IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 20, 2019, accepted November 13, 2019, date of publication November 20, 2019, date of current version December 3, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2954546

DMS: Dynamic Model Scaling for Quality-Aware Deep Learning Inference in Mobile and Embedded Devices

WOOCHUL KANG, (Member, IEEE), DAEYEON KIM, AND JUNYOUNG PARK
Department of Embedded Systems Engineering, Incheon National University, Incheon 22012, South Korea

Corresponding author: Woochul Kang (wchkang@inu.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) Grants Funded by the Ministry of Education under Grant NRF-2016R1D1A1B03934266, and in part by the Ministry of Science and ICT under Grant NRF-2019R1F1A1060959.

ABSTRACT Recently, deep learning has brought revolutions to many mobile and embedded systems that interact with the physical world using continuous video streams. Although there have been significant efforts to reduce the computational overheads of deep learning inference in such systems, previous approaches have focused on delivering 'best-effort' performance, resulting in unpredictable performance under variable environments. In this paper, we propose a runtime control method, called DMS (Dynamic Model Scaling), that enables dynamic resource-accuracy trade-offs to support various QoS requirements of deep learning applications. In DMS, the resource demands of deep learning inference can be controlled by adaptive pruning of computation-intensive convolution filters. DMS avoids irregularity of pruned models by reorganizing filters according to their importance so that varying number of filters can be applied efficiently. Since DMS's pruning method incurs no runtime overhead and preserves the full capacity of original deep learning models, DMS can tailor the models at runtime for concurrent deep learning applications with their respective resource-accuracy trade-offs. We demonstrate the viability of DMS by implementing a prototype. The evaluation results demonstrate that, if properly coordinated with system level resource managers, DMS can support highly robust and efficient inference performance against unpredictable workloads.

INDEX TERMS Deep learning, edge devices, embedded systems, energy efficiency, feedback control, filter pruning, mobile devices, model compression, quality-of-service, QoS.

I. INTRODUCTION

In the past few years, deep learning has emerged as a state-of-the-art approach that provides highly robust and accurate inference capability for many intelligent systems and services [1]. In particular, convolutional neural networks (CNNs or ConvNets) [2]–[4] have brought revolutions to computer vision applications [5]. Deep CNNs play as generic feature extractors for various visual recognition tasks such as image classification [3], object detection [6], semantic segmentation [7], and image retrieval [3]. Such visual recognition tasks are essential for many intelligent systems interacting with the physical world using continuous streaming of video inputs. Some examples are augmented reality wearables [8], camera-based surveillance, drones, autonomous vehicles [9], and live video analytics [5], to name a few.

The associate editor coordinating the review of this manuscript and approving it for publication was Bhaskar Prasad Rimal.

However, the size and complexity of deep learning models has been a major challenge for resource-constrained mobile and embedded devices, and there have been significant efforts to reduce the amount of computation of deep learning models either by compressing deep learning models at a modest loss of inference accuracy [10]–[13] or by offloading inference workloads to custom accelerators [14]–[16]. Although these approaches have demonstrated significant gains in performance and efficiency, their resource demands are predetermined at development stages, incurring unpredictable 'best-effort' performance in highly dynamic environments. For instance, when a person with a wearable cognitive-assistance device walks to a more crowded area, more objects needs to be classified, resulting in sudden increases of overall inference latency and energy consumption [8]. In such highly dynamic environments, there should be an effective and efficient method to manage the resource consumption of deep learning applications on a per-task basis according to their respective QoS goals, such as inference latency, energy consumption, and accuracy.

In this paper, we present *DMS (Dynamic Model Scaling),* a runtime control method that enables dynamic resource-accuracy trade-offs to support various QoS requirements of deep learning applications. With DMS, deep learning applications can trade their inference accuracy dynamically to accelerate inference speed or to reduce energy consumption. If enough resource is available, applications can also recover its maximum inference accuracy dynamically. To achieve this, DMS controls the computational cost of inference operations by scaling deep learning models dynamically at runtime. The scaling of deep learning models is accomplished by pruning convolution filters [13]. This is because the convolution layers of deep learning models are the most computationally intensive layers. Unlike static compression techniques, DMS does not prune filters of convolution layers at the development time. At off-line stages, DMS only rearranges the convolution filters of off-the-shelf CNN models according to their importance and preserves the ability of original models. At runtime, DMS dynamically adjusts the number of active convolution filters to control the computational cost of convolution operations. Since DMS's convolution filters are rearranged to avoid irregularities after pruning, DMS can exploit highly optimized vectorized libraries such as BLAS [17] for efficient convolution operations.

With this light-weight model scaling mechanism, DMS can be combined with runtime feedback control mechanisms to guarantee applications' QoS goals. Since DMS scales the computational cost of fully capable models without actually removing pruned filters, it can support different QoS levels for concurrent inference tasks even if they share a single deep learning model. Taking a traffic control system as an example [5], license plate readers at toll routes require high inference accuracy, but can tolerate several minutes of latency. On the other hand, counting cars to control traffic lights has stringent latency goals, but can tolerate a small loss of accuracy. With DMS, both applications can make systematic resource-accuracy trade-offs at runtime to support their respective QoS goals.

To show the viability of the proposed approach, we have implemented DMS by extending Caffe [18], an open-source deep learning framework. Using this prototype, we evaluate DMS for three representative deep learning models under various workloads. Our evaluation results demonstrate that DMS can control the resource consumption of deep learning applications at a modest loss of accuracy. For instance, ResNet-50 [4]'s energy consumption and inference latency can be reduced by 14.2% and 14.8%, respectively, at about 5% degradation of inference accuracy. Our evaluation results also show that, if properly coordinated with system level resource managers such as DVFS governors, DMS can support highly robust and efficient inference performance against unpredictable workloads. To the authors' best knowledge, this paper presents the first attempt that supports various

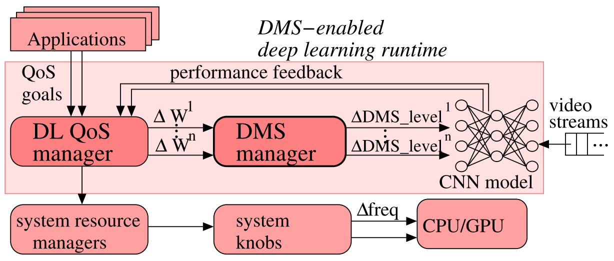


FIGURE 1. The QoS management architecture using DMS.

QoS goals, such as energy consumption, in deep learning inferences using dynamic scaling of deep learning models. The remainder of this paper is organized as follows. Section II gives an overview of DMS. Section III presents the details of DMS's model scaling mechanism and the QoS management using DMS. Section IV describes our evaluation results. The related work is discussed in Section V. Finally, Section VI concludes this paper and discusses future work.

II. OVERVIEW OF DMS

Figure 1 illustrates the QoS management architecture with DMS. As shown in Figure 1, the model scaling capability of DMS can be implemented in a deep learning runtime, such as Caffe and TensorRT [19], to support dynamic adaptation of resource consumption of deep learning inference tasks.

