DiReCtX: Dynamic Resource-Aware CNN Reconfiguration Framework for Real-Time Mobile Applications

Zirui Xu, Fuxun Yu, Zhuwei Qin, Chenchen Liu, and Xiang Chen

***Abstract*—Although convolutional neural networks (CNNs) have been widely applied in various cognitive applications, they are still very computationally intensive for resource-constrained mobile systems. To reduce the resource consumption of CNN computation, many optimization works have been proposed for mobile CNN deployment. However, most works are merely targeting CNN model compression from the perspective of parameter size or model structure, ignoring different resource constraints in mobile systems *with respect to* memory, energy, and real-time requirement. Moreover, previous works take accuracy as their primary consideration, requiring a time-costing retraining process to compensate the inference accuracy loss after compression. To address these issues, we propose *DiReCtX*—a dynamic resource-aware CNN model reconfiguration framework. *DiReCtX* is based on a set of accurate CNN profiling models for different resource consumption and inference accuracy estimation. With manageable consumption/accuracy tradeoffs, *DiReCtX* can reconfigure a CNN model to meet distinct resource constraint types and levels with expected inference performance maintained. To further achieve fast model reconfiguration in real-time, improved CNN model pruning and its corresponding accuracy tuning strategies are also proposed in *DiReCtX*. The experiments show that the proposed CNN profiling models can achieve 94.6% and 97.1% accuracy for CNN model resource consumption and inference accuracy estimation. Meanwhile, the proposed reconfiguration scheme of *DiReCtX* can achieve at most 44.44% computation acceleration, 31.69% memory reduction, and 32.39% energy saving, respectively. On field-tests with state-of-the-art smartphones, *DiReCtX* can adapt CNN models to various resource constraints in mobile application scenarios with optimal real-time performance.**

***Index Terms*—Convolutional neural network (CNN), mobile device, model reconfiguration, neuron pruning, resource-aware.**

I. Introduction

In THE past few years, convolutional neural networks (CNNs) have been widely applied in various intelligent applications, such as image classification [1]–[3], speech recognition [4]–[6], etc. Benefited by complex model structures with massive parameters, CNNs achieve high recognition inference accuracy. However, such model structures also hinder CNN applications' performance on resource-constrained computing platforms, especially mobile systems. For example, when implementing a CNN-based augmented reality application on a Nexus 5×, about 700-M memory, 45% CPU utilization, and 1860-mW battery power will be consumed [7]. Considering mobile systems are featured with multitasking, dynamic application utilization, and specific user interaction, such conspicuous resource consumption significantly compromises the system performance.

To reduce the resource consumption of CNN computation, it is intuitive to generate smaller model to replace the original one. Many "model-oriented" optimization works have been proposed, which mainly reconfigure CNN models in terms of parameter settings and model structures [7]–[11]. For example, Meng *et al.* [8] utilized parameter quantization method for model parameter size reduction; Li *et al.* [9] pruned part of the neurons with insignificant absolute values to reduce model computation workload. Although these *model-oriented* methods can reduce the CNN computation consumption with smaller model structure and less parameter size, they did not take specific resource constraints into consideration during optimization process. Therefore, they failed to continuously control the model resource consumption below the budgets given by model platform, resulting ineffective adaptability even to a single resource constraint's varying levels.

Recently, a series of "resource-aware" optimization works emerged. Different with previous ones, resource-aware methods directly took the specific computation resource levels as the optimization targets for CNN model configuration [11], [12]. Therefore, they can achieve a controllable model resource consumption after the optimization. For example, Wang *et al.* [12] added an energy loss into the accuracy loss, which guided the CNN training for particular system energy budgets; Yang *et al.* [11] associated the convolutional neurons with processor-specific inference time, and then utilized the greedy search algorithm to prune convolution filters for real-time requirement. However, these works also had certain limitations: On the one hand, they usually only took one single type of resource constraint as the optimization target, ignoring various resources in mobile systems, e.g., energy and real-time requirement. On the other hand, most of them set accuracy with highest priority, hence required another time-consuming retraining processes to compensate the inference accuracy loss after compression. However, tens of minutes of retraining is not affordable for mobile computation scenarios.

Manuscript received October 14, 2019; revised January 7, 2020 and March 16, 2020; accepted May 3, 2020. Date of publication May 20, 2020; date of current version January 20, 2021. This work was supported in part by NSF under Grant 1717775. This article was recommended by Associate Editor J. Xu. *(Corresponding author: Xiang Chen.)*

Zirui Xu, Fuxun Yu, Zhuwei Qin, and Xiang Chen are with the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030 USA (e-mail: zxu21@gmu.edu; fyu2@gmu.edu; zqin@gmu.edu; xchen26@gmu.edu).

Chenchen Liu is with the Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore, MD 21201 USA (e-mail: ccliu@umbc.edu).

Digital Object Identifier 10.1109/TCAD.2020.2995813

0278-0070 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

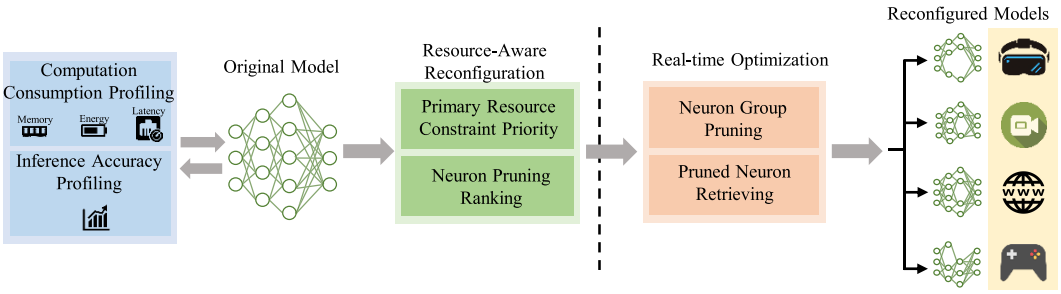


Fig. 1. Framework overview.

In this article, we will tackle these challenges by answering the following questions.

1) How to develop a generic CNN reconfiguration method that can control the model resource consumptions to distinct resource types and levels on mobile systems?
2) How to enable fast CNN reconfiguration to adapt dynamic resource constraint change in real-time computation scenarios?

Specifically, we proposed *DiReCtX*, an optimized dynamic resource-aware CNN model reconfiguration framework for real-time CNN-based mobile applications.

DiReCtX is based on a set of accurate CNN profiling models for resource consumption and inference accuracy estimation. With the above profiling models, we can easily estimate a reconfigured CNN model's resource consumption and inference accuracy, achieving a manageable consumption/accuracy tradeoffs. Therefore, *DiReCtX* can leverage neuron[1] pruning to reconfigure a pretrained CNN model to meet distinct resource constraints with expected inference performance maintained. To further achieve fast model reconfiguration for real-time requirement, improved CNN model pruning and accuracy tuning strategies are also proposed in *DiReCtX*. The framework overview of *DiReCtX* is shown in Fig. 1. With the neuron pruning-based reconfiguration scheme and corresponding optimization strategies, *DiReCtX* can real-timely reconfigure CNN models for different mobile computation scenarios, such as Web-browsing, video streaming, etc.

