be represented as:

$$s^R_i = N^R_{task}(W(f^c, O(I_{i-1}, I_i))) \tag{3.1}$$

$$s^U_i = N^U_{task}(N^U_{feat}(I_i)) \tag{3.2}$$

The score maps s^R_i and s^U_i represent two views on the correct class labels for the pixels in the current frame. These predictions are now merged in a 1×1 convolutional fusion step, which we refer to as *score fusion* (SF). s^R_i and s^U_i are stacked along the channel dimension, yielding an input $s^{stacked}_i \in R^{1 \times 2C \times h \times w}$. Applying a 1×1 convolutional layer with dimensions $C \times 2C \times 1 \times 1$ to $s^{stacked}_i$ yields an output $s_i \in R^{1 \times C \times h \times w}$. Notationally,

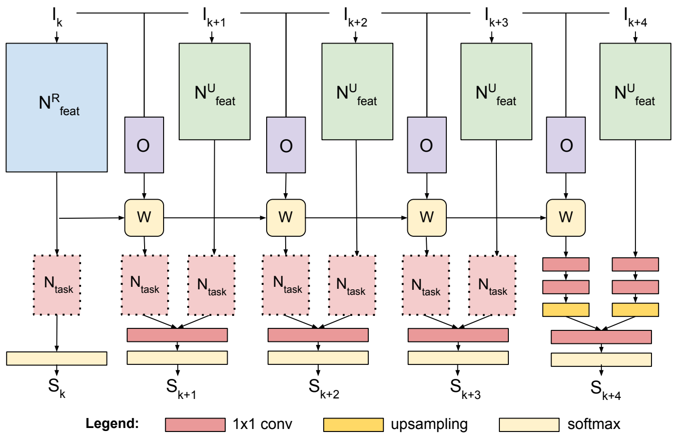


Figure 3.2: Accel consists of several components: (1) a reference feature net N^R_{feat} executed on keyframes, (2) an update feature net N^U_{feat} executed on intermediate frames, (3) an optical flow network O used for feature warping W, (4) two instantiations of N_{task} (reference and update), (5) a 1×1 convolutional network fusion layer, and (6) a final softmax layer.

$s_i = SF(s_i^{stacked}) = SF([s_i^R, s_i^U])$. Finally, applying the output block P to s_i yields the segmentation S_i of frame I_i.

Note that while the layer definitions of N^R_{feat} and N^U_{feat} differ in general, N^R_{task} and N^U_{task} are architecturally equivalent, albeit independent, instantiations. This makes Accel highly modular. Since the task network N_{task} has a fixed interface, Accel can accept any feature network N^U_{feat} that outputs representations f_i with the appropriate dimensionality.

Training

Accel can be trained end-to-end on sparsely annotated sequences of video frames. The entire network consists of the score fusion layer, along with three independently trainable components, N^R, N^U, and O, which we now discuss.

For our reference network N^R and update network N^U, we use a high-accuracy variant [25] of the DeepLab architecture [18]. DeepLab is a canonical architecture for semantic segmentation [25, 9, 62, 96], and a DeepLab implementation has consistently ranked first on the Pascal VOC segmentation benchmark [8]. N^R_{feat} and N^U_{feat} are first trained on ImageNet; N^R and N^U are then individually fine-tuned on a semantic segmentation dataset, such as Cityscapes [22]. In Accel, we fix N^R_{feat} as ResNet-101. We then build an ensemble of models, based on a range of update feature networks N^U_{feat}: ResNet-18, -34, -50, and -101. This forms a full spectrum of accuracy-throughput modalities, from a lightweight, competitive Accel based on ResNet-18, to a slow, extremely accurate Accel based on ResNet-101. For the third and last independently trainable component, the optical flow network O, we use the “Simple” architecture from the FlowNet project [34]. This network is pre-trained on the synthetic Flying Chairs dataset, and then jointly fine-tuned on the semantic segmentation task with N^R.

To train Accel, we initialize with weights from these three pre-trained models. In each mini-batch, we select a frame I_j. When training at keyframe interval n, we select frame $I_{j-(n-1)}$ from the associated video snippet, and mark it as the corresponding keyframe I_k for frame I_j. In a forward pass, we execute Accel's reference branch on frame I_k, and execute the update branch and fusion step on each subsequent intermediate frame until I_j. A pixel-level, cross-entropy loss [64] is computed on the predicted segmentation S_j and the ground-truth label for frame I_j. In the backward pass, gradients are backpropagated through time through the score fusion operator, the reference and update branches, and the warping operator, which is parameter-free but fully differentiable. Note that the purpose of joint training is to learn weights for the score fusion (SF) operator, and to optimize other weights (i.e. N^R_{task} and N^U_{task}) for the end-to-end task.

Design Choices

Recent work has explored *adaptive keyframe scheduling*, where keyframes are selected based on varying video dynamics and feature quality [106, 60, 93]. Here both rapid scene change and declining feature quality can trigger feature recomputation. We note that keyframe scheduling is an optimization that is orthogonal to network design, and therefore entirely compatible with the Accel architecture.

3.4 Experiments

Setup

We evaluate Accel on Cityscapes [22] and CamVid [12], the largest available datasets for complex urban scene understanding and standard benchmarks for semantic segmentation [18, 25, 97]. Cityscapes consists of 30-frame snippets of street scenes from 50 European cities, recorded at a frame rate of 17 frames per second (fps). Individual frames are 2048×1024 pixels in size. The train, validation, and test sets consist of 2975, 500, and 1525 snippets each, with ground truth labels provided for the 20th frame in each train and validation set snippet. The Cambridge-driving Labeled Video Database (CamVid) consists of over 10 minutes of footage captured at 30 fps. Frames are 960 by 720 pixels in size, and ground-truth labels are provided for every 30th frame. We use the standard train-test split of [85], which divides CamVid into three train and two test sequences, containing 367 and 233 frames, respectively.

To evaluate accuracy, we use the *mean intersection-over union* (mIoU) metric, standard for semantic segmentation [28]. mIoU is defined as the average achieved intersection-over-union value, or Jaccard index, over all valid semantic classes in the dataset. To evaluate performance, we report *average inference time* in seconds per frame (s/frame) over the entire dataset. Note that this is the inverse of throughput (frames per second).

We train Accel as described in Section 3.3 on Cityscapes and CamVid. We perform 50 epochs of joint training at a learning rate of $5 \cdot 10^{-4}$ in two phases. In phase one, all weights except SF are frozen. In phase two, after 40 epochs, all remaining weights are unfrozen. We train a reference implementation of [105] by jointly fine-tuning the same implementations of N^R and O. At inference time, we select an operational keyframe interval i, and in each snippet, choose keyframes such that the distance to the labeled frame rotates uniformly through $[0, i-1]$. This sampling procedure simulates evaluation on a densely labeled video dataset, where $\frac{1}{i}$ frames fall at each keyframe offset between 0 and $i-1$. Here we follow the example of previous work [105].

Finally, Accel is implemented in the MXNet framework [20]. All experiments are run on Tesla K80 GPUs, at keyframe interval 5, unless otherwise stated.

Results

Baselines

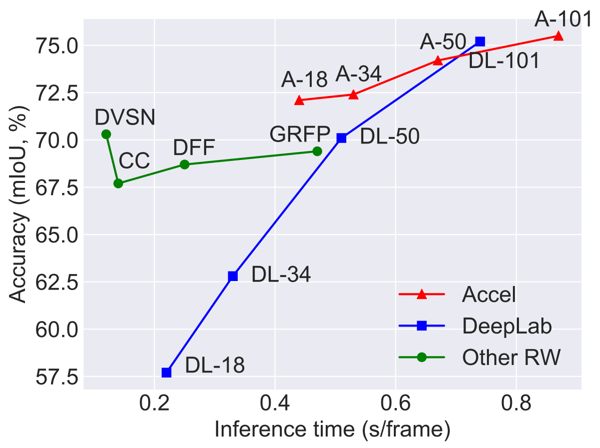
To generate our baseline accuracy-throughput curve, we run single-frame DeepLab [18] models based on ResNet-18, -34, -50, and -101 on the Cityscapes and CamVid test data. For both DeepLab and Accel, we use a variant of the ResNet architecture called Deformable ResNet, which employs *deformable convolutions* in the last ResNet block (conv5) to achieve significantly higher accuracy at slightly higher inference cost [25]. We refer to DeepLab models based on ResNet-x as DeepLab-x, and Accel models based on DeepLab-x as Accel-x.

Accuracy-throughput

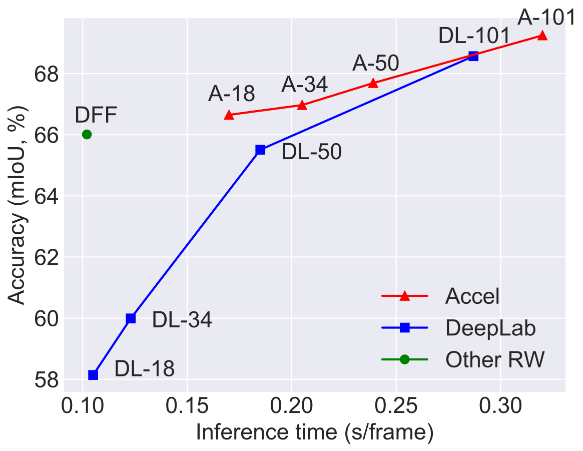
Using Accel, we achieve a new, state-of-the-art accuracy-throughput trade-off curve for semantic video segmentation (see Figs. 3.3a, 3.3b).

All Accel models, from Accel-18 to Accel-101, allow operation at high accuracy: above 72 mIoU on Cityscapes and above 66 mIoU on CamVid. At the high accuracy end, Accel-101 is by far the most accurate model, achieving higher mIoU than the best available DeepLab model, DeepLab-101. At the high throughput end, Accel-18 is *both faster and more accurate* than the closest comparable single-frame model, DeepLab-50. Notably, Accel-18 is over 40% cheaper than DeepLab-101, at only 2-3% lower mIoU. As a rule, each Accel-x model is more accurate than its single-frame counterpart, DeepLab-x, for all x.

Together, the four Accel models form an operational Pareto curve that clearly supersedes the Pareto curve defined by the four single-frame DeepLab models (Figs. 3.3a, 3.3b). Accel also visibly outperforms related work, including Clockwork Convnets [81], Deep Feature Flow [105], Gated Recurrent Flow Propagation [71], and Dynamic Video Segmentation Network [93] (see Fig. 3.3a). Though Deep Feature Flow (DFF) offers a strong accuracy-throughput trade-off in the low accuracy range, due to its fixed architecture, it is not a contender in the high accuracy regime. We provide a more detailed comparison with DFF in the next section.



(a) **Cityscapes**. Data from Table 3.3.



(b) **CamVid**. Data from Table 3.4

Figure 3.3: Accuracy vs. inference time on **Cityscapes** and **CamVid**. Comparing four variants of Accel (A-x) to single-frame DeepLab models (DL-x) and various other related work (RW). All results at keyframe interval 5.

We also briefly survey a range of recent, new single-frame segmentation networks. These include architectures based on spatial pyramid pooling, such as PSPNet, NetWarp, and DenseASPP [102, 37, 94], which achieve high accuracy (up to 80.6% mIoU on Cityscapes test) but at steep computational cost. One evaluation [37] finds that PSPNet operates at 3.00 seconds per Cityscapes frame, even barring any augmentations (e.g. NetWarp) or advanced settings (e.g. multi-scale ensembling), which is substantially slower than any DeepLab or Accel variant. Other relevant single-frame network families include the encoder-decoder architectures (e.g. U-Net [80]), which optimize for accuracy on high-resolution biomedical images, and the parameter-efficient DenseNets (e.g. FC-DenseNet [52]), for which segmentation inference times have not yet been reported.

Keyframe intervals

In this section, we extend our evaluation to a range of keyframe intervals from 1 to 10. Keyframe interval 1 corresponds to running the reference network N^R on every frame. As a result, Deep Feature Flow (DFF) [105] and the Accel variants report the same accuracy at this setting (see Fig. 3.4). At keyframe intervals above 1, we find that even the cheapest version of Accel, Accel-18, consistently offers higher accuracy than DFF. In particular, over keyframe interval 8, a wide accuracy gap emerges, as DFF's accuracy approaches 60 mIoU while all Accel models maintain roughly between 70 and 75 mIoU (Fig. 3.4).

This gap is an illustration of the compounding warping error that builds in DFF, but is corrected in Accel with the advent of the update branch. The trade-off is that Accel models are slower on intermediate frames: in addition to the inference cost of O and N^R_{task}, which

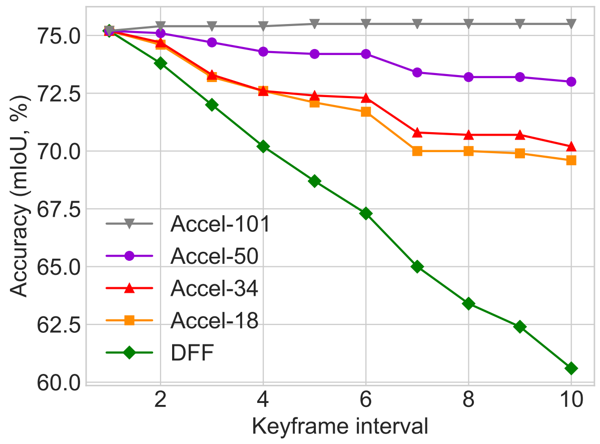


Figure 3.4: Accuracy vs. keyframe interval on **Cityscapes** for optical flow-based warping alone (DFF) and four variants of Accel. All five schemes use ResNet-101 in N^R.

is also paid by DFF, Accel models also incur the cost of N^U, which is low when N^U_{feat} is ResNet-18 and higher when N^U_{feat} is ResNet-101.

