ApproxNet: Content and Contention-Aware Video Object Classification System for Embedded Clients

RAN XU, Purdue University, USA
RAKESH KUMAR, Microsoft Corp, USA
PENGCHENG WANG, Purdue University, USA
PETER BAI, Purdue University, USA
GANGA MEGHANATH, Indian Institute of Technology Madras, India
SOMALI CHATERJI, Purdue University, USA
SUBRATA MITRA, Adobe Research, USA
SAURABH BAGCHI, Purdue University, USA

Videos take a lot of time to transport over the network, hence running analytics on the live video on embedded or mobile devices has become an important system driver. Considering such devices, e.g., surveillance cameras or AR/VR gadgets, are resource constrained, although there has been significant work in creating lightweight deep neural networks (DNNs) for such clients, none of these can adapt to changing runtime conditions, e.g., changes in resource availability on the device, the content characteristics, or requirements from the user. In this paper, we introduce ApproxNet, a video object classification system for embedded or mobile clients. It enables novel dynamic approximation techniques to achieve desired inference latency and accuracy trade-off under changing runtime conditions. It achieves this by enabling two approximation knobs within a single DNN model, rather than creating and maintaining an ensemble of models (e.g., MCDNN [MobiSys-16]). We show that ApproxNet can adapt seamlessly at runtime to these changes, provides low and stable latency for the image and video frame classification problems, and show the improvement in accuracy and latency over ResNet [CVPR-16], MCDNN [MobiSys-16], MobileNets [Google-17], NestDNN [MobiCom-18], and MSDNet [ICLR-18].

CCS Concepts: • **Computer systems organization → Embedded software**; *Real-time system architecture*; • **Computing methodologies → Computer vision**; **Machine learning**; **Concurrent algorithms**.

Additional Key Words and Phrases: Approximate computing, video analytics, object classification, deep convolutional neural networks

ACM Reference Format:
Ran Xu, Rakesh Kumar, Pengcheng Wang, Peter Bai, Ganga Meghanath, Somali Chaterji, Subrata Mitra, and Saurabh Bagchi. 2021. ApproxNet: Content and Contention-Aware Video Object Classification System for Embedded Clients. *ACM Trans. Sensor Netw.* 1, 1, Article 1 (January 2021), 27 pages. https://doi.org/10.1145/3463530

Authors' addresses: Ran Xu, Purdue University, 610 Purdue Mall, West Lafayette, Indiana, USA, 47907, xu943@purdue.edu; Rakesh Kumar, Microsoft Corp, One Microsoft Way, Redmond, Washington, USA, 98052, rakku@microsoft.com; Pengcheng Wang, Purdue University, USA, wang4495@purdue.edu; Peter Bai, Purdue University, USA, pbai@purdue.edu; Ganga Meghanath, Indian Institute of Technology Madras, Indian Institute Of Technology, Chennai, Tamil Nadu, India, 600036, gangamegha29@gmail.com; Somali Chaterji, Purdue University, USA, schaterji@purdue.edu; Subrata Mitra, Adobe Research, 345 Park Avenue, San Jose, California, USA, 95110-2704, subrata.mitra@adobe.com; Saurabh Bagchi, Purdue University, USA, sbagchi@purdue.edu.

1550-4859/2021/1-ART1 $15.00
https://doi.org/10.1145/3463530

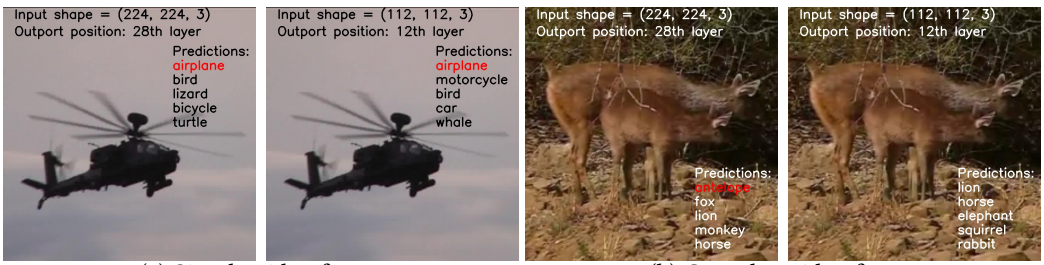
1 INTRODUCTION

There is an increasing number of scenarios where various kinds of analytics are required to be run on live video streams, on resource-constrained mobile and embedded devices. For example, in a smart city traffic system, vehicles are redirected by detecting congestion from the live video feeds from traffic cameras while in Augmented Reality (AR)/Virtual Reality (VR) systems, scenes are rendered based on the recognition of objects, faces or actions in the video. These applications require low latency for event classification or identification based on the content in the video frames. Most of these videos are captured at end-client devices such as IoT devices, surveillance cameras, or head-mounted AR/VR systems. Video transport over wireless network is slow and these applications often must operate under intermittent network connectivity. Hence such systems must be able to run video analytics in-place, on these resource-constrained client devices[1] to meet the low latency requirements for the applications.

State-of-the-art is too heavy for embedded devices: Most of the video analytics queries involve performing an inference over DNNs (mostly convolutional neural networks, *aka* CNNs) with a variety of functional architectures for performing the intended tasks like classification [22, 26, 68, 69], object detection [45, 62, 63, 66], face [57, 65, 70, 75] or action recognition [32, 43, 58, 67] etc. With advancements in deep learning and emergence of complex architectures, DNN-based models have become *deeper* and *wider*. Correspondingly their memory footprints and their inference latency have become significant. For example, DeepMon [28] runs the VGG-16 model at approximately 1-2 frames-per-second (fps) on a Samsung Galaxy S7. ResNet [22], with its 101-layer version, has a memory footprint of 2.8 GB and takes 101 ms to perform inference on a single video frame on the NVIDIA Jetson TX2. MCDNN, Mainstream, VideoStorm and Liu *et al.* [18, 33, 44, 83] require either the cloud or the edge servers to achieve satisfactory performance. Thus, on-device inference with a low and stable inference latency (i.e., 30 fps) remains a challenging task.

Content and contention aware systems: Content characteristics of the video stream is one of the key runtime conditions of the systems with respect to approximate video analytics. This can be leveraged to achieve the desired latency-accuracy tradeoff in the systems. For example, as shown in Figure 1, if the frame is very simple, we can downsample it to half of the original dimensions

[1]For end client devices, we will use the term "mobile devices" and "embedded devices" interchangeably. The common characteristic is that they are computationally constrained. While the exact specifications vary across these classes of devices, both are constrained enough that they cannot run streaming video analytics without approximation techniques.



(a) Simple video frame (b) Complex video frame

Fig. 1. Examples of using a heavy DNN (on the left) and a light DNN (on the right) for simple and complex video frames in a video frame classification task. The light DNN downsamples an input video frame to half the default input shape and gets prediction labels at an earlier layer. The classification is correct for the simple video frame (red label denotes the correct answer) but not for the complex video frame.

and use a shallow or small DNN model to make an accurate prediction. If the frame is complex, the same shallow model might result in a wrong prediction and would need a larger DNN model.

Resource contention is another important runtime condition that the video analytics system should be aware of. In several scenarios, the mobile devices support multiple different applications, executing concurrently. For example, while an AR application is running, a voice assistant might kick in if it detects background conversation, or a spam filter might become active, if emails are received. All these applications share common resources on the device, such as, CPU, GPU, memory, and memory bandwidth, and thus lead to *resource contention* [1, 2, 37] as these devices do not have advanced resource isolation mechanisms. It is currently an unsolved problem how video analytics systems running on the mobile devices can maintain a low inference latency under such variable resource availability and changing content characteristics, so as to deliver satisfactory user experience.

Single-model vs. multi-model adaptive systems: How do we architect the system to operate under such varied runtime conditions? Multi-model designs came first in the evolution of systems in this space. These created systems with an ensemble of multiple models, satisfying varying latency-accuracy conditions, and some scheduling policy to choose among the models. MCDNN [18], being one of most representative works, and well-known DNNs like ResNet, MobileNets [22, 23] all fall into this category. On the other hand, the single-model designs, which emerged after the multi-model designs, feature one model with tuning knobs inside so as to achieve different latency-accuracy goals. These typically have lower switching overheads from one execution path to another, compared to the multi-branch models. MSDNet [25], BranchyNet [71], and NestDNN [11] are representative works in this single-model category. However, none of these systems can adapt to runtime conditions, primarily, changes in content characteristics and contention levels on the device.

Our solution: ApproxNet. In this paper, we present ApproxNet, our content and contention aware object classification system over streaming videos, geared toward GPU-enabled mobile/embedded devices. We introduce a novel workflow with a set of integrated techniques to solve the three main challenges as mentioned above: (1) on-device real-time video analytics, (2) content and contention aware runtime calibration, (3) a single-model design. The fundamental idea behind ApproxNet is to perform approximate computing with tuning knobs that are changed automatically and seamlessly

Table 1. ApproxNet's main features and comparison to existing systems.

Solution	Single model	Considers switching overhead	Focused on video	Handles runtime conditions	Open-sourced	Replicable in our datasets
MCDNN [MobiSys'16]	✗	◐	✗	✗	◐	✓
MobileNets [ArXiv'17]	✗	✗	✗	✗	✓	✓
MSDNet [ICLR'18]	✓	✗	✗	✗	✓	✓
BranchyNet [ICPR'16]	✓	✗	✗	✗	✓	✗
NestDNN [MobiCom'18]	✓	◐	✗	◐	✗	✗
ApproxNet	✓	✓	✓	✓	✓	✓
✓ Supported ◐ Partially Supported ✗ Not Supported						
Notes for partially support: 1. MCDNN and NestDNN only consider the switching overhead in memory size, but in the execution latency. 2. NestDNN handles multiple concurrent DNN applications with joint optimization goals. 3. The core models in MCDNN are open-sourced while the scheduling components are not.						

within the same video stream. These knobs trade off the accuracy of the inferencing for reducing inference latency and thus match the frame rate of the video stream or the user's requirement on either a latency target or an accuracy target. The optimal configuration of the knobs is set, particularly considering the resource contention and complexity of the video frames, because these runtime conditions affects the accuracy and latency of the model much.