The contributions of this article are shown as follows.

1) We analyze individual CNN neuron's impact to different resource consumption types, *with respect to* the overall energy cost, memory usage, and computation time. Based on the analysis, we build multiple profiling models to estimate the resource consumption of CNN computation.
2) We analyze individual CNN neuron's impact to the overall inference accuracy and build a profiling model to estimate a CNN model's accuracy with various neuron configurations.
3) We propose a resource-aware CNN model reconfiguration scheme. With dedicated neuron pruning method, pretrained CNN models can be reconfigured into manageable computation consumption and inference accuracy, adapting to different resource constraint types and levels.
4) We further propose two optimization strategies for the CNN model reconfiguration scheme, namely, "neuron group pruning" for fast reconfiguration and "pruned neuron retrieving" for inference accuracy tuning. These two strategies effectively reduce the neuron pruning and accuracy tuning effort and, therefore, significantly improve the reconfiguration performance in real-time.
5) We comprehensively integrate the proposed *DiReCtX* into a state-of-the-art smartphone application. The application can real-timely reconfigure CNN models in various mobile computation scenarios for performance optimization and evaluation.

The experimental results show that: the proposed CNN profiling models can achieve expected accuracy of 94.6% and 97.1% for CNN model computation consumption and inference accuracy estimation. With *LeNet*, *CaffeNet*, and *VGG-13* as the major CNN models for evaluation, the proposed reconfiguration scheme can achieve at most 44.44% computation acceleration, 31.69% memory reduction, and 32.39% energy saving, respectively. On-field tests with state-of-the-art smartphones, *DiReCtX* can effectively adapt CNN models to various resource constraints in dynamic mobile application scenarios with optimal real-time performance.

II. Preliminary

A. CNN Model Computation Consumption

A typical CNN model usually consists of a set of layers, i.e., convolutional layers, fully connected layers, pooling layers, and nonlinear layers (e.g., *ReLu* layers) [2]. Since pooling layers and nonlinear layers occupy a small portion of model components, we only consider the computation optimization in convolutional layers and fully connected layers.

1) *Convolutional Layers:* The computation in each convolutional layer involves multiple neurons, input feature maps, and output feature maps. Each single neuron in convolutional layers is denoted as a convolutional filter, which represents a group of weights and usually plays as the fundamental computation component in convolutional layers. Leveraging the 2-D convolution operation between convolutional filters and input feature maps, each convolutional layer can achieve input feature extraction and generate output feature maps for following layers.
2) *Fully Connected Layers:* Fully connected layers transform the output feature maps from the last convolutional layer into linear vectors and performance the final confidence evaluation for each inference class.

When executing a CNN model in mobile systems, the computation in both kinds of layers are usually conducted by CPUs and measured with the amount of multiply accumulate operations (MACs). Fig. 2 depicts our evaluation of

[1]We collectively refer convolutional filters and fully connected layer neurons as "neurons", as both of them will be optimized together in this article.

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

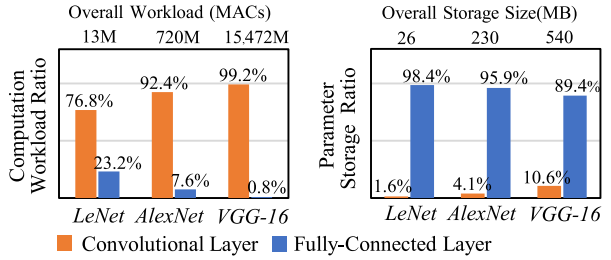


Fig. 2. Mobile CNN computation consumption analysis.

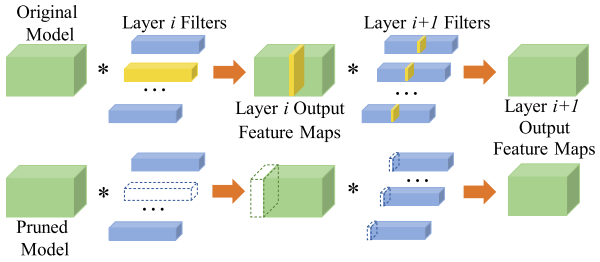


Fig. 3. Neuron pruning in convolutional layers.

both convolutional layers and fully connected layers regarding computation workload and parameter size. Three typical CNN models for image classification task are evaluated, i.e., *LeNet* [13], *AlexNet* [2], and *VGG-16* [14].

1) The three CNN models' overall computation workloads are 13 M, 720 M, and 15 472 M MACs, respectively. Convolutional layers contribute most of the computation workload (76.8% ∼99.2%). Considering the maximum mobile computing bandwidth is around 1∼2*G* MACs, such a computation workload may occupy the majority of computation resources [15].
2) Different with convolutional layers, fully connected layers contribute the most memory occupancy (89.4%∼98.4%). When computing fully connected layers, a large volume of system memory is required not only for native CNN model parameters but also for dynamically generated feature maps [16]. According to our evaluation, the real-time memory occupancy of *LeNet*, *AlexNet*, and *VGG-16* are at least 26 MB, 230 MB, and 540 MB for a single image inference, while the available run-time memory in a mobile system (*esp.* smartphone) is usually under 1 GB [15].

From the above analysis, we find distinct computation characters between convolutional layers and fully connected layers. In later sections, we will focus on these two kinds of layers to investigate expected CNN reconfiguration schemes.

B. Model-Oriented CNN Computation Optimization

To improve CNN computation performance on resource-constrained systems, a lot of CNN optimization methods are proposed. Most of these works fall into a "model-oriented" approach, which optimize CNN models in different model aspects, e.g., CNN model structure, parameter setting, and computation process.

1) Model design, such as *ShuffleNet* [17], *SqueezeNet* [18], and *Xception* [19]. These optimized CNN models adopt slim model structures or small model components to reduce computation workload.
2) Parameter compression, such as filter pruning [9] and weight sparsity [20]. These optimization methods identify and remove insignificant model components and, therefore, reduce both computation workload and memory occupancy.
3) Computation approximation, such as low-rank computation [21] and quantization [22]. These methods alleviate the CNN computation complexity by decoupling model structures with approximate computations.

However, since these works are mainly oriented by CNN model optimization perspectives, they overlook specific hardware and system constraints associated with diverse computation scenarios. In other words, these works did not take the specific resource constraints into optimization process. Therefore, many model-oriented works fail to optimize the model's resource consumption to specific constraints.

For example, Yang *et al.* [23] showed that, even *SqueezeNet* can reduce parameter size to 50× less than traditional CNN (i.e., *AlexNet*), it still consumes even more computation energy. Moreover, most works reconfigure CNN models targeting a certain accuracy expectation with a singular model configuration approach. Therefore, when different resource constraint levels exist, these works fail to achieve feasible manageability to *dynamic consumption/accuracy tradeoff* [23].