Ablation study

We now present a simple ablation study that isolates the contributions of the reference network N^R and the update network N^U to the accuracy of Accel (see Table 3.1). Disabling N^U corresponds to using only the optical flow-warped representations from the previous keyframe. Since all versions of Accel share the same N^R, this results in the same accuracy for all models (row 1). Disabling the reference network N^R corresponds to running only the single-frame update networks, DeepLab-18, -34, -50, or -101, on all frames (row 2). Disabling neither yields our original models (row 3). Notice the effect of the network fusion: each unmodified Accel model is more accurate than either of its component subnetworks. Moreover, Accel-18 observes a 6.8 point accuracy boost over N^R via the use of an update network N^U that is cheaper and *much less accurate* than N^R. This confirms the powerful synergistic effect of combining two contrasting sets of representations: one that is high-detail but dated, and one that is lower resolution but temporally current.

Fusion location

In this section, we evaluate the impact of fusion location on final network accuracy and performance. Accel, as described so far, uses a 1×1 convolutional layer to fuse pre-softmax class scores, but it was also possible to perform this fusion at an earlier stage. In Table 3.2, we compare accuracy values and inference times for two fusion variants: (1) feature fusion (fusion between N_{feat} and N_{task}) and (2) score fusion (fusion between the score upsampling block and the softmax layer).

Table 3.1: **Ablation study**. A breakdown of the accuracy contributions of N^R (reference branch) and N^U (update branch) to Accel. Results for keyframe interval $i = 5$, at the maximum offset (4) from the keyframe. Dataset: *Cityscapes*.

	Model			
Setting	A-18	A-34	A-50	A-101
N^R only	62.4	62.4	62.4	62.4
N^U only	57.7	62.8	70.1	75.2
Accel	**69.2**	**69.7**	**73.0**	**75.5**

Table 3.2: **Fusion location**. An evaluation of the impact of network fusion location on final accuracy values. Model: Accel-18. Results for keyframe interval $i = 5$, at the maximum offset (4) from the keyframe. Dataset: *Cityscapes*.

	Metric	
Location	Acc. (mIoU)	Time (s/frame)
Feature	**69.5**	0.46
Score	69.2	**0.44**

As Table 3.2 indicates, score (late) fusion results in slightly lower accuracy, but faster inference times. Recall that a 1×1 convolutional fusion layer is a mapping $R^{1\times 2C\times h\times w} \rightarrow R^{1\times C\times h\times w}$, where C is the channel dimensionality of the input. Feature (early) fusion results in higher accuracy ostensibly because it is executed on higher-dimensionality inputs, allowing for the discovery of richer channel correspondences (C is 2048 for ResNet feature maps, versus 19 for scores). Inference times, on the other hand, benefit from *lower* channel dimensionality: the fusion operator itself is cheaper to execute on scores as opposed to features. We use score fusion in all except the most accurate model (Accel-101), as in our view, the 5% difference in inference cost outweighs the more marginal gap in accuracy. Nevertheless, the choice between the two schemes is a close one.

Finally, we also experimented with the intermediate channel dimensionality, C. ResNets-50 and -101 traditionally have channel dimension 2048 after the fifth conv block, which is why $C = 2048$ was our default choice. In our experiments, we found that using smaller values of C, such as 512 or 1024, resulted in poorer segmentation accuracy, without noticeably reducing inference times.

Qualitative evaluation

In Figure 3.5, we compare the qualitative performance of DFF (Accel N^R), DeepLab (Accel N^U), and Accel ($N^R + N^U$) on two sequences of 10 frames (top and bottom).

3.5 Conclusions

Accel is a fast, high-accuracy video segmentation system that utilizes the combined predictive power of two network pathways: (1) a *reference branch* N^R that extracts high-quality features on a reference keyframe, and warps these features forward using incremental optical flow estimates, and (2) an *update branch* N^U that processes the current frame to correct accumulated temporal error in the reference representation. Comprehensive experiments demonstrate a full range of accuracy-inference speed modalities, from a high-throughput version of Accel that is both faster and more accurate than comparable single-frame models to a high-accuracy version that exceeds state-of-the-art. The full ensemble of Accel models consistently outperforms previous work on the problem at all keyframe intervals, while an ablation study demonstrates that Accel makes significant accuracy gains over its individual components. Finally, the Accel architecture is modular and end-to-end trainable, serving as a general example on how to perform dense prediction tasks efficiently on video.

3.6 Appendix

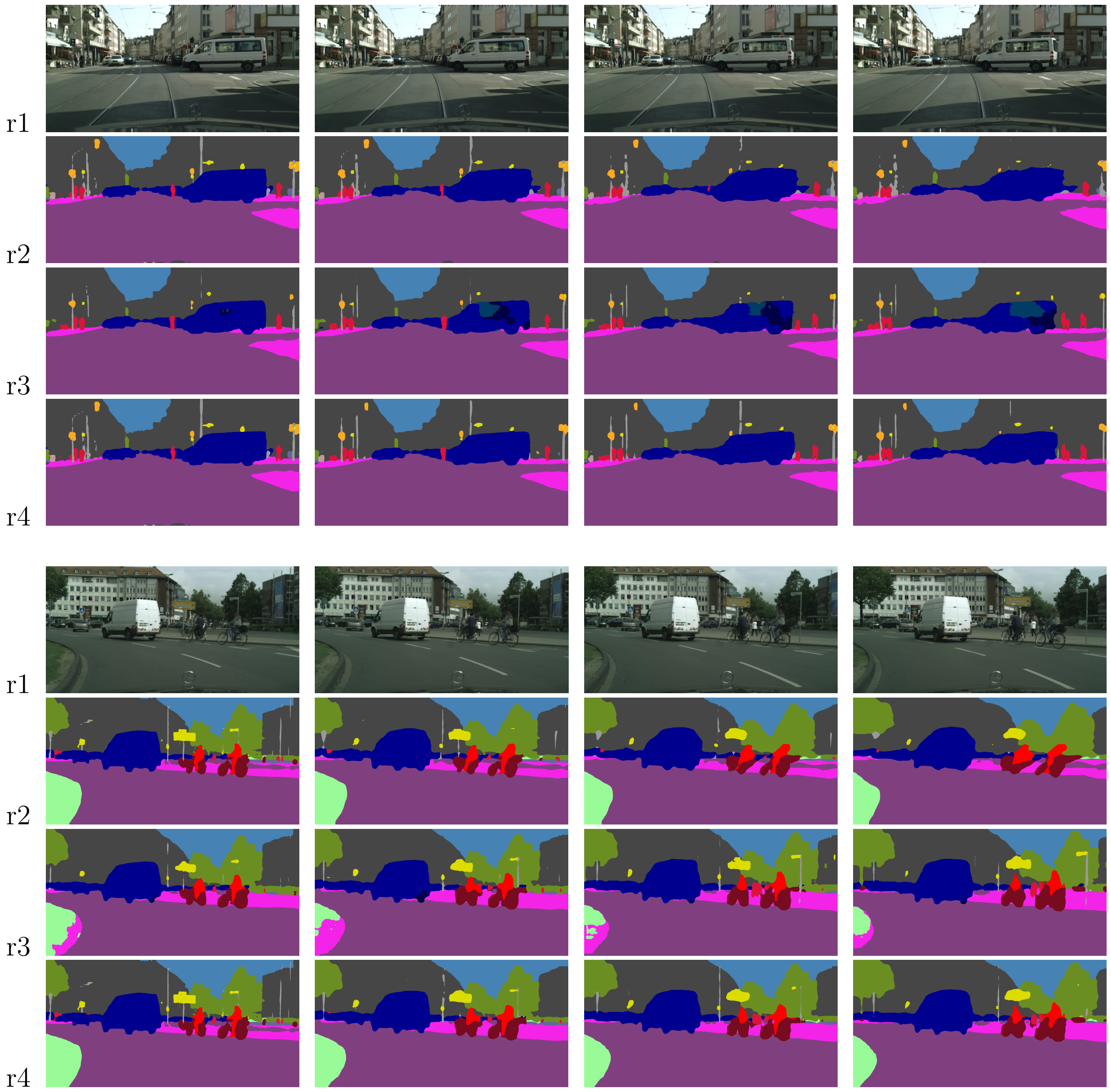
This section includes accuracy and inference times on **Cityscapes** and **CamVid** for Accel, DeepLab, and various related work (Table 3.3 and Table 3.4).

Table 3.3: Accuracy and inference times on **Cityscapes** for four single-frame DeepLab models (DL-x), four variants of Accel (A-x), and various related work. Table ordered by accuracy. All inference time standard deviations less than 0.01. Each Accel-x model is more accurate than its single-frame counterpart, DeepLab-x, for all x. Data plotted in Fig. 3.3a.

Model	Acc. (mIoU, %)	Time (s/frame)
DL-18	57.7	0.22
DL-34	62.8	0.33
CC (Shel. 2016)	67.7	0.14
DFF (Zhu 2017)	68.7	0.25
GRFP (Nils. 2018)	69.4	0.47
DL-50	70.1	0.51
DVSN (Xu 2018)	70.3	0.12
A-18	72.1	0.44
A-34	72.4	0.53
A-50	74.2	0.67
DL-101	75.2	0.74
A-101	75.5	0.87

Table 3.4: Accuracy and inference times on **CamVid**. Table ordered by accuracy. All inference time standard deviations less than 0.01. Each Accel-x model is more accurate than its single-frame counterpart, DeepLab-x, for all x. Data plotted in Fig. 3.3b.

Model	Acc. (mIoU, %)	Time (s/frame)
DL-18	58.1	0.105
DL-34	60.0	0.123
DL-50	65.5	0.185
DFF (Zhu 2017)	66.0	0.102
A-18	66.7	0.170
A-34	67.0	0.205
A-50	67.7	0.239
DL-101	68.6	0.287
A-101	69.3	0.320



(a) k (b) $k+3$ (c) $k+6$ (d) $k+9$

Figure 3.5: **Qualitative outputs**. Two frame sequences at keyframe interval 10. Column $k+i$ corresponds to the i^{th} frame past keyframe k. **First row:** input frames. **Second row:** Accel N^R branch / DFF [105]. **Third row:** Accel N^U branch / DeepLab-18. **Fourth row:** Accel-18. Note how Accel both corrects DFF's warping-related distortions in row 2, including the obscured pedestrians (top example) and the distorted vehicles (bottom example), *and* avoids DeepLab's misclassifications in row 3 on the van (top) and vegetation patch (bottom). Column (c) in the bottom example also qualifies as an error case for Accel, as unlike DeepLab, Accel misses the street sign on the right.

Part II

Large-Scale Video Analytics

Chapter 4

ReXCam: Resource-Efficient Cross-Camera Analytics

4.1 Introduction

Enterprises are increasingly deploying large camera networks for video analytics, for use cases ranging from public safety monitoring to patient oversight and business intelligence [88]. In Chicago, police access footage from 30,000 security cameras installed citywide to inform their responses to live crime reports [15]. In London, police tap into 12,000 cameras installed on its underground transit network to identify and investigate threats to public safety [11, 95]. In Paris, public hospitals plan to install 1,500 *new* cameras to protect staff and monitor patients [75].

Close analysis of the live video from these deployments, however, remains a costly undertaking. Human monitoring does not scale to 1000s of cameras, and state-of-the-art neural networks are too expensive to operate on each video feed in real-time. In particular, at 1 Nvidia K80 GPU per video feed, automated analytics on the Chicago Public Schools' 7,000-camera deployment [84] would require a \$28 million investment in GPU hardware, or cost \$6,300 per hour in GPU cloud time (about \$0.9 million per month) [76, 2], significant expenditures for a publicly-funded school system.

Recent work explores techniques to accelerate simple, per-frame tasks on single-camera video pipelines [55], and support low-latency, after-the-fact queries on indexed, historical video [45]. This work, however, does not address the key requirement of real video analytics applications: *cross-camera analytics* on *live* video. Multi-camera inference is the core capability that enables operators to monitor environments at scale, track and predict entity movement, and understand complex scenes, for applications such as flow control [39], suspect tracking [65, 100], and traffic management [65, 99].

