Towards Memory-Efficient Inference in Edge Video Analytics

Arthi Padmanabhan[★†], Anand Padmanabha Iyer[★], Ganesh Ananthanarayanan[★], Yuanchao Shu[★], Nikolaos Karianakis[★], Guoqing Harry Xu[†], Ravi Netravali[‡]

[★]Microsoft Research [†]UCLA [‡]Princeton University

Abstract

Video analytics pipelines incorporate *on-premise* edge servers to lower analysis latency, ensure privacy, and reduce bandwidth requirements. However, compared to the cloud, edge servers typically have lower processing power and GPU memory, limiting the number of video streams that they can manage and analyze. Existing solutions for memory management, such as swapping models in and out of GPU, having a common model stem, or compression and quantization to reduce the model size incur high overheads and often provide limited benefits. In this paper, we propose *model merging* as an approach towards memory management at the edge. This proposal is based on our observation that models at the edge *share* common layers, and that *merging* these common layers across models can result in significant memory savings. Our preliminary evaluation indicates that such an approach could result in up to 75% savings in the memory requirements. We conclude by discussing several challenges involved with realizing the model merging vision.

CCS Concepts: • **Information systems** → **Data analytics**; • **Computing methodologies** → *Neural networks.*

Keywords: video analytics, deep neural networks

ACM Reference Format:
Arthi Padmanabhan, Anand Padmanabha Iyer, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Guoqing Harry Xu, and Ravi Netravali. 2022. Towards Memory-Efficient Inference in Edge Video Analytics. In *3rd ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges (HotEdgeVideo'21), January 31-February 4, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3477083.3480150

1 Introduction

Typical tasks in live video analytics such as object counting, license plate recognition, and surveillance involve perpetual (i.e., long-running), real-time monitoring of live video streams from hundreds of cameras deployed in a geographic location [10, 20]. In order to practically deliver both fast and accurate responses using the latest deep learning models [12, 16, 26, 28], video analytics deployments are generally designed as *cloud-edge* systems, where *on-premise* edge servers complement more powerful servers in the cloud [33]. The edge servers maximally execute model inferences on the live video streams, saving the large bandwidth (and associated transfer delays) necessary to upload them to the cloud, and enabling compliance with privacy and data placement restrictions. Hence, edge servers play a crucial (and increasing) role in live video analytics [13, 15, 19, 27, 38].

Despite their benefits, edge servers must carefully balance functionality and cost in order to support the aforementioned use cases at scale. More specifically, to remain cost-efficient, edge servers typically possess weaker GPUs [3, 4] than their cloud counterparts. However, traffic monitoring for a even small city can involve analyzing hundreds of live streams in parallel [1], translating to the need to run inferences of many models concurrently, and over long time periods. Worse, vision processing models consume increasing amounts of memory resources [2, 22, 23, 34]; note that running a model involves loading its parameters into GPU memory, and reserving enough space for all intermediate data that it generates during inference. The result is that video analytics deployments at the edge are already *bottlenecked by GPU memory*[29], and the situation is only expected to worsen as deployments and model complexity grow.

A natural solution to edge GPU memory management when all of the models do not fit together in memory is to time-slice the inference execution [35]. In this approach, models are *swapped* in and out of GPU memory according to a given scheduling policy. The evicted models are stashed in CPU memory and the transfer is done through a PCIe interface. Unfortunately, the overheads of such swapping are prohibitive for practical video analytics scenarios, often exceeding the execution time of the inference itself, leading to unacceptable accuracy or latency degradations (§2.1).

We propose *model merging* as a technique to improve the efficiency of video analytics workloads that exceed the available GPU memory at an edge server. By merging multiple models (or components of them), the overall memory requirement can be drastically reduced compared to running them independently. This, in turn, can substantially reduce (or even avoid) the amount of swapping required, thereby increasing the number of models that can be supported while adhering to a given accuracy or latency constraint.