Applications can request the DMS-enabled deep learning runtime to initiate inference tasks with a deep learning model M. Model M is a preprocessed model at an off-line stage by rearranging its convolution filters according to their importance, but model M still maintains the full capacity of the original deep learning model. The applications can also specify desired QoS goals, such as the desired inference latency and energy consumption. To support QoS goals, the *QoS manager* in the deep learning runtime needs to cooperate with system-level resource managers, such as DVFS managers. For instance, the QoS manager might request the DVFS manager to set the clock frequency of device processors to the most energy efficient ones. If the QoS manager of the deep learning runtime is not properly coordinated with the system-level resource managers, the result might be unpredictable.

Inference tasks are usually periodic because they need to process incoming video streams continuously. For example, wearable cognitive-assistance devices need to analyze their surrounding environments at least with 5Hz frequency [20]. To meet such target frame rates, performance, such as inference latency, is continuously monitored and reported to the QoS manager. According to the gap between the QoS objectives and the monitored performance, the QoS manager requests the *DMS manager* to adjust the workload of inference tasks by ΔWs. If the system has high resource contention among tasks, the greater amount of workload adaptation ΔWs is required to support the QoS goals of the inference tasks. The DMS manager is responsible for controlling the resource demands of inference tasks by dynamically pruning convolution filters at runtime.

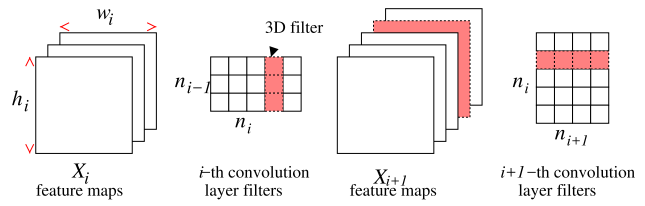


FIGURE 2. Pruning a 3D filter from the i-th convolution layer [13]. It also affects ($i+1$)-th convolution layer's filters and feature maps.

Therefore, as shown in Figure 1, the DMS manager translates the requested workload changes ΔWs to the model scaling levels ΔDMS_levels. For per-task QoS guarantees, each inference task maintains its own model scaling level. As will be discussed in Section III-B, the DMS manager adjusts the number of active convolution filters to achieve ΔDMS_levels without actually pruning filters.

III. DYNAMIC MODEL SCALING

A. ConvNets AND FILTER PRUNING

1) CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (ConvNets or CNNs) are special kind of deep learning models designed to extract visual features or patterns from input images with minimal preprocessing [2]–[4]. A CNN architecture consists of several types of layers, such as convolution layers, pooling layers, ReLU non-linearity layers, fully connected layers, etc. When a CNN model is used for inference, a cascade of such layers are applied to input images to transform them to higher-level features. Convolution layers are used as a feature extractor and are the most computation-intensive and frequent operations in CNN models.

Figure 2 illustrates two consecutive convolution layers. Let X_i denote the feature maps, or channels, for the i-th convolution layer and h_i and w_i are the height and the width of each feature map. The i-th convolution layer transforms feature maps $X_i \in \mathbb{R}^{w_i \times h_i \times n_{i-1}}$ into another set of of feature maps $X_{i+1} \in \mathbb{R}^{w_{i+1} \times h_{i+1} \times n_i}$ for the $(i+1)$th convolution layer. This transformation is performed by applying n_i 3D filters to the feature maps X_i. Each 3D filter has n_{i-1} 2D filters whose spatial size is $k \times k$. In this settings, the time complexity of convolution operations at i-th layer is as follows:

$$O\{n_{i-1} \times k^2 \times n_i \times (w_{i+1} \times h_{i+1})\}. \tag{1}$$

2) FILTER PRUNING

Pruning filters of convolution layers is a very effective method to scale the computational cost of deep learning inference [13]. If one 3D filter is pruned at the i-th convolution layer, the computation of $O\{n_{i-1} \times k^2 \times (w_{i+1} \times h_{i+1})\}$ is saved. Unlike other model compression techniques [21]–[24], filter-pruning does not incur severe model restructuring and sparsity. For instance, if the j-th 3D filter in the i-th convolution layer is pruned, only the j-th feature map of X_{i+1} needs to be removed to match inter-layer dimensions. This subsequent pruning of the feature map of X_{i+1} renders additional computational reduction of $O\{k^2 \times n_{i+1} \times (w_{i+2} \times h_{i+2})\}$ at the $(i+1)$-th convolution layer. For example, if 50% of feature maps and 50% of filters are pruned respectively from feature maps X_i and the filters of the i-th convolution layer, the total computational cost at the i-th convolution layer is reduced by 75%.

TABLE 1. Deep learning models.

	# of parameters	# of conv. layers	# of total layers	accuracy (top-5)
VGG-16	138M	13	16	90.0
ResNet-50	25.5M	49	50	92.9

3) ConvNet MODELS AND DATASETS

To present the details of DMS, in this work, two representative, but very contrasting, CNN models shown in Table 1 are considered. VGG-16 represents relatively shallow deep learning models that have a large number of parameters [25]. Even though *VGG-16* has only 16 layers, its three fully-connected layers occupy 90% of its 138 million parameters. In contrast, *ResNet-50* represents deep and highly computation-intensive deep learning models [4]. Even though *ResNet-50* has deep 50 layers, it requires only 25 million parameters and takes up 25.5 MBytes of storage. Both *VGG-16* and *ResNet-50* use 224 × 224 images as input and classify them into 1000 classes. These models are trained with the ImageNet 2012 classification dataset [26] that consists of 1.28 million training images and 50k validation images. In this paper, the accuracy of pruned models is the mean accuracy on the 50k images of the ImageNet 2012 validation dataset. The performance is evaluated on a Nvidia Jetson TX2 embedded board [1]. In Section IV-D.1, we extend this evaluation to a smaller-scale classification task with the CIFAR-10 dataset [27], the ResNet-32 model [4], and a more resource-constrained device.

B. DYNAMIC MODEL SCALING

1) IMPACT OF MODEL SCALING ON ACCURACY AND PERFORMANCE

Since DMS prunes filters dynamically at runtime, it cannot recover original accuracy through retraining and it needs to minimize the loss of accuracy by carefully selecting filters to be pruned both within a single layer and across layers.

Within a single layer, the importance of filters in a convolution layer can be measured by their $\ell 1$-norm, which is the sum of absolute weights [13]. Figure 3 shows the inference accuracy as varying percentages of filters are pruned in individual layers in the smallest $\ell 1$-norm order. In both VGG-16 and ResNet-50, the results show that the accuracy is decreased monotonically as the higher pruning ratio is applied. However, the effect of pruning is quite different for different models and their respective layers. ResNet-50's layers are less sensitive to the filter pruning than VGG-16's. For example, when 50% of filters of a layer are pruned, the accuracy of VGG-16 and ResNet-50 are decreased, respectively, by about 22% and 6% on average.

[1]Details of the experiment setup are in Section IV.

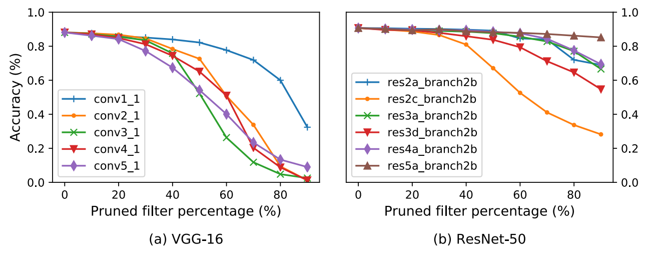


FIGURE 3. Accuracy with varying pruning ratios when the filters are pruned in the smallest $\ell 1$-norm order.