C. Resource-Aware CNN Computation Optimization

To achieve *comprehensive performance enhancement* in practical CNN computation, many resource-aware CNN model configuration works are proposed. Rather than merely focusing on CNN model parameter and structure optimization, these resource-aware works extend the configuration methodologies in model-oriented to more practical computing scenarios, directly setting various resource constraints, *with respect to* energy cost, memory occupancy and computation time as model configuration targets.

Gordan *et al.* [24] leveraged single resource constraint regularizer to automatically determine the layer-wise shrink rate with regarding to computation workload optimization or model size optimization. Yang *et al.* [23] identified the CNN layer-wise energy cost and implemented corresponding filter pruning schemes for energy-constrained systems. Liu *et al.* [25] considered memory constraint as optimization targets and leveraged reinforcement learning techniques to configure CNN model to meet certain memory budgets.

Although achieving effective computation optimization regarding resource consumption, these existing resource-aware CNN model optimization works usually focus on a single resource constraint type and static level, which can be hardly applied to practical computation scenarios with various dynamic constraints.

D. Neuron Pruning for CNN Reconfiguration

As aforementioned, the resource-aware optimization methods are based on the above model-oriented methods, such as parameter compression and computation approximation. Among those model-oriented CNN model configuration works, "neuron pruning" is considered as one of the most efficient and effective methods. It can directly reconstruct CNN models on neuron level. By removing insignificant neurons,

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

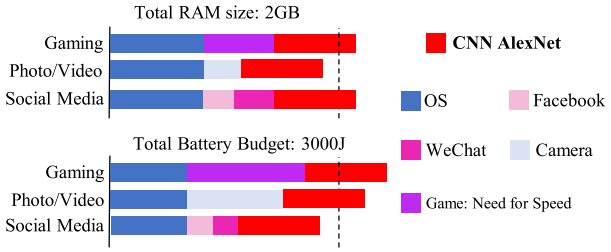


Fig. 4. Different application scenarios on an Android smartphone nexus 5×.

the model structure and functionality is trimmed with optimal performance retained.

Fig. 3 illustrates a pruning process targeting convolution filters in convolutional layers [26]. In the *i*th convolutional layer, the insignificant filters are identified by ranking their absolute filter matrix values. By pruning the insignificant filters, corresponding output feature maps will be also removed. As the *i*th layer's output feature maps are the inputs of the $(i+1)$th layer, all filters in the $(i+1)$th will have less sizes and, therefore, less computation workload. Such a pruning process can also apply to neurons in fully connected layers.

In this article, we combine the pruning schemes targeting convolutional filters and fully connected neurons together as the major configuration method for the proposed resource-aware CNN model reconfiguration.

III. Motivation

Although *resource-aware* optimization methods directly alleviate CNN computation consumption to a certain degree, complex mobile systems and various computation scenarios introduce more critical challenges.

Challenges:

1) *Distinct Resource Constraints:* In mobile systems, due to diverse mobile application scenarios, the computation resource constraints have distinct *types* and *levels*. One example is shown in Fig. 4: when deploying *AlexNet* on a Nexus 5× with different mobile applications in background, we can see distinct energy and memory budgets with different levels. The computation resource constraints for Gaming scenario are memory usage and energy cost while constraint for Social Media scenario is only memory usage. As for photograph/video scenario, it is only constrained by the energy. However, previous works only considered a singular type of resource constraint with a static level, lacking capability for distinct resource budgets under various computation scenarios.
2) *Real-Time Reconfiguration:* In mobile systems, the computation resource constraints are dynamically changed with varying computation scenarios. Therefore, the ideal CNN model optimization requires real-time model reconfigurations for different computation consumption adaptation. However, most existing CNN model optimization works take accuracy as the highest priority, which is aiming to obtain as high as possible accuracy. Therefore, they usually obey a reconfiguration-retraining strategy which requires an extra time-consuming retraining process after model reconfiguration. In that case, such methods cannot be well applied for dynamic mobile computation scenarios.

Proposed Solutions:

1) To address the first challenge, we build multiple profiling models to estimate CNN models' computation consumption and inference accuracy in Sections IV and V. With thorough understanding of CNN computation consumption/accuracy tradeoffs, we further propose a dedicated CNN model reconfiguration scheme based on neuron pruning in Section VI, which is designed to address distinct resource constrains with minimum CNN model reconfiguration effort. Specifically, we evaluate each neuron's importance not only from the perspective of accuracy contribution to the entire model but also considering its resource consumption.
2) To address the second challenge, we further improve the neuron pruning-based reconfiguration scheme with two optimization strategies: a) the neuron group pruning and b) pruned neuron retrieving in Section VII. The neuron group pruning can effectively enhance the reconfiguration speed and satisfy the real-time requirement. The pruned neuron retrieving can achieve fast accuracy tuning by recovering a few unnecessary pruned neurons removed by the first strategy.

By tackling the aforementioned challenges, we propose *DiReCtX* in Section VIII. *DiReCtX* integrates our resource-aware CNN model reconfiguration scheme and the two optimization strategies. *DiReCtX* can real-timely reconfigure a CNN model to dynamic and distinct computation resource constraints in mobile systems. The effectiveness and efficiency of *DiReCtX* is comprehensively evaluated in Section IX.

IV. Mobile CNN Computation Consumptions Profiling

To achieve manageable tradeoffs between CNN computation consumption and inference accuracy, a set of comprehensive CNN profiling models is highly required. Although some previous works have proposed certain models to profile CNN computation consumption, they merely focus on CNN models' inner parameter settings and structure designs, ignoring actual mobile system overheads [10], [25].

By taking mobile system computation overheads into consideration, we build three dedicate resource consumption profiling models for CNN consumption. The proposed profiling models can measure arbitrarily configured CNN models, *with respect to* the overall energy cost, memory usage, and computation time under various mobile computation scenarios.

These models are developed based on a general CNN design: 1) a CNN model has I convolutional layers and J fully connected layers; 2) for the ith convolutional layer, there are n^i_{Conv} filters with a size of $n^{i-1}_{\text{Conv}} \times s^i \times r^i$. During the convolution process, n^{i-1}_{Conv} input feature maps are fed from the $(i-1)$th layer to generate n^i_{Conv} output feature maps with a dimension of $h^i \times w^i$; and 3) the jth fully connected layer is composed of n^j_{FC} neurons.

A. Memory Usage Profiling Model for CNN Computation

1) Neuron-Level Memory Usage Profiling: Memory is one of the primary computation resources in mobile systems [27], which is mainly occupied by storing neurons' weights and generated feature maps. As our primary model configuration method is based on the neuron pruning, we first build profiling models to evaluate the memory usage when adding/removing individual CNN neurons.

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

Since neurons in the same layer have the identical memory consumption, we can formulate the memory usage M_{Conv} and M_{FC} for a single neuron in either convolutional layer or fully connected layer as

$$M_{\text{Conv}} = B_f\left(r^i s^i n_{\text{Conv}}^{i-1} + r^{i+1} s^{i+1} n_{\text{Conv}}^{i+1}\right) + B_a h^i w^i \quad (1)$$

$$M_{\text{FC}} = B_f\left(n_{\text{FC}}^{j-1} + n_{\text{FC}}^{j+1}\right) \quad (2)$$

where B_f and B_a are data bandwidths, which equal to 16-bit or 32-bit in common mobile systems. Equations (1) and (2) can be only used to evaluate the single neuron's occupancy on system memory without considering practical mobile system overheads yet.