Our proposal is motivated by the observation that vision processing models that are deployed at the edge routinely contain layers that are common, even across model families. For example, Faster RCNN (an object detector) and

Model	Load (GB)	Run (GB)		
		BS=1	BS=2	BS=4
YOLOv3	0.242	0.518	0.728	1.22
Faster RCNN	0.656	2.55	4.70	8.32
ResNet152	0.244	0.648	0.978	1.71
ResNet50	0.118	0.346	0.498	0.838
VGG16	0.536	0.738	0.890	1.18

Table 1. Memory requirements (in GB) for loading and running inference for various models for three different batch sizes (1, 2 and 4). Even simple models incur large memory footprints for inference. Frameworks add their own overhead, for instance PyTorch reserves 0.8GB for book-keeping. The run values include the load values.

Resnet50 (a classifier) have 50 layers in common. This is because both tasks start by extracting features using the same layer structure. Hence, *sharing* these common layers and their associated weights across models can reduce memory consumption. Further, as the number of layers that can be shared across models and the number of models that need to be supported at the edge increase, the opportunities for memory savings also increase.

Enabling efficient model merging requires solving several challenges. As an example, each model in a workload may possess many layers that can be shared with other models, leading to a combinatorial search space of sharing configurations. This is further exacerbated by the fact that shareable layers may come with different weights, and identifying a singular set of weights that preserves accuracy for all of the considered models, if any such weights exist, requires training the merged model. Such training can be costly and quickly discount the benefits of merging.

In the rest of this paper, we discuss the shortcomings of existing memory management techniques (§2), assess the potential of model merging (§3), and discuss many such challenges that stand in the way of efficient model merging (§4). We conclude with potential directions that we are exploring towards a solution (§5).

2 Motivation

While edge servers provide a number of advantages, such as bandwidth reduction, privacy compliance, and resilience to disconnection, they are inferior to the cloud in terms of compute and memory for cost- and power-efficiency reasons. For example, edge boxes are commonly equipped with GPUs that have only 2-16 GB of memory and operate at lower clock speeds [7]. To run inference, the model layers and parameters need to be loaded to the GPU memory with enough room for intermediate data (e.g., activations). The memory used by a model depends on the model architecture and the batch size; a higher batch size requires more memory in general. Real time video analytics tasks tend to use smaller batch sizes due to their stringent latency requirements [6, 36].

Model	Load (ms)	Run (ms)		
		BS=1	BS=2	BS=4
YOLOv3	49.5	17.0	17.8	18.1
Faster RCNN	95.5	82.3	84.1	85.2
ResNet152	73.25	24.81	26.27	26.70
ResNet50	27.1	8.41	8.50	8.52
VGG16	72.2	2.10	2.23	2.40

Table 2. Load/run time for various models for three different batch sizes. Loading results in significant overheads.

Existing vision models deployed at the edge vary widely in terms of their compute and memory requirements. Table 1 lists the memory required to load and run inference on several popular video analysis models using different batch sizes. As shown, even the lightest model imposes a non-negligible memory footprint (i.e., ResNet50 requires 0.12 GB), and other models such as VGG16 and Faster RCNN quickly push the boundaries of what existing edge servers can support. For instance, after accounting for the fixed memory that machine learning frameworks like PyTorch [9] reserve for use (i.e., 0.8 GB), an edge server with 2 GB of memory would be unable to house more than two Faster RCNN or VGG16 models – a drastic drop from the hundreds of feeds and models that typical deployments warrant today.

2.1 Existing GPU Memory Management Techniques

The common approach to running inference on multiple models is to share the available GPUs, either in space or in time. We describe these approaches in turn.

Existing deep learning frameworks recommend allocation at the granularity of an entire GPU—for example, PyTorch uses a worker-per-model architecture and advocates allocating a GPU per worker [22]. Space-sharing techniques, such as NVIDIA's Multi-Process Service (MPS) [5], TensorRT inference server [8] and SwapAdvisor [18] eschew this exclusivity and partition the GPU memory per model. Although space-sharing approaches are effective when the set of models in a workload can fit together in GPU memory, they are insufficient when that property does not hold.

