Mainstream: Dynamic Stem-Sharing for Multi-Tenant Video Processing

Angela H. Jiang, Daniel L.K. Wong, Christopher Canel, Lilia Tang, and Ishan Misra, *Carnegie Mellon University;* **Michael Kaminsky, Michael A. Kozuch, and Padmanabhan Pillai,** *Intel Labs;* **David G. Andersen and Gregory R. Ganger,** *Carnegie Mellon University*

https://www.usenix.org/conference/atc18/presentation/jiang

This paper is included in the Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC '18).

July 11–13, 2018 • Boston, MA, USA

ISBN 978-1-939133-02-1

Open access to the Proceedings of the 2018 USENIX Annual Technical Conference is sponsored by USENIX.

Mainstream: Dynamic Stem-Sharing for Multi-Tenant Video Processing

Angela H. Jiang, Daniel L.-K. Wong, Christopher Canel, Lilia Tang, Ishan Misra,
Michael Kaminsky[†], Michael A. Kozuch[†], Padmanabhan Pillai[†],
David G. Andersen, Gregory R. Ganger
Carnegie Mellon University; [†]*Intel Labs*

Abstract

Mainstream is a new video analysis system that jointly adapts concurrent applications sharing fixed edge resources to maximize aggregate result quality. Mainstream exploits partial-DNN (deep neural network) compute sharing among applications trained through transfer learning from a common base DNN model, decreasing aggregate per-frame compute time. Based on the available resources and mix of applications running on an edge node, Mainstream automatically determines at deployment time the right trade-off between using more specialized DNNs to improve per-frame accuracy, and keeping more of the unspecialized base model to increase sharing and process more frames per second. Experiments with several datasets and event detection tasks on an edge node confirm that Mainstream improves mean event detection F1-scores by up to 47% relative to a static approach of retraining only the last DNN layer and sharing all others ("Max-Sharing") and by 87X relative to the common approach of using fully independent per-application DNNs ("No-Sharing").

1 Introduction

Video cameras are ubiquitous, and their outputs are increasingly analyzed by sophisticated, online deep neural network (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.

This paper focuses on video processing on edge devices, which will be a common way to address bandwidth limitations, intermittent connectivity (e.g., in drones), and real-time requirements. Applications executing at the edge, though, face tighter bounds on resource availability than in datacenters. Naturally, optimal video application performance requires tuning for the resources available [48, 12, 51, 18, 26].

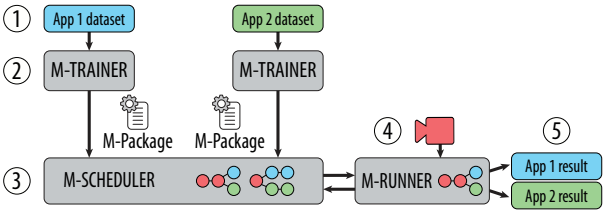


Figure 1: Mainstream Architecture. Offline, for each task, M-Trainer takes a labeled dataset and outputs an M-Package. M-Scheduler takes independently generated M-Packages, and chooses the task-specific degree of specialization and frame rate. M-Scheduler deploys the unified multi-task model to M-Runner, performing inference on the edge.

Unfortunately, what resources will be available to the application at deployment time is often unknown to the developer. Further, resource availability changes as additional applications arrive and depart. Instead, individual application developers typically develop their models in isolation, assuming either infinite resources or a predetermined resource allotment. When a number of separately tuned 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. However, due to the popularity of *transfer learning* (Sec. 2) [40, 47, 37, 43], contention can be reduced by eliminating redundant computation between concurrent applications [18].

Mainstream is a new system for video processing that addresses resource contention by dynamically tuning degrees of work sharing among concurrent applications. Specifically, it focuses on sharing portions of DNN inference, which consumes the majority of video processing cycles. Mainstream exploits the potential "shared stem" of computation that results from application developers' use of the standard DNN training approach of transfer learning. In transfer learning, training begins with an existing, *pre-trained* DNN, which is then re-trained for a different task. Typically, only a subset of the pre-trained DNN is specialized; when different applications start

with the same pre-trained DNN, Mainstream identifies the common layers and executes them only once per frame.

A critical challenge of exploiting shared stems well is determining how much to share. Application developers usually specialize as much of the pre-trained DNN as is necessary to achieve high model accuracy. More specialization, however, means that less of the pre-trained DNN can be shared. Thus, there is an explicit trade-off between the benefits of greater per-frame accuracy (via more-specialized DNNs) and processing more frames of the input video stream (via more sharing of less-specialized DNNs). The right choice depends on the edge device resources, the number of concurrent applications, and their individual characteristics.

Deployment time model selection. 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 has the necessary information to select the right amount of DNN specialization (and thus sharing) for each application. As applications come and go, Mainstream can dynamically modify its choices. Previous systems like VideoStorm [48] select models by considering each application independently. *The specialization-vs-sharing trade-off, however, can only be optimized when considering applications jointly.* Joint optimization produces a combinatorial set of options, which Mainstream navigates using application metadata and domain-specific models; the system uses this information to estimate the effects of DNN specialization and frame sampling rate on application performance. Unlike black-box approaches, Mainstream can jointly optimize for stem-sharing without needing to profile each combination.

Mainstream consists of three main parts (Fig. 1). The *M-Trainer* toolkit helps application developers manage their training process to produce the information needed to allow tuning the degree of specialization at deployment time. Current standard practice is for developers to experiment with different model types, hyperparameters, and degrees of re-training to find the best choice for an assumed resource allocation, discarding the trained DNN models not chosen. M-Trainer instead keeps "less optimal" *candidate* DNN models, together with associated training-time information (e.g., per-frame accuracy, event detection window). The *M-Scheduler* uses this information, together with a profile of per-layer runtime on the target edge device, to determine the best candidate for each application—including the degrees of specialization and, thus, sharing. It efficiently searches the option space to maximize application quality (e.g., average F1 score among the applications). The *M-Runner* runtime system runs the selected DNNs, sharing the identical unspecialized layers.

Experiments with several datasets and event detection tasks on an edge node confirm the importance of making deployment-time decisions and the effectiveness of Mainstream's approach. Results show that dynamic selection of shared stems can improve F1-scores by up to 87X relative to the common approach of using fully-independent per-application DNNs (No-Sharing) and up to 47% compared to a static approach of retraining only the last DNN layer and sharing all others (Max-Sharing). Across a range of concurrent applications, Mainstream adaptively selects a balance between per-frame accuracy and frame sampling rate that consistently provides superior performance over such static approaches.

