# Dynamic Stem-Sharing for Multi-Tenant Video Processing

Angela Jiang, Christopher Canel, Daniel Wong, Michael Kaminsky, Michael A. Kozuch,
Padmanabhan Pillai, David G. Andersen, Gregory R. Ganger
Carnegie Mellon University, Intel Labs

## 1 INTRODUCTION

Video cameras are ubiquitous, and their outputs are increasingly analyzed by sophisticated, online DNN inference-based applications. The ever-growing capabilities of video and image analysis techniques create new possibilities for what may be gleaned from any given video stream. Consequently, most raw video streams will be processed by multiple analysis pipelines. For example, a parking lot camera might be used by three different applications: reporting open parking spots, tracking each car's parking duration for billing, and recording any fender benders.

In this paper, we focus on shared processing on edge devices; processing video near the camera addresses issues such as bandwidth, intermittent connectivity (e.g., in drones), and real-time requirements, but leads to resource limitations. Thus, optimal video application performance requires tuning to the resources available [13, 2, 14, 4, 7]. However, application developers may be unable to predict easily what resources will be available when the application is deployed, particularly in "multi-tenant" environments where the set of concurrently deployed applications may vary. Instead, individual application developers typically develop their models in isolation, assuming either infinite resources or a predetermined set of static resources. When a number of such individually-tailored models are run concurrently, resource competition forces the video stream to be analyzed at a lower frame rate— leading to unsatisfactory results for the running applications, as frames are dropped and events in those frames are missed.

The *Mainstream* video processing system enables efficient execution of multiple independently-developed and incrementally-deployed video analysis applications on a given video stream. Mainstream shares execution of concurrent DNNs, yet does not rely on applications' DNNs to be trained collectively. Therefore, Mainstream provides collaborative execution, even when development and training data are not centralized in one organization.

**Dynamic Model Selection and Stem Sharing.** Mainstream moves the final DNN model selection step from application development time to deployment time, when the hardware resources and concurrent application mix are known. By doing so, Mainstream can select the best DNN for the resources available. Moreover, it can coordinate the selections to share DNN "stems", where doing so leads to greater aggregate application quality by reducing aggregate computational load. [1] For a given input, the shared stem need only be executed once [4]. While a less specialized DNN may provide worse accuracy than a fully-specialized DNN, it allows for stem-sharing with concurrent applications, leading to lower resource contention. Thus, constraining concurrent analytics applications to share parts of their machine learning models can actually improve the overall accuracy of a mix of applications.

To enable the selection of DNN models at deployment time, Mainstream requires an additional step in the application development process. Currently, developers experiment with different model types, hyperparameters, and degrees of re-training/fine-tuning to find the best choice for an assumed resource allocation, discarding the trained DNN models not chosen. With Mainstream, a number of *candidate* DNN models are kept and provided to the runtime system. In particular, for a given base model (e.g., InceptionV3 trained on ImageNet), candidate DNNs with different numbers of re-trained layers are provided. This allows Mainstream to select at deployment time— for a set of concurrent applications— the amount of stem sharing that maximizes application quality within the resource availability. At the same time, it avoids requiring that developers yield control over their training data and processes, as would be necessary for joint training of a multi-task DNN.

Experiments with a Mainstream deployment show that through its dynamic selection of shared stems, Mainstream delivers as much as a 93% higher F1-score, relative to the common approach of using fully-independent per-application DNNs (No-sharing) and up to 61% higher F1-score than a static approach of retraining only the last DNN layer and sharing all others (Max-sharing).

## 2 DESIGN OF *MAINSTREAM*

Mainstream consists of three components: M-Trainer, M-Planner, and M-Runner. To provide dynamism, M-Trainer trains a variety of models, each with different amounts of sharing. These are stored, so at runtime, as resource availability fluctuates, Mainstream can adapt the amount of sharing between applications.

### 2.1 Decentralized Training

**Fine-tuning Networks.** Developing a new DNN model by fine-tuning an existing "pre-trained" DNN model is a practical alternative to training from scratch. During fine-tuning, a subset of the pre-trained model weights (layers near the DNN input) are held frozen and not permitted to change. The remaining free parameters are then retrained for the given task with a new dataset. Fine-tuning by using one of a few popular neural networks is standard practice to reduce training time. However, this technique also provides an opportunity for sharing computation: given two models fine-tuned from the same base model, whatever layers were held frozen during training will have the same weights, will produce identical results for a given frame, and can be shared at inference time. Thus, models need not be *jointly* trained, they may be fine-tuned independently using the same base DNN model.