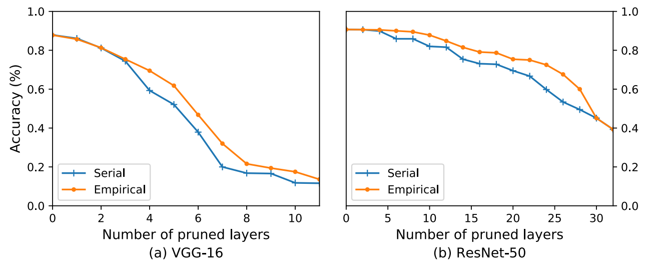


FIGURE 4. Accuracy while increasing number of layers are pruned with different ordering methods.

Since pruning filters across a network has multiplicative effect on final inference accuracy, the order of pruning across layers has a significant impact on the final inference accuracy. As shown in Figure 3, each convolution layer has very different sensitivity to pruning filters. Therefore, when pruning filters across the network, it is desirable to prune the layer first that incurs less accuracy loss. The experimental results in Figure 4 supports this claim. Figure 4 shows the inference accuracies when the varying number of convolution layers are pruned by 30% in two different orders. In *Empirical* order, each layer's pruning sensitivity on accuracy is considered and the layers are pruned in the order of minimal sensitivity to pruning. In *Serial* order, the convolution layers are pruned in the order that appears in the original CNN model, without considering pruning sensitivities. As expected, in Figure 4, the slope of accuracy loss is more gradual in *Empirical* order.

It should also be noted that the two CNN models show very different performance when the filters are pruned across the network. For example, the accuracy of VGG-16 drops to 0.6 when only 4 layers are pruned by 30%. In contrast, ResNet-50 maintains its accuracy over 0.6 until 30 layers are pruned by 30%. This result shows that CNN models with a small number of convolution layers, such as VGG-16, are less appropriate for DMS because pruning filters can incur significant loss of accuracy.

When k layers out of total d convolution layers are pruned, the expected computational cost, C_{scaled}, of convolution layers can be estimated as follows:

$$C_{scaled} = \sum_{i=1}^{k} s_{i-1} \times s_i \times C_i + \sum_{i=k+1}^{d} C_i, \qquad (2)$$

where C_i is the computation cost of the i-th layer without pruning, and s_i is the *scaling factor* for pruning filters. Since pruning filters reduces the feature maps in the next layer,

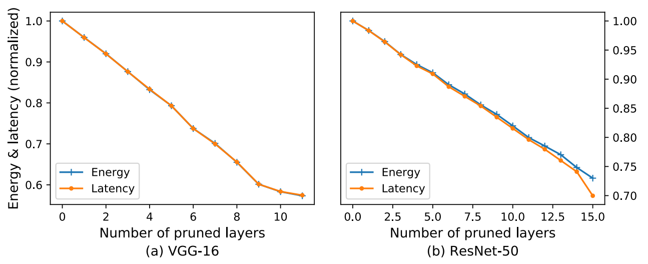


FIGURE 5. Performance while varying number of layers are pruned.

C_i is also scaled by the previous convolution layer's scaling factor s_{i-1}. As shown in Figure 3, convolution layers have different sensitivities to pruning, and, hence, we might consider applying different scaling factors in per-layer basis. For instance, He *et al.* proposed to learn optimal scaling factors via reinforcement learning [28]. However, in this paper, we apply fixed scaling factors for all target convolution layers for simplicity. In DMS, a scaling factor is chosen at a knee point where high pruning ratio can be achieved without too much loss of accuracy. For instance, in both VGG-16 and ResNet-50, scaling factors are set to 0.7 because both models maintain relatively high accuracy until 30% of filters of each layer are pruned, as shown in Figure 3.

Figure 5 shows the inference latency and energy consumption while varying number of layers are pruned by 30%. In both models, the inference latency and energy consumption are inversely proportional to the number of pruned layers. This experimental result is consistent with the analytic estimation in Equation 2. These linear relationships are highly desirable for DMS to make systematic resource-accuracy trade-offs. The resource demands of an inference task can be controlled by adjusting the number of pruned layers k in Equation 2.

2) EFFICIENT IMPLEMENTATION OF DYNAMIC FILTER PRUNING

In DMS, the effect of filter pruning is achieved by applying filters in the order of higher importance and by skipping less important filters when the computation cost needs to be reduced. For runtime efficiency of selecting important filters, DMS preprocesses a CNN model at an off-line stage by reordering its convolution filters in decreasing order of $\ell 1$-norm. For the preprocessed model, DMS can scale a convolution layer at runtime by adjusting the number of active filters and feature maps. For instance, Algorithm 1 shows the convolution operation at the i-th layer, and the number of active filters and feature maps (or channels) in the layer are scaled by adjusting two outermost loops by s_i and s_{i-1}, respectively. Since filters are ordered in the order of importance, less important filters are skipped.

In practice, however, convolution operations in deep learning frameworks are never implemented using nested *for* loops as in Algorithm 1. Instead, convolution operations are implemented to exploit vectorized operations of highly optimized libraries such as BLAS [17]. In most deep

Algorithm 1 Scaling convolution operations at i-th layer

```
Input: feature maps $X_i$ with $n_{i-1}$ channels
Input: $n_i$ 3D convolution filters
Input: scaling factor $s_i$ for filters ($0 \leq s_i \leq 1.0$)
Input: scaling factor $s_{i-1}$ for input feature maps
Output: feature maps $X_{i+1}$ with $s_i \times n_i$ channels
$num\_active\_filters = s_i \times n_i$ ;
$num\_active\_feature\_maps = s_{i-1} \times n_{i-1}$;
for $m$=1 to $num\_active\_filters$ do
    for $n$=1 to $num\_active\_feature\_maps$ do
        convolve ($m$, $n$)-th 2D filter with $n$-th channel of $X_i$;
    end
end
```

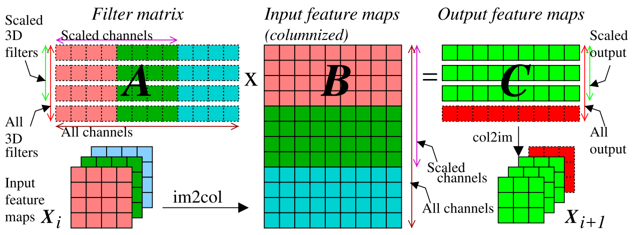


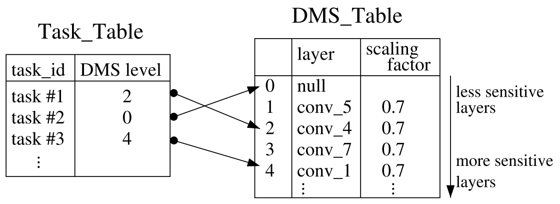
FIGURE 6. Efficient implementation of scaled convolution using a multiplication of dense sub-matrices.

learning frameworks, *im2col* (image to columns) is used to vectorize the convolution operations. Figure 6 shows the input feature maps transformed by im2col that expands the elements of individual convolution into columns in matrix **B**. Since 3D convolution filters are stored linearly in contiguous memory, they only need to be reshaped to a matrix, e.g., **A**, that matches the columnized input feature maps. Once feature maps are columnized by im2col, all convolution operations of the layer can be performed in a single general matrix to matrix multiplication (GEMM). The pruning operation of DMS can be efficiently performed in such vectorized convolution operations, as shown in Figure 6. Since the filters are rearranged in order of importance, important filters and corresponding feature maps are simply selected by accessing the sub-matrices of **A** and **B**. For example, if we want to apply 0.75 scaling factor to 4 3D filters in **A**, only first $3(= 4 * 0.75)$ rows of **A** are used for matrix multiplication, and this generates only 3 output feature maps in **C**. Similarly, the number of active input channels in the columnized feature maps **B** can be simply adjusted by selecting submatrices of **B**. This scaling mechanism does not incur runtime overhead because the sub-matrices can be accessed simply by changing the strides in memory accesses.