Cross-camera analytics, however, is compute and data intensive. Unlike simple, stateless tasks, such as object detection, cross-camera analytics entails discovering *associations*, across frames and across cameras. Whereas cost grows linearly with time for per-frame queries, cost

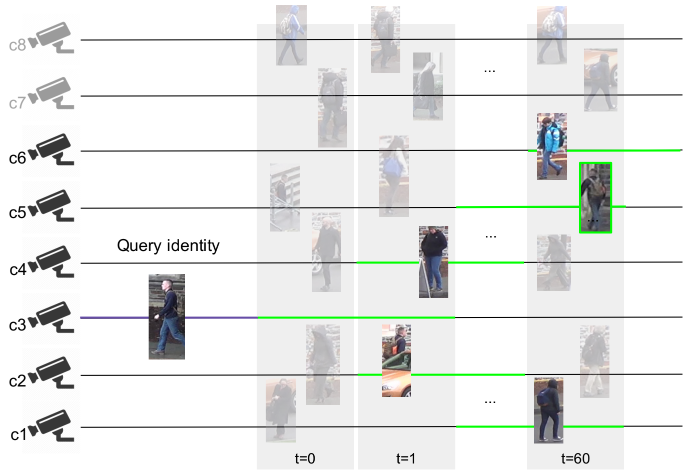


Figure 4.1: ReXCam exploits a *learned model of spatial and temporal correlations*, built on historical data, to reduce compute workload at inference time. In this figure, the camera network (compressed to 1-D) is represented on the y-axis, and time on the x-axis. In searching for a query identity, ReXCam eliminates some cameras entirely (spatial filtering), and searches the others only within a narrow time window (temporal filtering).

in cross-camera analytics also entails a second, spatial dimension – the number of cameras in the network. For example, in the suspect tracking example, flagging frames containing an identified perpetrator requires searching both *across* the camera network (which can span the entire city) and forward in time, at the throughput of the incoming video stream.

We can formalize the core problem here as follows: given a query instance of an object or entity of interest, we wish to return all subsequent instances of that identity in the live video, *while examining as few video frames as possible*. We adopt the current practice as our baseline, which is to search every nearby camera for the query identity. This baseline is both extremely data intensive, and prone to a high rate of false positive matches – by searching cameras indiscriminately, it encounters a large number of distractor instances, which derail inference *precision* [89].

To address these severe cost and accuracy challenges, we present ReXCam – a new system for efficient cross-camera video analytics. ReXCam exploits *spatial* and *temporal correlations* in large camera networks to reduce the size of the inference search space, and thus dramatically decrease compute cost. Spatial correlations indicate the degree of association between cameras – the probability that a source camera will send traffic *to* a particular destination camera. Temporal correlations indicate the degree of association between cameras *over time* – the probability that a source camera will send traffic to a particular destination camera *at a particular time*. These correlations, learned offline on historical data, enable ReXCam to guide its inference-time search toward cameras and frames most likely to contain the query identity. In doing so, ReXCam is able to both substantially cut down its inference time workload, and increase its rate of true positive detections (see **Figure 4.1**).

ReXCam operates in three phases. In an offline profiling phase, it constructs a *cross-camera correlation model* on unlabeled video data, which encodes the locality observed in historical traffic patterns. This is an expensive one-time operation that requires assigning *identifiers* to detected entities with an offline tracker, and then converting these identifiers into an aggregate profile of cross-camera correlations. At inference time, ReXCam uses this model to filter frames that are not spatially and temporally correlated to the query identity's current position, and thus unlikely to contain its next instance. On occasion, this pruning will cause ReXCam to miss query detections. In these cases, ReXCam performs a *fast-replay search* on recently filtered frames, which uncovers the skipped query instances, and enables it to gracefully recover into its live cross-camera search.

Together these techniques enable significant improvements over the all-camera baseline. Evaluating on the well-known DukeMTMC dataset [79], which contains footage from 8 cameras located on the Duke University campus, we find that ReXCam is able to reduce compute cost by a factor of 4.6× compared to the baseline, while improving precision (fraction of instances correct) by 27%, at the price of just 1.6% lower recall (fraction of instances found). Moreover, RexCam's fast-replay search scheme reduces delay by about 50% relative to a scheme that searches for missed detections at the video frame rate. ReXCam is able to achieve these gains at a one-time offline profiling cost equal to running 600 real-time queries, a small value relative to the annual workload in real video analytics operations [99, 100].

We also evaluate our achieved savings factor relative to an oracle that could predict with certainty at inference time the destination camera for a given query instance. Such an oracle could achieve at maximum a 7.0× workload reduction over the baseline on 8 cameras. In comparison, ReXCam, which is trained only on historical data, obtains a 4.6× workload reduction, a substantial fraction of the oracle's gains.

4.2 Problem & Motivation

Problem statement

In this section, we study a broad template of cross-camera applications that involve tracking a person or object of interest, in real time, through a camera network. We call this process *identity tracking*. In particular, given a single instance of a query identity q (e.g. a person) flagged in camera c at frame f, we wish to return all subsequent frames, across all cameras, in which q appears. This is a bounded process – since q will eventually exit the network, we must at some point cease our search for q.

While tracking q, we wish to perform well on four metrics:

1. **Recall (%)** – The ratio of (a) the number of instances of q successfully retrieved to (b) the total number of instances of q present in the footage after frame f.

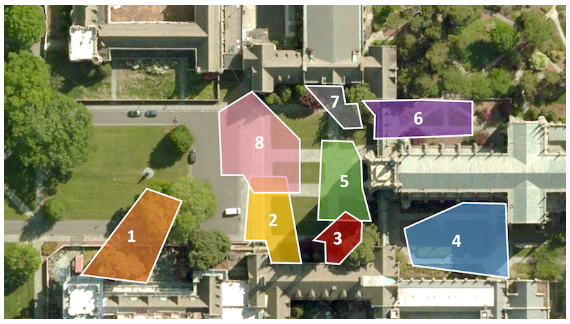


Figure 4.2: DukeMTMC camera network [79]. Labeled regions indicate the visual field of view of each camera.

2. **Precision (%)** – The ratio of (a) the number of instances of q successfully retrieved to (b) the total number of instances of any entity retrieved.

3. **Compute cost** (in 1000s of frames) – The total number of video frames processed. We especially wish to avoid processing frames that do *not* contain q.

   Note that processing a frame involves running a compute-intensive machine learning model (e.g. a convolutional neural network) on the image, to determine if it contains any instances of q. This is the most expensive component of the video analytics pipeline (which we describe in Section 4.3). Consequently, number of frames is the key cost factor we wish to minimize.

4. **Delay (sec.)** – The lag, in seconds, between receiving a video frame from a camera and returning an inference decision, i.e. deciding whether the frame contains (or does not contain) q. This lag, or delay, is a major component of total response latency.

A successful system will achieve high recall (return most instances of q), high precision (return few instances of other entities), low compute cost (process a small number of video frames), and low delay (track q in real-time).

Empirical motivation

We now present a short empirical study that establishes the presence of strong *cross-camera correlations* in real-world video surveillance data. This in turn motivates the design of a video analytics system, such as ReXCam, that leverages such correlations to reduce compute cost.

We conduct our study on the DukeMTMC dataset [79], one of the most popular benchmarks in computer vision for work on person re-identification and tracking [101, 92]. The dataset contains footage from eight cameras placed on the Duke University campus (see **Figure 4.2**), in an area with significant pedestrian traffic. The field of views of the cameras do not intersect, but the cameras are placed close enough that people frequently appear in

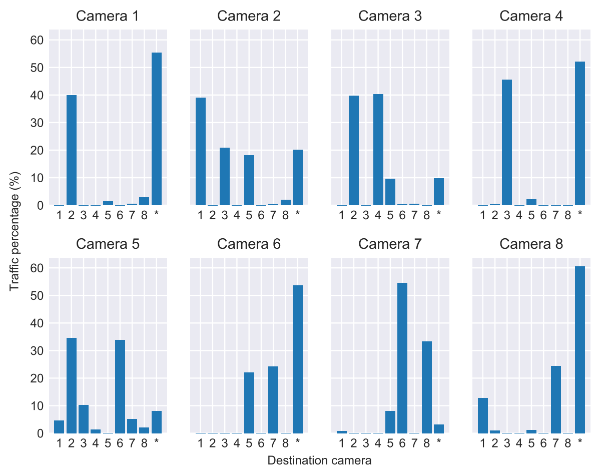


Figure 4.3: Spatial traffic patterns in the DukeMTMC dataset [79]. Plots display percentage of outbound traffic that appears at a particular destination. Each plot corresponds to a particular source camera; each bar to a destination camera. The final bar (*) represents outbound traffic that *exits* the network.

multiple cameras. The dataset contains over 2,700 unique identities across 85 minutes of footage, recorded at 60 frames per second [79].

Spatial locality

Our first finding is that cross-camera traffic demonstrates a high degree of spatial locality. Here, "traffic" between cameras A and B is defined as the set of unique individuals detected in camera A that are next detected in camera B. (Any people traversing from A to B via camera C are excluded from the $A \rightarrow B$ traffic count.) In particular, we find that *cameras generally only send traffic to a small number of their peers.* On the 8-camera Duke dataset, only 2.0 of 7 potential peers receive more than 10% of the total outbound traffic from a given camera on average, and only 2.9 of 7 peers receive more than 2%. The full spatial statistics for Duke are plotted in **Figure 4.3**.

Exploiting this insight can significantly reduce our compute workload, at little cost to accuracy, in a large class of surveillance applications. For example, consider a setting in which we must search for a query identity q (e.g. a person), first detected in camera c_i, among the video feeds of its $n-1$ peers. In comparison to a scheme that searches all $n-1$ peers indiscriminately, if we search only those peers that receive *at least 2%* of the traffic from c_i, we reduce our compute workload by almost 60% (we search only 2.9 cameras instead of 7), while still capturing 98.82% of all detections. (This accuracy figure is computed by tabulating the total traffic volume absorbed by all cameras receiving at least 2% of traffic, averaged over all possible source cameras.)

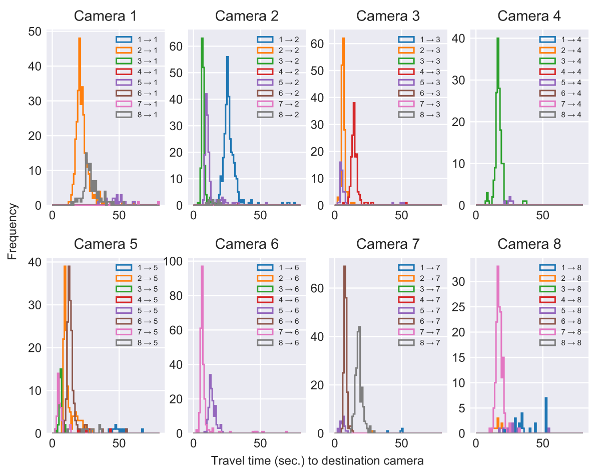


Figure 4.4: Temporal traffic patterns in the DukeMTMC dataset [79]. Plots display distribution of inter-camera travel times. Each plot corresponds to traffic to a particular destination camera. Each colored line represents a particular source camera.

Temporal locality

Our second finding is that cross-camera traffic demonstrates a high degree of *temporal* locality. As can be seen in **Figure 4.4**, travel times between a particular source camera and a particular destination camera in the Duke dataset are highly localized. This is in line with our expectations. Since these are static cameras, their pairwise distances $d_{i,j}$ are also static. Assuming that people in the network travel at an average pace p, we would expect travel times for a given camera pair (i, j) to be clustered around a mean $\mu_{i,j} = \frac{d_{i,j}}{p}$.

What is perhaps surprising is the degree of localization. We quantify localization as the *average standard deviation* in travel times $\overline{\sigma}$ across every camera pair :

$$\overline{\sigma} = \frac{1}{n^2} \sum_{i,j} \sigma_{i,j} \tag{4.1}$$

where $\sigma_{i,j}$ is the standard deviation in travel times for a particular particular camera pair (i, j) (the analog to $\mu_{i,j}$). Computing this quantity on the Duke dataset, we find that $\overline{\sigma} = 10.3$ seconds. This is relatively small compared to the average mean travel time $\overline{\mu} = 44.2$ s. across every camera pair, and the average *range* in travel times $\overline{r} = 85.0$ s.

Temporal locality, like spatial locality, implies potential compute savings. Given the task of locating a given query identity q, first identified in camera c_i, in one of the $n - 1$ possible destination cameras, one solution may be to simply search each of the $n - 1$ cameras for $\overline{r} = 85$ seconds, starting at $t = 0$, the time at which q first disappears from camera c_i. In such a scheme, we would stop the search as soon as q was rediscovered. In the worst case, we would search up to 85 seconds. $\overline{r} = 85$ s. would thus serve as our *exit threshold*.

With the above data on past network dynamics, however, we could instead do the following: we could begin our search for q on a particular camera j at $t_{i,j} - 2\sigma_{i,j}$, and end our search for q at $t_{i,j} + 2\sigma_{i,j}$. (As before, we could stop searching earlier if we discover q earlier.) Assuming that travel times are distributed normally, we would then capture 95% of the detections (by the $\pm 2\sigma \rightarrow 95\%$ rule), while searching only $\frac{4\overline{\sigma}}{\overline{r}} = 48\%$, of the frames that our fixed baseline searches.

Aggregate gains

Note that the 48% figure is conservative is one respect: we compared to a baseline that, by stopping the search at $\overline{r}$, was itself partially aware of past network dynamics! If we drop this assumption, however, it becomes harder to quantify potential gains. By exploiting temporal locality on the 8-camera Duke dataset, we can thus extract *at least* an additional 52% in compute savings, over the 60% gain achieved via spatial filtering. Assuming these gains are independent, a claim we will presently examine, spatial and temporal filtering could yield *up to 80.3% in total savings* over a baseline that searches all of the cameras for a fixed time interval (e.g. $\overline{r}$). As we will show in Section 4.8, ReXCam achieves up to a 4.75× reduction in compute cost, which is quite close to this computed upper bound of a 5× reduction.