**Adaptive Control of the Sharing Opportunity.** Figure 1 shows the relationship between specialization and Top-1 accuracy for three different DNN architectures and three classification tasks.

[1] Common DNN stems arise when application developers use *transfer learning* and *fine tuning* [9, 12, 8, 10] to speed model development and address training data limitations, wherein only a subset of a previously-trained DNN's layers are re-trained (fine-tuned) to the new application's task—the common non-retrained layers of two applications would be a shared stem.

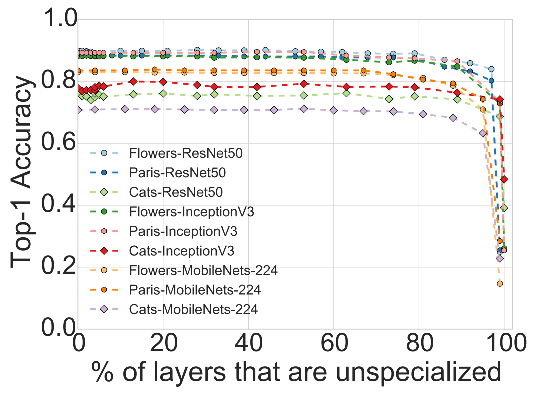

**Figure 1: Per-frame classification accuracy vs. (Potential) sharing for three different DNN networks (ResNet50 [5], InceptionV3 [11], and MobileNets [6]), each fine-tuned with three different data sets extracted from the ImageNet [3] database. The base DNN model was also trained using ImageNet. Note that approximately 80% of the model may remain unspecialized without significantly affecting accuracy; in other words, 80% of the computation may be shared.**

As more of the network is specialized, Top-1 accuracy increases, but less sharing is enabled. M-Trainer cannot anticipate how much sharing M-Scheduler will recommend. Therefore, M-Trainer trains several model *candidates*, each with a different number of layers held frozen. These models are stored so that at deployment time, M-Scheduler can determine the optimal degree of sharing to use.

**Training SDK.** M-Trainer is responsible for generating the models and metadata needed to deploy an application to M-Scheduler. To retain the privacy of users' data, M-Trainer is designed as an SDK that may be run under the developer's control. For a given task, M-Trainer generates a *Model Set* of trained model candidates in the context of some base model (e.g., InceptionV3 pre-trained with ImageNet). For each base model, M-Trainer has a set of *branchpoints* (essentially layer numbers) to which the application DNN model will be fine-tuned to generate a model candidate. For a model candidate trained via a branchpoint at layer $l$, the layers from 0 to $l$ will represent the (unspecialized) *stem* which may be shared with other tasks. Additionally, for each candidate in the Model Set, M-Trainer estimates the accuracy of that model using an input validation set. The Model Set and corresponding accuracies constitute the *M-Package*.

**Training Costs.** For each application, M-Trainer trains multiple models. While training a model from scratch can be expensive, fine-tuning is less costly. In our study, M-Trainer creates Model Sets with 15 model candidates in 8 hours on a single GPU.

## 2.2 Dynamic Scheduling

During scheduling, M-Scheduler must choose a model candidate from each application's M-Package to implement that task. Each candidate corresponds to a particular accuracy and degree of sharing, and more sharing enables a higher frame rate of analysis, which in turn leads to better end-to-end application quality. M-Scheduler chooses the various model candidates to produce an overall schedule that optimizes some objective function (e.g., maximize average F1-score across applications).