In the time-sharing category, techniques focus on slicing the time each model executes in the GPU by *swapping* them in and out of the GPU memory [18], and is more directly applicable to edge settings where the workloads exceed available GPU memory. The models are stashed in CPU before they are loaded into the GPU memory through a PCIe interface, which provides bandwidths of up to 64 GB/s. However, the swapping process can introduce prohibitive overheads. In table 2, we show the loading time and inference time for several models that are commonly used for video analysis. As we can see, the loading time can be large and often greater than the inference time. The loading time being larger than the inference time is particularly a problem with the smaller batch sizes that are common in live video analytics [36].

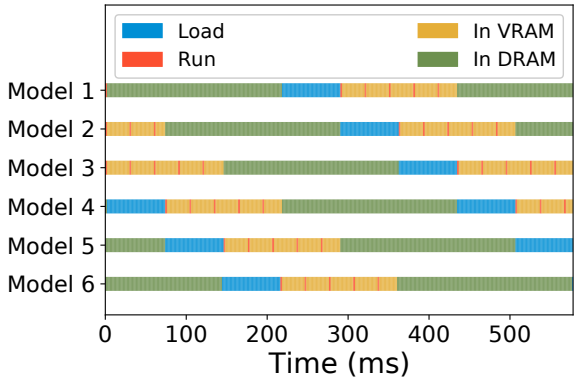

Figure 1. Six VGG16 models do not fit in memory at once and are swapped in and out of the GPU, accumulating significant lag between eviction and reloading.

Due to the prohibitive loading time, it is desirable to have models present in the GPU memory rather than loading them on-demand. Since the memory consumption of a model is higher during execution (i.e., due to the intermediates), a better (hybrid) approach is to *pack* models in the GPU memory and only execute as many as possible while ensuring that the loading costs for the next model is hidden (i.e., pipelining). For example, it is possible to load three YOLOv3 models in a GPU with 2GB memory and execute them one at a time.

While this hybrid approach avoids the cost of swapping for the models that are loaded on the GPU, it still suffers from many shortcomings. First, the number of models that can be packed in the GPU is dependent on the left over memory and the memory usage characteristics of the model. In GPUs with limited memory, the number of models that can be packed may not be enough to meet the latency requirements at the edge. Figure 1 shows a workload consisting of six VGG16 models that do not fit in the GPU memory all at a time. An optimal solution is to load three models at a time and execute one so that the GPU memory is fully utilized. As we notice, the time between when a model is evicted and when it is loaded again makes it such that each model is out of the GPU for longer than it is in. Second, while on one hand packing reduces swapping overhead, it also limits the number of models that can be executed in parallel. Finally, it may be possible to selectively pre-load models based on predictability of the considered workload (e.g., eschewing inference on certain video streams at night due to lack of activity). However, in video analytics at the edge, spatial correlation of streams results in model demands being correlated [21], e.g., during rush hour and at midnight, video streams of roads in city will exhibit similar processing demands.

Compression and quantization are alternative techniques that are commonly used to reduce memory requirements and speed up execution time. However, they have their own challenges. While some standard models have off-the-shelf compressed models available (e.g., TinyYOLO), they often trade off accuracy for reducing the memory footprint [11]. Designing an optimal compression scheme for a given workload requires expert knowledge and is time intensive and thus cannot be carried out in an online fashion. Further, in a workload with a mix of models, which is the common case in edge servers, it may not be possible to compress all the models. Even a handful of non-compressed models could result in the workload exhausting the GPU memory.

3 Our Proposal: Model Merging

To overcome the limitations of existing memory management strategies in the increasingly prevalent scenario in which a workload's models cannot fit together in a GPU's memory, we propose *model merging*. Our proposal is based on the observation that models deployed at the edge contain layers that are common. Thus, *sharing* these layers (weights) can reduce memory consumption, and improve the efficacy of existing time- and space-sharing approaches.

We note that, though they are similar in spirit, our proposal differs from stem-sharing strategies [24] that reduce computation by sharing the common *stem* located at the beginning of specialized models. In contrast, we seek to flexibly and maximally share layers across the entire model architecture to directly address memory (not compute) bottlenecks. In addition, our vision of model merging is complementary to existing systems that directly optimize the computation (not memory) required in video analytics pipelines through improved job scheduling strategies, approximation tolerance, and tuning pipeline configurations [17, 25, 37].