In Table 1, we compare ApproxNet with various representative prior works in this field. First of all, none of these systems [11, 18, 23, 25, 71] is able to adapt to dynamic runtime conditions (changes in content characteristics and contention levels) as we can. Second, although most systems are able to run at variable operation points of performance, MCDNN [18] and MobileNets [23] use a multi-model approach and incur high switching penalty. For those that works with a single model, namely, MSDNet [25], BranchyNet [71], and NestDNN [11], they do not consider switching overheads in their models (except partially for NestDNN, which considers switching cost in memory size), do not focus on video content, and do not show how their models can adapt to the changing runtime conditions (except partially for NestDNN, which considers joint optimization of multiple DNN workloads). For evaluation, we mainly compare to MCDNN, ResNet, and MobileNets, as the representatives of multi-model approaches, and MSDNet, as the single-model approach. We cannot compare to BranchyNet as it is not designed and evaluated for video analytics and thus not suitable for our datasets. BranchyNet paper evaluates it on small image dataset: MNIST and CIFAR. We cannot compare to NestDNN since it's models or source-code and architecture and hyperparameter details are publicly available and we need those for replicating the experiments.

To summarize, we make the following contributions in this paper:

(1) We develop an end-to-end, approximate video object classification system, ApproxNet, that can handle dynamically changing workload contention and video content characteristics on resource-constrained embedded devices. It achieves this through performing system context-aware and content-aware approximations with the offline profiling and online lightweight sensing and scheduling techniques.
(2) We design a novel workflow with a set of integrated techniques including the adaptive DNN that allows runtime accuracy and latency tuning *within a single model*. Our design is in contrast to ensemble systems like MCDNN that are composed of multiple independent model variants capable of satisfying different requirements. Our single-model design avoids high switching latency when conditions change and reduces RAM and storage usage.
(3) We design ApproxNet to make use of video features, rather than treating video as a sequence of image frames. Such characteristics that we leverage include the temporal continuity in content characteristics between adjacent frames. We empirically show that on a large-scale video object classification dataset, popular in the vision community, ApproxNet achieves a superior accuracy-latency tradeoff than the three state-of-the-art solutions on mobile devices, MobileNets, MCDNN, and MSDNet (Figures 9 and 10).

The rest of the paper is organized as follows. Section 2 gives the relevant background. Section 3 gives our high-level solution overview. Section 4 gives the detailed design. Section 5 evaluates our end-to-end system. Section 6 discusses the details about training the DNN. Section 7 highlights the related works. Finally, Section 8 gives concluding remarks.

2 BACKGROUND AND MOTIVATION

2.1 DNNs for Streaming Video Analytics

DNNs have become a core element of various video processing tasks such as frame classification, human action recognition, object detection, face recognition, and so on. Though accurate, DNNs are computationally expensive, requiring significant CPU and memory resources. As a result, these

DNNs are often too slow when running on mobile devices and become the latency bottleneck in video analytics systems. Huynh *et al.* [28] experimented with VGG [68] of 16 layers on the Samsung Galaxy S7 and noted that classification on a single image takes as long as 644 ms, leading to less than 2 fps for continuous classification. Motivated by the observation, we explore in ApproxNet how we can make DNN-based video analytics pipelines more efficient through content-aware approximate computation *within the neural network*.

ResNet: Deep DNNs are typically hard to train due to the vanishing gradient problem [22]. ResNet solved this problem by introducing a short cut identity connection in between layers, which helps achieve at least the same accuracy upon further increasing the number of network layers. The unit of such connected layers is called a ResNet block. We leverage this key idea of a deeper model producing no higher error than its shallower counterpart, for the construction of an architecture that provides more accurate execution as it proceeds deeper into the DNN.

Spatial Pyramid Pooling (SPP) [21]: Popular DNN models, including ResNet, consist of convolutional and max-pooling (CONV) layers and fully-connected (FC) layers and the shape of an input image is fixed. Changing the input shape in a CNN typically requires re-designing the architecture of the CNN. SPP is a special layer that eliminates this constraint. The SPP layer is added at the end of CONV layers, providing the following FC layers with a fixed-dimensioned feature representation by pooling the CONV layer output with bins whose shapes are proportional to the input shape. We use SPP layers to change input shape as an approximation knob.

Trading-off accuracy for inference latency: DNNs can have several variants due to different configurations, and these variants yield different accuracies and latencies. But these variants are trained and inferenced independently and cannot be switched efficiently at inference time to meet differing accuracy or latency requirements. For example, MCDNN [18] sets up an ensemble of (up to 68) model variants to satisfy different latency/accuracy/cost requirements. MSDNet [25] enables five early exits in a *single* model but does not evaluate on streaming video with any variable content or contention situations. Hence, we set ourselves to design a single-model DNN that is capable of handling the accuracy-latency trade-off at inference time and guarantees our video analytics system's performance under variable content and runtime conditions.

2.2 Content-aware Approximate Computing

IRA [40] and VideoChef [77] first introduced the notion of content-aware approximation and applied the idea, respectively to image and video processing pipelines. These works for the first time showed how to tune approximation knobs as content characteristics change, e.g., the video scene became more complex. In particular, IRA performs approximation targeting individual images, while VideoChef exploits temporal similarity among frames in a video to further optimize accuracy-latency trade-off. However, these works do not perform approximation for ML-based inferencing, which comprises the dominant form of video analytics. In contrast, we apply approximation to the DNN model itself with the intuition that depending on complexity of the video frame, we want to feed input of a different shape and output at a different depth of layers to achieve the target accuracy.

2.3 Contention-aware Scheduling

Managing the resource contention of multiple jobs on high-performance clusters is a very active area of work. Bubble-Up [52], Bubble-Flux [81], and Pythia [78] develop characterization methodologies to predict the performance degradation of latency-sensitive applications due to shared resources in the memory subsystem. SMiTe [86] and Paragon [8] further extend such concurrent resource contention scenario to SMT processors and thousands of different unknown applications, respectively. On the other hand, we apply contention-aware approximation to the DNN model on

the embedded and mobile devices, and consider the three major sources of contention – CPU, GPU, and memory bandwidth.

3 OVERVIEW

Here we give a high-level overview of ApproxNet. In Section 4, we provide details of each component.

3.1 Design Principles and Motivation

We set four design requirements for streaming video analytics on the embedded devices motivated by real-world scenarios and needs. *First*, the application should adapt to changing input characteristics, such as, the complexity of the video frames because the accuracy of the DNN may vary based on the content characteristics. We find such changes happen often enough within a single video stream and without any clear predictive pattern. *Second*, the application should adapt to the resource contention due to the shared CPU, GPU, memory, or memory bandwidth with other concurrent applications on the same device. Such contention can happen frequently with co-location due to limited resources and the lack of clean resource isolation on these hardware platforms. Again, we find that such changes can happen without a clear predictive pattern. *Third*, the application should support different target accuracies or latencies at runtime with little transition overhead. For example, the application may require low latency when a time-critical query, such as detection of a miscreant, needs to be executed and have no such constraint for other queries on the steam. Thus, the aggregate model must be able to make efficient transitions in the tradeoff space of accuracy and latency, and less obviously throughput, optionally using edge or cloud servers. *Fourth*, the application must provide real-time processing speed (30 fps) while running on the mobile/embedded device. To see three instances where these four requirements come together, consider mobile VR/AR games like Pokemon Go (some game consoles support multitasking, accuracy requirements may change with the context of the game), autonomous vehicles (feeds from multiple cameras are processed on the same hardware platform resulting in contention, emergency situations require lower latency than benign conditions such as for fuel efficiency) and autonomous drones (same arguments as for autonomous vehicles).

A *non-requirement* in our work is that multiple concurrent applications consuming the same video stream be jointly optimized. MCDNN [18], NestDNN [11], and Mainstream [33] bring significant design sophistication to handle the concurrency aspect. However, we are only interested in optimizing a single video analytics application.

3.2 Design Intuition and Workflow

To address these challenges in our design, we propose a novel workflow with a set of integrated techniques to achieve a content and contention-aware video object classification system. We show the overall structure with three major functional units: *executor*, *profiler*, and *scheduler* in Figure 2. ApproxNet takes a video frame and optional user requirement for target accuracy or latency as an input, and produces top-5 prediction labels of the object classes as outputs.

The executor (Section 4.1) is an approximation-enabled, single-model DNN. Specifically, the single-model design largely reduces the switching overhead so as to support the adaptive system. On the other hand, the multiple **approximation branches (ABs)**, each with variable latency and accuracy specs, are the key to support dynamic content and contention conditions. ApproxNet is designed to provide real-time processing speed (30 fps) on our target device (NVIDIA Jetson TX2). Compared to the previous single-model designs like MSDNet and BranchyNet, ApproxNet provides novelty in enabling both depth and shape as the approximation knob for run-time calibration.

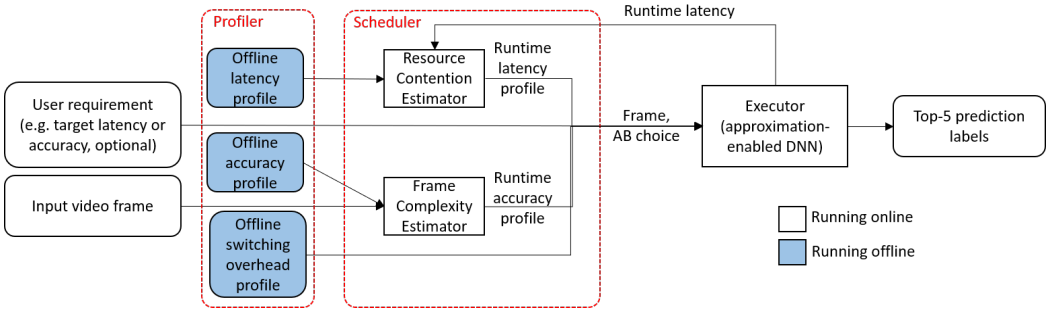


Fig. 2. Workflow of ApproxNet. The input is a video frame and an optional user requirement, and the outputs are prediction labels of the video frame. Note that the shaded profiles are collected offline to alleviate the online scheduler overhead.

The scheduler is the key component to react to the dynamic content characteristics and resource contention. Specifically, it selects an AB to execute by combining the precise accuracy estimation of each AB due to changing content characteristics via a **Frame Complexity Estimator** (**FCE**, Section 4.2), the precise latency estimation of each AB due to resource contention via a **Resource Contention Estimator** (**RCE**, Section 4.3), and the switching overhead among ABs (Section 4.4). It finally reaches a decision on which AB to use based on the user's latency or accuracy requirement and its internal accuracy, latency, and overhead estimation (Section 4.5).

Finally, to achieve our goal of real-time processing, low switching overhead, and improved performance under dynamic conditions, we design an offline profiler (Section 4.4). We collect three profiles offline — first, the accuracy profile for each AB on video frames of different complexity categories; second, the inference latency profile for each AB under variable resource contention, and third, the switching overhead between any two ABs.

Video-specific design. We incorporate these video-specific designs in ApproxNet, which is orthogonal to the existing techniques presented in the prior works on video analytics, e.g. frame sampling [33, 83] and edge device offloading [44].