2) CNN Model-Level Memory Usage Profiling: Based on (1) and (2), the overall memory usage from a CNN model can be formulated as

$$M_{\text{CNN}} = B_f\left(\sum_{i=1}^{I} r^i s^i n_{\text{Conv}}^{i-1} n_{\text{Conv}}^{i} + \sum_{j=1}^{J} n_{\text{FC}}^{j} n_{\text{FC}}^{j-1}\right) + B_a \sum_{i=1}^{I} h^i w^i n_{\text{Conv}}^{i}. \quad (3)$$

However, when executing a CNN model in practical mobile systems, certain overheads are introduced.

As the CNN model is programed into specific C++ libraries in mobile applications, these libraries will occupy extra system memory [28]. According to our preliminary experiments, the above overhead is linearly determined by the CNN model memory usage M_{CNN}. Therefore, we formed it as O_c. Meanwhile, since CNN-based mobile applications have specific user interfaces, graphic computation will also introduce certain overhead. Moreover, the mobile applications are developed with Java and native C/C++. Above two constant overheads for graphic computing and codes allocating are formed as O_o. Therefore, (3) can be improved as

$$M = \alpha_m(M_{\text{CNN}} + O_c) + O_o \quad (4)$$

where α_m is the coefficient parameter decided by the specific CNN-based application and the mobile device.

With these models, we can estimate the practical memory usage of a CNN model running in practical mobile systems.

B. Energy Cost Profiling Model for CNN Computation

Due to the limited mobile battery capacity, energy cost is also considered as one of the primary resource constraints [29]. Usually, the energy consumed by a CNN model includes two major parts: the computation energy cost E_{CPU} and the memory access energy cost E_{Mem}.

Mobile CPU Computation Energy Cost: E_{CPU} directly relates to the CNN model's total inference computation workload. Since most of the off-the-shelf mobile devices have not equipped with CNN-specific GPU or neural processing unit (NPU), they mainly compute CNN inference on CPUs. Therefore, certain CPU parameters, such as computation bandwidth and utilization rate, need to be taken into account. Considering mobile system is a dynamic process with changeable CPU utilization values. To simplify the modeling, we divide the total computation workload W into N parts and assume each part has an average utilization rate η_n. In that case, E_{CPU} can be modeled as following:

$$\begin{aligned} E_{\text{CPU}} &= \sum_{n=1}^{N}\left(\frac{W_n}{B_{\text{CPU}}^{\eta_n}} \times P_{\text{CPU}}^{\eta_n}\right) \\ &= \sum_{n=1}^{N}\left(\frac{W_n}{B_{\text{CPU}}^{\text{Peak}} \times \eta_n} \times \left(P_{\text{CPU}}^{\text{Peak}} \times \eta_n\right)\right) \\ &= \sum_{n=1}^{N} W_n \times \frac{P_{\text{CPU}}^{\text{Peak}}}{B_{\text{CPU}}^{\text{Peak}}} \end{aligned} \quad (5)$$

where B_{CPU} (MACs per second) and P_{CPU} (mW) are the accessible CPU computation bandwidth and the practical system power consumption; According to the [30], we can assume the current CPU computation bandwidth B_{CPU}^{η} equals to the product of CPU peak bandwidth and η_n. Moreover, if ignore the power overhead (we consider it can be included into total energy overhead in (11)), the current CPU power P_{CPU}^{η} equals to the product of CPU peak power and η_n. By offsetting η_n in both numerator and denominator, E_{CPU} is directly affected by peak CPU bandwidth, peak CPU power consumption, and total workload W. Since the peak CPU bandwidth and the power consumption are device-specific constants, the computation energy cost is mainly subject to the workload W.

Mobile Memory Access Energy Cost: During executing CNN model inference task, model weights and feature maps will introduce considerable memory access and, therefore, corresponding energy cost.

In mobile systems, the CNN model inference process obeys a layer by layer computing scheme. As aforementioned, since most off-the-shelf mobile systems still use normal CPUs for CNN computing, we consider a fundamental memory access scheme in here: during each layer's computing, the weights and feature maps are fetched from DRAM to the on-chip cache and the generated feature maps will be stored back to DRAM [31]. Also, unlike the advanced data reuse strategy in some latest NPUs [32], we adopt a simple yet widely used data reuse strategy mentioned in [33] to mimic the CNN inference flow in most of mobile systems: Each weight is reused $h^i w^i$ times for the same feature map. Meanwhile, each feature map pixel value is reused $r^i s^i$ times for the same neuron. Therefore, for each layer computing, we consider the weights are transferred from DRAM to Cache once and the feature maps are transferred twice between DRAM and Cache. Therefore, the memory access consumption E_{Mem} is determined by the total weight bit number M_w and feature map bit number M_{act}

$$E_{\text{Mem}} = 2\left(\varepsilon_f + \varepsilon_a\right) B_f M_{\text{act}} + \left(\varepsilon_f + \varepsilon_a\right) B_a M_w \quad (6)$$

where ε_c and ε_d denote the energy cost per bit when accessing on-chip cache and DRAM memory, respectively.

1) Neuron-Level Memory Access Energy Cost Profiling: Based on the above analysis, the energy cost caused by a single neuron in a convolutional layer or a fully connected layer can be formulated as E_{Conv} and E_{FC}

$$E_{\text{Conv}} = \frac{P_{\text{CPU}}^{\eta}}{B_{\text{CPU}}^{\eta}}\left(r^i s^i n_{\text{Conv}}^{i-1} h^i w^i + r^{i+1} s^{i+1} n_{\text{Conv}}^{i+1} h^{i+1} w^{i+1}\right) + \varepsilon_f B_f\left(r^i s^i n_{\text{Conv}}^{i-1} + r^{i+1} s^{i+1} n^{i+1}\right) + \varepsilon_a B_a h^i w^i \quad (7)$$

$$E_{\text{FC}} = \frac{P_{\text{CPU}}^{\eta}}{B_{\text{CPU}}^{\eta}} n_{\text{FC}}^{j-1} + \varepsilon_f B_f n^{j-1}. \quad (8)$$

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

2) CNN Model-Level Energy Cost Profiling: We further propose the overall energy cost profiling model of a certain CNN model running on mobile systems. The energy cost E_{CPU} is determined by CNN model's total computation workload, which can be formulated as

$$E_{\mathrm{CPU}} = \frac{P_{\mathrm{CPU}}^{\mathrm{Peak}}}{B_{\mathrm{CPU}}^{\mathrm{Peak}}}\left(\sum_{i=1}^{I} r^i s^i n_{\mathrm{Conv}}^{i-1} n_{\mathrm{Conv}}^{i} w^i h^i + \sum_{j=1}^{J} n_{\mathrm{FC}}^{j-1} n_{\mathrm{FC}}^{j}\right) \quad (9)$$

where $\sum_{i=1}^{I} r^i s^i n_{\mathrm{Conv}}^{i-1} n_{\mathrm{Conv}}^{i} w^i h^i + \sum_{j=1}^{J} n_{\mathrm{FC}}^{j-1} n_{\mathrm{FC}}^{j}$ is the term of the total computation workload W.