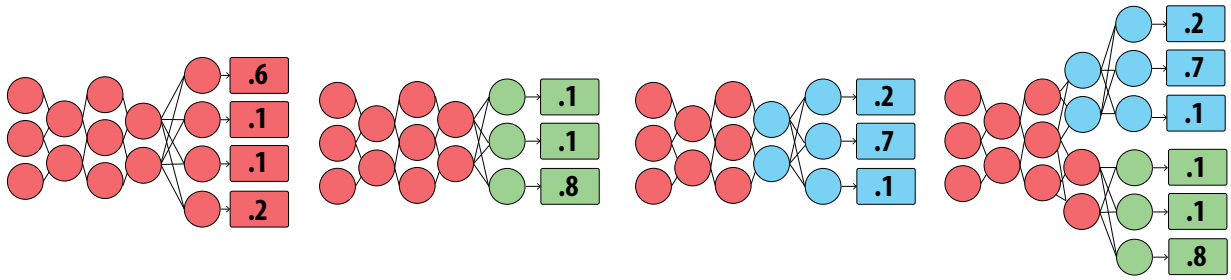
Figure 2: Example computation pipeline for event detection.

Contributions. This paper makes three main contributions. First, it highlights the critical importance of reducing aggregate per-frame CPU work of multiple independently developed video processing applications via stem-sharing; No-Sharing is unable to support even three concurrent applications on our edge node deployment. Second, it identifies the goodness trade-off between per-frame quality and the frame sampling rate dictated by the degree of DNN specialization (and thus the amount of sharing). Third, it describes and demonstrates the efficacy of the Mainstream approach for automatically deploying the right degree of specialization for each submitted application's DNN.

2 Background

DNNs are a powerful tool used in computer vision tasks such as human action recognition [43], object detection [15], scene geometry estimation [14], face recognition [45], etc. Fig. 2 shows a typical computation pipeline for an image classification application. Although frame ingest and image preprocessing are necessary stages of computation, they are low cost and easily shared between concurrent applications. DNNs, on the other hand, are typically unique to each application and computationally expensive: in one image classification application we run, the DNN inference incurs 25X more latency than the preprocessing steps.

DNNs and transfer learning. A machine learning (ML) *model* is a parameterized function that performs a task. *Training* is the process of learning parameter values (called weights) such that the model will approximate the desired function with some measure of accuracy. For example, when training an image classifier, one might examine labeled input images and use gradient descent to find a set of weights that minimizes a loss function over the labels. Using the trained model to find the function's



(a) Base DNN model **(b)** App. #1's DNN model **(c)** App. #2's DNN model **(d)** As executed in Mainstream

Figure 3: Fig. 3a depicts a base DNN trained from scratch for its task. Fig. 3b and Fig. 3c show two new task DNNs, fine-tuned against the base DNN. App. #1 freezes more layers during training than App. #2. Fig. 3d shows how Mainstream runs the applications concurrently. Layers frozen by both App. #1 and App. #2 can be shared.

Architecture	Number of Layers	ImageNet Top-1 Accuracy (%)
InceptionV3	314	78.0
MobileNets-224	84	70.7
ResNet-50	177	75.6

Table 1: Top-1 accuracy of three neural networks architectures trained on the ImageNet dataset.

output given a new, unlabeled input is called *inference*.

DNNs are a class of ML models that usually have a large input space, such as the pixels of an image. A DNN can be represented by a graph where nodes are organized into *layers*; each node computes a function of its inputs, which are outputs from the previous layer.

The "deep" in DNNs refers to their many layers. Increasingly, successful applications of DNNs have largely been the result of building models with more layers that take larger vectors of inputs [42, 29, 19, 44]. The success of these models has hinged critically upon the arrival of very large, labeled datasets for training [13, 32, 3].

Training these large models is notoriously hard. One often lacks sufficient labeled data or computational resources to train such a model. *Transfer learning* is an alternative to training a model from scratch. Here, a model that has already been trained on a similar task (a *base DNN* as in Fig. 3a), is used as an initialization point or feature extractor for the new *target DNN*. During training, a subset of the old parameters are *frozen* and do not change. The remaining free parameters are then retrained on the new task with a new training dataset (Fig. 3b and Fig. 3c). This process *fine-tunes* these parameters to achieve a result comparable to end-to-end training on the entire DNN, but does so with much less data and at a much lower computational cost. In practice, few practitioners train networks from scratch, let alone develop novel network architectures. Transfer learning via one of a few popular neural networks is standard practice.

DNNs for image classification and event detection. Although we believe that Mainstream's approach is generally applicable to video stream analysis, in this paper, we focus on applications that use image-classification DNNs to perform event detection.

Image classification aims to assign one label from a set of categories or *classes* to each image. For example, a 10-class classifier takes an input image and returns a 10-item vector of probabilities representing the likelihood that the image belongs to each class. *Top-N accuracy* is the probability that the correct label is among the top-N highest probability output labels. So, Top-1 accuracy indicates the fraction of images that the model classifies correctly. We refer to this metric as the *per-frame accuracy* in the context of video classification. Popular neural network architectures for image classification include ResNet [19], InceptionV3 [44], and MobileNets [20]. Table 1 describes these three neural networks and their Top-1 accuracy achieved on the ImageNet dataset [13]. Networks trained on ImageNet are popular base-DNNs for image classification tasks.

We define an event as a contiguous group of frames containing some visible phenomenon that we are trying to identify: e.g., a cyclist passing by, or a puff of smoke being emitted. One way of doing event detection is to perform image classification across a sequence of frames. An event is detected if at least one of the contiguous frames is sampled, analyzed, and correctly labeled. Previous works [31] have also used this existence metric to measure recall and precision of range-based queries. (Event detection is not to be confused with object detection, where the goal is to locate an object in a single frame. Indeed, object detection is another way of performing event detection.) We evaluate event classification applications by measuring the *event F1-score*, the harmonic mean between *event recall* and *event precision*. The event recall reports the proportion of ground truth events identified. The event precision reports the proportion of classified events that are true positives. Note that these metrics are relative

to the detection of events across multiple frames and are distinct from per-frame metrics (e.g., Top-1 accuracy).

3 Mainstream Approach: What and Why

Sharing computation between DNNs. When supporting multiple inference applications on a single infrastructure, the common approach is to execute every application's DNN model independently. We refer to this as a "No Sharing" approach. To avoid redundant work, Mainstream instead computes results for DNN layers common to multiple concurrent applications just once and distributes the outputs of shared stems to the specialized layers of all sharing applications. This is analogous to common subexpression elimination used in other domains, e.g., optimizing compilers or database query planners.