C. QoS MANAGEMENT USING DMS

1) DMS MANAGER

As shown in Section III-B, the computational cost at each convolution layer can be scaled simply by changing the number of active filters and feature maps, and, hence, DMS can control the computational cost of each individual inference



task_id	DMS level
task #1	2
task #2	0
task #3	4
⋮	

	layer	scaling factor
0	null	
1	conv_5	0.7
2	conv_4	0.7
3	conv_7	0.7
4	conv_1	0.7
	⋮	⋮

FIGURE 7. Data structures for the DMS manager.

task without extra overhead. For per-task QoS management, the *DMS manager* maintains two data structures shown in Figure 7. In *DMS_Table*, the convolution layers in the model are sorted in ascending order of sensitivity to pruning filters. Each layer in the table specifies the DMS scaling factor s_i, which is the ratio of active filters when the pruning is applied to the layer. The scaling factor is determined so that each layer in the table yields equal amount of savings via pruning. In *Task_Table*, each task maintains its own DMS level as an index to *DMS_Table*. The *DMS_level* indicates how many convolution layers will be pruned during the task's inference. When inference tasks are executed, *Task_Table* and *DMS_Table* are used by a deep learning runtime to apply proper scaling factors for chosen convolution layers. For example, in Figure 7, when task #3's DMS level is 2, two convolution layers (*conv_5* and *conv_4*) in *Task_Table* are pruned with their respective scaling factors. If the task wants to decrease the computation cost either for further energy saving or for reducing the latency, it might increase its *DMS_level* at runtime. Conversely, if the task needs full inference accuracy, its *DMS_level* can be set to 0, as task #2.

2) QoS MANAGEMENT USING DMS

As shown in Figures 4 and 5, the scaling of convolution layers manifests highly predictable resource-accuracy trade-offs. These relationships can be exploited by DMS to support QoS at runtime. For instance, Figure 8 shows a feedback control loop to support a desired inference latency as a QoS goal. In the feedback control loop, the QoS manager requests the DMS manager to adapt the workload of the inference task by ΔW according to the gap between the target latency and the monitored latency. The DMS manager translates ΔW to ΔDMS_level to scale the CNN model. Since each inference task maintains its own *DMS_level* in *Task_Table*, this feedback control loop can be executed concurrently on a shared CNN model to provide per-task QoS guarantees.

In our prototype implementation, we use a PI (proportional integral) controller that relates the error in latency directly to ΔDMS_level. In Section IV, we demonstrate the effectiveness of DMS for QoS management with this prototype implementation. The feedback control loop using the target latency as a QoS metric in Figure 8 is proposed only to demonstrate the usability of DMS for QoS management. Other control architectures (e.g., Multiple Inputs/Multiple Outputs control [29]) and QoS metrics (e.g., energy consumption) can also be considered for QoS management. The details

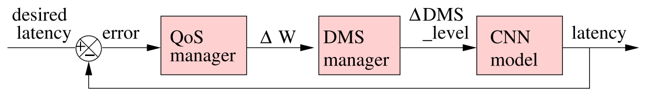


FIGURE 8. Feedback control of QoS using DMS.

of system modeling and controller design are out of scope of this paper, and readers are referred to [29]–[31].

IV. EVALUATION

In this section, we evaluate DMS and compare it with a state-of-the-art deep learning inference runtime. The objectives of the performance evaluation are 1) to test the effects of uncoordinated interactions between DMS and system-wide resource managers, 2) to investigate if DMS can support QoS under unpredictable conditions, 3) to test the effectiveness of DMS in supporting concurrent inference tasks, and finally 4) to investigate if DMS can be applied to other models, platforms, and datasets.

A. EXPERIMENT SETUP

We have implemented DMS by extending Caffe [18], an open source deep learning framework. Most deep learning frameworks, such as Caffe, provide runtime environments for efficient inference, but they do not support QoS. Though our implementation of DMS is based on Caffe, QoS management using DMS is framework-neutral and applicable to any deep learning frameworks such as TensorFlow [32].

In our testbed, DMS-enabled Caffe runs on a NVIDIA Jetson TX2 mobile platform. The operating system of Jetson TX2 is Ubuntu 18.04 Linux, and it supports several CPU frequency scaling policies through DVFS governors [33]. For example, the *on-demand* DVFS governor adjusts the CPU clock frequency according to CPU utilization. For experiments, we use CPU processors to investigate the interaction between DMS and the system-wide resource managers such as DVFS governors. Unlike CPUs, Jetson TX2 does not provide DVFS governors for GPUs, and, hence, their clock frequency must be set manually by users. During the evaluation, the power/energy consumption of the Jetson TX2 is monitored by a Yokogawa WT310E power meter. Real-time energy measurements are reported to the Jetson TX2 via the USBTMC protocol.

For the evaluation, we use only ResNet-50 since, as discussed in Section III-B.1, shallow models such as VGG-16 suffer from drastic degradation of accuracy for pruning filters. We believe that such drastic degradation of accuracy is not acceptable for most deep learning applications. Although pruned VGG-16 models can recover some accuracy through off-line retraining [13], it is not applicable at runtime due to the high cost of retraining.

B. UNCOORDINATED INTERACTION WITH SYSTEM-LEVEL RESOURCE ADAPTERS

Modern mobile and embedded devices provide various system-level adaptive resource managers, such as DVFS governors, to efficiently use shared resources such as energy. For instance, the on-demand DVFS governor of Linux systems automatically controls the frequency and voltage of processors according to the processor utilization [33]. However, these system level approaches cannot support per-task QoS goals such as inference latency because they are oblivious to application-level QoS goals.

To show the effect of uncoordinated interactions between DMS and system-wide resource managers, the same experiment, as in Section III-B.1, is performed, while varying number of layers are pruned by 30%. The result in Section III-B.1 was obtained while the system-level DVFS governor was disabled. In contrast, in this evaluation, the on-demand DVFS governor is activated during the experiment.

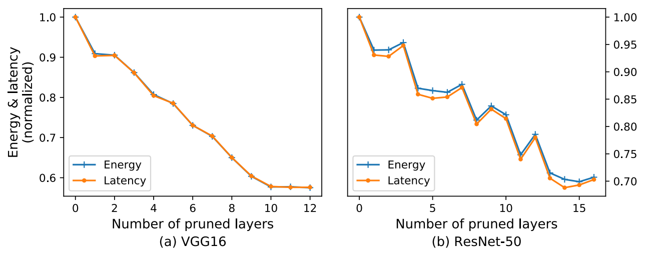


FIGURE 9. Uncoordinated interaction of DMS with the system's on-demand DVFS governor.