(1) The FCE uses a Scene Change Detector (SCD) as a preprocessing module to further alleviate its online cost. This optimization is beneficial because it reduces the frequency with which the FCE is invoked (only when the SCD flags a scene change). This relies on the intuition that discontinuous jumps in frame complexity are uncommon in a video stream.
(2) The scheduler decides whether to switch to a new AB or stay depending on how long it predicts the change in the video stream to last and the cost of switching.
(3) We drive our decisions about the approximation knobs by the goal of keeping up with the streaming video rate (30 fps). We achieve this under most scenarios when evaluated with a comprehensive video dataset (the ILSVRC VID dataset).

4 DESIGN AND IMPLEMENTATION

4.1 Approximation-enabled DNN

ApproxNet's key enabler, an approximation-enabled DNN, is designed to support multiple accuracy and latency requirements at runtime *using a single DNN model*. To enable this, we design a DNN that can be approximated using two approximation knobs. The DNN can take an input video frame in different shapes, which we call *input shapes*, our first approximation knob and it can produce a classification output at multiple positions in the intervening layers, which we call

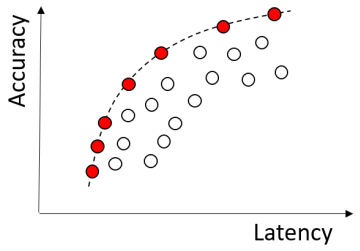


Fig. 3. A Pareto frontier for trading-off accuracy and latency in a particular frame complexity category and at a particular contention level.

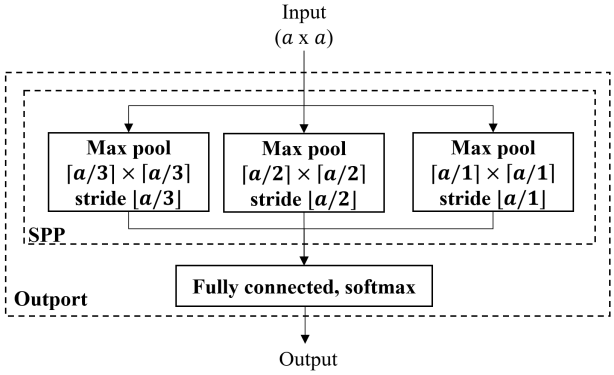
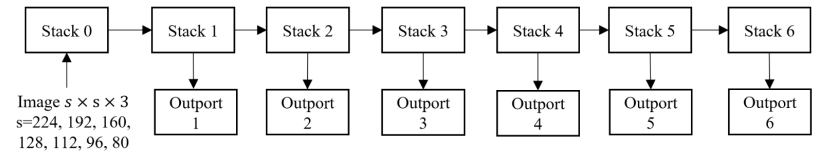


Fig. 4. The outport of the approximation-enabled DNN.



Stack 0	Stack 1	Stack 2	Stack 3	Stack 4	Stack 5	Stack 6
Conv, 7x7, 64 stride 2 Relu	ResNetBlock 3x3, 128 stride 2	ResNetBlock 3x3, 128 stride 1	ResNetBlock 3x3, 256 stride 2	ResNetBlock 3x3, 256 stride 1	ResNetBlock 3x3, 256 stride 1	ResNetBlock 3x3, 512 stride 2
Max pool 3x3 stride 2	ResNetBlock 3x3, 128 stride 1	ResNetBlock 3x3, 128 stride 1	ResNetBlock 3x3, 256 stride 1	ResNetBlock 3x3, 256 stride 1	ResNetBlock 3x3, 256 stride 1	ResNetBlock 3x3, 512 stride 1
ResNetBlock*3 3x3, 64 stride 1						ResNetBlock 3x3, 512 stride 1

Fig. 5. The architecture of the approximation-enabled DNN in ApproxNet.

outports, our second approximation knob. There are doubtless other approximation knobs, e.g., model quantization, frame sampling, and others depending on specific DNN models. These can be incorporated into ApproxNet and they will all fit within our novelty of the one-model DNN to achieve real-time on-device, adaptive inference. The appropriate setting of the tuning knobs can be determined on the device (as is done in our considered usage scenario) or, in case this computation is too heavyweight, this can be determined remotely and sent to the device through a reprogramming mechanism such as [54].

Combining these two approximation knobs, ApproxNet creates various **approximation branches (ABs)**, which trade off between accuracy and latency, and can be used to meet a particular user requirement. This tradeoff space defines a set of Pareto optimal frontiers, as shown in Figure 3. Here, the scatter points represent the accuracy and latency achieved by all ABs. A Pareto frontier defines the ABs which are either superior in accuracy or latency against all other branches.

We describe our design using ResNet as the base DNN, though our design is applicable to any other mainstream CNN consisting of convolutional (CONV) layers and fully-connected (FC) layers such as VGG [68], DenseNet [26] and so on. Figure 5 shows the design of our DNN using ResNet-34 as the base model. This enables 7 input shapes ($s \times s \times 3$ for $s = 224, 192, 160, 128, 112, 96, 80$) and

Table 2. The list of the total 30 ABs supported for a baseline DNN of ResNet-34, given by the combination of the input shape and the outport from which the result is taken. "–" denotes the undefined settings.

Input shape	Outport 1	Outport 2	Outport 3	Outport 4	Outport 5	Outport 6
224x224x3	28x28x64	28x28x64	14x14x64	14x14x64	14x14x64	7x7x64
192x192x3	24x24x64	24x24x64	12x12x64	12x12x64	12x12x64	–
160x160x3	20x20x64	20x20x64	10x10x64	10x10x64	10x10x64	–
128x128x3	16x16x64	16x16x64	8x8x64	8x8x64	8x8x64	–
112x112x3	14x14x64	14x14x64	7x7x64	7x7x64	7x7x64	–
96x96x3	12x12x64	12x12x64	–	–	–	–
80x80x3	10x10x64	10x10x64	–	–	–	–

6 outports (after 11, 15, 19, 23, 27, and 33 layers). We adapt the design of ResNet in terms of the stride, shape, number of channels, use of convolutional layer or maxpool, and connection of the layers. In addition, we create *stacks*, with stacks numbering 0 through 6 and each stack having 4 or 6 ResNet layers and a variable number of blocks from the original ResNet design (Table 2). We then design an outport (Figure 4), and connect with stacks 1 to 6, whereby we can get prediction labels by executing only the stacks (i.e., the constituent layers) till that stack. The use of 6 outports is a pragmatic system choice—too small a number does not provide enough granularity to approximate in a content and contention-aware manner and too many leads to a high training burden. Further, to allow the approximation knob of downsampling the input frame to the DNN, we use the SPP layer at each outport to pool the feature maps of different shapes (due to different input shapes) into one unified shape and then connect with an FC layer. The SPP layer performs max-pooling on its input by three different levels $l = 1, 2, 3$ with window size $\lceil a/l \rceil$ and stride $\lfloor a/l \rfloor$, where a is the shape of the input to the SPP layer. Note that our choice of the 3-level pyramid pooling is a typical practice for using the SPP layer [21]. In general, a higher value of l requires a larger value of a on the input of each outport, thereby reducing the number of possible ABs. On the other hand, a smaller value of l results in coarser representations of spatial features and thus reduces accuracy. To support the case $l = 3$ in the SPP, we require that the input shape of an outport be no less than 7 pixels in width and height, *i.e.*, $a \geq 7$. This results in ruling out some input shapes as in Table 2. Our model has 30 configuration settings in total, instead of 7×6 (number of input shapes $\times$ number of outports) because too small input shapes cannot be used when the outport is deep.

To train ApproxNet towards finding the optimal parameter set θ, we consider the softmax loss $L_{s,i}(\theta)$ defined for the input shape $s \times s \times 3$ and the outport i. The total loss function $L(\theta)$ that we minimize to train ApproxNet is a weighted average of $L_{s,i}(\theta)$ for all s and i, defined as

$$L(\theta) = \sum_{\forall i} \frac{1}{n_i} \sum_{\forall s} L_{s,i}(\theta) \tag{1}$$

where the value of n_i is the factor that normalizes the loss at an outport by dividing by the number of shapes that are supported at that port i. This makes each outport equally important in the total loss function. For mini-batch, we use 64 frames for each of the 7 different shapes. To train on a particular dataset or generalize to other architectures, we discuss more details in Section 6.

4.2 Frame Complexity Estimator (FCE)

The design goal of the Frame Complexity Estimator (FCE), which executes online, is to estimate the expected accuracy of each AB in a content-aware manner. It is composed of a Frame Complexity Categorizer (FCC) and a Scene Change Detector (SCD) and it makes use of information collected by the offline profiler (described in Section 4.4). The workflow of the FCE is shown in Figure 6.

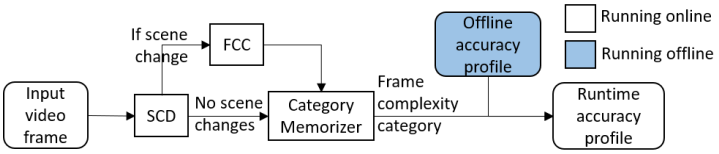


Fig. 6. Workflow of the Frame Complexity Estimator.

Fig. 7. Sample frames (first row) and edge maps (second row), going from left to right as simple to complex. Normalized mean edge values from left to right: 0.03, 0.24, 0.50, and 0.99 with corresponding frame complexity categories: 1, 3, 6, and 7

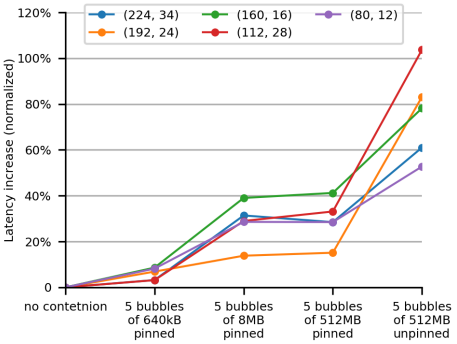


Fig. 8. Latency increase of several ABs in ApproxNet under resource contention with respect to those under no contention. The input shape and outport depth of the branches are labeled.

Frame Complexity Categorizer (FCC). FCC determines how hard it is for ApproxNet to classify a frame of the video. Various methods have been used in the literature to calculate frame complexity such as edge information-based methods [51, 82], compression information-based methods [82] and entropy-based methods [4]. In this paper, we use mean edge value as the feature of the frame complexity category, since it can be calculated with very low computation overhead (3.9 ms per frame on average in our implementation). Although some counterexamples may show the edge value is not relevant, we show empirically that with this feature, the FCE is able to predict well the accuracy of each AB with respect to a large dataset.