Now we address our independence assumption. Gains due to spatial filtering and gains due to temporal filtering will be independent if (a) the distribution of travel times on a camera's closely correlated peers *is similar to* (b) the distribution of times on the remaining cameras. Our analysis of the Duke travel times shows that this is indeed the case.

4.3 Background

Video analytics

Video analytics pipelines traditionally consist of a series of *modules*, which successively decode, filter, and or run inference on video feeds. A typical surveillance pipeline may include: (1) a *decode* module, which decompresses MPEG-4 video from the camera into individual JPEG image frames, (2) a *difference detector* module, which drops frames that have not changed perceptibly from their preceding frames, (3) an *object detection* module, which extracts and classifies objects of interest in each video frame (e.g. people, vehicles), and (4) an *re-identification* or *tracking* module, which given a query image (e.g. of a person), returns the frames and cameras in which the identity is present. The distinction between re-identification and tracking is that the latter is iterative, and involves *repeatedly* re-identifying an entity, in real-time, through the camera network.

This last module is the most challenging step of most tracking applications. There are two reasons for this.

Accuracy. First, re-identification (re-id) is highly error-prone [103, 89]. Accurate re-id is particularly difficult in crowded scenes, and in large camera networks, with significant lighting and viewpoint differences across cameras. In particular, surveillance footage is typically too low-resolution to apply facial recognition techniques, which can be used to distinguish and link identities [104]. Instead, re-id models must rely primarily on clothing and profile, which are much weaker unique identifiers than biometric traits.

Cost. Second, tracking in large camera networks is computationally expensive. Even tracking a single object through a camera network can potentially require processing every subsequent frame, in every camera, after an initial detection (in the absence of good heuristics for geographic localization). This translates to a large search space, in both the spatial dimension (number of cameras) and in the temporal dimension (number of frames). Moreover, unlike a stateless, per-frame task such as object detection (e.g. "flag all frames containing [buses / trucks / SUVs]"), identity tracking cannot easily be batched (each query is independent), parallelized (associations span cameras), or pipelined (tracking is stateful). These two properties – (1) the large inference space and (2) the sequential execution requirement – make cost-efficient live execution crucial for tracking workloads.

Key applications

Extensive networks of cameras are already installed in major cities such as London, Beijing, and Chicago [95, 1, 15] – on rapid transit systems, public buses, airports, corporate campuses, and city streets [65, 100]. In this section, we briefly survey the main use cases for intelligent cross-camera analytics systems, which operate on such networks.

Security and counter-terrorism

A key use case for cross-camera re-identification and tracking capabilities is localizing suspects in the aftermath of a security breach or major attack. For example, an on-site camera may record and flag a trespassing violation or burglary. Given that the perpetrator will then attempt to exit the premises, re-identification techniques can be used to locate the suspect in the surrounding network of cameras.

Alternatively, after a major public attack (e.g. on a public transport system), law enforcement may wish to track the accomplices of an identified perpetrator [100]. As a first step, they may scan a database of stored video for people frequently associated with the identified assailant [100]. Discovering these people in the aftermath of the attack, however, among the 12,000 cameras feeds installed on the London Underground [11] in a timely, scalable manner is a daunting live data analytics challenge. Here, cross-camera re-identification and tracking enables both initial discovery, and subsequent tracking, to allow for police apprehension.

Vehicle tracking

In the U.S. and Europe, AMBER alerts are raised if a child abduction is suspected [4]. Alerts containing the license plate number, model, and color of the captor's vehicle are broadcast by radio, television, and text messages to all citizens in the area [4]. Given camera installments along highways and city streets, vehicle re-identification and tracking techniques can be used to locate and keep tabs on the suspect's vehicle, as police attempt intervention [65].

Retail and business intelligence

Automated checkout systems, such as Amazon Go, rely on computer vision techniques to map actions (e.g. picking up an item, returning an item) to people, enabling Amazon to accurately charge customers for their purchases [3]. In other settings (e.g. large stores, theme parks), cross-camera analytics techniques can be used to track shopper movement (to optimize inventory placement), count the number of customers standing in lines (to plan staff shifts), and identify repeat visitors (to analyze retention) [78, 104, 100]. All of these applications benefit from improvements in the accuracy and compute efficiency of re-identification and tracking.

Setup and compute model

In our assumed setup, a camera network consists of k nodes. Each node hosts a high-definition, closed-circuit television (CCTV) camera with an on-board secure digital (SD) card, offering a small amount of local disk storage. All nodes are connected to each other over a high-speed local area network (e.g. Ethernet, Wi-Fi [100]). We assume sufficient bandwidth on inter-camera network links to transmit video and query metadata. Queries are issued by an operator in a surveillance center, which could be either located on-site with the cameras or in a remote location.

For the purposes of this section, we assume that all video is streamed to the cloud for analytics. In particular, all re-identification and tracking queries are executed in the cloud, and inference results are streamed to the operator (e.g. in a private web interface) in step with the live video. This is the most common setup for intelligent video processing applications (e.g. home security, public surveillance) today. Cloud-based processing has the benefit of offering a simple, elastic, centralized compute abstraction, which eases some aspects of implementation (e.g. cross-camera inference).

ReXCam, however, is not bound to this model. We briefly discuss two possible alternatives. First, video could be analyzed on an on-site deployment of server hardware ("edge cluster"), managed by the same enterprise running the analytics operation (e.g. local police). Second, video could be analyzed *on the camera itself*, given a deployment of AI cameras ("smart cameras"), each of which posses a small processor and hardware accelerator.

A key tradeoff between cloud-based analytics and edge-based analytics is the cost model. Cloud processing incurs time-rated or usage-rated pricing. Reducing workload translates directly to fewer GPU instance hours spent on processing, and thus proportionally lower costs, assuming effective resource utilization. Edge-based processing, in contrast, requires upfront investment in expensive hardware (e.g. GPU clusters, smart cameras). Reducing average and peak workloads here enables more video feeds to be processed per GPU, which in turn reduces hardware requirements. In this model, cost savings could be particularly substantial for enterprises planning to setup large *new* analytics operations.

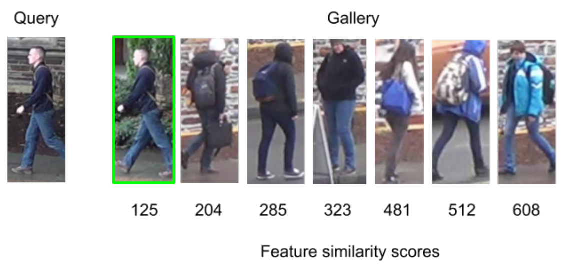


Figure 4.5: Illustration of identity re-identification.

4.4 System Operation

Identity re-identification

Identity tracking in ReXCam is implemented on a basic computer vision primitive known as *identity re-identification*. Given an image of a query identity q, a re-identification (re-id) algorithm ranks every image g_i in a gallery G based on its *feature distance* to q – a Euclidean distance metric defined on the space $R^{1 \times h \times w \times c}$ of image features (see **Figure 4.5**). Typically, these features are the intermediate representation of a deep neural network trained to associate instances of the same (co-identical) entity, and differentiate instances of different (non-co-identical) entities.

A successful re-id algorithm will rank co-identical (positive) instances to the query more highly in the list than non-co-identical (negative) instances. By extension, in a *perfect* re-id ranking, all i co-identical instances to q present in G appear in the top i list entries. Over the full set of queries $q \in Q$, a perfect ranking satisfies:

$$\forall q \in Q \quad \max_{p \sim q} d(q,p) < \min_{n \nsim q} d(q,n) \tag{4.2}$$

where $d(\cdot,\cdot)$ denotes the feature distance metric, and p and n denote positive and negative instances, respectively [78].

A re-id ranking is typically evaluated on two metrics. The first is *rank-k accuracy*, which is the percentage of the top k list entries that consist of positive examples. Rank-1 accuracy, for example, indicates how often the top ranked entry in the gallery matches the query. The second metric is *mean average precision* (mAP). mAP is a finer-grained accuracy metric from information retrieval and computer vision that sums the product of (a) precision and (b) change in recall across every position in the ranking. Both rank-k accuracy and mAP values fall between 0% and 100%, where 100% indicates a perfect ranking.

Identity tracking

Given the ability to rank a set of detections based on their similarity to a query image, we can now define and implement *cross-camera identity tracking*. In tracking, the input consists of a query image q, extracted from frame f_q on camera c_q. The goal is to flag all subsequent frames, on all cameras, that contain co-identical instances to q, while maximizing two metrics: (1) recall – the fraction of positive instances successfully retrieved, and (2) precision – the fraction of retrieved instances that are positive. Note that q can appear again on the same camera ($c = q^c$), different cameras ($c \neq q^c$), or else exit the network altogether at any point. Tracking stops when q has deemed to have exited.

Algorithm 3 Tracking in ReXCam.

```
1: input: video feeds $\{V_c\}$ for camera $c$
2: for query $(q, f_q, c_q) \in Q$ do
3:     $q_{\text{feat}} = \text{features}(q)$                          ▷ extract image features
4:     $f_{\text{curr}} = f_q + 1$                                   ▷ init current frame index
5:     $M_q = []$                                                   ▷ init query match array
6:     while $(f_{\text{curr}} - f_q) \leq \text{exit\_t}$ do
7:         frames = get_frames($V, f_{\text{curr}}$)
8:         gallery = extract_entities(frames)
9:         ranked = rank_reid($q_{\text{feat}}$, gallery)
10:        if ranked[0][dist] < match_thresh then
11:            $M_q = \text{append}(M_q, \text{ranked}[0][\text{img}])$
12:            $q_{\text{feat}} = \text{update\_rep}(q_{\text{feat}}, \text{ranked}[0][\text{feat}])$
13:            $f_q = f_{\text{curr}}$
14:        end if
15:        $f_{\text{curr}} = f_{\text{curr}} + 1$
16:    end while
17: end for
18: output: matched detections $\{M_q\}$
```

We propose **Algorithm 3** for cross-camera identity tracking. Given a set of video feeds V_c, we wish to execute $|Q|$ separate tracking queries. For each tracking query q, we begin by extracting image features q_{feat} and initializing an empty array of discovered matches M_q. We then proceed to repeatedly: (1) retrieve the current frame from each video feed, (2) extract entities from each frame using an object detection model, (3) rank the detections based on their feature distance to q using a re-id model, and (4) check if the distance to the top ranked detection is within a *match threshold.*

The match threshold is a binary decision cutoff we impose to convert re-id, a ranking algorithm, into a *classifier*. In particular, since it is possible for q to fail to appear in *any* camera at a given frame index f_{curr} (due to occlusions, blind spots in the camera network), the detection gallery may contain no co-identical instances. Thus, we must determine if the top match is in fact co-identical. If we decide that it is, we add the detection to our array of matches M_q, update our query *representation* q_{feat} to incorporate the features of the new instance of q, update the query frame index f_q to f_{curr}, and proceed with tracking q. If we instead decide that no co-identical instances are present at f_{curr}, we increment the current frame index f_{curr}, and proceed to search the next set of frames for q.

We continue searching until the gap between the last detected instance of q and our current frame index exceeds a pre-defined *exit threshold* (defined as exit_t in Algorithm 3). At this point, we conclude that q must have exited the camera network, and cease tracking q. We then repeat this process for the next query in Q.

4.5 Spatio-Temporal Correlations

Overview

ReXCam exploits two forms of cross-camera correlations to improve cost efficiency and inference accuracy in multi-camera video analytics. **Spatial correlations** capture long-term associations between camera pairs. These include, but are not limited to, associations arising from camera topology (e.g. nearby cameras tend to send more traffic to each other than distant cameras). The degree of spatial correlation d_{sc} between two cameras c_s, c_d is quantified by the ratio of (a) the number of entities leaving the source camera for the destination camera, $n(c_s, c_d)$, to (b) the total number of entities leaving the source camera:

$$d_{sc}(c_s, c_d) = \frac{n(c_s, c_d)}{\sum_i n(c_s, c_i)} \tag{4.3}$$

In particular, a camera c_i that receives a large fraction of the outgoing traffic from source camera c_s is said to be *highly correlated* to camera c_s. Note that spatial correlations may be asymmetric. In the previous example, it is possible that camera c_s is *not* highly correlated with camera c_i, even if the converse is true. ReXCam exploits spatial correlations in its search for a query identity q by prioritizing destination cameras that are highly correlated to the query camera c_q.

Temporal correlations capture associations between camera pairs *over time*. If a large percentage of the traffic leaving camera c_s for camera c_d arrives between t_1 and t_2, then camera c_d is said to be *highly correlated in* $[t_1, t_2]$ to camera c_s. The degree of temporal correlation d_{tc} between two cameras c_s, c_d during a time interval $[t_1, t_2]$ is quantified by the ratio of (a) entities reaching c_d from c_s during $[t_1, t_2]$ to (b) total entities reaching c_d from c_s:

$$d_{tc}(c_s, c_d, [t_1, t_2]) = \frac{n(c_s, c_d, [t_1, t_2])}{n(c_s, c_d)} \tag{4.4}$$

Like spatial correlations, temporal correlations can be asymmetric, with arrival counts peaking at different time intervals in the different directions. Note also that camera c_d can be temporally correlated to c_s at time t, without c_d being *spatially* correlated to c_s, if most of the small amount of traffic from c_s to c_d arrives around t. ReXCam exploits temporal correlations in its search for q by prioritizing the *time window* $[t_1, t_2]$ in which a destination camera is most correlated with the query camera c_q.