The results in Figure 9 show that uncoordinated interactions between the adaptive systems manifest unstable and unpredictable behavior, particularly in ResNet-50. Even though computational costs of inference is decreased steadily by the incremental pruning across layers, the energy consumption and inference latency change in an unpredictable manner, as shown in Figure 9-(b). With this unpredictable behavior, DMS cannot support the desired QoS, as will be shown in the following experiments.

The coordination between DMS and the system-wide resource managers is an interesting research problem. Prior studies demonstrated that uncoordinated interaction between adaptive systems can lead to unstable and oscillatory behavior even when individual adaptive components are provably well-behaved [34]. However, our goal, in this work, is not to fully explore the spectrum of available solutions. We leave this problem as our future work, and, in this work, we limit the role of system-wide resource managers to determine the optimal system configuration to achieve system-wide goals, such as energy efficiency. The energy efficiency (EE) of inference tasks is defined as follows:

$$EE = \frac{\text{inference performance}}{\text{power(W)}} = \frac{\frac{\text{images}}{\text{time(sec)}}}{\text{power(W)}} = \frac{\text{images}}{\text{energy(J)}} \tag{3}$$

Figure 10 shows the energy efficiency while clock frequency is varied. The result shows that the higher energy efficiency is achieved at the higher clock frequencies. This result is consistent with previous empirical studies showing that *race-to-idle* is often the most energy efficient resource allocation strategy in practice [35] [36]. Therefore, in the

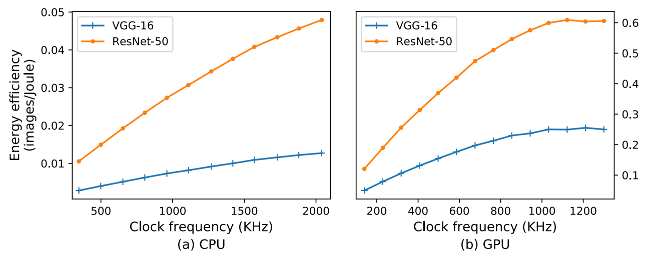


FIGURE 10. Energy efficiency of inference tasks while clock frequencies of processors are varied.

following experiments, we set the the processor clock frequency to the maximum. Once an optimal system-wide configuration is fixed by a system-level manager, the QoS manager for DMS dynamically scales CNN models to support per-task QoS goals.

C. SUPPORTING QoS WITH DMS

In this set of experiments, we test if DMS is effective in supporting QoS goals under varying environments. We choose to use ResNet-50 for these experiments because, as shown in Section III-B.1, the accuracy loss of ResNet-50 for model scaling is much less than VGG-16's. Further, since ResNet-50 has 53 convolution layers, it is easier to select less sensitive convolution layers across multiple layers.

1) SUPPORTING QoS UNDER UNPREDICTABLE WORKLOADS

In this experiment, we test if DMS can support QoS goals against unpredictable workloads. For this experiment, an inference task runs continuously to process input images, and the target inference latency is set to 3.0 seconds as a QoS goal. Performance is monitored as the inference task runs continuously over 150 monitoring periods.[2] At 50th monitoring instant, a disturbance thread is activated and continues until the 100th monitoring instant. The disturbance thread performs matrix multiplications of random sizes to interfere with the inference task. The accuracy of a model is plotted by relating the number of pruned layers monitored during the experiment to the validation accuracy in Figure 4.

Experiments are performed with 4 different versions of Caffe runtime. First, *DMS(coord)* is the DMS-enabled Caffe runtime that supports QoS goals using dynamic model scaling. In *DMS(coord)*, clock frequency is set to the maximum. *DMS(uncoord)* is the same as *DMS(coord)*, but on-demand governor is activated for system-wide power management. The on-demand governor and the DMS manager are not aware of each other, and, hence, they make control decisions independently. In *Sys_only*, the inference task runs on the vanilla Caffe runtime without QoS support, and the on-demand governor runs during the experiment. *MaxFreq* also runs the inference task on the vanilla Caffe runtime, but the CPU clock frequency is set to the maximum. Both *Sys_only* and *MaxFreq* are not aware of task's QoS goal.

[2]One monitoring period is the time to complete a single inference.

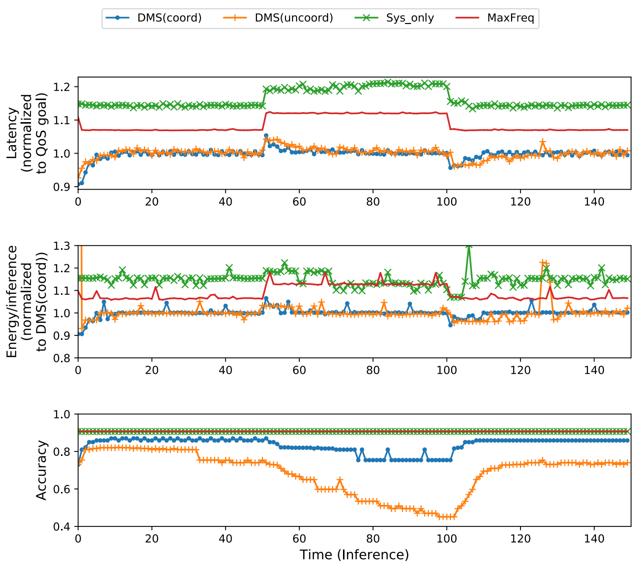


FIGURE 11. Transient behavior of DMS and baselines.

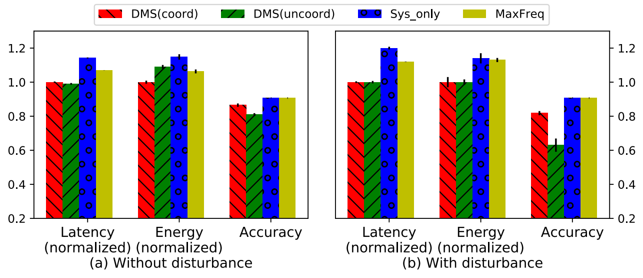


FIGURE 12. Average performance of DMS and baselines with and without disturbance.

Figure 11 shows the transient behavior during the experiment, and Figure 12 shows the average of the performance. The results demonstrate that *DMS(coord)* supports the target latency with the highest energy efficiency. For instance, Figure 12-(a) shows that *DMS(coord)* consumes at least 6% less energy than other baseline approaches when there is no disturbance. Although *MaxFreq* runs the inference task at maximum processor speed, its latency is longer than *DMS(coord)* and it cannot support the target latency. In contrast, *DMS(coord)* can do additional acceleration to meet the target latency at modest accuracy loss. For example, in the absence of disturbance, the accuracy of *DMS(coord)* is 0.865, which is about 4% less than the original inference accuracy of 0.907. This result shows that DMS is very effective in achieving additional performance gains, such as energy savings and reduced latency, while minimizing the loss of accuracy.

When the disturbance thread is active between the 50th and the 100th monitoring periods, additional performance degradation is observed in *Sys_only* and *MaxFreq*. For instance, the energy consumption and the latency of *MaxFreq* are increased by 5% and 7%, respectively. In contrast, *DMS(coord)* closely supports the target latency despite this disturbance. *DMS(coord)* makes further resource-accuracy trade-offs to support the target latency. For example, Figure 12-(b) shows that the accuracy of *DMS(coord)* is decreased to 0.818 while the disturbance is active. The energy consumption of *DMS(coord)* does not change much despite the disturbance because the scaled model reduces the resource demand.