3.1 Commonality of Layers

All of the models running on the edge servers in our target video analytics deployments are vision processing models. We leverage the fact that vision processing models have common functionality, and thus importantly, common layers. Each layer in a model governs what to do with an input video frame. For example, a convolutional layer will apply some (learned) filter to the image and send the result to the next layer. The properties of each layer, such as input size, output size, kernel size, and stride determine how the layer looks through the pixels of an image. We therefore consider two layers a match when all of these properties are identical. Within a model, layers are typically grouped by functionality, for example extracting features or finding boxes that contain objects. As a result, we often find not just individual layers in common between models, but groups of layers.

Figure 2 shows the percentage of layers that are common and the potential memory savings by sharing the common layers across pairs of models. These model pairs fall into one of three categories: (1) instances of the same model (e.g., for different detection tasks or training datasets), (2) different models in the same model family (e.g., multiple ResNet models), and (3) different models in different families. Multiple instances of the same model clearly match on every layer; this favorable scenario is not uncommon in video analytics

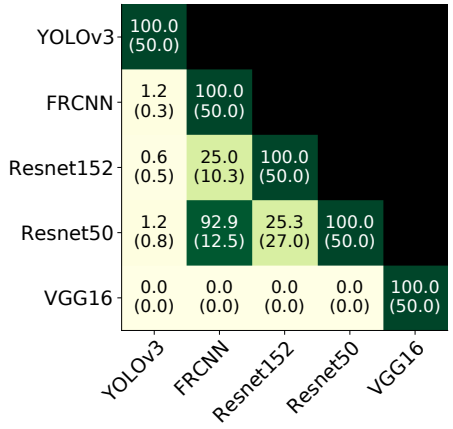

Figure 2. Percentage of layers that are common among pairs of models, along with the potential memory savings by sharing the common layers in parentheses, e.g., two YOLOv3 models (first row, first column) have all layers in common (hence 100%), and can save 50% memory by merging compared to running individually.

deployments, as several models tend to dominate the landscape [1] and a given model may be employed for the same task on different video feeds. However, even across models in different families that are designed for different tasks, such as the Faster RCNN object detector and the Resnet50 classifier, we (perhaps surprisingly) find commonalities. This is because both tasks start by extracting features using the same layer structure, so they share the first 50 layers. Similarly, Resnet50 and Resnet152 have different architectures but they share 50 common layers as well.

3.2 Potential Memory Savings with Merging

Figure 2 also depicts the memory shared across only single pairs of different models, i.e., translating layer sharing into memory savings. As expected, these pair-wise merging benefits can result in substantial memory savings in real-world workloads. The amount of memory saved depends on the number of models considered for merging and the number of layers that can be shared, and increases with they increase. To understand the potential savings that can be achieved using a merging approach, we study four typical edge workloads, analyzing video feeds from traffic cameras. In each workload, the edge server runs a mixture of the models shown in tables 1 and 2. The number of models in a workload range from 4 to 8. Table 3 shows the potential savings that can be obtained in these workloads: up to 74% savings if we were able to merge all common layers across all of the models in the workloads. Note that, beyond the alleviation of memory pressure on edge GPUs, merging benefits also reduce the costs associated with swapping by (1) enabling more models to be run without swapping, and (2) reducing the amount that must be loaded into memory when transitioning to another model that was merged with a previously loaded one.

Workload	No sharing (GB)	Max sharing (GB)	Savings (%)
1	2.02	0.656	67.5
2	0.93	0.244	73.8
3	1.212	0.896	26.1
4	0.13	0.090	30.76

Table 3. Potential of merging in four sample workloads.

4 Challenges in Model Merging

While seemingly simple, model merging poses a set of challenges that stand in the way of practically realizing the aforementioned benefits.

Challenge #1: Combinatorial search space

Models typically have many layers – e.g., YOLOv3 and Resnet152 have 106 and 152 layers, respectively – and determining the appropriate combination of layers to share in a way that maximizes memory savings while adhering to accuracy constraints is difficult for several reasons. First, there exists a combinatorial search space of potential layer sharing combinations for a given set of models. More formally, for a set of m models each with n layers, the number of possible layer combinations is $({}^{n}C_1)^m + ({}^{n}C_2)^m + ... + ({}^{n}C_n)^m$.