Spatio-temporal model

Given a source camera c_s, the current frame index f_{curr}, which serves as a timestamp, and a destination camera c_d, our proposed spatio-temporal model M outputs `true` if c_d is both spatially correlated and temporally correlated with c_s at f_{curr}, and `false` otherwise.

The thresholds for being spatially correlated with c_s, and temporally correlated with c_s at time f_{curr}, are model parameters set by the system operator. As an example, we may first wish to search cameras receiving at least $s_{\text{thresh}} = 5\%$ of traffic from c_s, during the time window containing the first $1 - t_{\text{thresh}} = 98\%$ of traffic from c_s. These parameter settings exclude both *outlier cameras* (cameras receiving less than 5% of the traffic from c_s) and *outlier frames* (frames containing either the last 2% of traffic or no traffic from c_s).

Once built, M will only output `true` (i.e. 1) if both conditions s_{thresh} and t_{thresh} are met at f_{curr}. Formally:

$$M(c_s, c_d, f_{\text{curr}}) = \begin{cases} 1, & d_{sc}(c_s, c_d) \geq s_{\text{thresh}} \\ & \quad \textbf{and} \\ & d_{tc}(c_s, c_d, [f_0, f_{\text{curr}}]) \leq 1 - t_{\text{thresh}} \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

Here f_0 is the frame index at which the first historical arrival at c_d from c_s was recorded. Our temporal filter checks if the volume of historical traffic that arrived at c_d between $[f_0, f_{\text{curr}}]$ is less than $1 - t_{\text{thresh}}\%$ of the total traffic. This ensures that f_{curr} falls in the "dense" part of the travel time distribution, where we are likely to find q.

Our model M encodes the spatial and temporal locality inherent in the camera network (see Section 4.2). By first examining spatio-temporally correlated camera frames, we explore the part of the inference space most likely to contain q. A "cache hit" reduces inference cost, as we avoid searching the entire space of detections at f_{curr}. On the other hand, on a "cache miss", we must subsequently process the remainder of the inference space. On these rarer cache misses, using M incurs a penalty, as this procedure introduces *delay*: instead of detecting q in real-time, we must find q in past video frames in our second pass through V. If cache misses are rare enough, and we can mitigate the delay they introduce (Section 4.7), then such a system will outperform one that is locality-agnostic.

Cost savings

We can quantify the savings achieved by spatio-temporal filtering, compared to a baseline that applies no filtering and thereby searches all cameras, with the following cost ratio:

$$r_c = \frac{\mathrm{c}^{\mathrm{det}}_{\mathrm{baseline}} + \mathrm{c}^{\mathrm{feat}}_{\mathrm{baseline}} + \mathrm{c}^{\mathrm{reid}}_{\mathrm{baseline}}}{\mathrm{c}^{\mathrm{det}}_{\text{st-filter}} + \mathrm{c}^{\mathrm{feat}}_{\text{st-filter}} + \mathrm{c}^{\mathrm{reid}}_{\text{st-filter}}} \tag{4.6}$$

$$= \frac{\mathrm{c}^{\mathrm{det}}|V| + \mathrm{c}^{\mathrm{feat}}|V| \cdot \bar{d} + \mathrm{c}^{\mathrm{reid}}|V| \cdot \bar{d}}{\mathrm{c}^{\mathrm{det}}|V_{\mathrm{corr}}| + \mathrm{c}^{\mathrm{feat}}|V_{\mathrm{corr}}| \cdot \bar{d} + \mathrm{c}^{\mathrm{reid}}|V_{\mathrm{corr}}| \cdot \bar{d}} \tag{4.7}$$

$$= \frac{|V|}{|V_{\mathrm{corr}}|} \tag{4.8}$$

where $|V|$ refers to the total number of cameras, $|V_{\mathrm{corr}}|$ the number of cameras correlated with the query camera c_q at f_{curr}, and $\bar{d}$ the average number of entity detections per camera frame, while $\mathrm{c}^{\mathrm{det}}$, $\mathrm{c}^{\mathrm{feat}}$, $\mathrm{c}^{\text{re-id}}$ represent the costs of running object detection, extracting features, and computing re-id feature distance to q, respectively, for a single frame or detection (see Algorithm 3). **Equation 4.8** signifies that the achieved savings factor r_c reduces to the ratio of (a) the total number of cameras $|V|$ and (b) the number of correlated cameras $|V_{\mathrm{corr}}|$ at a given frame step. While $|V_{\mathrm{corr}}|$ varies based on c_q and f_{curr}, we see that, on average, achieved savings are proportional to the degree of correlation filtering.

Applying the model

Applying such a spatio-temporal model involves a series of small modifications to the cross-camera tracking algorithm (see **Algorithm 4**). First, in addition to the video feeds $\{V_c\}$, we must pass as input the spatio-temporal model itself. The model is represented as two filters, both of which return {`true`, `false`} values: (1) **spatial_corr**(c_s, c_d), which given a source camera c_s and a destination camera c_d returns `true` if c_d is correlated with c_s, and (2) **temporal_corr**(c_s, c_d, f), which given a source camera c_s, a destination camera c_d, and a frame index f, returns `true` if c_d is correlated with c_s at f. At query time, these two functions are passed to a higher-order **filter** function, which given a list of video feeds V, returns the subset of V that is *both* spatially and temporally correlated to c_q at f_{curr}.

Applying **filter** reduces the inference search space, at each frame step f_{curr}, from all entity detections at f_{curr} on every camera to all entity detections at f_{curr} on *correlated* cameras. This allows us to abstain from running object detection and feature extraction models on non-correlated cameras, and reduces the size of the re-id gallery in the ranking step.

The penalty paid for this reduced compute cost is missed true positive detections. While we expect instances of q to appear on correlated cameras at peak times *in general*, we also expect occasions where this will not be the case. When q reappears on non-correlated cameras or at non-peak times, we will fail to rediscover q, and at $f_{\mathrm{curr}} = f_q + \text{exit\_t}$, incorrectly declare that q has exited. To address this issue, we introduce a conditional second phase to Alg. 4,

Algorithm 4 Tracking with the spatio-temporal model

```
1: **input**: video feeds $\{V_c\}$ for camera $c$,
2:         $\text{spatial\_corr}(c_s, c_d) \rightarrow \{\texttt{true}, \texttt{false}\}$
3:         $\text{temporal\_corr}(c_s, c_d, f) \rightarrow \{\texttt{true}, \texttt{false}\}$
4: **for** query $(q, f_q, c_q) \in Q$ **do**
5:     $q_{\text{feat}} = \text{features}(q)$                         ▷ extract image features
6:     $f_{\text{curr}} = f_q + 1$                                    ▷ init current frame index
7:     $M_q = []$                                                    ▷ init query match array
8:     $\text{phase} = 1$                                            ▷ start phase one
9:     **while** $(f_{\text{curr}} - f_q) \leq \text{exit\_t}$ **do**
10:        $V_{\text{corr}} = \textbf{filter}(\text{sp\_corr}, \text{tp\_corr}, c_q, f_{\text{curr}}, V)$
11:        $\text{frames} = \text{get\_frames}(V_{\text{corr}}, f_{\text{curr}})$
12:        $\text{gallery} = \text{extract\_entities}(\text{frames})$
13:        $\text{ranked} = \text{rank\_reid}(q_{\text{feat}}, \text{gallery})$
14:        **if** $\text{ranked}[0][\text{dist}] < \text{match\_thresh}$ **then**
15:            $M_q = \text{append}(M_q, \text{ranked}[0][\text{img}])$
16:            $q_{\text{feat}} = \text{update\_rep}(q_{\text{feat}}, \text{ranked}[0][\text{feat}])$
17:            $f_q = f_{\text{curr}}$
18:            $\text{phase} = 1$                                    ▷ reset to phase one
19:        **end if**
20:        $f_{\text{curr}} = f_{\text{curr}} + 1$
21:        **if** $\text{phase} = 1$ **and** $(f_{\text{curr}} - f_q) > \text{exit\_t}$ **then**
22:            $f_{\text{curr}} = f_q + 1$                            ▷ reset frame index
23:            $\text{sp\_corr} = \neg\text{sp\_corr}$                 ▷ invert spatial filter
24:            $\text{phase} = 2$                                    ▷ start phase two
25:        **end if**
26:    **end while**
27: **end for**
28: **output**: matched detections $\{M_q\}$
```

which we call *replay search*. Given no matches on the correlated cameras from $f_{\text{curr}} = f_q + 1$ to $f_{\text{curr}} = f_q + \text{exit\_t}$, we regress to searching the cameras *not* correlated with c_q. One way to implement this is to negate the output of the correlation filter $\textbf{spatial\_corr}(c_s, c_d)$, and instead filter the cameras in V that we have *already searched*.

In particular, phase two of Alg. 4 (lines 21-24) initiates when we cross the exit threshold (exit_t), which signifies that either we have missed q by pruning the search space, or that q has in fact exited the camera network. To rule out the former possibility, we reset f_{curr} to the query frame index f_q, and invert our spatial correlation filter $\textbf{spatial\_corr}(c_s, c_d)$. We then restart the tracking procedure from $f_{\text{curr}} = f_q + 1$, looking for the next instance of q in video feeds *not* spatially correlated with c_q. If we do discover an instance of q, we proceed with tracking from that detection, initiating a new phase one. If we do not, then we cease our search for q at the exit threshold, as in the original algorithm (Alg. 3).

Note that regressing to the baseline involves searching for q in *historical video*. Doing this efficiently, and mitigating the delay we accumulate by searching for q in the past, while the live video stream progresses, is a key challenge introduced by spatio-temporal filtering. We discuss our solution, a fast-replay search mode, in Section 4.7.

4.6 Offline Profiling

How do we generate a model of spatio-temporal correlations? One approach that builds on standard techniques from computer vision is to use an offline *multi-target, multi-camera* (MTMC) *tracker* to label every entity detection in a dataset of historical video, collated from the same camera deployment on which the live tracking is executed. The goal of an MTMC tracker is to accurately map instances of the same entity, detected across frames and across cameras, to the same *entity identifier*. In the output of the tracker, each detected entity instance i is represented as a tuple, (c_i, f_i, e_i), containing the camera identifier c_i, frame index f_i, and entity identifier e_i for the detection, respectively.

Using this labeling, one can then compute two quantities:

1. $n(c_s, c_d)$ – the total number of entities leaving a source camera c_s for a destination camera c_d
2. $n(c_s, c_d, [t, t+1])$ – the total number of entities reaching c_d from c_s within the time interval $[t, t+1]$

for all cameras c_s and c_d, and each time interval $[t, t+1]$.

These quantities translate directly to our spatio-temporal model M (see Section 4.5). In particular, by normalizing (1) and imposing a specific spatial traffic cutoff s_{thresh} (e.g. 5%), we obtain our spatial filter:

$$\textbf{spatial\_corr}(c_s, c_d) = \frac{n(c_s, c_d)}{\sum_i n(c_s, c_i)} \geq s_{\text{thresh}} \tag{4.9}$$

By normalizing (2) and imposing a specific temporal cutoff t_{thresh} (e.g. 2%), we obtain our temporal filter:

$$\textbf{temporal\_corr}(c_s, c_d, f) = \frac{n(c_s, c_d, [f_0, f])}{n(c_s, c_d)} \leq 1 - t_{\text{thresh}} \tag{4.10}$$

Note that these are simply the two sub-conditions, passed to our tracking executor, of our full model M from Equation 4.5.

A multi-target, multi-camera (MTMC) tracker differs from the tracking module outlined in Algorithms 3 and 4 in that it tracks *all entities* in the dataset. This is needed to build a robust model of cross-camera traffic patterns. (In contrast, Algorithms 3 and 4 implement single-target tracking, which is the key application of interest in real-time security applications.) MTMC tracking is a highly intensive profiling operation that is typically performed *offline* on a static dataset [101, 78]. MTMC tracking exploits techniques from both computer vision (e.g. appearance matching, motion correlation) and combinatorial optimization (e.g. maximum bipartite matching, correlation clustering) [101, 78] to find the best possible assignment of identities to people. One accuracy metric that is commonly used to evaluate an MTMC tracker is its F1 score, the harmonic mean of its recall and precision [78].

Note that a tradeoff exists between the robustness of offline profiling and the accuracy of subsequent single-target tracking using the generated model. In particular, profiling cost can be reduced by labeling fewer frames with the MTMC tracker (e.g. by selecting a lower frame sampling rate or choosing a smaller subset of the data to label). This, however, magnifies the impact of labeling error and biased sampling, which in turn can translate to a weaker spatio-temporal model. For example, if too few frames are labeled, certain cross-camera correlations may be excluded or exaggerated (e.g. the spatial association between two cameras c_s and c_d). We explore this tradeoff between one-time profiling cost and tracking accuracy in Section 4.8.

4.7 Fast-Replay Search

Utilizing spatio-temporal correlations has a fundamental cost: missed true positive detections of the query identity q, which would be discovered by a baseline that searches all of the cameras. Our solution in Section 4.5 is to initiate a second pass through the video frames on cameras we did not previously examine, which we call *replay search*. This introduces *delay* in our cross-camera tracking, as we search past frames while the live video progresses.