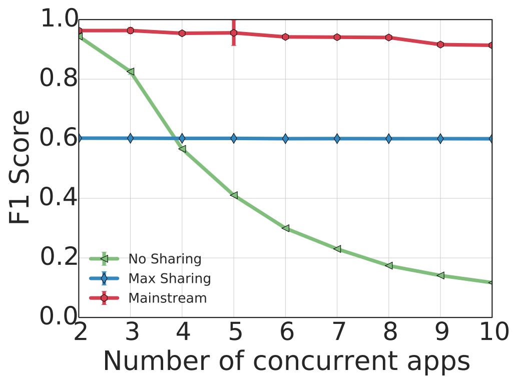

**Figure 2: Mainstream improves application quality (average F1-score) relative to both (a) no sharing between applications (No-sharing) *and* (b) sharing all layers but the last one (Max-sharing).**

The optimal schedule is dictated by the set of applications to be scheduled and the resources available. M-Scheduler models resource usage with a latency-based cost. Specializing more layers in an application incurs a cost but improves utility. Using observations from earlier rounds of deployment, M-Scheduler determines the "cost budget" (a measure of the available resources) and uses a greedy approach to find the optimal schedule that fits within this budget. While the space of possible schedules is combinatorically large, M-Scheduler efficiently navigates it with a cost-benefit-based heuristic. M-Scheduler starts with the least costly schedule: Max-sharing (applications share all but the last layer). It continually refines this schedule by making one application share fewer resources (use more fine-tuned layers) until there are no more objective-improving moves, or no more moves that are feasible under the cost budget.

## 3 EVALUATION

We evaluate our system on a a near-edge compute node shared by up to 10 independent DNN-based video processing applications, and we find that Mainstream improves result quality relative to both the predominant No-sharing and hypothetical Max-sharing approaches. See Fig. 2.

**Benefit to F1-Score.** The tested applications use image classifiers to detect events on a 14FPS video stream. M-Planner maximizes application quality (end-to-end F1-score measured as harmonic mean of event precision and event recall) by varying the amount of sharing. As applications are added, resource contention increases—forcing Mainstream to pick a different balance between accuracy and frame rate. Mainstream delivers as much as a 93% higher F1-Score than No-sharing and as much as 61% higher than Max-sharing. No-sharing exhibits low recall due to its low throughput (approx. 1.4FPS for 10 apps)—the system has fewer opportunities to detect the event. Max-sharing has higher throughput (over 12FPS for 10 apps) but a worse recall due to low model accuracy (70% lower than fully-tuned). Mainstream strikes a balance between accuracy and frame rate depending on load (choosing 8.7FPS with an accuracy 18% lower than fully-tuned for 10 apps). Overall, Mainstream is able to efficiently use limited edge compute resources by sharing computation among multiple concurrent DNN tasks, yet allows these tasks to be independently developed, trained, and deployed.

## REFERENCES

[1] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. 2017. Real-time video analytics: the killer app for edge computing. *Computer*, 50, 10, 58–67. ISSN: 0018-9162. DOI: 10.1109/MC.2017.3641638.

[2] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: a low-latency online prediction serving system. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*. USENIX Association, Boston, MA. ISBN: 978-1-931971-37-9. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *Cvpr09*.

[4] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints. In *Conference mobisys'16 the 14th annual international conference on mobile systems, applications, and services* (MobieSys '16). ACM.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 770–778.

[6] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: efficient convolutional neural networks for mobile vision applications. *Corr*, abs/1704.04861. http://arxiv.org/abs/1704.04861.

[7] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. Noscope: optimizing neural network queries over video at scale. In *Proceedings of the vldb endowment, vol. 10, no. 11*.

[8] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. 2014. Learning and transferring mid-level image representations using convolutional neural networks. In *The ieee conference on computer vision and pattern recognition (cvpr)*. (June 2014).

[9] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. 2014. CNN features off-the-shelf: an astounding baseline for recognition. In *IEEE conference on computer vision and pattern recognition, CVPR workshops 2014, columbus, oh, usa, june 23-28, 2014*, 512–519. DOI: 10.1109/CVPRW.2014.131. http://dx.doi.org/10.1109/CVPRW.2014.131.

[10] Karen Simonyan and Andrew Zisserman. 2014. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, 568–576.

[11] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the inception architecture for computer vision. *Corr*, abs/1512.00567. arXiv: 1512.00567. http://arxiv.org/abs/1512.00567.

[12] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? In *Proceedings of the 27th international conference on neural information processing systems* (NIPS'14). MIT Press, Montreal, Canada, 3320–3328. http://dl.acm.org/citation.cfm?id=2969033.2969197.

[13] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)* (NSDI '17). Boston, MA, 16 pages.

[14] Shlomo Zilberstein. 1996. Using anytime algorithms in intelligent systems. In *Ai magazine, 17(3):73-83*.