To expand, we extract an edge map by converting a color frame to a gray-scale frame, applying the Scharr operator [31] in both horizontal and vertical directions, and then calculating the L2 norm in both directions. We then compute the mean edge value of the edge map and use a pre-trained set of boundaries to quantize it into several frame complexity categories. The number and boundaries of categories is discussed in Section 4.4. Figure 7 shows examples of frames and their edge maps from a few different complexity categories.

Scene Change Detector (SCD). The Scene Change Detector is designed to further reduce the online overhead of FCC by determining if the content in a frame is significantly different from that in a prior frame in which case the FCC will be invoked. SCD tracks a histogram of pixel values, and declares a scene change when the mean of the absolute difference across all bins of the histograms of two consecutive frames is greater than a certain threshold (45% of the total pixels in our design). To bound the execution time of SCD we use only the R-channel and downsample the shape of the frame to 112×112. We empirically find that such optimizations do not reduce the accuracy of detecting new scenes but do reduce the SCD overhead, to only 1.3 ms per frame.

4.3 Resource Contention Estimator (RCE)

The design goal of the Resource Contention Estimator (RCE), which also executes online, is to estimate the expected latency of each AB in a contention-aware manner. Under resource contention, each AB is affected differently and we call the latency increase pattern the *latency sensitivity*. As shown in Figure 8, five approximation branches have different ratios of latency increase under a certain amount of CPU and memory bandwidth contention.

Ideally we would use a sample classification task to probe the system and observe its latency under the current contention level C. The use of such micro-benchmarks is commonly done in datacenter environments [46, 78]. However, we do not need the additional probing since the inference latencies of the latest video frames form a natural observation of the contention level of the system. Thus we use the averaged inference latency $\overline{L_B}$ of the current AB B across the latest N frames. We then check the latency sensitivity $L_{B,C}$ of branch B (offline profile as discussed in Section 4.4) and get an estimated contention level $\hat{C}$ with the nearest neighbor principle,

$$\hat{C} = \underset{C}{\operatorname{argmin}}\, abs(L_{B,C} - \overline{L_B}) \tag{2}$$

By default, we use $N = 30$. This will lead to an average over last one second when frame-per-second is 30. Smaller N can make ApproxNet adapt faster to the resource contention, while larger N make it more robust to the noise. Due to the limited observation data (one data point per frame), we cannot adapt to resource contention that is changing faster than the frame rate.

Specifically in this work, we consider CPU, GPU and memory contention among tasks executing on the device (our SoC board shares the memory between the CPU and the GPU), but our design is agnostic to what causes the contention. Our methodology considers the resource contention as a black-box model because we position ourselves as an application-level design instead of knowing the execution details of all other applications. We want to deal with the effect of contention, rather than mitigating it by modifying the source of the contention.

4.4 Offline Profiler

Per-AB Content-Aware Accuracy Profile. The boundaries of frame complexity categories are determined based on the criteria that all frames within a category should have an identical Pareto frontier curve (Figure 3) and frames in different categories should have distinct curves. We start with considering the whole set of frames as belonging to a single category and split the range of mean edge values into two in an iterative binary manner, till the above condition is satisfied. In our video datasets, we derive 7 frame complexity categories with 1 being the simplest and 7 the most complex. To speedup the online estimation of the accuracy on any candidate approximation branch, we create the offline accuracy profile $A_{B,F}$ given any frame complexity categories F and any ABs B, after the 7 frame complexity categories are derived.

Per-AB Contention-Aware Latency Profile. ApproxNet is able to select ABs at runtime in the face of resource contention. Therefore, we perform offline profiling of the inference latency of each AB under different levels of contention. To study the resource contention, we develop our synthetic contention generator (CG) with tunable "contention levels" to simulate resource contention and help our ApproxNet profile and learn to react under such scenarios in real-life. Specifically, we run each AB in ApproxNet with the CG in varying contention levels to collect its contention-aware latency profile. To reduce the profiling cost, we quantize the contention to 10 levels for GPU and 20 levels for CPU and memory and then create the offline latency profile $L_{B,C}$ for each AB B under each contention level C. Note that contention increases latency of the DNN but does not affect its accuracy. Thus, offline profiling for accuracy and latency can be done independently and parallelly and profiling overhead can be reduced.

Switching Overhead Profile. Since we find that the overhead of switching between some pairs of ABs is non-negligible, we profile the overhead of switching latency between any pairs of approximation branches offline. This cost is used in our optimization calculation to select the best AB.

4.5 Scheduler

The main job of the scheduler in ApproxNet is to select an AB to execute. The scheduler accepts user requirement on either the minimum accuracy, the maximum latency per frame. The scheduler requests from the FCE a runtime accuracy profile (B is the variable for the AB and $\hat{F}$ is the frame category of the input video frame) $A_{B,\hat{F}} \forall B$. It then requests from the RCE a runtime latency profile ($\hat{C}$ is the current contention level) $L_{B,\hat{C}} \forall B$. Given a target accuracy or latency requirement, we can easily select the AB to use from drawing the Pareto frontier for the current $(\hat{F}, \hat{C})$. If no Pareto frontier point satisfies the user requirement, ApproxNet picks the AB that achieves metric value closest to the user requirement. If the user does not set any requirement, ApproxNet sets a latency requirement to the frame interval of the incoming video stream. One subtlety arises due to the cost of switching from one AB to another. This cost has to be considered by the scheduler to avoid too frequent switches without benefit to outweigh the cost.

To rigorously formulate the problem, we denote the set of ABs as $\mathcal{B} = \{B_1, ...B_N\}$ and the optimal AB the scheduler has to determine as B_{opt}. We denote the accuracy of branch B on a video frame with frame complexity F as $A_{B,F}$, the estimated latency of branch B under contention level C as $L_{B,C}$, the one-time switch latency from branch B_p to B_{opt} as $L_{B_p \to B_{opt}}$, and the expected time window over which this AB can be used as W (in the unit of frames). For W, we use the average number of frames for which the latest ABs can stay unchanged and this term introduces hysteresis to the system so that the AB does not switch back and forth frequently. The constant system overhead per frame (due to SCD, FCC, and resizing the frame) is L_0. Thus, the optimal branch B_{opt}, given the latency requirement L_τ, is:

$$B_{opt} = \underset{B \in \mathcal{B}}{\operatorname{argmax}} A_{B,F}, \; s.t. \; L_{B,C} + \frac{1}{W} L_{B_p \to B} + L_0 \leq L_\tau \tag{3}$$

When the accuracy requirement A_τ is given,

$$B_{opt} = \underset{B \in \mathcal{B}}{\operatorname{argmin}} [L_{B,C} + \frac{1}{W} L_{B_p \to B} + L_0], s.t. \; A_{B,F} \geq A_\tau \tag{4}$$

5 EVALUATION

5.1 Evaluation Platforms

We evaluate ApproxNet by running it on an NVIDIA Jetson TX2 [7], which includes 256 NVIDIA Pascal CUDA cores, a dual-core Denver CPU, a quad-core ARM CPU on a 8GB unified memory [20] between CPU and GPU. The specification of this board is close to what is available in today's high-end smart phones such as Samsung Galaxy S20 and Apple iPhone 12. We train the approximation-enabled DNN on a server with NVIDIA Tesla K40c GPU with 12GB dedicated memory and an octa-core Intel i7-2600 CPU with 24GB RAM. For both the embedded device and the training server, we install Ubuntu 16.04 and TensorFlow v1.14.

5.2 Datasets, Task, and Metrics

5.2.1 ImageNet VID dataset. We evaluate ApproxNet on the video object classification task using ILSVRC 2015 VID dataset [64]. Although the dataset is initially used for object detection, we convert

the dataset so that the task is to classify the frame into one of the ground truth object categories. If multiple objects exist, the classification is considered correct if matched with any one of the ground truth classes and this rule applies to both ApproxNet and baselines. According to our analysis, 89% of the video frames are single-object-class frames and thus the accuracy is still meaningful under such conversion.

For the purpose of training, ILSVRC 2015 VID training set contains too many redundant video frames, leading to an over-fitting issue. To alleviate this problem, we follow the best practice in [36] such that the VID training dataset is sub-sampled every 180 frames and the resulting subset is mixed with ILSVRC 2014 detection (DET) training dataset to construct a new dataset with DET:VID=2:1. We use 90% of this video dataset to train ApproxNet's DNN model and keep aside another 10% as validation set to fine-tune ApproxNet (offline profiling). To evaluate ApproxNet's system performance, we use ILSVRC 2015 VID validation set – we refer to this as the "test set" throughout the paper.

5.2.2 ImageNet IMG dataset. We also use ILSVRC 2012 image classification dataset [9] to evaluate the accuracy-latency trade-off of our single DNN. We use 10% of the ILSVRC training set as our training set, first 50% of the validation set as our validation set to fine-tune ApproxNet, and the remaining 50% of the validation set as our test set. The choices made for training-validation-test in both the datasets follows common practice and there is no overlap between the three. **Throughout the evaluation, we use ImageNet VID dataset by default, unless we explicitly mention the use of the ImageNet IMG dataset.**

5.2.3 Metrics. We use latency and top-5 accuracy as the two metrics. The latency includes the overheads of the respective solutions, including the switching overhead, the execution time of FCE, RCE and scheduler.

5.3 Baselines

We start with the evaluation on the static models without the ability to adapt. This is because we want to reveal the relative accuracy-latency trade-offs in the traditional settings, compared to the single approximation branches in ApproxNet. The baselines for this static experiment include model variants, which are designed for different accuracy and latency goals, i.e., ResNet [22] MobileNets [23] and MSDNets [25] for which we use 5 execution branches in the single model that can provide different accuracy and latency goals. We use the ILSVRC IMG dataset to evaluate these static models, since this dataset is larger and with more classes.

We then proceed with the evaluation on the streaming videos, under varying resource contention. This brings two additional aspects for evaluation – (1) how the video frames are being processed in a timely and streaming manner as frames in this case cannot be batched like images, and (2) how the technique can meet the latency budget in the presence of resource contention from other application that can raise the processing latency. The baselines we use are: MCDNN [18] as a representative of the multi-model approach, and MSDNets, a representative of the single-model approach (with multiple execution branches). We also compare the switching overhead of our single-model design with the multi-capacity models in NestDNN [11]. Unfortunately, we were not able to use BranchyNet [71], because their DNN is not designed for large images in the ImageNet dataset. BranchyNet was evaluated on MNIST and CIFAR datasets in thir paper and it does not provide any guidance on the parameter settings for training and makes it impossible to use on different datasets.