Fig. 3 illustrates how compute sharing can be leveraged when two DNNs are fine-tuned from a common pre-trained model and have some unspecialized layers in common. Compute sharing can significantly affect per-frame computation cost and improve throughput for a given CPU resource. Fig. 4a quantifies this effect. It shows the throughput achieved by Mainstream running up to eight concurrent InceptionV3-based event detection pipelines, as a function of how many DNN layers they have in common (i.e., their common degree of specialization). With no sharing (the left-most points), adding a second application halves throughput, which continues to degrade geometrically as more applications are added. Moving to the right, throughput improves as more layers are shared. When all but the last layer are shared, additional applications can be run at very low marginal cost.

On the other hand, there are costs to enabling sharing by leaving many layers unspecialized. In particular, the per-frame accuracy of a model may be lower when only a few layers are specialized. Fig. 4b shows the effect of specialization on per-frame accuracy for several combinations of DNN architectures and classification tasks. As expected, accuracy decreases slowly as less-specialized networks are employed (and hence more sharing is enabled)— with a large decrease occurring only when the fraction of the network specialized is very small. This characteristic enables Mainstream to share large portions of the network with low accuracy loss.

Adaptive management of the sharing opportunity. Since transfer learning is so commonly used by ML developers, and base models are shared within organizations and on the Internet, there may be many opportunities to exploit inter-DNN redundancy in the unspecialized layers. Most developers either use a popular default of specializing only the last layer (which is great for sharing potential, at the potential cost of model accuracy) or determine the degree of specialization based on the amount of training data available, since retraining too many layers without sufficient training data leads to over-fitting. Notably, each developer decides independently.

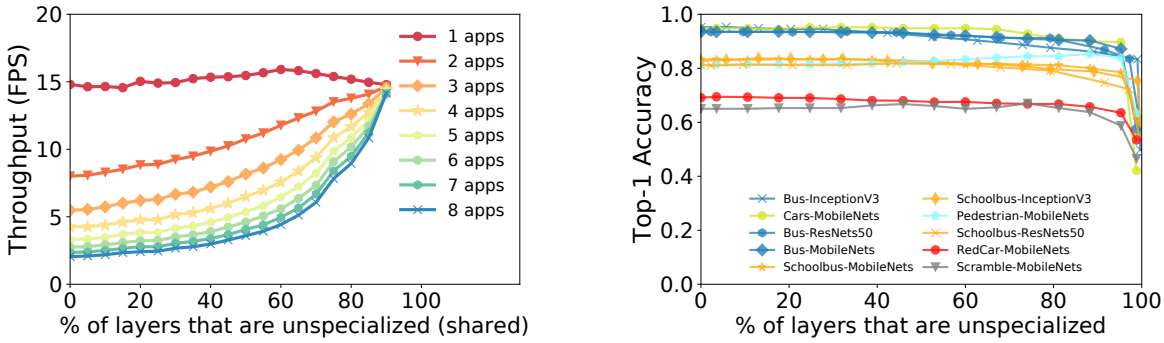
The problem with this approach is that the impact of sharing computation on application quality depends on factors only known at deployment time: the set of concurrent applications and the resources of the edge node on which they are run. Hence, Mainstream defers the decision regarding how much specialization to employ from application development time to deployment time.

Impact of sampling rate for event detection. Given the trade-off between per-frame accuracy and frame processing throughput, picking the right degree of specialization is challenging. Consider an application for monitoring environmental pollution from trains, which is being built using a train detector we deployed. When the application detects a train coming into view, it triggers the capture of high fidelity frames of the train's smoke stack (for subsequent pollution analysis).

Increasing specialization to improve per-frame accuracy increases the probability of correctly classifying frames containing trains— but reduces shared computation. This, in turn, leads to less frequent sampling, which removes opportunities to analyze frames containing a particular view of a train. A higher frame rate increases the probability that an event will be observed in more frames, creating more opportunities for detection. So, the question becomes: should one sample more frames using a less accurate model or sample fewer frames using a more accurate model?

Analytical model for event detection. The Mainstream scheduler (Sec. 6) navigates this "accuracy vs. sampling rate" space by evaluating various candidate *{specialization, frame rate}* tuples. To do so, however, the scheduler must be able to interpret the benefit at the application (not per-frame) level. Hence, we propose an analytical model (sketched in Equations 1-4, below) that approximates the *event* F1-score for a given DNN, given estimates of (a) event length, (b) event frequency, (c) the correlation between frames (discussed below), (d) per-frame DNN accuracy, and (e) DNN analysis frame rate. The frame rate (e) comes directly from the scheduler's proposed tuple; similarly, the accuracy (d) associated with a given specialization proposal will be available to the scheduler (see Sec. 5). Values for event length (a), frequency (b), and correlation (c) can either be measured using representative video samples, or they can be estimated by the developer.

We are able to predict the application's F1-score by estimating the expected number of frames per event that we will have the opportunity to analyze and computing the probability that analyzing the set of frames will result in a detection. The expected number of frames analyzed per event is dependent on the event length and frame rate. The per-frame Top-1 accuracy represents the probability that



(a) Throughput vs. Sharing

(b) Per-frame accuracy vs. (Potential) sharing

Figure 4: Conflicting consequences of DNN compute sharing. (a) shows the frame processing rate for 1–8 concurrent event detection applications as a function of the fraction of the InceptionV3 DNN they share, from No-Sharing on the left to sharing all but the last layer (Max-Sharing) on the right. The experiments are run on the hardware described in Section 7. (b) shows Top-1 accuracy as a function of the fraction of unspecialized layers for three popular DNN architectures (ResNet-50 [19], InceptionV3 [44], and MobileNets-224 [20]) using six of the datasets described in Table 2. We trained each classifier using all three network architectures but omitted some curves for brevity. The horizontal axis starts from fully specialized DNNs on the left to only the last layer being specialized on the right; recall that potential computation sharing is limited to the unspecialized layers.

we will classify any individual frame correctly. However, this does not factor in the fact that sequential frames of an event may be correlated in some way. We therefore introduce and estimate the *inter-frame correlation*, which measures the marginal benefit of analyzing more frames of a single event.

The inter-frame correlation, *corr*, is based on conditional probabilities. For frames corresponding to an event, if $P(X_i)$ and $P(\neg X_i)$ are the probabilities of detecting or not detecting the event in frame i, respectively, then $P(\neg X_2 \mid \neg X_1)$ is the probability of not detecting the event in frame X_2, given that we did not detect it in frame X_1. This conditional probability can be measured empirically from training data. Relying on the Kolmogorov definition, we can calculate the probability the event is detected in at least one of the two frames as $1 - P(\neg X_2 \mid \neg X_1) * P(\neg X_1)$. This logic can be extended to approximate the probability of detecting the event in N tries and to estimate recall.