Second, even for different instances of the same model, the weights for shared layers are usually different, as models are usually specialized to a specific task (e.g., detect cars at 4th & Main). Keeping one copy of a layer means finding a set of weights for that layer that allows all models sharing it to maintain accuracy above a certain target. Unfortunately, as a result, the straightforward approach of simply sharing all possible layers and training them together results in unacceptable accuracy drops. For example, we jointly trained two Faster RCNN models that detected cars and people at two nearby intersections in the same city. When sharing all layers, neither model was able to reach even 75% of its pre-sharing accuracy.

We then randomly selected combinations of layers to share to see if there was potential for saving memory while meeting accuracy. Figure 4 shows the combinations and the amount of memory saved for each along with whether the models were able to reach 95% accuracy. The successes towards the right show that there is clearly potential for saving memory while meeting a realistic accuracy target. However, the scatter of successes and failures also shows that it is not trivial to find combinations of layers that train successfully. Exhaustively searching through all of the possible combinations in the combinatorial space to find the best combination that meets the accuracy would be computationally infeasible, especially as the number of layers in each model and the number of models that are merged increase.

Challenge #2: Retraining cost

For any given combination of layers to share, conclusively determining (1) whether it can meet an accuracy target, and

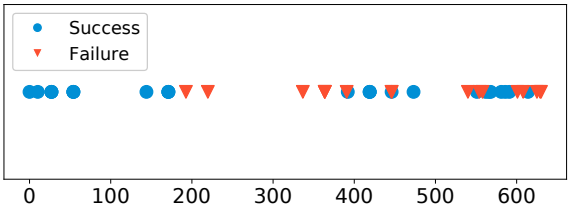

Figure 3. Memory savings and success in meeting 95% accuracy for random combinations of layers sampled from all shareable layers. There is potential to meet accuracy targets while saving memory, but many combinations fail.

if so, (2) the corresponding weights to use, requires retraining the merged models. The amount of training necessary depends on the number of layers being shared, and models merged. Such retraining costs can be prohibitive and discount the benefits of merging. For instance, in our earlier joint training example, each retraining epoch with Faster RCNN consumed ≈35 minutes, and different combinations of layer sharing required between 1-10 epochs to converge.

In video analytics deployments, it is typical for users to introduce and retire models periodically. Such scenarios also require retraining (e.g., when a new model is introduced). Hence, it is essential for the retraining process to be efficient.

We also note the difference in our task from multi-task learning, which enables the learning of multiple *related* tasks simultaneously [14], potentially with layer sharing [30, 32]. Multi-task learning is usually studied in the context of transfer learning, where one model does not have enough data to train on and trains together with another model, rather than two sets of pretrained weights that must be shared. Additionally, the related tasks are usually variations of the same model (e.g., spam classification) and thus the data for each task can be pooled together. In contrast, objectives vary widely among video analytics workloads at the edge, which consist of several tasks (e.g. object detection, classification), models (e.g. Resnet50, YOLOv3) and video streams.

Challenge #3: Incremental merging

These retraining delays are magnified by the fact that merged models may need to be retrained periodically to account for data drift [11, 27]. Data drift is common in real-time video analytic pipelines, where over time the live video data differs from the training data thereby degrading the performance of the models. The general approach to handling data drift is *continuous learning*, where the models are periodically retrained on new data.

In the case of model merging, handling data drift requires a different approach—it is not enough to just retrain the merged model on new data, but it is also necessary to assess the efficacy of the original merging decisions as the data changes. Thus, it is necessary for an efficient model merging solution to be *incremental* in nature, since naive approaches such as restarting the merging from scratch could be prohibitive.

Challenge #4: Merged model memory management

While a merged model can help reduce the memory requirements significantly, an efficient model merging scheme must be aware of the memory requirements of the merged models and account for scenarios where (1) the merged model may not fit in the memory of a given edge GPU, and (2) one or more of the models that are to be merged are large enough to not fit in the GPU by themselves.