The details of each baseline are as follows,

ResNet: ResNet is the base DNN architecture of many state-of-the-art image and video object classification tasks, with superior accuracy to other architectures. While it was originally meant

for server-class platforms, as resources on mobile devices increase, ResNet is also being used on such devices [47, 73, 85]. We use ResNet of 18 layers (ResNet-18) and of 34 layers (ResNet-34) as base models. We modify the last FC layer to classify into 30 labels in the VID dataset and fine-tune the whole model. ResNet-34 plays a role as the reference providing the upper bound of the target accuracy. ResNet architectures with more than 34 layers ([22] has considered up to 152 layers) become impractical as they are too slow to run on the resource-constrained mobile devices and their memory consumption is too large for the memory on the board.

MobileNets: This refers to 20 model variants (trained by the original authors) specifically designed for mobile devices ($\alpha = 1, 0.75, 0.5, 0.35, shape = 224, 192, 160, 128, 96$).

MSDNets: This refers to the 5 static execution branches to meet the different latency budgets in their anytime evaluation scenario. We have enhanced MSDNets with a scheduler to dynamically choose the static branches for dynamic runtime conditions. The former is compared with static models in the IMG dataset and the latter is compared with adaptive systems in the VID dataset. For the sake of simplicity, we reuse the same term (MSDNets) to refer to both.

NestDNN: This solution provides multi-capacity models with ResNet-34 architecture by varying the number of filters in each convolutional layer. We compose 9 descendant models, where (1) the seed model, or the smallest model, reduces the number of filters of all convolutional layers uniformly by 50%, (2) the largest descendant model is exactly ResNet of 34 layers, and (3) the other descendant models reduce the number of filters of all convolutional layers uniformly by a ratio equally distributed between 50% and 100%. We only compare the switching overhead of NestDNN inside its descendant models with ApproxNet because NestDNN is not open-sourced and the paper does not provide enough details about the training process or the architecture.

MCDNN: We change the base model in MCDNN from VGG to the more recent ResNet for a fairer comparison. This system chooses between MCDNN-18 and MCDNN-34 depending on the accuracy requirement. MCDNN-18 uses two models: a specialized ResNet-18 followed by the generic ResNet-18. The specialized ResNet-18 is the same as the ResNet-18 except the last layer, which is modified to classify the most frequent N classes only. This is MCDNN's key novelty that most inputs belong to the top N classes, which can be handled by a reduced-complexity DNN. If the top-1 prediction label of the specialized model in MCDNN is not among the top N frequent classes, then the generic model processes the input again and outputs its final predictions. Otherwise, MCDNN uses the top-5 prediction labels of the specialized model as its final predictions. We set $N = 20$ that covers 80% of training video frames in the VID dataset.

5.4 Typical Usage Scenarios

We use a few usage scenarios to compare the protocols, although ApproxNet can support finer-grained user requirements in latency or accuracy.

- **High accuracy, High latency (HH)** refers to the scenario where ApproxNet has less than 10% (relative) accuracy loss from ResNet-34, our most accurate single model baseline. Accordingly, the runtime latency is also high to achieve such accuracy.
- **Medium accuracy, Medium latency (MM)** has an accuracy loss less than 20% from our base model ResNet-34.
- **Low accuracy, Low latency (LL)** can tolerate an accuracy loss of up to 30% with a speed up in its inferencing.
- **Real time (RT)** scenario, by default, means the processing pipeline should keep up with 30 fps speed, i.e., maximum 33.33 ms latency. This is selected if no requirement is specified.

5.5 Accuracy-latency Trade-off of the Static Models

We first evaluate ApproxNet on ILSVRC IMG dataset on the accuracy-latency trade-off of each AB in our single DNN as shown in Figure 9. Our AB with higher latency (satisfying 30 fps speed though higher) has close accuracy to ResNet-18 and ResNet-34 but much lower latency than ResNet-34. Meanwhile, our AB with reduced latency (25 ms to 30 ms) has close accuracy to MobileNets. And finally, our AB is superior to all baselines in achieving extremely low latency (< 20 ms). However, the single execution branches in MSDNet are much slower than ApproxNet and other baselines. The latency ranges from 62 ms to 153 ms, which cannot meet the real-time processing requirement and will be even worse in face of the resource contention. MobileNets can keep up with the frame rate but it lacks the configurability in the latency dimensional that ApproxNet has. Although MobileNets does win on the IMG dataset at higher accuracy, it needs an ensemble of models (like MCDNN) when it comes to the video where content characteristics, user requirement, and runtime resource contention change.

5.6 Adaptability to Changing User Requirements

We then, from now on, switch to the ILSVRC VID dataset and show how ApproxNet can meet different user requirements for accuracy and latency. We list the averaged accuracy and latency of Pareto frontier branches in Table 3(a), which can serve as a lookup table in the simplest scenario, i.e., without considering frame complexity categories and resource contention. ApproxNet, provides content-aware approximation and thus keeps a lookup table for each frame complexity category, and to be responsive to resource contention, updates the latency in the lookup table based on observed contention.

We perform our evaluation on the entire test set, but without the baseline protocols incurring any switching penalty. Figure 10 compares the accuracy and latency performance between ApproxNet and baselines in three typical usage scenarios “HH”, “MM”, and “LL” (AN denotes ApproxNet). In this experiment, ApproxNet uses the content-aware lookup table for each frame complexity category and chooses the best AB at runtime to meet the user accuracy requirement. MCDNN and MSDNet use similar lookup tables (Table 3(b) and (c)) to select among model variants or execution branches to satisfy the user requirement. We can observe that “AN-HH” achieves the accuracy

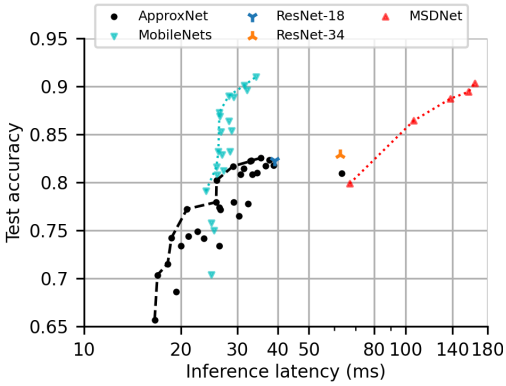


Fig. 9. Pareto frontier for test accuracy and inference latency on the ImageNet IMG dataset for ApproxNet compared to ResNet and MobileNets, the latter being specialized for mobile devices.

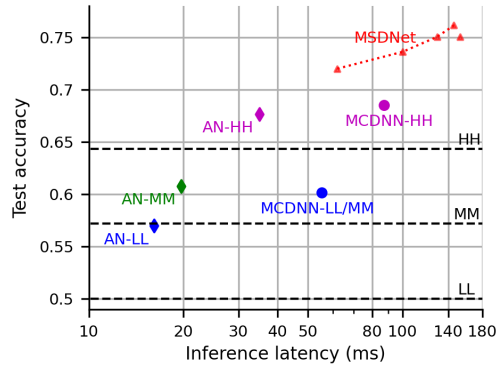


Fig. 10. Comparison of system performance in typical usage scenarios. ApproxNet is able to meet the accuracy requirement for all three scenarios. User requirements are shown in dashed lines.

Table 3. Averaged accuracy and latency performance of ABs on the Pareto frontier in ApproxNet and those of the baselines on validation set of the VID dataset. Note that the accuracy on the validation dataset can be higher due to its similarity with the training dataset. Thus the validation accuracy is only used to construct the look up table in ApproxNet and baselines and does not reflect the true performance.

(a) Averaged accuracy and latency performance in ApproxNet.

Usage Scenario (rate)	Shape	Layers	Latency	Accuracy
HH (32 fps)	128x128x3	24	31.42 ms	82.12%
	160x160x3	20	31.33 ms	80.81%
	128x128x3	20	27.95 ms	79.35%
	112x112x3	20	26.84 ms	78.28%
MM (56 fps)	128x128x3	12	17.97 ms	70.23%
	112x112x3	12	17.70 ms	68.53%
	96x96x3	12	16.78 ms	67.98%
LL (62 fps)	80x80x3	12	16.14 ms	66.39%

(b) Lookup table in MCDNN's scheduler.

Scenario (rate)	Shape	Layers	Latency	Accuracy
HH (11 fps)	224x224x3	34	88.11 ms	77.71%
MM/LL (17 fps)	224x224x3	18	57.83 ms	71.40%

(c) Lookup table in MSDNet's scheduler.

Scenario (rate)	Shape	Layers	Latency	Accuracy
HH (5.2 fps)	224x224x3	191	153 ms	96.79%
MM/LL (16 fps)	224x224x3	63	62 ms	95.98%

(d) Reference performance of single model variants or execution branches.

Model name (rate)	Shape	Layers	Latency	Accuracy
ResNet-34 (16 fps)	224x224x3	34	64.44 ms	85.86%
ResNet-18 (22 fps)	224x224x3	18	45.22 ms	84.59%
MSDNet-branch5 (5.2 fps)	224x224x3	191	153 ms	96.79%
MSDNet-branch4 (5.6 fps)	224x224x3	180	146 ms	96.55%
MSDNet-branch3 (7.8 fps)	224x224x3	154	129 ms	96.70%
MSDNet-branch2 (10 fps)	224x224x3	115	100 ms	96.89%
MSDNet-branch1 (16 fps)	224x224x3	63	62 ms	95.98%

of 67.7% at a latency of 35.0 ms, compared to "MCDNN-HH" that has an accuracy of 68.5% at the latency of 87.4 ms. Thus, MCDNN-HH is 2.5X slower while achieving 1.1% accuracy gain over ApproxNet. On the other hand, MSDNet is more accurate and slower than all ApproxNet's branches. The lightest branch and heaviest branch achieve 4.3% and 7.3% higher accuracy respectively, and incur 1.8X and 4.4X higher latency respectively. In "LL" and "MM" usage scenarios, MCDNN-LL/MM is 2.8-3.3X slower than ApproxNet, while gaining in accuracy 3% or less. MSDNets, on the other hand, is running with much higher latency (62 ms to 146 ms) and higher accuracy (72.0% to 76.2%). Thus, compared to these baseline models, ApproxNet wins by providing lower latency, satisfying the real-time requirement, and flexibility in achieving various points in the (accuracy, latency) space.