Delay is the gap, in seconds, between the position of the tracker and the position of the live video stream. This quantity is 0 for a query if ReXCam never performs replay search. On the other hand, each instance of replay search introduces d seconds of delay. This can

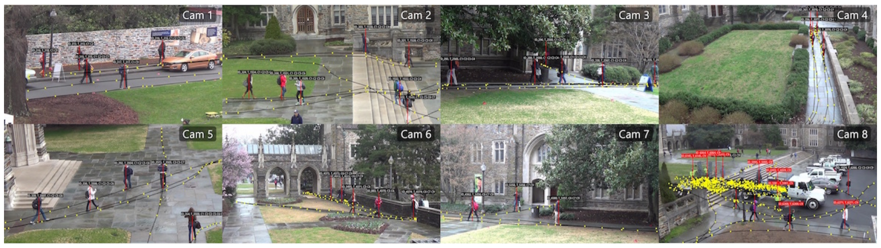


Figure 4.6: DukeMTMC camera network [79]

be calculated as follows:

$$d = \frac{(f_q + \text{exit\_t}) - f_q}{r} \tag{4.11}$$

$$= \frac{\text{exit\_t}}{r} \tag{4.12}$$

where r is the frame rate of the tracker during fast-replay search. This is the time required to return to the tracker's original position in the video (before it began replay search), $f_q + \text{exit\_t}$. Since we start our replay search at f_q, this is an effective distance of exit_t.

Note that can *choose* our tracker's frame rate, r. Typically, this is just the video frame rate (e.g. 30 fps). However, in fast-replay search, we explicitly choose to operate at a *faster-than-real-time* frame rate to minimize the per-instance delay d.

In ReXCam, a higher frame rate is achieved by assuming one of two operational modes:

1. *Skip frame mode* – Employs a lower *frame sampling rate* on historical video frames to increase throughput, at the cost of lower accuracy.

2. *Fast-forward mode* – Employs a higher *frame processing rate* (e.g. via parallelization) to increase throughput, at the cost of increased resource usage.

Both the skip frame mode and the fast-forward mode have trade-offs: the former raises the likelihood of missed detections, while the later increases resource usage.

We implement both solutions, and investigate their trade-offs, in our experimental evaluation (Section 4.8).

4.8 Experiments

Dataset

We evaluate on the DukeMTMC dataset [79], a large-scale video surveillance dataset with footage from eight cameras installed on the Duke University campus (see **Figure 4.6**). The data consists of 85 minutes of 1080p video from each camera recorded at 60 frames per second. In all, the footage contains over 2,700 unique identities and over 4 million distinct person detections. The dataset is meticulously labeled with person identities and bounding boxes, with annotations available for 2 million frames. The data is split into a 50 minute `train/val` set and a 35 minute `test` set.

Implementation

For our re-identification model, we use an open-source, ResNet-50-based implementation of person re-identification [26], trained in PyTorch on a subset of the Duke dataset called DukeMTMC-reID [27]. We then propose and implement our own version of tracking (see Algorithms 3, 4), which applies this model iteratively at inference time to discover all instances of a query identity in the Duke dataset. Our cross-camera person tracking testbed is open-source on GitHub [24].

To build our spatio-temporal model on unlabeled video data (simulating real deployment conditions), we apply an offline multi-target multi-camera (MTMC) tracker [69], as described in Section 4.6, to label every person detection in a subset of the Duke `train` set. We then implement a *profiler* to extract spatial and temporal correlation statistics from these labels. Note that the profiler implements Equations 4.9 and 4.10, yielding the spatial and temporal correlation filters used at inference time.

Evaluation Setup

Our evaluation procedure consists of running a set of 100 tracking queries, $\{q_i\}$, drawn from the `test` query partition of the DukeMTMC-reID dataset [27]. Each tracking query consists of multiple *iterations*. Each iteration involves searching for the next *instance*, q_i^j, of the query identity in the dataset, starting with the initial instance q_i^0. A tracking query terminates when no more instances can be found.

We report four metrics – compute cost, recall, precision, and delay – which are computed over the entire 100 query `test` set. As described in the Problem Statement (Section 4.2), these metrics correspond to the following quantities:

1. **Compute cost** – Number of video frames processed, aggregated over all queries $\{q_i\}$.
2. **Recall (%)** – Ratio of query instances retrieved to all query instances in dataset, q_i^j.
3. **Precision (%)** – Ratio of query instances retrieved to all retrieved instances, r_i^j.
4. **Delay (sec.)** – Lag between position of tracker and current video frame, converted to seconds, at the end of a tracking query. Note that this will be 0 for a query if no replay search was performed.

Compute cost, recall, and precision are reported in aggregation. Delay is reported as an average value per query.

We conducted our experiments on AWS EC2 p2.xlarge instances, which each contain one Nvidia Tesla K80 GPU.

Results

Spatio-temporal filtering

To evaluate our core spatio-temporal filtering scheme, we compare two high-level systems:

1. **Baseline** - Searches for query identity q in all the cameras at every frame step. Implementation of Alg. 3. Uses state-of-the-art person re-identification model [26].

2. **ReXCam** - Searches for query identity q only on cameras that are currently *spatio-temporally correlated* with c_q. Implementation of Alg. 4. Uses same person re-identification model as baseline [26].

In particular, we consider various versions of (2) corresponding to different levels of spatio-temporal filtering:

(a) **Spatial-1%** - Filters cameras that receive less than **1%** of the traffic from query camera c_q. (**S1**)

(b) **Spatial-5%** - Filters cameras that receive less than **5%** of the traffic from query camera c_q. (**S5**)

(c) **Spatial-1%, Temporal-1%** - Filters cameras that receive less than **1%** of the traffic from query camera c_q. In addition, filter frames outside the time window containing the **first 99%** of traffic from c_q. (**S1-T1**)

(d) **Spatial-5%, Temporal-1%** - Filters cameras that receive less than **5%** of the traffic from query camera c_q. In addition, filter frames outside the time window containing the **first 99%** of traffic from c_q. (**S5-T1**) (**ReXCam-O**ptimal)

(e) **Spatial-5%, Temporal-2%** - Filters cameras that receive less than **5%** of the traffic from query camera c_q. In addition, filter frames outside the time window containing the **first 98%** of traffic from c_q. (**S5-T2**)

Note that the **baseline** utilizes *no* spatio-temporal filtering. ReXCam versions **S1** and **S5** utilize only spatial filtering. RexCam versions **S1-T1**, **S5-T1**, and **S5-T2** utilize spatio-temporal filtering.

As discussed in Section 4.5, the level of spatio-temporal filtering is quantified by two model parameters. Spatial filtering is quantified by the spatial traffic threshold s_{thresh}, which represents the minimum percentage of traffic a camera must receive to be searched. In our evaluation, we consider two possible settings, $s_{\text{thresh}} = 1\%$ and $s_{\text{thresh}} = 5\%$.

Temporal filtering is quantified by the temporal traffic threshold, t_{thresh}, which specifies the time window of frames that we search on a destination camera c_d. In all cases, we begin our search at the time t_0 at which the first historical arrival at c_d was recorded (e.g. $t_0 = 3.2$ s). We terminate our search at the time exit_t, which marks the point at which $1 - t_{\text{thresh}}$ percent of the historical traffic had arrived. For example, if $t_{\text{thresh}} = 2\%$, we search until

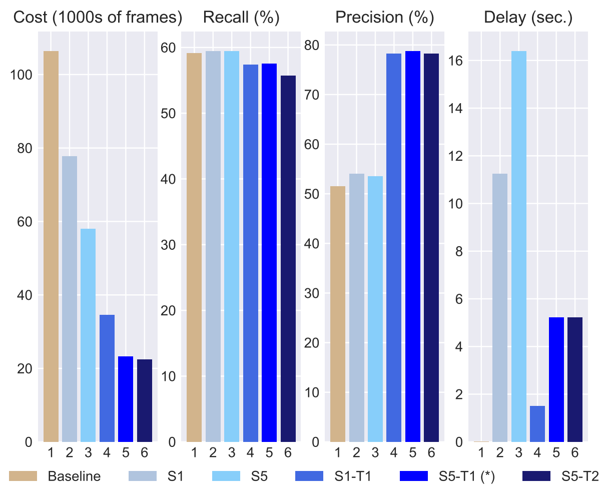


Figure 4.7: Results for all-camera baseline (tan) vs. five versions of ReXCam (blues). Each ReXCam version is coded as Ss-Tt, where s indicates the spatial filtering threshold (e.g. $s = 5\%$) and t indicates the temporal filtering threshold. Higher values of s and t indicate more aggressive filtering. (No t value indicates no temporal filtering.) All ReXCam schemes (1) reduce compute cost and (2) improve precision over the baseline. We argue S5-T1 (*) offers the best trade-off on all four metrics.

the time exit_t such that 98% of the historical traffic had arrived at camera c_d. We do this to avoid searching frames in which traffic is unlikely to arrive. (Our goal is to cut down on compute cost without impacting accuracy.) In our evaluation, we consider two possible settings, $t_{\text{thresh}} = 1\%$ and $t_{\text{thresh}} = 2\%$.

In **Figure 4.7**, we compare the performance of the baseline and ReXCam versions (a) - (e). In general, we find that ReXCam *significantly outperforms the baseline*, by (1) reducing compute cost and (2) improving precision, while maintaining comparable recall. As we will show in the following discussion, we believe that ReXCam version (d), in particular, offers the best trade-off between compute cost, recall, precision, and delay. We term this scheme **ReXCam-O**(ptimal). We now compare the six schemes:

1. **Compute cost** – The baseline is by far the most compute-intensive system, processing 106,300 frames to execute 100 queries on the Duke dataset (Figure 4.7). Each successive version of ReXCam ((a) - (e)) achieves lower compute cost than its predecessor. The most aggressive version of ReXCam, S5-T2, processes only 22,400 frames, and *achieves 4.75× lower compute cost* on 8 cameras than the all-camera baseline.

   In comparison, **ReXCam-O** processes 23,200 frames, which translates to 4.58× lower compute cost than the all-camera baseline.

2. **Recall (%)** – Note here an interesting effect. Our baseline achieves recall of 59.1%, which published results for the DukeMTMC dataset [26]. Recall *improves* slightly

over the all-camera baseline with the advent of spatial filtering (**S1**, **S0**), but *declines* slightly when temporal filtering is introduced.

Details: Both spatial-only schemes achieve 59.4% recall. **ReXCam-O** achieves 57.5%, a 1.6% drop from the baseline. S5-T2 achieves 55.7%, a 3.4% drop.

Spatial filtering improves recall because we search fewer irrelevant cameras, reducing false positive matches, which in turn derail subsequent tracking accuracy. Temporal filtering *reduces* recall because we deliberately skip over outlier detections (the last 1-2% of traffic) to reduce compute cost. In general, both effects are small.

In particular, **ReXCam-O**'s 1.6% drop in recall is a small price to pay for a 4.58× decrease in compute cost, and significant gains in precision, discussed next.

3. **Precision (%)** – Our baseline achieves precision of 51.5%. All version of ReXCam improve on this, but **ReXCam-O** in particular achieves 78.7% precision, which is a *gain of 27.2% over the baseline.*

   Higher precision is one of the two key ways in which spatio-temporal filtering improves on the current practice in cross-camera video analytics. By searching fewer irrelevant cameras, and fewer irrelevant frames, ReXCam is less likely to declare matches on persons that do not actually match the query. ReXCam thus addresses a well-known challenge in large-scale, image retrieval.

4. **Delay (sec.)** – Here we report total cumulative lag (lag in the absence of fast-replay search (Section 4.7)), averaged over all 100 queries.

   We find that delay is highest with spatial-only filtering (e.g. S5) because every time ReXCam regresses to the baseline, it must search every skipped camera for a fixed duration. Lacking any camera-specific temporal information (i.e. t_{thresh}), this is the best it can do. This introduces a fixed delay of exit_t seconds with every instance of regression (see Algorithm 4).

   Delay is also higher with more spatial filtering (e.g. S5, S5-T1) as there are more instances of regressions. **ReXCam-O** in particular incurs moderate delay – less delay than S1 and S5 but more delay than S1-T1.

Given this analysis, we believe that **ReXCam-O** offers the best possible tradeoffs between the four metrics – achieving nearly the lowest compute cost (4.6× lower), nearly the highest precision (27% higher), competitive recall (1.6% lower), and moderate cumulative lag (5.2 seconds), when compared to the locality-agnostic, all-camera baseline.