We plan to investigate a number of ways to address them. The first case is more common, where a potential solution is to introduce a constraint in the merging process to account for the memory of the GPU where the merged model is to be deployed. As a result, the merging process could produce multiple (smaller) merged models which can then be swapped in and out using existing time-sharing techniques. The second case is more difficult to address. A possible approach here is to consider leveraging techniques that are able to load a model layer-by-layer, there by avoiding the need to load the model in its entirety.

Challenge #5: Joint optimization across cloud & edge

Finally, an efficient model merging solution must also consider jointly optimizing the model merging process with its deployment. Since the merging process is computationally heavy and time-consuming, it is natural to do it at the cloud, where the resources are plentiful. However, this means that the deployment of the merged model and the retraining and incremental merging process needs participation from the edge. One possible direction is to have a feedback channel between the cloud and the edge where the edge updates the merging engine of the need to retrain, and then design an intelligent scheduler which can incrementally load/swap the merged models based on the memory availability and real-time query requirements.

5 Final Thoughts & Future Directions

The significant memory saving potential of model merging makes it a compelling direction towards memory management at the edge. Here, we outline our vision towards a solution. We envision the solution to span the edge and the cloud, with the merging happening at the cloud using hints from the edge. To tackle the challenge of search space explosion, we plan on taking a greedy approach to deciding which layers to share. While sub-optimal, we believe it can still provide substantial savings. Retraining costs can potentially be reduced using variations of early-exit techniques [31] and/or sampling the data. We plan on employing light-weight checkpointing mechanism in the merging process to enable incremental merging. Finally, the edge can incorporate an intelligent scheduler that loads the merged models at a layer granularity, thus respecting the memory constraints in real-time. We are actively pursing all of these directions.

References

[1] 2021. https://www.microsoft.com/en-us/research/publication/traffic-video-analytics-case-study-report/

[2] 2021. https://techcommunity.microsoft.com/t5/internet-of-things/live-video-analytics-with-microsoft-rocket-for-reducing-edge/ba-p/1522305

[3] 2021. AWS Outposts. https://aws.amazon.com/outposts/

[4] 2021. Azure Stack Edge. https://azure.microsoft.com/en-us/services/databox/edge/

[5] 2021. CUDA Multi-Process Service. https://web.archive.org/web/20200228183056/https://docs.nvidia.com/deploy/mps/index.html

[6] 2021. Microsoft Rocket Video Analytics Platform. https://github.com/microsoft/Microsoft-Rocket-Video-Analytics-Platform

[7] 2021. NVIDIA JetsonNano. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/

[8] 2021. NVIDIA TensorRT. https://developer.nvidia.com/tensorrt

[9] 2021. PyTorch. https://pytorch.org/

[10] Ganesh Ananthanarayanan, Victor Bahl, Yuanchao Shu, Franz Loewenherz, Daniel Lai, Darcy Akers, Peiwei Cao, Fan Xia, Jiangbo Zhang, and Ashley Song. 2019. *Traffic Video Analytics - Case Study Report*. Technical Report MSR-TR-1970-3. Microsoft and City of Bellevue. https://www.microsoft.com/en-us/research/publication/traffic-video-analytics-case-study-report/

[11] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Nikolaos Karianakis, Yuanchao Shu, Kevin Hsieh, Victor Bahl, and Ion Stoica. 2020. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. arXiv:2012.10557 [cs.DC]

[12] Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos. 2015. Learning Complexity-Aware Cascades for Deep Pedestrian Detection. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15)*. IEEE Computer Society, Washington, DC, USA, 3361–3369. https://doi.org/10.1109/ICCV.2015.384

[13] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G. Andersen, Michael Kaminsky, and Subramanya R. Dulloor. 2019. Scaling Video Analytics on Constrained Edge Nodes. In *2nd SysML Conference*.

[14] Rich Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997), 41–75.