To estimate recall, we calculate the probability that our DNN will correctly classify at least one frame of an event using the following steps:

$$N = \begin{cases} \lceil \frac{d}{stride} \rceil & \text{w.p. } \frac{d-(stride\%d)}{stride} \\ \lfloor \frac{d}{stride} \rfloor & else \end{cases} \quad (1)$$

$$P_{miss\_1} = 1 - accuracy_{per-frame} \quad (2)$$

$$P_{miss\_N} = corr^{N-1} * P_{miss\_1} \quad (3)$$

$$recall = 1 - P_{miss\_N} \quad (4)$$

We use Eq. 1 to calculate N, the expected number of frames of the event that the model will process. Here, d is the event length, and *stride* is the inverse of the frame rate. Equation 3 estimates the probability the DNN misclassifies all N analyzed frames. Recall is the complement: the probability that we correctly classify at least one frame of the event.

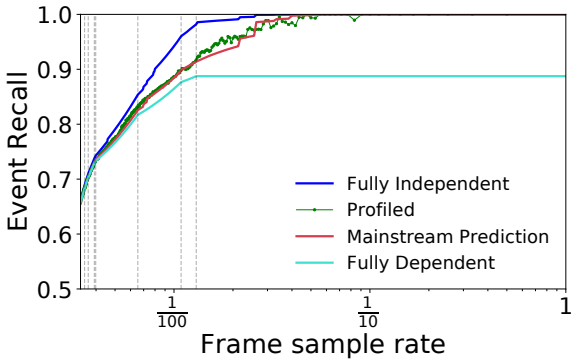


Figure 5: Effect of sample-rate on recall, for different inter-frame correlations. The dotted vertical lines represent each train in the dataset, denoting $\frac{1}{trainlength}$, which is the sample rate required to ensure that one frame of that train is analyzed. The "Profiled" line is measured directly from the TRAIN video dataset, and the other three are approximations based on different inter-frame correlations (uncorrelated, fully correlated, and the empirical correlation observed in the TRAIN dataset).

To estimate the false positive rate, we repeat this calculation, except that d is the number of frames between events (derived from the event frequency), and P_{miss_1} is the probability of *true* negatives. The true positive rate, the false positive rate, and the event frequency are used to calculate the precision. The F1-score is the harmonic mean between precision and recall.

To evaluate our model, we profile an application and

measure the actual recall observed when running the event detector application (e.g., train detection) on the video stream at different sampling rates (Fig. 5). This was measured by averaging the recall achieved from 10,000 trials of sampling at each sample rate. The result is plotted by the "Profiled" line. Our analytical model ("Mainstream Prediction") is sufficiently accurate to describe the complex relationship between frame sample rate and application recall. Mainstream uses this analytical model to efficiently optimize for the trade-off between per-frame accuracy and frame rate.

We use Mainstream for event detection but believe its approach can be generally applied to DNN-based tasks where application quality depends not only on its model, but also on its input sampling rate (e.g., object tracking, action recognition.)

4 Mainstream Architecture

We have developed Mainstream, a training and runtime system for DNN-based live analytics of camera streams, which (a) enables efficient sharing of computation between detection applications, (b) maximizes event F1-score across all tasks, and (c) allows each task to be independently developed, trained, and deployed. Fig. 1 shows the architecture of Mainstream, which consists of three components: M-Trainer, M-Scheduler, and M-Runner.

To deploy a new application to Mainstream, the user provides a corresponding labeled training dataset to their local instance of M-Trainer (Step 1). M-Trainer uses the dataset to train a number of potential models, with varying numbers of specialized layers. This *Model Set* and associated metadata are then assembled into an "M-Package" (Step 2). Note that these are offline steps, performed just once per application prior to deployment, independent of all other tasks. At runtime, M-Scheduler uses the M-Packages of all currently-deployed applications to determine, for each application, the degree of DNN sharing and sampling rate such that, across all applications, the event F1-score is maximized, subject to the resource limits of the edge platform (Step 3). M-Scheduler runs in the datacenter, and is executed once for each scheduling event (e.g., a change in the deployed set of applications, or in available hardware resources). M-Runner then executes the selected model configuration on edge devices (Step 4) and returns app-specific results in real-time (Step 5).

M-Runner is a relatively straightforward execution system for running visual processing pipelines. It accepts a DAG, where each node represents a unit of independent computation, and connections represent data flow. Fig. 2 illustrates the logical DAG for an image classification application. Most of the computation is expected in the "DNN" process, which evaluates the merged DNN of all concurrent tasks. This combined DNN, as illustrated in Fig. 3d, represents the set of models selected by M-Scheduler across all tasks. This DNN is structured as a tree, with sets of layers branching from the shared stem. M-Runner executes the shared stem once per frame, reducing the total processing costs of the deployed tasks.

We next describe how M-Trainer independently trains model candidates for potential sharing (Sec. 5) and how M-Scheduler dynamically chooses among them (Sec. 6).

5 Distributed Sharing-Aware Training

M-Trainer produces a set of models for each task so that they can be combined dynamically at runtime to maximize collective performance. Application developers use M-Trainer independently at different times and locations.

One approach to sharing computation between applications would be to jointly train them using a multi-task network. This, however, requires centralized training of applications. MCDNN [18] proposed fine-tuning models independently and sharing the unspecialized DNN layers. This static approach prevents M-Scheduler from dynamically tailoring stem-sharing to the available resources. In contrast, M-Trainer generates a *set of models* that vary in the number of specialized layers. These models compose an application-specific *Model Set*. The generation of Model Sets allows for the late binding of the degree of specialization to deployment time, when the platform characteristics and co-deployed applications are known.

To construct a Model Set, M-Trainer first analyzes the structure of the base DNN to identify *branchpoints*, the potential boundaries between frozen and specialized layers. Using the app-specific training data provided by the developer, M-Trainer generates a set of fine-tuned DNN models, one per branchpoint, where layers up to the branchpoint are frozen, and the rest are specialized. Only the models at the Pareto-optimal frontier with respect to number of layers specialized and estimated accuracy are actually included in the M-Package. This eliminates from consideration models that reduce accuracy, while requiring more specialization. For example, an overfitted model, caused by specializing too many layers with insufficient training data, will not be included.

Model Sets are bundled together with application metadata into an *M-Package*. This metadata includes the measured per-frame accuracy of each model (we use a portion of the data as the validation dataset.) The expected minimum event duration, event frequency, and inter-frame correlation are optionally measured from the training data and included in the M-Package, or directly provided by the application developer.

The construction of the M-Package is an offline operation, which is run just once per application. For each application, M-Trainer must train multiple models. Although training a model from scratch can be very resource

intensive, fine-tuning is much quicker. M-Trainer creates Model Sets with 15 model options in 8 hours on a single GPU (Sect. 7). Note that the computation for generating all of the models is easily parallelized in a datacenter setting, and may not be significantly higher than traditional fine-tuning. For example, to find the right number of layers to specialize in order to maximize accuracy, one may need to generate these models anyway. The key difference here is that intermediate runs are not discarded, and the final selection is made at run time by M-Scheduler.