Fast-replay search

In this section, we evaluate the efficacy of fast-reply search in eliminating lag (see **Figure 4.8**). In particular, we consider the two proposed schemes from Section 4.7:

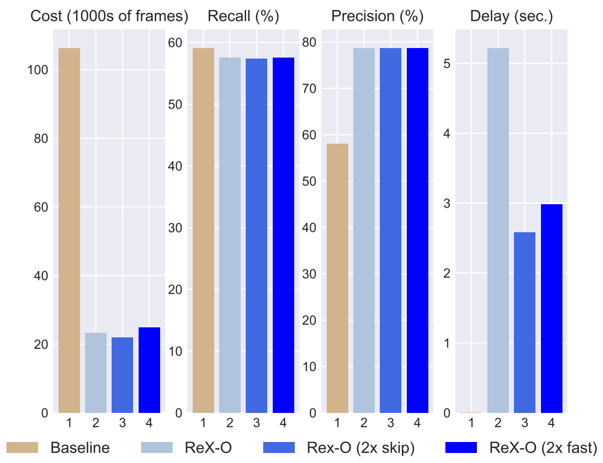


Figure 4.8: Fast-replay search evaluation. Schemes compared: baseline, ReXCam-O (no fast-replay), ReXCam-O (2× skip), ReXCam-O (2× fast-forward). We find that scheme 2× skip outperforms 2× fast-forward on both compute cost and delay reduction.

1. *Skip frame mode* - Employ a $\frac{x}{2}$ frame sampling rate to increase throughput on historical frames, at the price of lower accuracy (via missed detections). (**2x skip**)

2. *Fast-forward mode* - Employ a $2x$ frame processing rate to increase throughput, at the price of increased compute cost (via increased resource usage). (**2x ff**)

Both schemes are applied to **ReXCam-O**, and compared to (a) the all-camera baseline and (b) **ReXCam-O** with the default *real-time* replay search, which incurs 5.2 s of delay.

We find that both **2x skip** and **2x ff** achieve similar delay reductions, decreasing final cumulative lag to 2.6 and 3.0 seconds, respectively. However, these reductions come with different tradeoffs. **2x skip** reduces recall by 0.1% to 57.4%, but *increases compute cost savings* from 4.58× to 4.84× better than the baseline (by processing fewer historical frames). **2x ff** does not impact recall, but reduces compute cost savings from 4.58× to only 4.28× better than the baseline. Neither scheme impacts precision.

Taken together, these results demonstrate that **2x skip** is the stronger scheme, as it reduces compute cost (instead of increasing it) and slightly outperforms **2x ff** on delay reduction, while recording negligible impact on recall.

In general, by implementing fast-replay search (**2x skip**), we are able to reduce delay by exactly 50% (from 5.2 to 2.6 seconds), at the cost of only 0.1% lower recall.

Profiling cost vs. tracking accuracy

In this final experiment, we investigate the trade-off between profiling cost and subsequent tracking accuracy. Noting that offline profiling cost scales with the number of frames that must be processed by the MTMC tracker (Section 4.6), we test whether we can build a robust spatio-temporal model on successively smaller subsets of the training data.

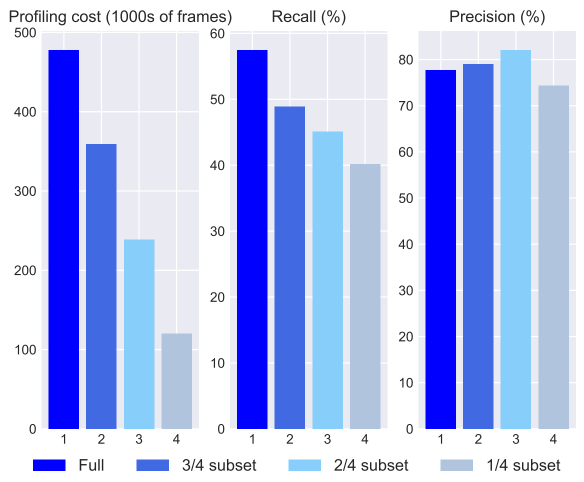


Figure 4.9: Offline profiling cost vs. online accuracy. Profile intervals compared (in minutes of data used per camera): 16.6 min. (full), 12.5 min., 8.3 min. (half), 4.15 min.

As **Figure 4.9** indicates, there is a clear trade-off between cost and tracking recall (accuracy). Our default setting, from **ReXCam-O**, is to run the MTMC tracker on the full 0.48 million frames that comprise the `trainval-mini` partition of the Duke dataset (intended for offline profiling), which results in 1.19 million labeled detections. Building the spatio-temporal model M from these detections results in tracking recall of 57.5%.

This figure declines steadily, from 57.5% to 40.2%, as we confine our MTMC tracker to label first three-fourths, then one-half, and finally one-fourth of the full 0.48 million frames in the `trainval-mini` partition. In general, we see that tracking recall varies roughly linearly with profiling cost. On the other hand, precision shows no clear trend (besides its initial rise), fluctuating between 74% and 82%.

End-to-end evaluation

The profiling cost figures from Section 4.8 allow us to evaluate the end-to-end gains achieved with spatio-temporal filtering. To obtain ReXCam-O's accuracy numbers requires us to process 0.48 million frames. Running ReXCam saves us on average 830 detections *per query* at inference time (see Figure 4.7), compared to the all-camera baseline, which requires no offline profiling. At that rate, *ReXCam would need to run 580 live tracking queries to break-even with locality-agnostic tracking.* This represents a small fraction of the expected annual workload in large video analytics operations (e.g. the Chicago Police Department's surveillance system [15]) [100, 99].

4.9 Related Work

Systems for Video Analytics

Since 2016, a sizable body of work on video analytics has emerged in the systems and data management community. We briefly survey five key papers. Optasia parallelizes video query plans and de-duplicates the work of common modules (e.g. background subtraction) in a dataflow framework to improve query completion time and reduce resource usage [65]. VideoStorm investigates the variance in quality-lag requirements between common video analytics queries (e.g. scanning license plates for billing on toll routes vs. issuing AMBER Alerts), and proposes an offline profiler and online scheduler for their optimal execution [99]. NoScope accepts specialized queries (e.g. "find all frames with buses in the Taipei feed"), and constructs a model cascade exploiting difference detectors and specialized models to achieve speedups on most inputs [55]. Focus invokes object clustering and low-cost models to cheaply index video at ingest time, and thereby support low-latency, after-the-fact queries on historical video [45]. Chameleon exploits correlations in camera content (e.g. velocity and sizes of detected objects) to amortize profiling costs across cameras over time [54].

This preceding work leaves three key problem areas unexplored.

First, all of these papers focus on single-frame analytics tasks (e.g. license plate recognition, binary frame classification, object detection), which are stateless and easily parallelized. In contrast, many real surveillance applications involve interactive or long-running queries (e.g. multi-frame tracking), where future questions are dependent on past inference results. In particular, ReXCam executes real-time tracking, a task that is difficult to parallelize or pipeline, and entails compounding classification errors (i.e. misidentifying a person at time t affects all future tracking behavior).

Second, all of these papers study single-*camera* analytics tasks. While some propose joint execution plans and shared profiling [65, 54] as means to reduce redundant work, none explore the complexities involved in cross-camera inference (e.g. occlusions, perspective shifts) or collaborative execution (e.g. intermediate state sharing), a defining component of many key applications, such as person re-identification. Moreover, none model the dynamics of the camera network itself to inform future inference decisions, as ReXCam does by profiling cross-camera traffic patterns.

Third, all focus on standard classification tasks, where objects or activities of interest at inference time fall neatly into classes seen at training time. In contrast, many real security applications involve searching for new object instances (e.g. a suspicious person), or detecting highly anomalous behavior (e.g. a bomb setup), given training data skewed overwhelmingly toward negative examples. ReXCam focuses exclusively on tasks for the first type, termed *instance retrieval*, on which current techniques achieve low precision [89]. This is because, in large datasets, many detected entities tend to match against the query identity, of which there are only a few instances, and which was likely never seen at training time. Given this property, systems level insights – such as our observation that cameras tend to share traffic with only a small set of neighboring nodes – can yield particularly substantial accuracy gains.

Efficient Machine Learning

Techniques in machine learning that addresses the bandwidth, memory, and compute costs of deploying large models fall largely into two categories: (1) training-time optimizations and (2) inference-time optimizations.

Training-time optimizations can be broadly characterized as efforts to *compress* accurate but expensive models. Proposed techniques include parameter and filter pruning [43, 59], compact architecture design [47, 63], knowledge distillation [44, 40, 6], and model specialization [55, 45]. In general, these techniques are orthogonal to ReXCam, which would gain from any reduction in the inference cost or memory size of the models it deploys.

Work on efficient inference generally aims to address a constrained optimization problem: maximize accuracy or minimize resource usage, given specific constraints on resource availability or latency (i.e. SLOs). Prior work explores resource-aware scheduling [42, 29], low-latency prediction serving [23], edge-cloud compute partitioning [100, 21], hardware-specific optimizations [46, 68], and multi-tenant resource sharing [61, 53]. Unlike these systems, ReXCam does not aim to multiplex heterogeneous models, nor does it espouse a particular compute model (e.g. mobile, edge-cloud hybrid). Instead, ReXCam entails a new approach altogether to reducing resource usage: instead of operating cheaper models, run inference on *less data*. Its mechanism for doing so is to exploit spatial and temporal locality in the data source (the camera network).

Computer Vision

ReXCam is most closely related to computer vision literature on person re-identification and multi-target, multi-camera (MTMC) tracking. Papers in this area generally make one of three types of contributions: (1) new datasets [79, 104, 92, 89], (2) new neural network architectures [104, 92, 89], or (3) new training schemes [78, 104, 92].

Examples of new architectures include networks for joint detection and re-identification [104, 92], and networks that enable better generalization (i.e. transfer learning) to new datasets [89]. Examples of new training schemes include new loss functions [78, 92] and new data sampling techniques (e.g. hard-identity mining [78], confidence weighting [104]).

In general, the vision literature does not address the inference cost of re-identification and MTMC tracking, nor does it study *online* tracking, a key application of interest in real surveillance systems. While prior work has explored the use of network topology information to improve tracking accuracy, it has generally confined itself to explicitly learning epipolar geometry in offline settings with classical vision techniques [51, 13, 58, 67, 14].

Visual Data Management

A body of work also exists in the data management space on storing querying content in image and video databases. These systems explore the use of classical computer vision techniques (e.g. clustering by low-level features, such as color and texture) to index image and video efficiently, and focus on relational, historical data stores [72, 35, 7, 16, 73]. We build on this tradition, revisiting large-scale visual analytics in the context of *cross-camera inference* on *live video* with *modern computer vision* (e.g. deep learning-based) *techniques*, a setting that entails substantially different challenges than the target domain of older work.

4.10 Conclusions

Cross-camera analytics is a capability that underpins a range of real video analytics applications, from public safety monitoring and suspect tracking to intelligent retail and automated checkout. To address the cost challenges of processing every raw video frame in a large camera deployment, we present ReXCam, an efficient cross-analytics video analytics system that leverages a *learned model of cross-camera correlations* to drastically reduce the size of the inference time search space. ReXCam builds this model on unlabeled video data, by aggregating data on cross-camera traffic patterns into spatial and temporal filters. In the case of occasional missed detections, ReXCam performs a fast-replay search to uncover skipped detections on recently filtered frames. Combining these techniques, ReXCam is able to reduce compute workload by 4.6$\times$ and improve inference precision by 27% on an eight camera dataset, while maintaining with 1-2% of the recall of a locality-agnostic baseline.

Chapter 5

Conclusion

In this thesis, I discuss two lines of research: high-throughput perception on video streams and low-cost analytics on multiple live video feeds. These two bodies of work are united by a common theme: the efficient execution of deep convolutional neural networks on video.

On the perception side, I leverage temporal structure in video to reuse intermediate frame representations, and thereby amortize feature computation cost across frames. On the analytics side, I leverage spatial and temporal correlations in video networks to carefully constrain the scope of queries at inference time, thereby substantially reducing workload.

At a high level, work on performance-oriented computer vision is motivated by new application requirements. The control module in an autonomous vehicle demands detailed object and scene annotations from the upstream perception module, at high temporal resolution. This necessitates high-frame rate object detection, scene segmentation, and prediction. In a similar vein, cross-camera object tracking requires efficiently scanning large volumes of incoming live video. This necessitates low-latency inference, but also low, per-camera compute cost, to support execution on large camera networks.

This thesis, which addresses the joint challenges of high accuracy and high performance, represents a step toward realizing these and other emerging computer vision applications.

Bibliography

[1] *Absolutely everywhere in Beijing is now covered by police video surveillance.* https://qz.com/518874/. Accessed: 2018-10-27.

[2] *Amazon EC2 P2 Instances.* https://aws.amazon.com/ec2/instance-types/p2/. Accessed: 2018-10-27.

[3] *Amazon Go.* https://www.cnn.com/2018/10/03/tech/amazon-go/index.html. Accessed: 2018-10-27.

[4] *AMBER Alert.* https://en.wikipedia.org/wiki/AMBER_Alert. Accessed: 2018-10-27.

[5] *An explosion of online video could triple bandwidth consumption again in the next five years.* https://www.recode.net/2017/6/8/15757594/future-internet-traffic-watch-live-video-facebook-google-netflix. Accessed: 2019-04-28.

[6] Rohan Anil et al. "Large scale distributed neural network training through online distillation". In: *ICLR*. 2018.

[7] Walid Aref et al. "Video query processing in the VDBMS testbed for video database research". In: *ACM MMDB*. 2003.

[8] Yusuf Aytar. *PASCAL VOC Challenge performance evaluation and download server.* http://host.robots.ox.ac.uk:8080/leaderboard. Accessed: 2018-03-06. 2018.

[9] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *TPAMI*. 2017.

[10] Piotr Bilinski and Victor Prisacariu. "Dense Decoder Shortcut Connections for Single-Pass Semantic Segmentation". In: *CVPR*. 2018.