Also, the memory access energy cost E_{Mem} can be modeled as

$$E_{\mathrm{Mem}} = 2\big(\varepsilon_f + \varepsilon_a\big)B_f\left(\sum_{i=1}^{I} r^i s^i n_{\mathrm{Conv}}^{i-1} n_{\mathrm{Conv}}^{i} + \sum_{j=1}^{J} n_{\mathrm{FC}}^{j-1} n_{\mathrm{Conv}}^{j}\right) + \big(\varepsilon_f + \varepsilon_a\big)B_a \sum_{i=1}^{I} h^i w^i n_{\mathrm{Conv}}^{i}. \quad (10)$$

Besides the energy cost formulated in (9) and (10), we also take practical mobile system overheads into account. In mobile systems, a certain amount of energy is also consumed by basic CPU running consumption and mobile application inner processes, such as calling camera components for image recognition, executing Java code for run-time support, etc. We define such overheads as E_o, which will be determined by practical measurement with specific mobile devices. Therefore, the practical energy cost E of a CNN-based application running on the mobile system is modeled as

$$E = E_{\mathrm{CPU}} + E_{\mathrm{Mem}} + E_o. \quad (11)$$

For different CNN-based applications, the E_o will be different, causing various E value.

C. Computation Time Profiling Model for CNN Computation

Computation time is considered as one comprehensive performance evaluation criterion for CNN computation, which involves with computation bandwidth, computation workload, and real-time constraints. To profile the entire CNN model computation time, we first propose time-centric models based on (9), which evaluate a single neuron's computation time. We then estimate the practical computation time of a CNN model running on mobile systems with certain mobile system overheads involved. We borrow the insights from roofline model in [34] and assume the models we discuss in this article are in the compute-bound area. Therefore, the computation time cost is mainly affected by the computation workload.

1) Neuron-Based Computation Time Profiling: We formulate the theoretical computation time changes by adding/removing a single neuron in convolutional layer and fully connected layer as T_{Conv} and T_{FC}

$$T_{\mathrm{Conv}} = \frac{1}{B_{\mathrm{CPU}}^{\mathrm{Peak}} \times \eta_a}\left(r^i s^i n_{\mathrm{Conv}}^{i-1} h^i w^i + r^{i+1} s^{i+1} n_{\mathrm{Conv}}^{i+1} h^{i+1} w^{i+1}\right) \quad (12)$$

$$T_{\mathrm{FC}} = \frac{1}{B_{\mathrm{CPU}}^{\mathrm{Peak}} \times \eta_a} n_{\mathrm{FC}}^{j-1} \quad (13)$$

where η_a is the average CPU utilization rate for a specific mobile system.

2) CNN-Based Computation Time Profiling: By considering practical computation scenarios, we define R as a certain CNN model's execution number in a mobile application. Therefore, the computation time for the entire CNN model computation can be modeled as (14)

$$T = \frac{R}{B_{\mathrm{CPU}}^{\mathrm{Peak}} \times \eta_a}\left(r^i s^i n_{\mathrm{Conv}}^{i-1} n_{\mathrm{Conv}}^{i} w^i h^i + n_{\mathrm{FC}}^{j-1} n_{\mathrm{FC}}^{j}\right) + T_o^{\eta} \quad (14)$$

where T_o^{η} is the current computation time overhead introduced by mobile systems. For mobile CNN-based image classification applications, such overheads might be caused by the time delay of calling camera components and transmitting image data between processors and memory components [35]. According to our preliminary experiments, T_o^{η} can be considered as a fixed value for each CNN-based application. Please be noted that the proposed time profiling model is suitable for both serial computing and parallel computing since the CPU average computing bandwidth value exactly reflects the computing configuration. In other words, serial computing leads a lower CPU average computing bandwidth while parallel computing leads a higher bandwidth.

Equations (4), (11), and (14) provide the precise computation consumption profiling for various CNN models with arbitrary neuron configurations in practical mobile systems. When CNN models are reconfigured, the proposed models can help to reestimate resources required on mobile systems, *with respect to* memory usage, energy cost, and computation time. However, as many mobile system overheads are taken into consideration, we further introduce particular measurement process to determine the overhead parameters under specific CNN-based mobile application scenarios.

D. Mobile-Specific Overhead Parameter Retrieval

Certain overhead parameters, such as α_m and O_c in proposed profiling models are specified by CNN-based application and the mobile system specifications, therefore, we conduct on-device computation consumption measurement for parameter retrieval.

1) Mobile-Specific Parameter Test Platform Setup: We employ a Google Nexus 5× smartphone as the test platform as shown in Fig. 5. A Monson power monitor is used to measure the practical overall device energy cost [36]. Meanwhile, an Android kernel analysis tool is modified to breakdown the overall power consumption and monitor the real-time system resource automatically, such as memory usage and CPU usage rate [37].

2) Overhead Model Parameter Retrieval: Since above defined overhead parameters and coefficient parameters are usually constant values regarding certain mobile applications and mobile systems, they can be calculated by conducting multiple CNN model computation consumption measurements for a given application and mobile system.

Therefore, we use Tensorflow [38] to generate 200 CNN models with random structure configurations (i.e., layer number, neuron sizes, neuron amount per layer, etc.), regardless of actual network functionalities. All these 200 CNN models are further converted to flatbuffers-based model files through Tensorflow Lite [39] and deployed into a real-time image classification application on the Nexus 5× smartphone. The real-time image classification application can capture images through on-board camera and further crop them to the size of 32×32 for CNN inference.

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

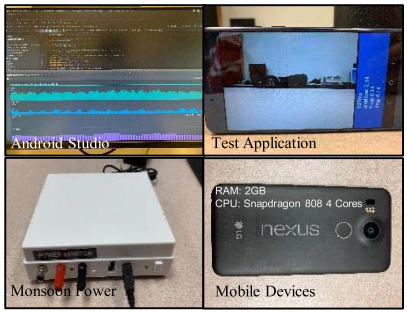


Fig. 5. Mobile-specific parameter test platform setup.

TABLE I
MODEL PARAMETERS RETRIEVED FROM MEASUREMENT

Parameter	Value	Parameter	Value
B_f	32bit	B_a	32bit
α_m	1.53	O_c	16.2MB
O_o	7.1MB	ε_a	18pJ
ε_f	450pJ	T_O^{η}	13.2 ms

For Nexus 5X, $P_{CPU}^{Peak} = 5100mV$ $B_{CPU}^{Peak} = 4.5$ MACs.