As each application's models are independently trained and analyzed, no coordination or sharing of training data is needed between developers of different tasks. The resulting M-Package, however, contains enough information that M-Scheduler, at run time, can optimize across the independently-developed tasks.

6 Dynamic Sharing-Aware Scheduling

At each *scheduling event* (typically, an application submission or termination), M-Scheduler uses the M-Packages created by the various per-application M-Trainers to produce a new overall schedule that optimizes some global objective function, subject to resource constraints (currently, M-Scheduler maximizes average event F1-score across applications). The *schedule* consists of a DNN model selection (indicating the degree of sharing) and target frame-rate for every running application.[1] The final schedule is a tree-like model with applications splitting from a shared stem at potentially different branch points, with each application able to run at its own frame rate.

M-Scheduler optimization algorithm. With N applications to schedule, S possible specialization settings per application, and R frame-rate settings per application, the number of possible schedules is $(S \cdot R)^N$. Although this space is large (e.g., $N \approx 10$, $R \approx 10$, and $S \approx 10$ in our experiments), M-Scheduler can efficiently determine a good schedule using a greedy heuristic (Algo. 1). We compare the schedules generated by our greedy scheduler to those of an exhaustive scheduler in >4,800 workloads each consisting of up to 10 applications, and find that the greedy schedules are on average within 0.89% of optimal.

Essentially, at each step of our iterative algorithm, the scheduler considers making a *move* which improves the application quality of a single application by tweaking its frame rate and/or model specialization. The algorithm greedily selects the move that yields the best ratio of *benefit* to *cost*, defined below. Naturally, before this iterative refinement, the schedule is initialized to the lowest cost configuration— Max-Sharing with minimum frame rate. At any iteration step, the number of possible moves is bounded by $S \cdot R \cdot N$. The total number of moves per invocation of the scheduler is similarly bounded by $S \cdot R \cdot N$, but in practice is much fewer as the set of potential moves that fit within the computational budget is exhausted.

[1]Here, we assume that some admission control policy (outside the scope of this work) has been applied to ensure that some schedule is feasible for the set of running applications.

Algorithm 1 Scheduler optimization algorithm

```
function GetNextMove(schedule)
    ▷ Finds change to schedule with the highest benefit/cost
function Schedule(budget)
    sched ← GetSchedule(max_sharing, min_fps)
    while True do
        next_move ← GetNextMove(sched)
        cost ← cost + GetCost(next_move)
        if cost < budget then
            sched ← UpdateSchedule(next_move)
        else return sched
```

Measuring the Benefit of a Move. The benefit associated with a move captures the improvement in F1-score for the application associated with that move. This value is calculated using the analytical model presented in Sect. 3 and the application metadata in the M-Package.

Measuring the Cost of a Move. The cost value considered represents the computational resources (e.g., CPU-seconds per second) consumed by a given schedule arrangement and depends on the number of shared subgraphs, the number of task-specific subgraphs, and the intended throughput (frame-rate) of each subgraph. The relative cost of a schedule is the sum of the execution time of each model layer, multiplied by the desired throughput. Consider for example a schedule with two applications, both executing at F FPS. Assume they share a compute stem A, and then branch to specialized subgraphs B_1 and B_2. If C_A represents the execution cost (in CPU-seconds per frame, say) of A, and C_B the execution cost of B_1 and B_2, then the total cost of the schedule is $F \cdot C_A + 2F \cdot C_B$. Adding a third application based on the same network, using the same branchpoint and frame rate will add another factor of $F \cdot C_B$ to the schedule cost.

To most accurately model the compute costs (e.g., C_A and C_B), a forward pass execution of the base DNN should be executed and measured once on the target hardware. Note that as cost is relatively insensitive to the assigned model weights, each base DNN need only run one time (ever) per target hardware, not once per trained application.

Max-Min Fairness Among Applications. Although stem-sharing improves overall system efficiency, maximizing a global objective may lead to an inequitable allocation of resources for individual applications. Thus, M-Scheduler can also be run in max-min fairness mode, which maximizes the minimum benefit among applications, instead of the average. Max-min fairness is scheduled by searching the space using dynamic programming.

X-Voting To Improve Precision. Mainstream uses *voting* across frames to improve precision, and consequentially F1-score. With X-voting, Mainstream requires X consecutive positives to identify an event. While X-voting decreases false positives, it is not guaranteed to increase precision. For X-voting to improve precision, it must decrease false positives at a higher rate than true positives. The ideal X-voting configuration again depends on the applications and the resources available. A higher X incurs fewer false positives, but requires more cost to sustain high recall (either by increasing FPS or increasing specialization). We evaluate the effect of various X-voting configurations in Sect. 8.

7 Experimental Setup

To evaluate our system, we implement seven different event detection applications. We refer to the set of applications as 7-Hybrid. These are listed in Table 2, along with the datasets we used to train and test them. A pedestrian-detection application (Pedestrian) is trained based on the fully-labeled, publicly-available Urban Tracking video dataset [25]. Our application to classify car models (Car) uses the Stanford Cars image dataset [28]. Train detection (Train) is based on video of nearby train tracks that we have captured in our camera deployment, and have hand labeled. The remaining classifiers are trained on a video of a nearby intersection, also captured in our camera deployment. We have obtained the necessary permissions and plan to make our Trains and Intersection video dataset available publicly. We reserve a portion of these datasets to create synthetic video workloads for testing.

We use M-Trainer to produce a task-specific M-Package for each application. Model candidates are fine-tuned using the MobileNets-224 model pretrained on ImageNet as a base DNN (implemented in Keras [8] using TensorFlow [2]). Each M-Package contains several models with different degrees of fine-tuning as described in Section 5. We evaluate Mainstream using the M-Packages and hold-out validation sets from our datasets. Our experiments use the applications in Table 2.

Hardware. Training is performed on nodes equipped with Intel® Xeon® E5-2698Bv3 processors (2.0 GHz, 16 cores) and an Nvidia Titan X GPU. All end-to-end experiments use an Intel® NUC with an Intel® Core™ i7-6770HQ processor and 32 GiB DRAM, which is intended to represent an edge processing device. The Train and Intersection videos were captured using an Allied Vision Manta G-1236 GigE Vision camera.

8 Evaluation

We evaluate our system in the context of independent DNN-based video processing applications sharing a fixed-resource edge computer. In our evaluation, we show that Mainstream's dynamic approach outperforms static solutions in all of our experimental settings, across various application workloads, computational budgets, and numbers of concurrent applications. Mainstream's X-voting is capable of improving F1-score but, like model specialization, must also be dynamically configured to the resources available. In addition to our benchmarked applications, we show an end-to-end Mainstream deployment of a train detection application used for environmental pollution monitoring.