[11] *British Transport Police: CCTV.* http://www.btp.police.uk/advice_and_information/safety_on_and_near_the_railway/cctv.aspx. Accessed: 2018-10-27.

[12] Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla. "Semantic Object Classes in Video: A High-Definition Ground Truth Database". In: *Pattern Recognition Letters* 30.2 (2009), 8897.

[13] Yinghao Cai and Grard Medioni. "Exploring Context Information for Inter-Camera Multiple Target Tracking". In: *IEEE WACV*. 2014.

[14] Simone Calderara, Rita Cucchiara, and Andrea Prati. “Bayesian-Competitive Consistent Labeling for People Surveillance”. In: *TPAMI* 30 (2 2007).

[15] *Can 30,000 Cameras Help Solve Chicago’s Crime Problem?* `https://www.nytimes.com/2018/05/26/us/chicago-police-surveillance.html`. Accessed: 2018-10-27.

[16] Marco La Cascia and Edoardo Ardizzone. “Jacob: Just a content-based query system for video databases”. In: *IEEE ICASSP*. 1996.

[17] Abhishek Chaurasia and Eugenio Culurciello. “LinkNet: Exploiting Encoder Representations For Efficient Semantic Segmentation”. In: *CVPR*. 2018.

[18] Liang-Chieh Chen et al. “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs”. In: *PAMI*. 2017.

[19] Liang-Chieh Chen et al. “Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs”. In: *ICLR*. 2016.

[20] Tianqi Chen et al. “MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems”. In: *NIPS Workshop on Machine Learning Systems*. 2016.

[21] Tiffany Yu-han Chen et al. “Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices Categories and Subject Descriptors”. In: *ACM SenSys*. 2015.

[22] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *CVPR*. 2016.

[23] Daniel Crankshaw et al. “Clipper: A Low-Latency Online Prediction Serving System”. In: *USENIX NSDI*. 2017.

[24] *Cross-camera person tracker*. `https://github.com/SamvitJ/Deep-Person-ReId`. Accessed: 2018-10-28.

[25] Jifeng Dai et al. “Deformable Convolutional Networks”. In: *ICCV*. 2017.

[26] *Deep Person Reid*. `https://github.com/KaiyangZhou/deep-person-reid/`. Accessed: 2018-10-28.

[27] *DukeMTMC-reID*. `https://github.com/layumi/DukeMTMC-reID_evaluation`. Accessed: 2018-10-28.

[28] Mark Everingham et al. “The Pascal Visual Object Classes Challenge: A Retrospective”. In: *IJCV* 111.1 (2015), pp. 98–136.

[29] Biyi Fang, Xiao Zeng, and Mi Zhang. “NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision”. In: *ACM MobiCom*. 2018.

[30] Christoph Feichtenhofer, Axel Pinz, and Richard P. Wildes. “Spatiotemporal Residual Networks for Video Action Recognition”. In: *NIPS*. 2016.

[31] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. “Convolutional Two-Stream Network Fusion for Video Action Recognition”. In: *CVPR*. 2016.

[32] Christoph Feichtenhofer et al. "What have we learned from deep representations for action recognition?" In: *CVPR*. 2018.

[33] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. "Efficient graph-based image segmentation". In: *IJCV* 59.2 (2004), pp. 167–181.

[34] Phillip Fischer et al. "FlowNet: Learning Optical Flow with Convolutional Networks". In: *ICCV*. 2015.

[35] Myron Flickner et al. "Query by Image and Video Content: The QBIC System". In: *Computer* 28 (9 1995).

[36] Katerina Fragkiadaki et al. "Learning to segment moving objects in videos". In: *CVPR*. 2015.

[37] Raghudeep Gadde, Varun Jampani, and Peter V. Gehler. "Semantic Video CNNs through Representation Warping". In: *ICCV*. 2017.

[38] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* 32.11 (2013).

[39] *Genetec Announces Airport Sense*. `https://www.genetec.com/about-us/news/press-center/press-releases/`. Accessed: 2018-10-30.

[40] Urban Gregor et al. "Do Deep Convolutional Nets Really Need To Be Deep And Convolutional?" In: *ICLR*. 2017.

[41] Matthias Grundmann et al. "Efficient Hierarchical Graph-Based Video Segmentation". In: *CVPR*. 2010.

[42] Seungyeop Han et al. "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints". In: *ACM MobiSys*. 2016.

[43] Song Han, Huizi Mao, and William J. Dally. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". In: *ICLR*. 2016.

[44] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the Knowledge in a Neural Network". In: *NIPS*. 2014.

[45] Kevin Hseih et al. "Focus: Querying Large Video Datasets with Low Latency and Low Cost". In: *USENIX OSDI*. 2018.

[46] Loc N Huynh, Youngki Lee, and Rajesh K Balan. "DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications". In: *ACM MobiSys*. 2017.

[47] Forrest N. Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size". In: *arXiv:1602.07360* (2016).

[48] Max Jaderberg et al. "Spatial Transformer Networks". In: *NIPS*. 2015.

[49] Samvit Jain and Joseph Gonzalez. “Fast Semantic Segmentation on Video Using Block Motion-Based Feature Interpolation”. In: *ECCV Workshop on Video Segmentation*. 2018.

[50] Suyog Dutt Jain, Bo Xiong, and Kristen Grauman. “FusionSeg: Learning to combine motion and appearance for fully automatic segmentation of generic objects in videos”. In: *CVPR*. 2017.

[51] Omar Javed et al. “Modeling inter-camera space-time and appearance relationships for tracking across non-overlapping views”. In: *CVIU* 109 (2 2008).

[52] Simon Jégou et al. “The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation”. In: *CVPR Workshop on Computer Vision in Vehicle Technology*. 2017.

[53] Angela H Jiang et al. “Mainstream: Dynamic Stem-Sharing for Multi-Tenant Video Processing”. In: *USENIX ATC*. 2018.

[54] Junchen Jiang et al. “Chameleon: Video Analytics at Scale via Adaptive Configurations and Cross-Camera Correlations”. In: *ACM SIGCOMM*. 2018.

[55] Daniel Kang et al. “NoScope: Optimizing Neural Network Queries over Video at Scale”. In: *VLDB*. 2017.

[56] Andrej Karpathy et al. “Large-scale Video Classification with Convolutional Neural Networks”. In: *CVPR*. 2014.

[57] Philipp Krähenbühl and Vladlen Koltun. “Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials”. In: *NIPS*. 2011.

[58] Cheng-Hao Kuo, Chang Huang, and Ram Nevatia. “Inter-camera Association of Multi-target Tracks by On-Line Learned Appearance Affinity Models”. In: *ECCV*. 2010.

[59] Hao Li et al. “Pruning Filters for Efficient ConvNets”. In: *ICLR*. 2017.

[60] Yule Li, Jianping Shi, and Dahua Lin. “Low-Latency Video Semantic Segmentation”. In: *CVPR*. 2018.

[61] Robert LiKamWa and Lin Zhong. “Starfish: Efficient Concurrency Support for Computer Vision Applications”. In: *ACM MobiSys*. 2015.

[62] Guosheng Lin et al. “RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation”. In: *CVPR*. 2017.

[63] Min Lin, Qiang Chen, and Shuicheng Yan. “Network In Network”. In: *ICLR*. 2014.

[64] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *CVPR*. 2015.

[65] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. “Optasia: A Relational Platform for Efficient Large-Scale Video Analytics”. In: *ACM SoCC*. 2016.

[66] Behrooz Mahasseni, Sinisa Todorovic, and Alan Fern. "Budget-Aware Deep Semantic Video Segmentation". In: *CVPR*. 2017.

[67] Dimitrios Makris, Tim Ellis, and James Black. "Bridging the gaps between cameras". In: *CVPR*. 2004.

[68] Akhil Mathur et al. "DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models Using Wearable Commodity Hardware". In: *ACM MobiSys*. 2017.

[69] *Multi-Target, Multi-Camera Tracking*. https://github.com/ergysr/DeepCC. Accessed: 2018-10-28.

[70] Naveen Shankar Nagaraja, Frank R. Schmidt, and Thomas Brox. "Video Segmentation with Just a Few Strokes". In: *CVPR*. 2015.

[71] David Nilsson and Cristian Sminchisescu. "Semantic Video Segmentation by Gated Recurrent Flow Propagation". In: *CVPR*. 2018.

[72] Virginia E. Ogle and Michael Stonebraker. "Chabot: Retrieval from a relational database of images". In: *Computer* 28 (9 1995).

[73] JungHwan Oh and Kien A. Hua. "Efficient and cost-effective techniques for browsing and indexing large video databases". In: *ACM SIGMOD*. 2000.

[74] Anestis Papazoglou and Vittorio Ferrari. "Fast object segmentation in unconstrained video". In: *ICCV*. 2013.

[75] *Paris hospitals to get 1,500 CCTV cameras to combat violence against staff*. https://bit.ly/2OYiBz2. Accessed: 2018-10-27.

[76] *Pricing - Linux Virtual Machines*. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/. Accessed: 2018-10-27.

[77] Iain E. Richardson. *H.264 and MPEG-4 video compression: video coding for next-generation multimedia*. Wiley, 2008.

[78] Ergys Ristani and Carlo Tomasi. "Features for Multi-Target Multi-Camera Tracking and Re-Identification". In: *IEEE CVPR*. 2018.

[79] Ergys Ristani et al. "Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking". In: *ECCV Workshops*. 2016.

[80] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *MICCAI*. 2015.

[81] Evan Shelhamer et al. "Clockwork Convnets for Video Semantic Segmentation". In: *Video Semantic Segmentation Workshop at ECCV*. 2016.

[82] Jamie Shotton et al. "TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context". In: *IJCV* 81.1 (2009), pp. 2–23.

[83] Karen Simonyan and Andrew Zisserman. "Two-Stream Convolutional Networks for Action Recognition in Videos". In: *NIPS*. 2014.

[84] *State of the Art Security Cameras Installed Ahead of Schedule, Under Budget at 14 Schools*. https://cps.edu/News/Press_releases/Pages/04_05_2012_PR1.aspx. Accessed: 2018-10-27.

[85] Paul Sturgess et al. "Combining Appearance and Structure from Motion Features for Road Scene Understanding". In: *BMVC*. 2009.

[86] Pavel Tokmakov, Karteek Alahari, and Cordelia Schmid. "Learning Video Object Segmentation with Visual Memory". In: *CVPR*. 2017.

[87] Yi-Hsuan Tsai, Ming-Hsuan Yang, and Michael J. Black. "Video Segmentation via Object Flow". In: *CVPR*. 2016.

[88] *Video meets the Internet of Things*. https://www.mckinsey.com/industries/high-tech/our-insights/video-meets-the-internet-of-things. Accessed: 2018-10-28.

[89] Longhui Wei et al. "Person Trasfer GAN to Bridge Domain Gap for Person Re-Identification". In: *IEEE CVPR*. 2018.

[90] Chao-Yuan Wu et al. "Compressed Video Action Recognition". In: *CVPR*. 2018.

[91] Zuxuan Wu et al. "Modeling Spatial-Temporal Clues in a Hybrid Deep Learning Framework for Video Classification". In: *ACM-MM*. 2015.

[92] Tong Xiao et al. "Joint Detection and Identification Feature Learning for Person Search". In: *IEEE CVPR*. 2017.

[93] Yu-Syuan Xu et al. "Dynamic Video Segmentation Network". In: *CVPR*. 2018.

[94] Maoke Yang et al. "DenseASPP for Semantic Segmentation in Street Scenes". In: *CVPR*. 2018.

[95] *You're being watched: there's one CCTV camera for every 32 people in UK*. https://www.theguardian.com/uk/2011/mar/02/cctv-cameras-watching-surveillancel. Accessed: 2018-10-27.

[96] Fisher Yu and Vladlen Koltun. "Multi-scale Context Aggregation by Dilated Convolutions". In: *ICLR*. 2016.

[97] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. "Dilated Residual Networks". In: *CVPR*. 2017.

[98] Bowen Zhang et al. "Real-time Action Recognition with Enhanced Motion Vector CNNs". In: *CVPR*. 2016.

[99] Haoyu Zhang et al. "Live Video Analytics at Scale with Approximation and Delay-Tolerance". In: *USENIX NSDI*. 2017.

[100] Tan Zhang et al. "The Design and Implementation of a Wireless Video Surveillance System". In: *ACM MobiCom*. 2015.

[101] Zhimeng Zhang et al. "Multi-Target, Multi-Camera Tracking by Hierarchical Clustering: Recent Progress on DukeMTMC Project". In: *arXiv:1712.09531* (2017).

[102] Hengshuang Zhao et al. “Pyramid Scene Parsing Network”. In: *CVPR*. 2017.

[103] Liang Zheng, Yi Yang, and Alexander G Hauptmann. “Person Re-identification: Past, Present and Future”. In: *arXiv:1610.02984* (2015).

[104] Liang Zheng et al. “Person Re-identification in the Wild”. In: *IEEE CVPR*. 2017.

[105] Xizhou Zhu et al. “Deep Feature Flow for Video Recognition”. In: *CVPR*. 2017.

[106] Xizhou Zhu et al. “Toward High Performance Video Object Detection”. In: *CVPR*. 2018.