[15] John Emmons, Sadjad Fouladi, Ganesh Ananthanarayanan, Shivaram Venkataraman, Silvio Savarese, and Keith Winstein. 2019. Cracking Open the DNN Black-Box: Video Analytics with DNNs across the Camera-Cloud Boundary. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges* (Los Cabos, Mexico) *(HotEdgeVideo'19)*. Association for Computing Machinery, New York, NY, USA, 27–32. https://doi.org/10.1145/3349614.3356023

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. 2017. Mask R-CNN. *CoRR* abs/1703.06870 (2017). arXiv:1703.06870 http://arxiv.org/abs/1703.06870

[17] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. 2018. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 269–286. https://www.usenix.org/conference/osdi18/presentation/hsieh

[18] Chien-Chin Huang, Gu Jin, and Jinyang Li. 2020. SwapAdvisor: Pushing Deep Learning Beyond the GPU Memory Limit via Smart Swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 1341–1355. https://doi.org/10.1145/3373376.3378530

[19] C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose. 2018. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. 115–131. https://doi.org/10.1109/SEC.2018.00016

[20] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. 2020. Spatula: Efficient Cross-camera Video Analytics on Large Camera Networks. In *ACM/IEEE Symposium on Edge Computing (SEC)*.

[21] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Victor Bahl, and Joseph Gonzalez. 2020. Spatula: Efficient cross-camera video analytics on large camera networks. In *ACM/IEEE Symposium on Edge Computing (SEC 2020)*. https://www.microsoft.com/en-us/research/publication/spatula-efficient-cross-camera-video-analytics-on-large-camera-networks/

[22] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 947–960. https://www.usenix.org/conference/atc19/presentation/jeon

[23] Myeongjae Jeon, Shivaram Venkataraman, Junjie Qian, Amar Phanishayee, Wencong Xiao, and Fan Yang. 2018. Multi-tenant gpu clusters for deep learning workloads: Analysis and implications. *Technical report, Microsoft Research* (2018).

[24] Angela H. Jiang, Daniel L.-K. Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A. Kozuch, Padmanabhan Pillai, David G. Andersen, and Gregory R. Ganger. 2018. Mainstream: Dynamic Stem-Sharing for Multi-Tenant Video Processing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 29–42. https://www.usenix.org/conference/atc18/presentation/jiang

[25] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1586–1597. https://doi.org/10.14778/3137628.3137664

[26] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. 2015. A convolutional neural network cascade for face detection. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5325–5334.

[27] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 359–376. https://doi.org/10.1145/3387514.3405874

[28] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. 2017. Feature Pyramid Networks for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 936–944. https://doi.org/10.1109/CVPR.2017.106

[29] SB Shriram, Anshuj Garg, and Purushottam Kulkarni. 2019. Dynamic memory management for gpu-based training of deep neural networks. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 200–209.

[30] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. 2019. Adashare: Learning what to share for efficient deep multi-task learning. *arXiv preprint arXiv:1911.12423* (2019).

[31] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2464–2469.

[32] Simon Vandenhende, Stamatios Georgoulis, Bert De Brabandere, and Luc Van Gool. 2019. Branched multi-task networks: deciding what layers to share. *arXiv preprint arXiv:1904.02920* (2019).

[33] Junjue Wang, Ziqiang Feng, Shilpa George, Roger Iyengar, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2019. Towards Scalable Edge-Native Applications. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing* (Arlington, Virginia) *(SEC '19)*. Association for Computing Machinery, New York, NY, USA, 152–165. https://doi.org/10.1145/3318216.3363308

[34] Mengdi Wang, Chen Meng, Guoping Long, Chuan Wu, Jun Yang, Wei Lin, and Yangqing Jia. 2019. Characterizing Deep Learning Training Workloads on Alibaba-PAI. arXiv:1910.05930 [cs.PF]

[35] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 595–610. https://www.usenix.org/conference/osdi18/presentation/xiao

[36] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. 2017. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. 1–13.

[37] 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)*. USENIX Association, Boston, MA, 377–392. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang

[38] Tan Zhang, Aakanksha Chowdhery, Paramvir Bahl, Kyle Jamieson, and Suman Banerjee. 2015. The Design and Implementation of a Wireless Video Surveillance System. 426–438. https://doi.org/10.1145/2789168.2790123