5.7 Adaptability to Changing Content Characteristics & User Requirements

We now show how ApproxNet can adapt to changing content characteristics and user requirements within the same video stream. The video stream, typically at 30 fps, may contain content of various

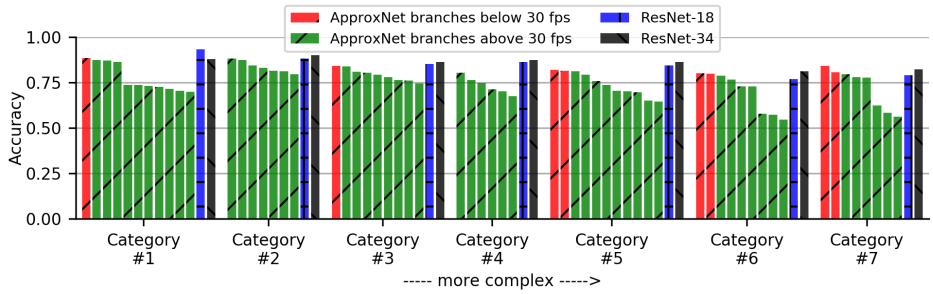


Fig. 11. Content-specific accuracy of Pareto frontier branches. Branches that fulfill real-time processing (30 fps) requirement are labeled in green. Note that both ResNet-18 and ResNet-34 models, though with the higher accuracy, cannot meet the 30 fps latency requirement.

complexities and this can change quickly and arbitrarily. Our study with the FCC on the VID dataset has shown that in 97.3% cases the frame complexity category of the video will change within every 100 frames. Thus, dynamically adjusting the AB with frame complexity category is beneficial to the end-to-end system. We see in Figure 11 that ApproxNet with various ABs can satisfy different (accuracy, latency) requirements for each frame complexity category. According to user's accuracy or latency requirement, ApproxNet's scheduler picks the appropriate AB. The majority of the branches satisfy the real-time processing requirement of 30 fps and can also support high accuracy quite close to the ResNet-34.

In Figure 12, we show how ApproxNet adapts for a particular representative video from the test dataset. Here, we assume the user requirement changes every 100 frames between "HH", "MM", and "LL". This is a synthetic setting to observe how models perform at the time of switching. We assume a uniformly distributed model selection among 20 model variants for MCDNN's scheduler (in [18], the MCDNN catalog uses 68 model variants) while the embedded device can only cache two models in the RAM (more detailed memory results in Section 5.9). In this case, MCDNN has a high probability to load a new model variant into RAM from Flash, whenever the user requirement changes. This results in a huge latency spike, typically from 5 to 20 seconds at each switch. It is notable that for some cases, there are also small spikes in MCDNN following the larger spikes because the generic model is invoked due to the specialized model's prediction of "infrequent" class. On the other hand, ApproxNet and MSDNets incur little overhead in switching between any two branches, because they are all available within the same single-model DNN. Similar to the results before, ApproxNet wins over MSDNets at lower latency to meet the real-time processing requirement even though its accuracy is slightly lower.

To see in further detail the behavior of ApproxNet, we profile the mean transition time of all Pareto frontier branches under no contention as shown in Figure 13 (a). Most of the transition overheads are extremely low, while only a few transitions are above 30 ms. Our optimization algorithm (Equations 3 and 4) filters out such expensive transitions if they happen too frequently. In Figure 13 (b), we further show the transition time between the descendant models in NestDNN, which uses a multi-capacity model with varying number of filters in the convolutional layers. The notation c50 stands for the descendant model with 50% capacity of the largest variant, and so on. We can observe that the lowest switching cost of NestDNN is still more than an order of magnitude higher than the highest switching cost of ApproxNet. The highest switching cost of NestDNN compared to the highest of ApproxNet is more than three orders of magnitude—the highest cost is important when trying to guarantee latencies with the worst-case latency spikes. We can observe

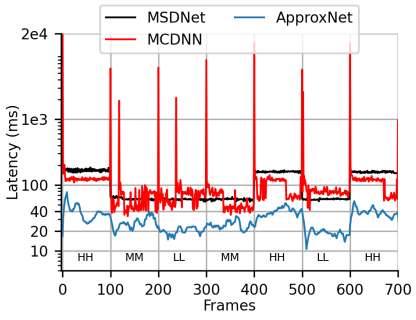


Fig. 12. Latency performance comparison with changing user requirements throughout video stream.

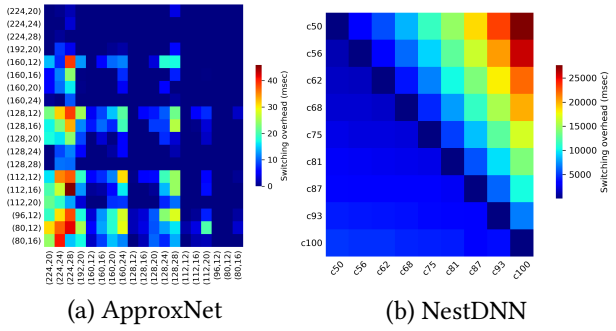
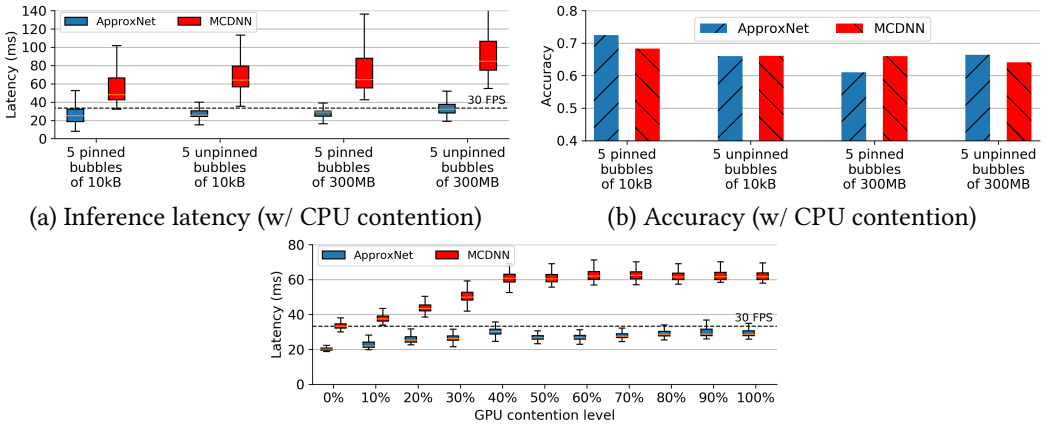


Fig. 13. Transition latency overhead across (a) ABs in ApproxNet and (b) descendant models in NestDNN. "from" branch on Y-axis and "to" branch on X-axis. Inside brackets: (input shape, outport depth). Latency unit is millisecond.



(a) Inference latency (w/ CPU contention)

(b) Accuracy (w/ CPU contention)

(c) Inference latency (w/ GPU contention)

Fig. 14. Comparison of ApproxNet vs MCDNN under resource contention. (a) and (b) inference latency and accuracy on the whole test dataset. (c) inference latency on a test video.

that the transition overhead can be up to 25 seconds from the smallest model to the largest model, and is generally proportional to the amount of data that is loaded into the memory. This is because NestDNN only keeps a single model, the one in use in memory and loads/unloads all others when switching. In summary, the benefit of ApproxNet comes from the fact that (1) it can accommodate multiple (accuracy, latency) points within one model through its two approximation knobs while MCDNN has to switch between model variants, and (2) switching between ABs does not load large amount of data and computational graphs.

5.8 Adaptability to Resource Contention

We evaluate in Figure 14, the ability of ApproxNet to adapt to resource contention on the device, both CPU and GPU contention. First, we evaluate this ability by running a *bubble application* [52, 78] on the CPU that creates stress of different magnitudes on the (shared) memory subsystem while the video analytics DNN is running on the GPU. We generate bubbles, of two different memory sizes 10

KB (low contention) and 300 MB (high contention). The bubbles can be "unpinned" meaning they can run on any of the cores or they can be "pinned" in which case they run on a total of 5 CPU cores leaving the 6th one for dedicated use by the video analytics application. The unpinned configuration causes higher contention. We introduce contention in phases—low pinned, low unpinned, high pinned, high unpinned.

As shown in Figure 14(a), MCDNN with its fastest model variant MCDNN-18, runs between 40ms and 100 ms depending on the contention level and has no adaptation. For ApproxNet, on the other hand, our mean latency under low contention (10 KB, pinned) is 25.66 ms, and it increases a little to 34.23 ms when the contention becomes high (300 MB, unpinned). We also show the accuracy comparison in Figure 14(b), where we are slightly better than MCDNN under low contention and high contention (2% to 4%) but slightly worse (within 4%) for intermediate contention (300 MB, pinned).

To further evaluate ApproxNet with regard to GPU contention, we run a synthetic matrix manipulation application concurrently with ApproxNet. The contention level is varied in a controlled manner through the synthetic application, from 0% to 100% in steps of 10%, where the control is the size of the matrix, and equivalently, the number of GPU threads dedicated to the synthetic application. The contention value is the GPU utilization when the synthetic application runs alone as measured through `tegrastats`. For baseline, we use the MCDNN-18 model again since among the MCDNN ensemble, it comes closest to the video frame rate (33.3 ms latency). As shown in Figure 14(c), without the ability to sense the GPU contention and react to it, the latency of MCDNN increases by 85.6% and is far beyond the real-time latency threshold. The latency of ApproxNet also increases with gradually increasing contention, 20.3 ms at no contention to 30.77 ms at 30% contention. However, when we further raise the contention level to 50% or above, ApproxNet's scheduler senses the contention and switches to a lighter-weight approximation branch such that the latency remains within 33.3 ms. The accuracy of MCDNN and ApproxNet were identical for this sample execution. Thus, this experiment bears out the claim that ApproxNet can respond to contention gracefully by recreating the Pareto curve for the current contention level and picking the appropriate AB.

5.9 Solution Overheads

With the same experiment as in Section 5.7, we compare the overheads of ApproxNet, MCDNN, and MSDNets in Figure 15. For ApproxNet, we measure the overhead of all the steps outside of the core DNN, i.e., frame resizing, FCE, RCE, and scheduler. For MCDNN, the dominant overhead is the model switching and loading. The model switching overhead of MCDNN is measured at each switching point and averaged across all frames in each scenario. We see that ApproxNet, including overheads, is 7.0X to 8.2X faster than MCDNN and 2.4X to 4.1X faster than MSDNets. Further, we can observe that in "MM" and "LL" scenarios, ApproxNet's averaged latency is less than 30 ms and thus ApproxNet can achieve real-time processing of 30 fps videos. As mentioned before, MCDNN may be forced to reload the appropriate models whenever the user requirement changes. So, in the best case for MCDNN the requirement never changes or it has all its models cached in RAM. ApproxNet is still 5.1X to 6.3X faster.