With sufficient measurement, the CNN-based model application's energy cost, memory usage, and computation consumption are well recorded by a set of automatic measurement scripts. Based on the measured data, model parameter values are calculated. Table I shows the retrievaled coefficient parameters in the proposed computation consumption profiling models. We can find that α_m is 1.53 for the under-test mobile system, whereas it equals to 1 in previous memory usage estimation models [25]. It shows that the actual memory usage for CNN-based mobile application is 1.53× than the pure CNN's memory usage. Therefore, our profiling models have better computation consumption profiling ability than the previous works on the practical mobile computation scenarios. The retrieved profiling models' estimation accurateness and effectiveness will be further evaluated in Section IX.

V. MOBILE CNN MODEL INFERENCE ACCURACY ANALYSIS AND PROFILING

Besides the CNN models' computation consumption profiling, we further conduct the CNN models' inference accuracy profiling to obtain thorough CNN model consumption/accuracy tradeoff analysis. Specifically, by evaluating each neuron's impact to the overall CNN model inference accuracy, we are able to profile the inference accuracy of arbitrarily configured CNN models.

A. Neuron-Level Inference Accuracy Impact Evaluation

We first evaluate the accuracy changes introduced by a single neuron's configuration. However, different from the computation consumption profiling, each single neuron in a CNN model has different inference accuracy impact, even they are in the identical layer. Such impact differences come from neurons' different matrix size, weight distribution, layer location, feature extraction preference, etc. Considering the neuron differences, we propose a normalized accuracy contribution index (ACI). ACI is defined by the accuracy impact from a neuron's absence to the CNN model's final loss Z

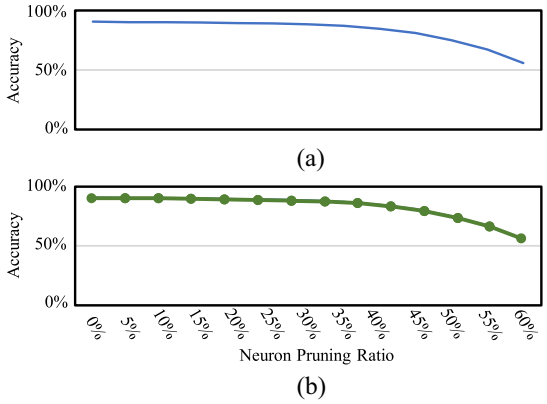


Fig. 6. Gradient-based neuron pruning and inference accuracy approximation. (a) Accuracy change regarding pruning ratio. (b) Original accuracy loss curve can be approximated as multiple lines.

$$Z\left(A_i^j + \delta\right) = Z\left(A_i^j\right) + \frac{\partial Z\left(A_i^j\right)}{\partial A_i^j} * \Delta \tag{15}$$

$$\mathrm{ACI}_i^j = \sum \left| \frac{\partial Z\left(A_i^j\right)}{\partial A_i^j} \right| \tag{16}$$

where A_i^j denotes the activation of the jth neuron in the ith layer; $Z(A_i^j)$ means its corresponding final CNN model loss from the inference loss function; the coefficients matrix $([\partial Z(A_i^j)]/\partial A_i^j)$ represents the neuron's differential impact to the final classification target.

Based on (16), we use the average $\ell 1$-norm of coefficients as ACI, which can indicate the neuron's average impact to the final accuracy. With higher ACI value, the neuron is considered having more accuracy impact.

To directly examine each neuron's actual accuracy impact, we prune neurons in both convolutional layers and fully connected layers in the entire *CaffeNet* model on *CIFAR-10* according to the ACI and the pruning result is shown in Fig. 6(a). The figure demonstrates the relationship between the inference accuracy loss and the neuron pruning ratio, reflecting each neuron's actual accuracy impact.

B. Configured CNN Inference Accuracy Profiling

Although we can evaluate the accuracy impact for every single neuron theoretically, it is hard to precisely measure it practically, since the inference accuracy change generated by pruning a single neuron is negligible. Therefore, we group N_i neurons together according to the calculated ACI ranking and measure the total accuracy loss of N_i neurons. Furthermore, we consider each neuron in the same group has the identical accuracy impact. Based on the above analysis, the CNN accuracy loss result obtained from ACI analysis can be approximated as the combination of m linear accuracy loss lines (m is the number of neuron group). Fig. 6(b) shows the approximation result for *CaffeNet*. We replace the original accuracy loss curve with $m = 13$ green lines, and each line represents the accuracy loss caused by pruning a group of neurons. It should be noticed that, the inference accuracy will drop more significantly when the model size becomes smaller. In other words,

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

the accuracy estimation error will gradually increase with a larger neuron pruned ratio. Therefore, in order to keep high profiling accurateness, our proposed method is supposed to profile the neuron's accuracy impact below $n\%$ pruning ratio. According to our empirical study, the value of n can be set as 60 for most of CNN models, such as *LeNet*, *CaffeNet*, and *VGG-13*.

Supported by the above approximation process, the configured CNN model's inference accuracy can be formulated as

$$\mathrm{AC} = \mathrm{AC}_o - \sum_{1}^{I} a_i N_i \tag{17}$$

where AC_o is the accuracy of the original CNN model. I represents the number of green lines that compose the accuracy loss curve. a_i is the average accuracy impact for a single neuron in group N_i. Through (17), we can build the configured CNN inference accuracy profiling model and its profiling performance will be evaluated in Section IX.

VI. Resource-Aware Mobile CNN Model Reconfiguration Scheme

In former sections, we build profiling models for both CNN model computation consumption and inference accuracy for thorough analysis. Based on the above analysis, we propose a resource-aware CNN model reconfiguration scheme in this section. It leverages neuron pruning methodology to reconfigure CNN models with manageable consumption/accuracy tradeoffs, and therefore adapts to distinct resource constraint types and levels with expected inference performance.

To accomplish such a scheme, we first build a neuron pruning priority to guide the CNN model reconfiguration process by considering each neuron's computation consumption and accuracy impact.

Second, considering distinct resource constraint types and levels in different model computation scenarios, the proposed CNN model reconfiguration scheme leverages various resource constraint priorities to reconfigure CNN model with minimum optimization effort. More specifically, the proposed reconfiguration scheme can set different resource constraints as the primary optimization targets according to specific model computation scenarios.

A. Resource-Aware Neuron Pruning Priority

Since we already evaluated each neuron's significance in terms of computation consumptions and inference accuracy, we leverage the manageable consumption/accuracy tradeoffs to determine the neuron pruning priority during the CNN model reconfiguration.

According to neuron-level CNN model computation consumption profiling, each neuron's computation consumption can be profiled and mapped. Fig. 7(a)–(c) represents each neuron's corresponding energy cost, memory usage, and computation time in *CaffeNet*, respectively. We can find that all eight layers in *CaffeNet* have distinct computation consumption focus. For example, pruning one neuron in the first fully connected layer can lead the largest energy cost and memory usage reduction. On the contrary, pruning a neuron in the first convolutional layer will provide a largest benefit to the computation time.