The last thing to note in this experiment is the unpredictable behavior of *DMS(uncoord)*. Although *DMS(uncoord)* seems to support the target inference latency throughout the experiment, it manifests unpredictable changes in inference accuracy, especially when the disturbance thread is active. As shown in Figure 11, when the disturbance is present, the accuracy drops slowly from 0.75 to 0.47. Since the QoS support via DMS is not coordinated with the on-demand DVFS governor, these two adapters interact with each other in an uncontrolled manner. For example, if the DMS manager scales down the model to reduce computational costs, the on-demand governor might reduce the processor speed due to the decreased processor utilization, and this, in turn, leads to further model scaling by the DMS manager, to accelerate the inference task again. However, these interactions are not predictable, if not coordinated. As this result shows, the QoS support using DMS needs careful coordination with system-wide resource managers to avoid unnecessary loss of accuracy.

2) SUPPORTING QoS FOR CONCURRENT INFERENCE TASKS

Another advantage of DMS is that multiple concurrent inference tasks with their respective QoS goals can share a CNN model. To demonstrate this advantage, we run three inference tasks concurrently for 150 monitoring periods.[3] The target inference latency of *Task A*, *B*, and *C* are set to 2.9, 3.2, and 3.5 seconds, respectively. These 3 tasks share a single ResNet-50 model, and they are scheduled continuously by DMS-enabled Caffe runtime on a CPU core in a round-robin fashion; for example, if one task completes an inference, the next task immediately starts its inference. As previous experiments, a disturbance thread is activated between the 50th and the 100th monitoring periods. During the experiment, the CPU clock frequency is set to the maximum for energy efficiency.

Figures 13 and 14 show the results. In Figure 13, all three inference tasks satisfy closely their respective latency goals. However, Figure 14-(a) shows that these 3 tasks manifest different resource-demands to satisfy their respective latency goals. For example, *Task A* has the shortest latency goal, and it reduces the computational cost of the inference by scaling down the model until about 20% accuracy loss is incurred. *Task C*, in contrast, has the longest latency goal, and it needs only 0.1% accuracy loss to meet the target latency.

Figure 13 shows that all three tasks react promptly to the disturbance to support the target latency. As shown in Figure 14-(b), all three tasks satisfy the target latency despite

[3]One monitoring period is the time for the three tasks to complete their inference once.

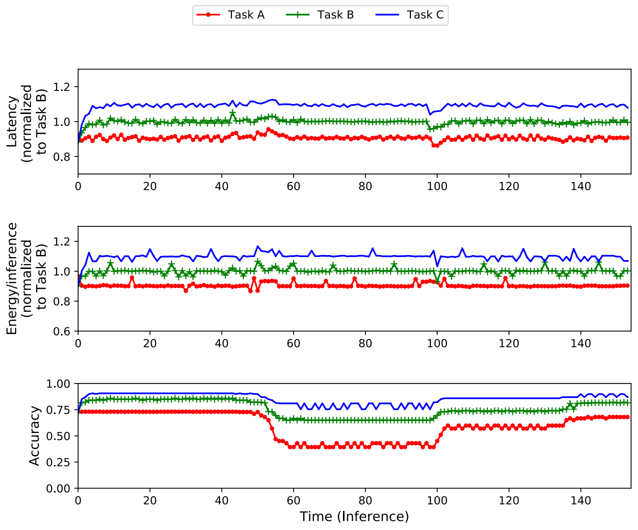


FIGURE 13. Transient behavior of concurrent tasks.

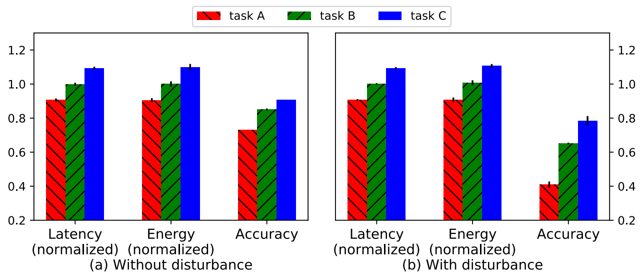


FIGURE 14. Average performance of concurrent inference tasks with and without disturbance.

TABLE 2. ResNet-32 model for CIFAR-10 dataset.

	# of parameters	# of conv. layers	# of total layers	accuracy
ResNet-32	0.46M	31	32	91.8

the disturbance. They make additional resource-accuracy trade-offs to handle the disturbance. For instance, the accuracies of *Task A*, *B*, and *C* are decreased to 0.41, 0.65, and 0.78, respectively.

D. SMALL SCALE CLASSIFICATION TASKS USING CIFAR-10 DATASET AND ResNet-32

1) CIFAR-10 DATASET AND ResNet-32 MODEL

To demonstrate the generality of our approach, we conduct further studies on small scale classification tasks using the CIFAR-10 dataset [27], the ResNet-32 deep learning model [4], and a Jetson TK1 embedded board. The ResNet-32 model, shown in Table 2, is a tiny CNN model adapted from the original ResNet models to process small input images of 32 $\times$ 32 sizes. The ResNet-32 model has only 0.46M parameters, which is about 1/55 of original ResNet-50 models, and, hence, has far less resource demands. The ResNet-32 model is trained and tested with CIFAR-10 dataset that consists of 50k training images and 10k test images in 10 classes. For DMS, the filters of the trained ResNet-32 model is reorganized in

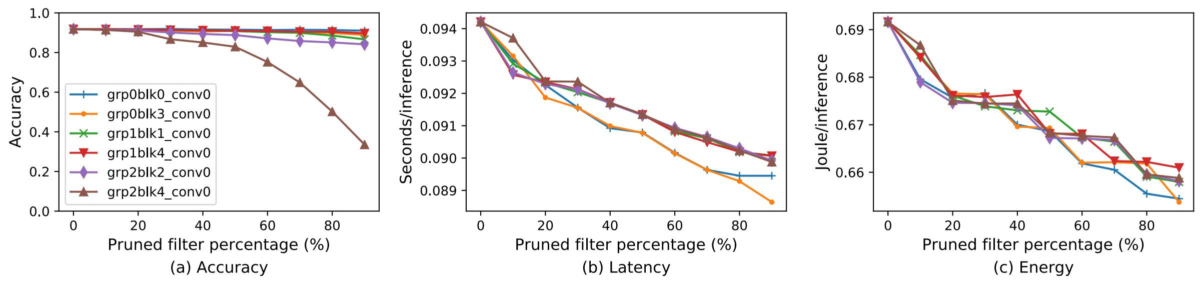


FIGURE 15. **Accuracy and performance while varying pruning ratios are applied to the layers of ResNet-32 (CIFAR-10) running on an Nvidia Jetson TK1 device.**

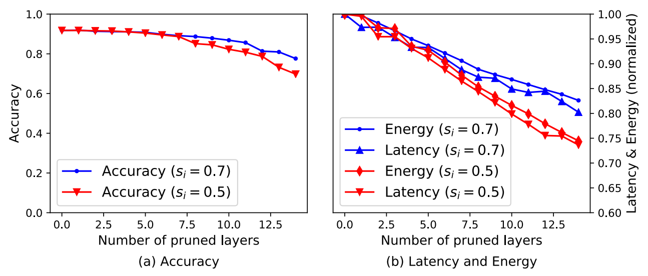


FIGURE 16. **Accuracy and performance while varying number of layers of ResNet-32 (CIFAR-10) are pruned with scaling factor s_i.**

the smallest ℓ1-norm order in the off-line phase. For this set of experiments, we choose to use a Jetson TK1 embedded board that has less computing power than the Jetson TX2. The main memory of the Jetson TK1 is 2GB, which cannot accommodate large models such as VGG-16, but is enough for tiny models like ResNet-32.