Figure 16 compares the peak memory consumption of ApproxNet and MCDNN in typical usage scenarios. ApproxNet-mixed, MCDNN-mixed, and MSDNet-mixed are the cases where the experiment cycles through the three usage scenarios. We test MCDNN-mixed with two model caching strategies: (1) the model variants are loaded from Flash when they get triggered (named "re-load"), simulating the minimum RAM usage (2) the model variants are all loaded into the RAM at the beginning (named "load-all"), assuming the RAM is large enough. We see that ApproxNet in going from "LL" to "HH" requirement consumes 1.6 GB to 1.7 GB memory and is lower than

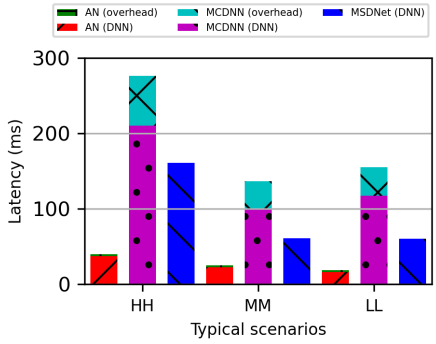


Fig. 15. System overhead in ApproxNet and MCDNN.

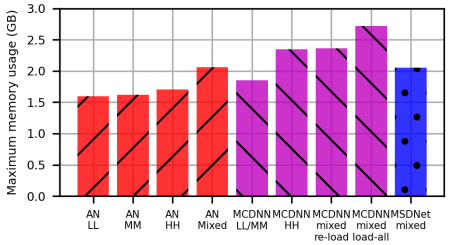
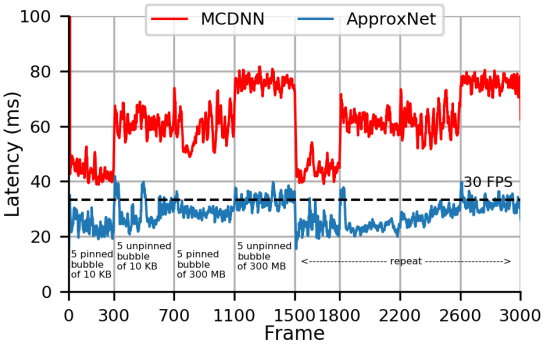
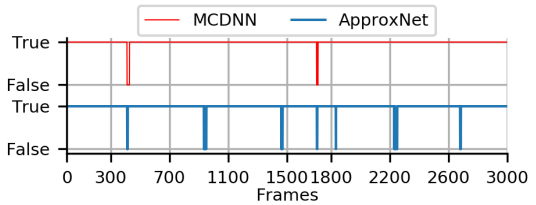


Fig. 16. Memory consumption of solutions in different usage scenarios (unit of GB).



(a) Inference latency



(b) Accuracy

Fig. 17. Case study: performance comparison of ApproxNet vs MCDNN under resource contention for a Youtube video.

MCDNN (1.9 GB and 2.4 GB). MCDNN's cascade DNN design (specialized model followed by generic model) is the root cause that it consumes about 15% more memory than our model even though they only keep one model variant in the RAM and it consumes 32% more memory if it loads two. For the mixed scenario, we can set an upper bound on the ApproxNet memory consumption—it never exceeds 2.1 GB no matter how we switch among ABs at runtime, an important property for proving operational correctness in mobile or embedded environments. Further, ApproxNet, with tens of ABs available, offers more choices than MCDNN and MSDNet, and MCDNN cannot accommodate more than two models in the available RAM.

Storage is a lesser concern but it does affect the pushing out of updated models from the server to the mobile device, a common use case. ApproxNet's storage cost is only 88.8 MB while MCDNN with 2 models takes 260 MB and MSDNet with 5 execution branches takes 177 MB. A primary reason is the duplication in MCDNN of the specialized and the generic models which have identical architecture except for the last FC layer. Thus, ApproxNet is well suited to the mobile usage scenario due to its low RAM and storage usage.

5.10 Ablation Study with FCE

To study the necessity of FCE, we conduct an ablation study of ApproxNet with a content agnostic scheduler. Different from the content-aware scheduler, this scheduler picks the same AB for all video frames regardless of the content (the contention level is held unchanged). The test dataset obviously has variations in the content characteristics. With the same experiment as in Section 5.6, we compare the accuracy and latency performance between ApproxNet with FCE and that without FCE in three typical usage scenarios "HH", "MM", and "LL". Figure 18 shows that ApproxNet with

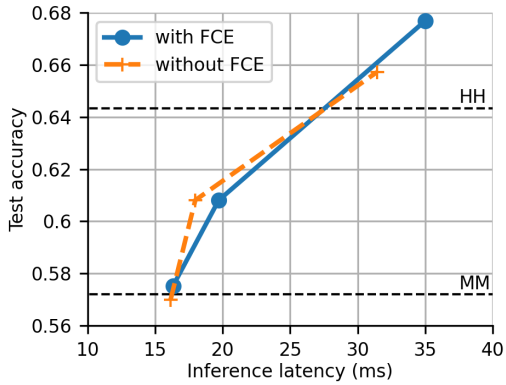


Fig. 18. Comparison of system performance in typical usage scenarios between ApproxNet with FCE and ApproxNet without FCE.

FCE is able to improve the accuracy by 2.0% under a "HH" scenario with an additional 3.57 ms latency cost. ApproxNet with FCE under "MM" and "LL" scenarios is either slightly slower or more accurate than that without FCE. To summarize, ApproxNet's FCE is more beneficial when the accuracy goal is higher, and however the latency budget is also higher.

5.11 Case Study with YouTube Video

As a case study, we evaluate ApproxNet on a randomly picked YouTube video [59], to see how it adapts to different resource contention scenarios at runtime (Figure 17). The video is a car racing match with changing scenes and objects, and thus, we want to evaluate the object classification performance. The interested reader may see a demo of ApproxNet and MCDNN on this and other videos at https://starsthu2016.github.io/projects/approxnet/approxnet_demo.html. Similar to the control setup in Section 5.8, we test ApproxNet and MCDNN for four different contention levels. Each phase is 300-400 frames and the latency requirement is 33 ms to keep up with the 30 fps video. We see ApproxNet adapts to the resource contention well—it switches to a lightweight AB while still keeping high accuracy, comparable to MCDNN (seen on the demo site). Further, ApproxNet is always faster than MCDNN, while MCDNN, with a latency of 40-80 ms and even without switching overhead, has degraded performance under resource contention, and has to drop approximately every two out of three frames. As for the accuracy, there are only some occasional false classifications in ApproxNet (in total: 51 out of 3,000 frames, or 1.7%). MCDNN, in this case, has slightly better accuracy (24 false classifications in 3,000 frames, or 0.8%). We believe commonly used post-processing algorithms [19, 36] can easily remove these occasional classification errors and both approaches can achieve very low inaccuracy.

6 DISCUSSION

Training the approximation-enabled DNN of ApproxNet may take longer than conventional DNNs, since at each iteration of training, different outports and input shapes try to minimize their own softmax loss, and thus, they may adjust internal weights of the DNN in conflicting ways. In our experiments with the VID dataset, we observe that our training time is around 3 days on our evaluation edge server, described in Section 5.1, compared to 1 day to train a baseline ResNet-34 model. However, training being an offline process, the time is of less concern. However, training

can be sped up by using one of various actively researched techniques for optimizing training, such as [41].

Generalizing the approximation-enabled DNN to other architectures. The shape and depth knobs are general to all CNN-based architectures. Theoretically, we can attach an outport (composed of SPP to adjust the shape and fully-connected layer to generate classification) to any layers. The legitimate input shape can be tricky and dependent on the specific architecture. Considering the training cost and exploration space, we select 7 shapes in multiples of 16 and 6 outports, in mostly equally spaced positions of the layers. More input shapes and outports enable finer-grained accuracy-latency trade-offs and the granularity of such a trade-off space depends on the design goal.

7 RELATED WORK

System-wise optimization: There have been many optimization attempts to improve the efficiency of video analytics or other ML pipelines by building low power hardware and software accelerators for DNNs [6, 13, 16, 39, 55, 56, 61, 84] or improving application performance using database optimization, either on-premise [49] or on cloud-hosted instances [48]. These are orthogonal and ApproxNet can also benefit from these optimizations. VideoStorm [83], Chameleon [34], and Focus [24] exploited various configurations and DNN models to handle video analytics queries in a situation-tailored manner. ExCamera [12] and Sonic [50] enabled low-latency video processing on the cloud using serverless architecture (AWS Lambda [30]). Mainstream [33] proposed to share weights of DNNs across applications. These are all server-side solutions, requiring multiple models to be loaded simultaneously, which is challenging with resource constraints. NoScope [35] targeted to reduce the computation cost of video analytics queries on servers by leveraging a specialized model trained on a small subset of videos. Thus, its applicability is limited to a small subset of videos that the model has seen. Closing-the-loop [79] uses genetic algorithms to efficiently search for video editing parameters with lower computation cost. VideoChef [77] attempted to reduce the processing cost of video pipelines by dynamically changing approximation knobs of preprocessing filters in a content-aware manner. In contrast, ApproxNet, and concurrently developed ApproxDet [80] (for video-specific object detection), approximate in the core DNN, which have a significantly different and computationally heavier program structure than filters. Due to this larger overhead of approximation in the core DNN, ApproxNet's adaptive tuning is challenging. Thus, we plan on using either distributed learning [14] or a reinforcement learning-based scheduler for refining this adaptive feature [72].

DNN optimizations: Many solutions have been proposed to reduce computation cost of a DNN by controlling the precision of edge weights [15, 27, 60, 87] and restructuring or compressing a DNN model [3, 5, 10, 17, 23, 29, 74]. These are orthogonal to our work and ApproxNet's one-model approximation-enabled DNN can be further optimized by adopting such methods. There are several works that also present similar approximation knobs (input shape, outport depth). BranchyNet, CDL, and MSDNet [25, 53, 71] propose early-exit branches in DNNs. However, BranchyNet and CDL only validate on small datasets like MNIST [42] and CIFAR-10 [38] and have not shown practical techniques to selectively select these early-exit branches in an end-to-end pipeline. Such an adaptive system, in order to be useful (especially on embedded devices), needs to be responsive to resource contention, content characteristics, and users' requirement, e.g., end-to-end latency SLA. MSDNet targets a very simple image classification task without a strong use case and does not show a data-driven manner of using the early exits. It would have been helpful to demonstrate the system's end-to-end latency on either a server-class or embedded device. BlockDrop [76] trains a policy network to determine whether to skip the execution of several residual blocks at inference

time. However, its speedup is marginal and it cannot be applied directly to mobile devices for real-time classification.