Since the accuracy impact of each neuron (which is referred as a_k) has been analyzed and profiled in Section V, we can directly define each neuron's pruning priority as

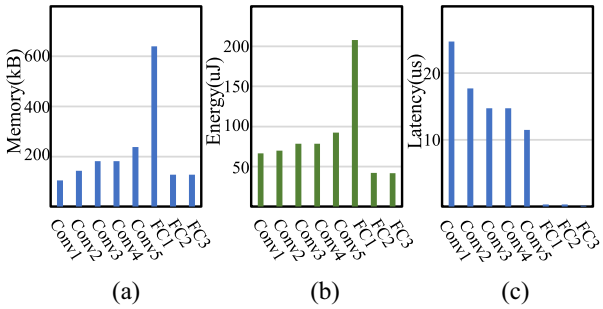


Fig. 7. Neuron resource consumption analysis. (a) Memory. (b) Energy. (c) Latency.

$$\mathrm{PI}_k = \frac{\mathrm{norm}(a_k)}{\beta_e \mathrm{norm}(E_k) + \beta_m \mathrm{norm}(M_k) + \beta_t \mathrm{norm}(T_k)} \tag{18}$$

where $\mathrm{norm}(E_k)$, $\mathrm{norm}(M_k)$, and $\mathrm{norm}(T_k)$ are normalized memory, energy, and time consumption, respectively. β_e, β_m, and β_t are computation consumption weights determined by the distinct practical resource constraints. We will specifically discuss how to determine their values for different resource constraint types and levels in the following part.

With the established neuron pruning priority, the neuron with a lower PI_k value is predicted that has less accuracy impact but higher computation consumption, which can be pruned first during the CNN model reconfiguration process.

B. Model Reconfiguration for Distinct Constraints

As aforementioned, during CNN-based mobile applications execution, the primary computation resource constraint will be changed for different computation scenarios because of their distinct demand preferences. For example, in the gaming scenario, the energy cost will become the primary concern because of the high energy requirement from CPU and GPU computation. On the contrary, the primary constraint in the mobile system changes to the memory usage for VR video scenario because of huge graphic rendering and caching load [40]. Therefore, it is critical to take the distinct computation resource constraint priorities into account for particular CNN model optimization targets.

To achieve effective reconfiguration for different computation scenarios, we set consumption weights β_m, β_e, and β_t in (18) with different values, *with respect to* various computation resource constraint priorities. For example, we assume the energy constraint has highest priority while the memory constraint has lowest priority in the gaming scenario. Therefore, the values of β_e, β_t, and β_m can be set as 3, 2, and 1, separately. Neuron with smallest PI value (the larger energy cost and the lower accuracy impact) will be first pruned during the CNN model reconfiguration process.

Table II is the reconfigured CNN model's computation consumption performance under three different primary computation resource constraint priorities. *VGG-13* is feeding with images from *CIFAR-10* for 1000 times and we measure the average computation consumption. The results show that our model reconfiguration can always achieve corresponding distinct computation consumption reduction for different computation resource constraint priorities. For example, when the pruning ratio equals to 60%, the energy cost under the energy priority reconfiguration will decrease to 290 mJ, while the

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

TABLE II
RESOURCE CONSUMPTIONS FOR VARIOUS CONSTRAINT PRIORITIES IN *VGG-13*

	Memory Priority				Energy Priority				Time Priority			
Pruning Ratio	M(MB)	E(mJ)	L(ms)	A	M(MB)	E(mJ)	L(ms)	A	M(MB)	E(mJ)	L(ms)	A
15%	118	384	79	89.85%	119	382	78	89.77%	119	383	78	89.77%
30%	87	357	70	88%	89	350	68	87.59%	89	358	67	87.47%
45%	62	340	62	81.52%	65	331	58	80.78%	66	336	56	80.66%
60%	49	321	55	55.74%	51	309	47	56.47%	51	319	46	55.60%

Original *VGG-13* computation cost baseline: Memory(M):151*MB*; Energy(E):448*mJ*; Computation Time(L):88*ms*; Accuracy(A):91.00%.

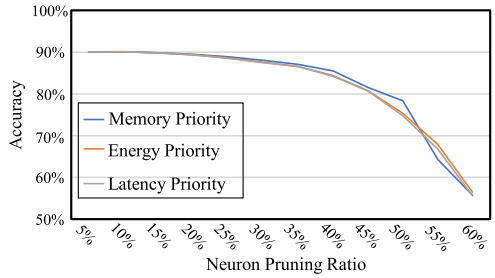


Fig. 8. Pruning accuracy performance for various primary constraint priorities in *VGG-13*.

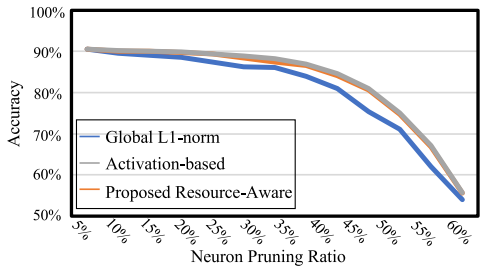


Fig. 9. Accuracy performance comparison between conventional $\ell 1$-norm method and the proposed schemes.

memory priority and the time priority reconfiguration still have 308 mJ and 299 mJ energy cost, respectively. Therefore, the proposed resource-aware CNN model reconfiguration scheme can lead to a specific computation resource reduction under certain resource constraint priority.

To evaluate the accuracy retaining capability of the proposed CNN model reconfiguration scheme, we conduct the preliminary evaluation and Fig. 8 shows the inference accuracy results comparison between three different primary resource constraint priority settings for *VGG-13* in *CIFAR-10*. We can find that accuracy loss rates are similar under three different priority settings. Only the reconfiguration with memory priority can achieve slightly higher prediction accuracy when the neuron pruning ratio less than 52%. Therefore, distinct resource constraint priorities will not affect the inference accuracy of our resource-aware reconfiguration scheme.

C. Model Reconfiguration Inference Accuracy Analysis

To further investigate the inference accuracy performance of the proposed resource-aware CNN model reconfiguration, we compare it in *VGG-13* with the global ℓ_1-norm-based pruning and gradient-based pruning [41], two of the state-of-the-art CNN model configuration methods [42]. The dataset is still *CIFAR-10*. For the global ℓ_1-norm-based pruning, it calculates the ℓ_1-norm value of each neuron in the entire CNN model and iteratively prunes the neuron with the currently lowest ℓ_1-norm value. For the gradient-based pruning, it calculates each neuron's impact to the final logits and iteratively prunes the neuron with the lest impact. Since the global ℓ_1-norm-based method and gradient-based method do not take the computation resource constraint priority into consideration, we set all three resource constraints with equivalent priorities in our scheme to achieve a more equitable comparison.

Fig. 9 illustrates the comparison results regarding inference accuracy and neuron pruning ratio between our proposed reconfiguration scheme and the other two methods. From the figure we can find, our proposed resource-aware reconfiguration scheme always demonstrates a better inference accuracy performance than the global ℓ_1-norm-based method with neuron pruning ratios increasing from 0% to 60%. Meanwhile, compared with gradient-based method, our method can achieve a comparable accuracy performance. When the neuron pruning ratio approximates to 47%, the inference accuracy difference between our propose scheme and the global ℓ_1-norm-based method achieves a largest value. Therefore, comparing to the global ℓ_1-norm-based pruning and gradient-based pruning, our proposed method not only considers the distinct computation resource constraints in different types and levels, but also shows an optimal performance on inference accuracy.