2) SCALING WITHIN AND ACROSS ConvNet LAYERS

Figure 15 shows the accuracy and performance while a few chosen layers of ResNet-32 are pruned with varying ratios. As shown in Figure 15-(a), the accuracy of ResNet-32 decreases monotonically when the higher pruning ratios are applied. Most layers of ResNet-32 are robust against the filter pruning. For example, pruning 90% of the filters results in less than 9% accuracy degradation in most layers. An exception is the *grp2blk4_conv0* layer that has an accuracy of 0.335 after pruning 90% of the filters. This is because *grp2blk4_conv0* is the closest layer to the output layer, so pruning filters of the layer directly affects the final output. Figures 15-(b) and -(c), respectively, show the inference latency and energy consumption in the same experiment. As with large models, we can observe gradual and monotonic decreases of both latency and energy consumption. For instance, ResNet-32 can achieve approximately 5-9% acceleration and energy savings by pruning 90% of the filters of a layer.

Figure 16 shows the inference accuracy and performance when filters are pruned across the layers with 2 different scaling factors, 0.7 and 0.5. Pruning filters across a network has a multiplicative effect both in accuracy and performance. Therefore, some sensitive layers such as *grp2blk4_conv0* should be excluded from the candidate layers of pruning to prevent drastic degradation of accuracy. As shown in Figure 16-(b), ResNet-32 also achieves linear performance gains as the number of pruned layers increases. These results are consistent with the results from the large models. The higher gain in power consumption and inference acceleration can be achieved with greater accuracy loss. For example, when the scaling factor is 0.5, it can achieve further 8.2% energy saving and 6.6% latency reduction compared to the scaling factor 0.7.

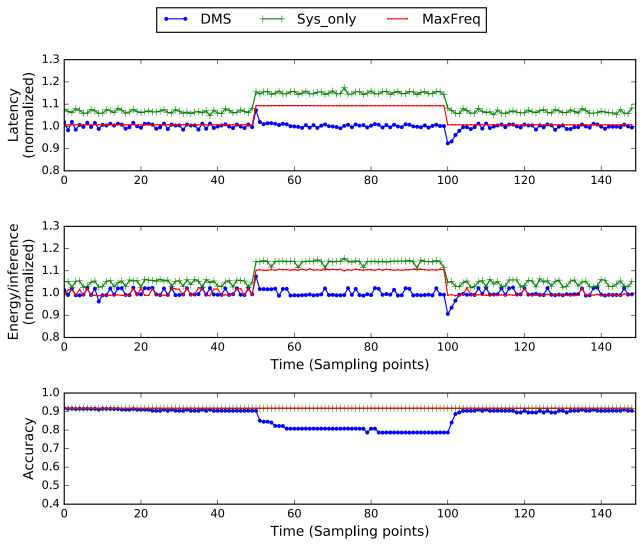


FIGURE 17. **Transient behavior of DMS and baselines with ResNet-32 on the Jetson TK1 device.**

3) QoS MANAGEMENT ON A JETSON TK1 DEVICE

In this experiment, we test the effectiveness of DMS's QoS support on a more resource-constrained Jetson TK1 device. As in Section IV-C, an inference task runs continuously to process input images, and the target latency is set to 90 msec as a QoS goal. At the 50th monitoring instant, a disturbance thread is activated and continues until the 100th monitoring instant.

Figure 17 shows the transient behavior of DMS and baseline approaches. The result shows that DMS supports the target latency with the highest energy efficiency. For instance, in the absence of disturbance, DMS closely supports the target latency while consuming about 4.9% less energy than *Sys_only*. When there is disturbance, this gap widens to about 14.0%. *MaxFreq* runs the inference task at the most energy-efficient CPU frequency, and, hence, its energy consumption is similar to DMS's when there is no disturbance. However, while the disturbance thread is activated, its latency and energy consumption are increased due to resource contention. For example, *MaxFreq*'s latency and energy consumption increase by about 10% during the presence of disturbance. Unlike these baseline approaches, DMS maintains the target latency with a modest loss of accuracy despite the disturbance. For example, during the presence of disturbance, DMS's inference accuracy drops from about 0.909 to 0.801.

V. RELATED WORKS

The computational complexity and size of deep learning models continues to increase in order to achieve higher inference accuracy, and this poses a major challenge for resource-constrained mobile and embedded devices. To address this problem, there has been significant effort to develop efficient and light-weight deep learning models, methods, and runtime environments for mobile and embedded devices.

Some small footprint deep learning models have been designed for resource-constrained mobile devices [37], [38]. However, handcrafting a small footprint network from scratch requires enormous development time and efforts. Further, the fixed computation cost of such models cannot meet the wide and varying resource requirements of mobile and embedded applications.

Inspired by early work of LeCun *et al.* [39], there have been several works focusing on pruning deep learning models to reduce their complexity [13], [21], [22]. Anwar *et al.* [22] proposed to use particle filters to measure the importance of connections and paths. Han *et al.* [21] iteratively removes weights having small magnitudes to obtain a pruned model without loss of accuracy. After pruning, these approaches perform a large amount of retraining to recover the original accuracy. Most of these pruning methods generate irregular network structures after pruning, rendering little saving of actual computation costs. Han *et al.* [14] proposed a specialized hardware accelerator for efficient processing of such irregular network structures.

Filter-based pruning of CNN models was proposed by Li *et al.* [13]. Unlike weight-based pruning approaches, filter-based pruning does not produce irregular network structure after pruning. In *network slimming* scheme [40], the importance of convolution filters and channels are automatically identified during training by imposing sparsity-induced regularization on the scaling factors in batch normalization layers. The filters are pruned according to the learned importance. *AMC* [28] exploits reinforcement learning to learn proper pruning ratios in a layer-by-layer manner. Both *network slimming* and *AMC* are complementary to DMS, and they can be used by DMS to learn the importance of filters and optimal per-layer scaling factors.

Another line of work explored dynamically scaling computation of CNN models either through early termination [41]–[43] or conditional execution [44]–[46]. For instance, in [42], a cascade of intermediate classifiers are used to allow early termination when samples can already be inferred with high confidence. In [46], gating modules are added to backbone networks to regulate skipping convolution layers on a per-input basis. Lin *et al.* [45] proposed to use a RNN-based decision network that decides how to prune convolution kernels conditioned on the input image and current feature maps. These dynamic scaling approaches demonstrated high accuracy comparable to the original network since they skip some computation on easy input images. However, their pruning decisions are based on input images, not on the QoS goals and resource-demands of applications. We believe that these conditional execution mechanisms can be combined with the QoS management architecture of this work to support QoS goals while maintaining high accuracy.