8 CONCLUSION

There is a push to support streaming video analytics close to the source of the video, such as, IoT devices, surveillance cameras, or AR/VR gadgets. However, state-of-the-art heavy DNNs cannot run on such resource-constrained devices. Further, the runtime conditions for the DNN's execution may change due to changes in the resource availability on the device, the content characteristics, or user's requirements. Although several works create lightweight DNNs for resource-constrained clients, none of these can adapt to changing runtime conditions. We introduced ApproxNet, a video analytics system for embedded or mobile clients. It enables novel dynamic approximation techniques to achieve desired inference latency and accuracy trade-off under changing runtime conditions. It achieves this by enabling two approximation knobs within a single DNN model, rather than creating and maintaining an ensemble of models. It then estimates the effect on latency and accuracy due to changing content characteristics and changing levels of contention. We show that ApproxNet can adapt seamlessly at runtime to such changes to provide low and stable latency for object classification on a video stream. We quantitatively compare its performance to ResNet, MCDNN, MobileNets, NestDNN, and MSDNet, five state-of-the-art object classification DNNs.

ACKNOWLEDGMENTS

This work is supported in part by the Wabash Heartland Innovation Network (WHIN) award from Lilly Endowment, NSF grant CCF-1919197, and Army Research Lab Contract number W911NF-20-2-0026. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] Rachata Ausavarungnirun, Vance Miller, Joshua Landgraf, Saugata Ghose, Jayneel Gandhi, Adwait Jog, Christopher J Rossbach, and Onur Mutlu. 2018. Mask: Redesigning the GPU memory hierarchy to support multi-application concurrency. In *ACM SIGPLAN Notices*, Vol. 53. ACM, 503–518.

[2] Saurabh Bagchi, Tarek F Abdelzaher, Ramesh Govindan, Prashant Shenoy, Akanksha Atrey, Pradipta Ghosh, and Ran Xu. 2020. New Frontiers in IoT: Networking, Systems, Reliability, and Security Challenges. *IEEE Internet of Things Journal* 7, 12 (2020), 11330–11346.

[3] Sourav Bhattacharya and Nicholas D Lane. 2016. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems (Sensys)*. ACM, 176–189.

[4] Maurizio Cardaci, Vito Di Gesù, Maria Petrou, and Marco Elio Tabacchi. 2009. A fuzzy approach to the evaluation of image complexity. *Fuzzy Sets and Systems* 160, 10 (2009), 1474–1484.

[5] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*. 2285–2294.

[6] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[7] NVIDIA Corporation. 2018. *Jetson TX2 Module*. Retrieved May 5, 2020 from https://developer.nvidia.com/embedded/buy/jetson-tx2

[8] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGPLAN Notices* 48, 4 (2013), 77–88.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee, 248–255.

[10] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*. 1269–1277.

[11] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 115–127.

[12] Sadjad Fouladi, Riad S Wahby, Brennan Shacklett, Karthikeyan Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads.. In *NSDI*. 363–376.

[13] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. Tetris: Scalable and efficient neural network acceleration with 3d memory. *ACM SIGOPS Operating Systems Review* 51, 2 (2017), 751–764.

[14] Asish Ghoshal, Ananth Grama, Saurabh Bagchi, and Somali Chaterji. 2015. An ensemble svm model for the accurate prediction of non-canonical microrna targets. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*. 403–412.

[15] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *International Conference on Machine Learning*. 1737–1746.

[16] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 243–254.

[17] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[18] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 123–136.

[19] Wei Han, Pooya Khorrami, Tom Le Paine, Prajit Ramachandran, Mohammad Babaeizadeh, Honghui Shi, Jianan Li, Shuicheng Yan, and Thomas S Huang. 2016. Seq-nms for video object detection. *arXiv preprint arXiv:1602.08465* (2016).

[20] Mark Harris. 2017. *Unified Memory for CUDA Beginners*. Retrieved May 5, 2020 from https://devblogs.nvidia.com/unified-memory-cuda-beginners/

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2014. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *european conference on computer vision*. Springer, 346–361.

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

[23] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[24] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. 2018. Focus: Querying large video datasets with low latency and low cost. *arXiv preprint arXiv:1801.03493* (2018).

[25] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. 2018. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations (ICLR)*.

[26] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[27] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *Journal of Machine Learning Research* 18 (2017), 187–1.

[28] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 82–95.

[29] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *ICLR* (2016).

[30] Amazon Web Services Inc. 2018. *AWS Lambda*. Retrieved May 5, 2020 from https://aws.amazon.com/lambda/

[31] Bernd Jähne, Horst Haussecker, and Peter Geissler. 1999. *Handbook of computer vision and applications*. Vol. 2. Citeseer.

[32] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 2013. 3D convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence* 35, 1 (2013), 221–231.

[33] 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.

[34] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 253–266.

[35] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1586–1597.

[36] Kai Kang, Hongsheng Li, Junjie Yan, Xingyu Zeng, Bin Yang, Tong Xiao, Cong Zhang, Zhe Wang, Ruohui Wang, Xiaogang Wang, et al. 2017. T-CNN: Tubelets with convolutional neural networks for object detection from videos. *IEEE Transactions on Circuits and Systems for Video Technology* (2017).

[37] Onur Kayiran, Nachiappan Chidambaram Nachiappan, Adwait Jog, Rachata Ausavarungnirun, Mahmut T Kandemir, Gabriel H Loh, Onur Mutlu, and Chita R Das. 2014. Managing GPU concurrency in heterogeneous architectures. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 114–126.

[38] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[39] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press, 23.

[40] Michael A Laurenzano, Parker Hill, Mehrzad Samadi, Scott Mahlke, Jason Mars, and Lingjia Tang. 2016. Input responsiveness: using canary inputs to dynamically steer approximation. *ACM SIGPLAN Notices* 51, 6 (2016), 161–176.

[41] Quoc V Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y Ng. 2011. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress, 265–272.

[42] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[43] Jun Liu, Amir Shahroudy, Dong Xu, and Gang Wang. 2016. Spatio-temporal lstm with trust gates for 3d human action recognition. In *European Conference on Computer Vision*. Springer, 816–833.

[44] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge assisted real-time object detection for mobile augmented reality. (2019).

[45] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.

[46] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving resource efficiency at scale. In *International Symposium on Computer Architecture (ISCA)*, Vol. 43. ACM, 450–462.

[47] Zongqing Lu, Swati Rallapalli, Kevin Chan, and Thomas La Porta. 2017. Modeling the resource requirements of convolutional neural networks on mobile devices. In *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 1663–1671.

[48] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. OptimusCloud: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 189–203.

[49] Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2019. Sophia: Online reconfiguration of clustered nosql databases for time-varying workloads. In *2021 USENIX Annual Technical Conference (USENIX ATC 19)*. 223–240.

[50] Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2021. SONIC: Application-aware Data Passing for Chained Serverless Applications. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 1–15.

[51] I Mario, M Chacon, D Alma, and S Corral. 2005. Image complexity measure: a human criterion free approach. In *NAFIPS 2005-2005 Annual Meeting of the North American Fuzzy Information Processing Society*. IEEE, 241–246.

[52] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 248–259.

[53] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. 2016. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 475–480.

[54] Rajesh Krishna Panta, Saurabh Bagchi, and Samuel P Midkiff. 2011. Efficient incremental code update for sensor networks. *ACM Transactions on Sensor Networks (TOSN)* 7, 4 (2011), 1–32.

[55] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. 2017. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, Vol. 45. ACM, 27–40.

[56] Eunhyeok Park, Dongyoung Kim, Soobeom Kim, Yong-Deok Kim, Gunhee Kim, Sungroh Yoon, and Sungjoo Yoo. 2015. Big/little deep neural network for ultra low power inference. In *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 124–132.

[57] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. 2015. Deep face recognition.. In *BMVC*, Vol. 1. 6.

[58] Ronald Poppe. 2010. A survey on vision-based human action recognition. *Image and vision computing* 28, 6 (2010), 976–990.

[59] Canal Max Power. 2016. *Sport Cars Drag Race Video.* Retrieved May 5, 2020 from https://www.youtube.com/watch?v=Qj21A8HLQ0M

[60] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.

[61] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 267–278.

[62] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[63] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

[64] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[65] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

[66] Karthick Shankar, Pengcheng Wang, Ran Xu, Ashraf Mahgoub, and Somali Chaterji. 2020. JANUS: Benchmarking Commercial and Open-Source Cloud and Edge Platforms for Object and Anomaly Detection Workloads. In *Proceedings of the IEEE International Conference on Cloud Computing*. 1–9.

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

[68] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[69] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[70] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1701–1708.

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

[72] Tara Elizabeth Thomas, Jinkyu Koo, Somali Chaterji, and Saurabh Bagchi. 2018. Minerva: A reinforcement learning-based technique for optimal scheduling and bottleneck detection in distributed factory operations. In *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*. IEEE, 129–136.

[73] Robert J Wang, Xiang Li, and Charles X Ling. 2018. Pelee: A real-time object detection system on mobile devices. In *Advances in Neural Information Processing Systems*. 1963–1972.

[74] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*. 2074–2082.

[75] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. 2016. A discriminative feature learning approach for deep face recognition. In *European Conference on Computer Vision*. Springer, 499–515.

[76] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. 2018. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8817–8826.

[77] Ran Xu, Jinkyu Koo, Rakesh Kumar, Peter Bai, Subrata Mitra, Sasa Misailovic, and Saurabh Bagchi. 2018. Videochef: efficient approximation for streaming video processing pipelines. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, 43–56.

[78] Ran Xu, Subrata Mitra, Jason Rahman, Peter Bai, Bowen Zhou, Greg Bronevetsky, and Saurabh Bagchi. 2018. Pythia: Improving Datacenter Utilization via Precise Contention Prediction for Multiple Co-located Workloads. In *Proceedings of the 19th International Middleware Conference*. ACM, 146–160.

[79] Ran Xu, Haoliang Wang, Stefano Petrangeli, Viswanathan Swaminathan, and Saurabh Bagchi. 2020. Closing-the-Loop: A Data-Driven Framework for Effective Video Summarization. In *Proceedings of the 22nd IEEE International Symposium on Multimedia (ISM)*. 201–205.

[80] Ran Xu, Chen-lin Zhang, Pengcheng Wang, Jayoung Lee, Subrata Mitra, Somali Chaterji, Yin Li, and Saurabh Bagchi. 2020. ApproxDet: content and contention-aware approximate object detection for mobiles. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 449–462.

[81] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. 2013. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *International Symposium on Computer Architecture (ISCA)*, Vol. 41. ACM, 607–618.

[82] Honghai Yu and Stefan Winkler. 2013. Image complexity and spatial information. In *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*. IEEE, 12–17.

[83] 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 *NSDI*, Vol. 9. 1.

[84] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-x: An accelerator for sparse neural networks. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 20.

[85] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 6848–6856.

[86] Yunqi Zhang, Michael A Laurenzano, Jason Mars, and Lingjia Tang. 2014. Smite: Precise qos prediction on real-system smt processors to improve utilization in warehouse scale computers. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 406–418.

[87] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).