8.1 Mainstream Improves App Quality

Our goal in event detection is to maximize per-*event* F1-score. We compare the F1-score achieved by Mainstream with two baselines: No-Sharing and Max-Sharing. *No-Sharing* is the default behavior for a multi-tenant environment and is the approach taken by systems like TensorFlow Serving [1] and Clipper [12]. No-Sharing maximizes classification accuracy at the cost of a reduced sampling rate and requires no coordination between tenants. *Max-Sharing* is the sharing approach used by MCDNN [18]. It uses partial-DNN sharing by fine-tuning the final layer of concurrent DNNs. In many cases, Max-Sharing provides better F1-score relative to No-Sharing when a non-trivial number of applications share the infrastructure; it sacrifices classification accuracy to maximize the number of frames processed. We show, however, that Max-Sharing is less effective than making deliberate runtime decisions about how much sharing to use.

In order to observe the effects of increasingly constrained resources without a large number of distinct applications, we generate additional applications by augmenting our application set. Each of the seven classification tasks in Table 2 has a corresponding "accuracy-tradeoff curve", which represents the relationship between per-frame accuracy and the shared stem size (Fig. 4b). For each application in our experiments, we randomly choose one of the seven classifiers (and its corresponding accuracy-tradeoff curve) and parameterize it with a different event length, event frequency and inter-frame correlation. To capture the effects of diverse application characteristics, the parameters are uniformly sampled from a range of possible values. Each workload consists of up to 30 concurrent applications. In most experiments, we show the behavior averaged across 100 workloads. Our video capture rate for all experiments is 10 FPS.

Mainstream outperforms static approaches. M-Scheduler maximizes per-event F1-score by varying the sampling rate and amount of sharing. Each additional application introduces more resource contention, forcing the system to pick a different balance between accuracy and sampling rate to achieve the best average F1-score.

Detection Task	Dataset Description	Number of Images	Avg Event Length	Min Event Length	Event Frequency
PEDESTRIANS	Urban Tracker atrium video	4538	59	2	0.63
BUS	Intersection near CMU video	4762	1039	141	0.27
RED CAR	Intersection near CMU video	9172	228	46	0.08
SCRAMBLE	Intersection near CMU video	1500	412	382	0.16
SCHOOLBUS	Intersection near CMU video	2600	854	92	0.03
TRAINS	Train tracks near CMU video	3066	132	20	0.01
CARS	Images of 23 car models	3042	—	—	—

Table 2: Labeled datasets used to train classifiers for event detection applications. Average and minimum event lengths are reported in number of frames. Event length and event frequency only apply to video datasets and not CARS.

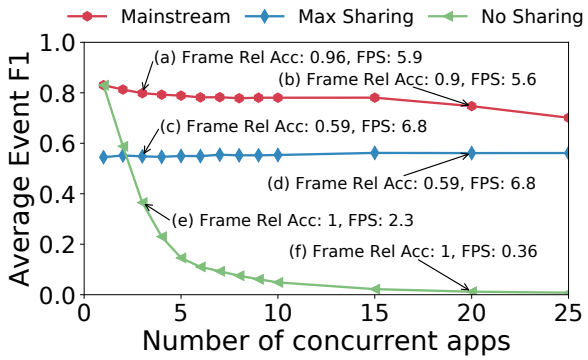


Figure 6: Mainstream improves F1-scores vs. No-Sharing between applications or conservatively sharing all layers but the last one. "Frame Rel Acc" is the relative image-level accuracy of the model deployed, compared to the best performing model candidate. "FPS" is the average observed throughput of the deployed applications.

Fig. 6 compares Mainstream with our baseline strategies. Mainstream delivers as much as a 87X higher per-event F1-score than No-Sharing and as much as a 47% higher score vs Max-Sharing. Fig. 6 reports F1-scores averaged across 100 workloads. The relationship between the three schedulers holds when examining individual workloads. No-Sharing exhibits low recall because of its low throughput—the system has fewer opportunities to detect the event. Max-Sharing has high throughput but a worse precision because the underlying model accuracy is lower—it evaluates many frames but does so inaccurately. Mainstream outperforms by striking a balance, sometimes choosing a more accurate model, and sometimes choosing to run at a higher throughput.

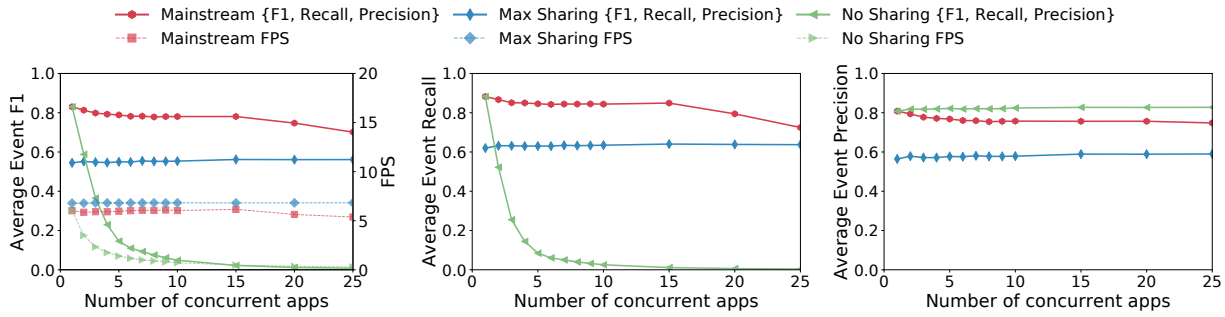
Mainstream dynamically balances precision and recall. Fig. 7 delves into the system effects of Mainstream more deeply. F1-score, recall, and precision are plotted. The average application frame rate is plotted, showing how Mainstream dynamically tailors resource usage to the workload. (Not shown is the varying model accuracy.) Optimizing for precision requires careful tuning of the application frame rate. While higher FPS always leads to higher recall, it does not always lead to higher precision. (A high frame rate may only increase false positives without increasing expected true positives.) For instance, No-Sharing's low frame rate and high per-frame accuracy allows it to have the highest precision of the three approaches. When given just a few applications, Mainstream runs specialized models, while throttling the stream rate to avoid unnecessary false positives. As resources become scarce, many applications begin to share more of the network.

Mainstream improves upon Max-Sharing even under tight resource constraints. Fig. 8 shows the effect of Mainstream, Max-Sharing, and No-Sharing on a range of computational budgets. We average the event F1-scores across 100 workloads, each with 3 applications. The right-most points represent the scenario of running on computational resources equivalent to an Intel® NUC. With a small workload of three applications, No-Sharing performs better than Max-Sharing, as it is able to run expensive models at a high enough frame rate. As we decrease the available budget, Max-Sharing's conservative sharing approach allows it to be more scalable than No-Sharing. However, even after the computational budget is reduced by 83%, Mainstream still improves application performance, compared to the overly conservative Max-Sharing approach.