Several runtime environments have been proposed for efficient and QoS-aware deep learning inference in mobile and embedded devices [10], [19], [29], [31], [47]. *DeepX*, for example, is a software accelerator that decomposes a deep learning model into unit-blocks for efficient execution on heterogeneous local device processors [10]. *NestDNN* [47] provides flexible resource-accuracy trade-offs by nesting a few models having different resource-accuracy trade-offs in a single model and selecting an optimal model at runtime. Kang and Chung [29], [31] proposed a deep learning runtime, called *DeepRT*, that provides predictable inference latency and power consumption by controlling the speeds of CPUs and GPUs simultaneously. Kang *et al.*'s work has a similar goal to ours. However, they only consider system-level resource management issues to support predictable inference, not exploiting the characteristics of deep learning models.

DMS's resource-accuracy trade-off follows the principle of *approximate computing* [48], in which applications trade accuracy for reduced resource usage, which, in turn, contributes to the benefits in power, energy, execution time, etc. For instance, Hoffman [34] proposed a runtime optimization systems, called JouleGuard, that takes an energy goal and dynamically configures both application and system to ensure the goal is met and application accuracy is near optimal. Chen *et al.* [49] proposed several approximate computing techniques for deep neural networks (DNNs), such as quantization and gradient compression, and demonstrated that DNNs are resilient to numerical errors from approximate computing. DMS's pruning-based approach to approximate computing is orthogonal to such previous techniques, and can be combined with them for further benefits.

VI. CONCLUSION

In this paper, we presented DMS, a runtime control mechanism that supports QoS for deep learning applications.

In order to control the resource demands of deep learning applications, DMS scales deep learning models at runtime by adaptive pruning of convolution filters. Since the model scaling mechanism of DMS does not incur runtime overhead, it can be combined with runtime feedback control mechanisms to guarantee QoS goals. We presented performance evaluation using experimental prototypes. The results are very encouraging in that DMS can control the resource consumption of concurrent deep learning applications at a modest loss of accuracy. Our evaluation results also show that, if properly coordinated with system level resource managers, DMS can support highly robust and efficient inference performance against unpredictable workloads.

While we believe that this work is a significant step towards QoS-aware deep learning for mobile and embedded devices, further study is needed in several directions. First, DMS needs to be expanded to support more diverse deep learning models such as DNNs and *recurrent neural networks (RNNs)*. Second, the coordination between DMS and system-wide resource managers deserves more in-depth exploration. In particular, our result shows that pruning filters of relatively shallow CNN models such as VGG-16 can significantly degrade the inference accuracy. For such shallow models, we plan to investigate methods that combine system-level resource management techniques with DMS to minimize the loss of accuracy. Finally, we are interested in combining input-dependent dynamic network structures to DMS. We believe that the conditional execution mechanisms of input-dependent dynamic networks can be exploited by the QoS management architecture of DMS to support various QoS goals with minimal loss of accuracy.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[2] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[5] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 377–392.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 779–788.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[8] *This Powerful Wearable is a Life-Changer for the Blind*. Accessed: Sep. 2019. [Online]. Available: https://blogs.nvidia.com/blog/2016/10/27/wearable-device-for-blind-visually-impaired/

[9] F. Falcini, G. Lami, and A. M. Costanza, "Deep learning in automotive software," *IEEE Softw.*, vol. 34, no. 3, pp. 56–63, Feb. 2017.

[10] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2016, Art. no. 23.

[11] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1269–1277.

[12] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.* Cambridge, MA, USA: MIT Press, 2015, pp. 1135–1143.

[13] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–13.

[14] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.

[15] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 92–104.

[16] N. P. Jouppi C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, Jun. 2017, pp. 1–12.

[17] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, "Intel math kernel library," in *High-Performance Computing on the Intel Xeon Phi: How to Fully Exploit MIC Architectures*. Cham, Switzerland: Springer, 2014, pp. 167–188, doi: 10.1007/978-3-319-06486-4_7.

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[19] (2019). *Nvidia TensorRT*. [Online]. Available: https://developer.nvidia.com/tensorrt

[20] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009.

[21] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–14.

[22] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, p. 32, 2017.

[23] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," in *Proc. Interspeech*, 2013, pp. 2365–2369.

[24] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. Brit. Mach. Vis. Conf. (BMVA)*, 2014, pp. 1–12.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.

[28] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 784–800.

[29] W. Kang and J. Chung, "Power- and time-aware deep learning inference for mobile embedded devices," *IEEE Access*, vol. 7, pp. 3778–3789, 2019.

[30] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Hoboken, NJ, USA: Wiley, 2004.

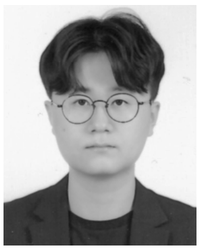
[31] W. Kang and J. Chung, "DeepRT: Predictable deep learning inference for cyber-physical systems," *Real-Time Syst.*, vol. 55, no. 1, pp. 106–135, 2019.

[32] M. Abadi P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[33] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proc. Linux Symp.*, vol. 2, 2006, pp. 215–230.

[34] H. Hoffmann, "JouleGuard: Energy guarantees for approximate applications," in *Proc. 25th Symp. Oper. Syst. Princ. (SOSP)*, New York, NY, USA, 2015, pp. 198–214.
[35] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.
[36] D. H. K. Kim, C. Imes, and H. Hoffmann, "Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics," in *Proc. IEEE 3rd Int. Conf. Cyber-Phys. Syst. Netw. Appl.*, Aug. 2015, pp. 78–85.
[37] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: https://arxiv.org/abs/1602.07360
[38] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: https://arxiv.org/abs/1704.04861
[39] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 598–605.
[40] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis.*, Jun. 2017, pp. 2736–2744.
[41] S. Teerapittayanon, B. McDanel, and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit. (ICPR)*, Dec. 2016, pp. 2464–2469.
[42] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–14.
[43] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2017, pp. 1039–1048.
[44] Y. Rao, J. Lu, J. Lin, and J. Zhou, "Runtime network routing for efficient image classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 10, pp. 2291–2304, Oct. 2019.
[45] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2181–2191.
[46] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "SkipNet: Learning dynamic routing in convolutional networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 409–424.
[47] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, New York, NY, USA, 2018, pp. 115–127.
[48] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, 2016, Art. no. 62.
[49] C.-Y. Chen, J. Choi, K. Gopalakrishnan, V. Srinivasan, and S. Venkataramani, "Exploiting approximate computing for deep learning acceleration," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*. Mar. 2018, pp. 821–826.

WOOCHUL KANG received the Ph.D. degree in computer science from the University of Virginia, in 2009. He was a Senior Researcher with the Electronics and Telecommunications Research Institute, South Korea, from 2000 to 2004 and from 2009 to 2012, and a Postdoctoral Research Associate with the University of Illinois at Urbana-Champaign, USA, from 2012 to 2013. He is currently an Associate Professor with the Department of Embedded Systems Engineering, Incheon National University, Incheon, South Korea. His research interests include real-time systems, distributed middleware, feedback control of computing systems, and energy-efficient deep learning.



DAEYEON KIM received the B.S. degree from the Department of Embedded Systems Engineering, Incheon National University, South Korea, in 2018, where he is currently pursuing the master's degree. His current research interest is deep learning model compression for energy-efficient embedded systems.

JUNYOUNG PARK received the B.S. degree from the Department of Embedded Systems Engineering, Incheon National University, South Korea, in 2019, where he is currently pursuing the master's degree. His current research is focused on the classification of vector mosquitoes using deep learning.

• • •