8.2 Tuned X-Voting improves F1-score

Applications that suffer from low per-frame precision will generate many false positives. An X-voting approach can greatly decrease the incidence of false positives, as X consecutive classifications are needed in order to report a detection. Too large a value of X can hurt recall, causing real events to go unreported. By using X-voting and optimizing the parameter X, Mainstream can improve the overall average event F1-score.

Fig. 9 shows the effects of X-voting on F1-scores as X and the number of applications are varied, while total resources are kept fixed. With just a few concurrent applications, running at high frame rates, 7-voting and 5-voting yield the highest F1-scores. With more resource



(a) F1-score **(b)** Recall **(c)** Precision

Figure 7: The average event F1-score (Fig. 7a), recall (Fig. 7b) and precision (Fig. 7c) across 100 deployed workloads are shown (solid lines) alongside the average frame-rate across applications (dotted lines). Mainstream dynamically balances recall and precision to maximize aggregate F1-score. With high numbers of concurrent applications, Mainstream sacrifices small amounts of both specialization and frame-rate.

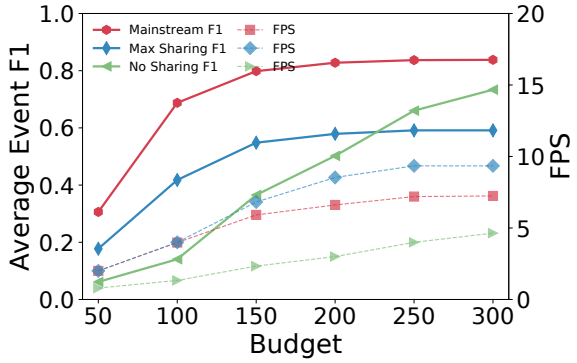


Figure 8: Mainstream improves F1-score of workloads under varying computational budgets. The rightmost points on the X axis represent the resources available on an Intel® NUC. Even under heavy resource constraints, there is available capacity for Mainstream to perform optimizations.

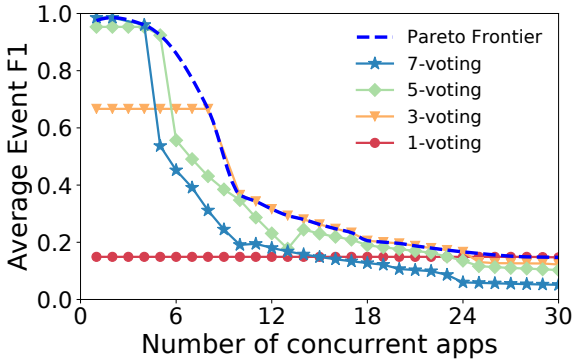


Figure 9: X-voting increases precision and helps Mainstream achieve a higher F1-score, but only if frame rate is high enough to avoid hurting recall. Thus, the effects vary by the level of resource contention. The Pareto frontier shows the F1-scores achievable given the dynamic selection of an optimal X-voting scheme for the resource scenario.

contention, and lower throughput, 3-voting becomes the best choice as the cost of dropping true positives outweighs the benefit of reducing false positives for higher values of X. When resources become too constrained, this approach is less viable, e.g., 1-voting becomes the best approach at 25 concurrent apps. Fig. 9 also shows the Pareto frontier of F1-scores achievable across all values of X for a given number of concurrent applications.

8.3 Mainstream Deployment

We deployed our environmental pollution monitor application and nine other concurrent applications using both Mainstream and a conventional No-Sharing approach for one week on the hardware setup described in Section 7. Fig. 10 shows the trace of both approaches on a *single* train event sequence, indicating the frames analyzed. A hit represents a correct classification of the train, a miss represents an incorrect classification. We see that Mainstream's deployment samples the stream more frequently, yielding many more hits (and misses) than No-Sharing; the result, though, is that Mainstream detects the train event earlier and more confidently.

We control the false positive rate with 2-voting, requiring Mainstream to have two positive samples before an event is classified. The false positive rate of the TRAIN video drops from 0.028 to 0.00056. No-Sharing and Mainstream achieve a 0 and a 0.00056 false positive rate, respectively. In the analyzed deployment in Fig 10, we see that Mainstream still detects the train easily and quickly.

9 Additional Related Work

Several recent systems have attempted to tackle the problem of optimizing execution of visual computing pipelines.

VideoStorm [48] is a video analytics system for large-scale clusters and workloads. It analyzes resource use and application-goal-based metrics as a function of tunable parameters of the analytics pipelines, building models for each application independently. It uses these models

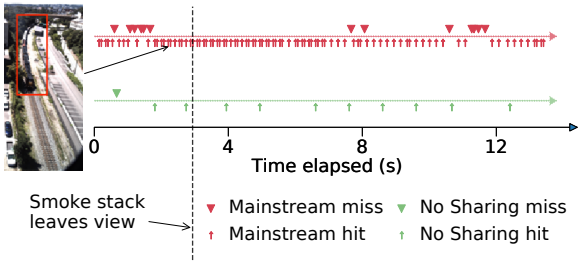


Figure 10: Timeline of Mainstream running a train detector app with 9 concurrent applications. Our goal is to detect the train as early as possible, before the smoke stack is out of view (end of window represented by the dotted line). Mainstream detects the train earlier and more confidently than No-Sharing.

to allocate resources and select parameters for deployed applications on a target platform, in order to maximize application quality metrics. VideoStorm takes a black-box view of the applications, and assumes that quality and resource consumption of co-deployed applications are independently determined. Therefore, it cannot take into account computation sharing, or optimize the sharing vs. degree of specialization tradeoff. In contrast, Mainstream takes a white-box approach to modeling application quality, and can explicitly tune computation sharing to improve application quality metrics. Compared to VideoStorm, Mainstream sacrifices some generality to solve the joint optimization problem.

MCDNN [18] introduces a static approach to sharing DNN computation, in which each application developer independently determines their amount of model specialization. MCDNN opportunistically shares any identical unspecialized layers between applications. In contrast, Mainstream's training and scheduling components allow late binding and jointly-optimized selection of the degrees of specialization at run time, when resource availability and co-deployed tasks are known.

Inference serving systems. Mainstream is an inference serving system for running neural networks on resource-constrained nodes. Other inference serving systems include Clipper [12], NoScope [26], and TensorFlow Serving [1]. Like Mainstream, these systems optimize for latency and throughput gains. Clipper caches results from multiple models, dynamically chooses from the results, and optimizes the batch size. NoScope replaces expensive neural networks for object detection with cheaper difference detectors and specialized models. TensorFlow Serving increases throughput with batching and hardware acceleration. LASER [4] and Velox [11] are inference serving systems for non-DNN models. LASER deploys linear models while Velox deploys personalized prediction algorithms using Apache Spark.

Unlike Mainstream, these inference serving systems do not share computation between independently trained models. They also target cluster environments. Mainstream targets edge devices with limited resources, where achieving the right degree of DNN computation sharing is particularly important, though such sharing would also be valuable in large data centers.

Reducing DNN inference time. Approaches to reducing DNN inference time for vision applications can be broadly classified into those that reduce model precision [16, 50, 10, 9, 7, 23, 39], use efficient network architectures [20, 24], use anytime prediction methods [22, 21], or employ model compression and sparsification [17, 33, 46]. All of these methods are orthogonal to Mainstream's adaptive DNN computation sharing technique, but share its goal of selecting the right trade-off between per-frame quality and frame throughput.

Multi-task networks. Multi-task learning [6, 49, 5, 36, 35, 34, 27, 38] is a ML approach in which a single model is trained to perform multiple tasks. Using multiple tasks to train a single model helps achieve better accuracy because of better generalization and complementary information [6, 41]. In the context of DNNs, a multi-task network can have a varying number of shared layers across tasks and task-specific layers [36, 35]. Multi-task learning assumes that all of the tasks are known *a priori*, and that training data for all of the tasks is available for use in a single training process. In contrast, Mainstream allows each task to be developed, trained, and deployed independently, and avoids the need to share or expose proprietary or privacy-sensitive training data between task developers. Note that one can run a multi-task network as a single large application in Mainstream.

10 Conclusion

Mainstream adaptively orchestrates DNN stem-sharing among concurrent video processing applications sharing the limited resources on an edge device, resulting in much higher aggregate application quality. Experiments with several event detection tasks confirm that Mainstream significantly increases overall event F1-score relative to current approaches over a range of concurrency levels.

11 Acknowledgments

We thank the member companies of the PDL Consortium (Alibaba, Broadcom, Dell EMC, Facebook, Google, HPE, Hitachi, IBM Research, Intel, Microsoft, MongoDB, NetApp, Oracle, Salesforce, Samsung, Seagate, Toshiba, Two Sigma, Veritas and Western Digital) for their interest, insights, feedback, and support. This work is supported in part by funding from Intel as part of the Intel STC for Visual Cloud Systems (ISTC-VCS). We also thank Michael Sussman for help in developing statistical models.

References

[1] Tensorflow Serving. https://www.tensorflow.org/serving/.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[3] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan. YouTube-8M: A large-scale video classification benchmark. *CoRR*, abs/1609.08675, 2016. URL http://arxiv.org/abs/1609.08675.

[4] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang. Laser: A scalable response prediction platform for online advertising. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14. ACM, 2014. URL http://doi.acm.org/10.1145/2556195.2556252.

[5] K. Ahmed and L. Torresani. Branchconnect: Large-scale visual recognition with learned branch connections. *CoRR*, abs/1704.06010, 2017. URL http://arxiv.org/abs/1704.06010.

[6] R. Caruna. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

[7] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045361.

[8] F. Chollet et al. Keras. https://keras.io, 2015.

[9] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *Advances in Neural Information Processing Systems*, 2016.

[10] M. Courbariaux, Y. Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. 2015.

[11] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan. The missing piece in complex analytics: Low latency, scalable model management and serving with Velox. In *CIDR*, 2015.

[12] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA, 2017. USENIX Association. ISBN 978-1-931971-37-9. URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[14] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2016.

[16] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014. URL http://arxiv.org/abs/1412.6115.

[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: Efficient inference engine on compressed deep neural network. *SIGARCH Comput. Archit. News*, 44(3):243–254, June 2016. ISSN 0163-5964. doi: 10.1145/3007787.3001163. URL http://doi.acm.org/10.1145/3007787.3001163.

[18] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. 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, 2016.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[21] H. Hu, D. Dey, J. A. Bagnell, and M. Hebert. Anytime neural networks via joint optimization of auxiliary losses. *arXiv preprint arXiv:1708.06832*, 2017.

[22] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *arXiv preprint arXiv:1703.09844*, 2017.

[23] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, 2016. URL http://arxiv.org/abs/1609.07061.

[24] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. 2016. URL http://arxiv.org/abs/1602.07360.

[25] Jodoin, J.-P., Bilodeau, G.-A., and N. Saunier. Urban tracker: Multiple object tracking in urban mixed traffic.

In *IEEE Winter conference on Applications of Computer Vision (WACV14)*, March 2014.

[26] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. Noscope: Optimizing neural network queries over video at scale. In *Proceedings of the VLDB Endowment, Vol. 10, No. 11*, 2017.

[27] I. Kokkinos. Ubernet: Training a 'universal' convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. *CoRR*, abs/1609.02132, 2016. URL http://arxiv.org/abs/1609.02132.

[28] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[30] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[31] T. J. Lee, J. Gottschlich, N. Tatbul, E. Metcalf, and S. Zdonik. Precision and recall for range-based anomaly detection. *SysML*, Feb 2018.

[32] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. pages 740–755", 2014.

[33] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy. Sparse convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[34] M. Long and J. Wang. Learning multiple tasks with deep relationship networks. *CoRR*, abs/1506.02117, 2015. URL http://arxiv.org/abs/1506.02117.

[35] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. S. Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. *CVPR*, 2016. URL http://arxiv.org/abs/1611.05377.

[36] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch Networks for Multi-task Learning. In *CVPR*, 2016.

[37] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[38] R. Ranjan, V. M. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *arXiv preprint arXiv:1603.01249*, 2016.

[39] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[40] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. 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*, pages 512–519, 2014. doi: 10.1109/CVPRW.2014.131. URL http://dx.doi.org/10.1109/CVPRW.2014.131.

[41] S. Ruder. An overview of multi-task learning in deep neural networks. abs/1706.05098, 2017. URL http://arxiv.org/abs/1706.05098.

[42] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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

[44] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[45] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, pages 1701–1708, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.220. URL http://dx.doi.org/10.1109/CVPR.2014.220.

[46] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. *Advances in Neural Information Processing Systems*, 2016. URL http://arxiv.org/abs/1608.03665.

[47] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3320–3328, Cambridge, MA, USA, 2014. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969033.2969197.

[48] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, NSDI '17, 2017.

[49] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Facial landmark detection by deep multi-task learning. In *ECCV*, 2014.

[50] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[51] S. Zilberstein. Using anytime algorithms in intelligent systems. In *AI Magazine, 17(3):73-83*, 1996.