VII. CNN MODEL RECONFIGURATION OPTIMIZATION FOR REAL-TIME MOBILE SYSTEMS

The proposed scheme is still based on iterative single neuron pruning which is time-consuming since it need to continuously calculate CNN model's latest computation consumption and inference accuracy.

To improve the proposed reconfiguration scheme to meet the real-time challenge, we further propose two optimization strategies targeting neuron pruning: 1) neuron group pruning and 2) pruned neuron retrieving. The key methodology of these two strategies is pruning neuron as groups to achieve faster reconfiguration performance while recovery few unnecessary pruned neurons for slightly accuracy compensation. It is notable that we did not take the regular retraining process into account after the reconfiguration like other works. This is because the real-time requirement is our main concern while a regular time-consuming retraining is not practical for dynamic mobile computation scenarios.

A. Neuron Group Pruning for Fast Reconfiguration

Pruning neurons individually ensures a accurate inference accuracy loss during reconfiguration process. However, it hurts the CNN-based application's real-time ability because the CNN model's current computation consumptions are estimated after pruning each neuron, which is extremely time-consuming if neuron number is large. Therefore, to solve the above

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

TABLE III
PERFORMANCE EVALUATION FOR TWO OPTIMIZATION STRATEGIES

	LeNet		CaffeNet		VGG-13	
	Time	Accuracy	Time	Accuracy	Time	Accuracy
Original	820ms	94%	1610ms	65%	2340ms	82.5%
S1	330ms	93.3%	730ms	64.4%	1170ms	82.1%
S1+S2	380ms	94%	800ms	65%	1210ms	82.5%

Neuron Pruning Ratio is set as 45%. S1 represents the first strategy ("Neuron Group Pruning") and S2 represents the second strategy ("Unnecessary Pruned Neuron Recovery").

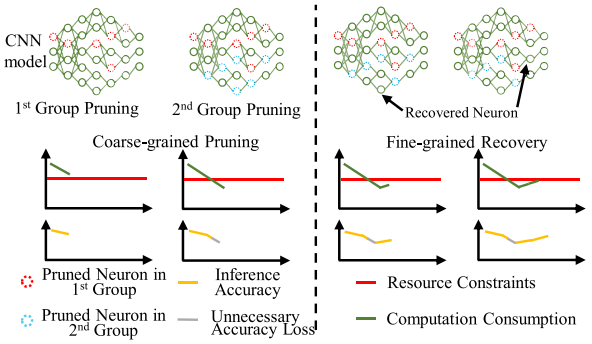


Fig. 10. Neuron group pruning and pruned neuron retrieving.

problem, we introduce the neuron group pruning strategy, which simultaneously prunes a group of neurons instead of a single one during each pruning process.

The methodology of neuron group pruning strategy is shown in the left part of Fig. 10. We group N neurons together according to the pruning priority PI built in Section VI. Therefore, all neurons in the CNN model can be classified to I_m groups. In Fig. 10, the red nodes and the purple nodes represent the first group and second group, respectively. During the reconfiguration, we prune all neurons in the same group simultaneously. Based on the resource consumption profiling models and inference accuracy profiling models, we can fast estimate the pruned CNN model's resource consumptions and inference accuracy after each pruning iteration. Therefore, without accounting other overheads, the computation workload introduced by estimating computation consumption will approximately decrease N times, leading a significant reconfiguration time cost reduction. The number of neurons in each group is different regarding various CNN model types. Usually, to ensure the real-time reconfiguration, CNN models with more neuron numbers have larger N values.

B. Pruned Neuron Retrieving for Accuracy Compensation

Sine the neurons are pruned as groups, more neurons are pruned than actually required in the last pruned group, causing an unnecessary inference accuracy loss (shown as the gray line in Fig. 10). To recovery such accuracy loss, we further propose pruned neuron retrieving strategy.

The right part of Fig. 10 illustrates the optimization process of the proposed strategy. After pruning the second group, we identify that the estimated CNN model computation consumptions are below the system resource constraints. It is clear to see, we pruned extra neurons than actual required. Therefore, we recovery the unnecessary pruned neurons in the second pruning group one by one according to their inverted pruning order PI. After this recovery process, the CNN model's inference accuracy can be slightly compensated.

We conduct the preliminary experiment to evaluate the optimal performance of our optimization strategies and the result is shown in Table III. From the table we can find that: under 45% neuron pruning ratio setting with affordable accuracy loss (approximate 5% drop), neuron group pruning can accelerate the CNN reconfiguration around 2.48× for *LeNet*, 2.21× for *CaffeNet*, and 2× for *VGG-16* with at most 0.7% extra accuracy loss. When combining the two strategies together, the extra inference accuracy loss will be compensated and the two strategies can achieve an average 2.2× speedup.

VIII. *DiReCtX* FRAMEWORK INTEGRATION

In this section, we propose a comprehensive reconfiguration paradigm to integrate the proposed resource-aware CNN model reconfiguration scheme and two optimization strategies into *DiReCtX*. Before introducing the integration design, we need to identify computation resource constraints in the real-time mobile system to enable *DiReCtX*'s practical deployment.

A. Mobile Computation Resource Constraints Analysis

We first define several mobile system variables as references as follows.

1) $E_A(t)$: Available mobile battery energy at time t.
2) E_B^n: Average energy cost for the nth background application.
3) $M_A(t)$: Available memory at time t.
4) T_{Max}: Maximum tolerant latency for the CNN-based application.

Based on the definitions of these system variables, we model mobile system computation resource constrains as

$$E_A(t) - \sum E_B^n T > E \tag{19}$$

$$M_A(t) > M \tag{20}$$

$$T_{\text{Max}} > T \tag{21}$$

where $\sum E_B^n T$ indicates the total amount of energy consumed by other background applications in the CNN-based application executing time. $E_A(t)$, E_B^n, and $M_A(t)$ can be obtained by calling the inner components in the mobile system. For example, we can leverage *ComponentCallbacks2* in the Android system to fetch memory footprints.

B. DiReCtX Reconfiguration Paradigm

For a mobile system with various and dynamic computation resource constraints analyzed above, we propose *DiReCtX* to integrate the resource-aware CNN model reconfiguration scheme and two optimization strategies. Algorithm 1 illustrates the *DiReCtX* reconfiguration process for a given CNN-based application in the mobile system.

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 *DiReCtX* Resource-Aware CNN Reconfiguration Paradigm

Input: 1.Original model Mod_0. 2.Specific scenario time constraint T_{Max}.
Output: $model_r$ with highest accuracy and can meet all system constraints.

Determine resource constraint priorities, *e.g.*, $E > M > T$.
$PI = \frac{norm(CI)}{\beta_e norm(E)+\beta_m norm(M)+\beta_t norm(T)}$
Group neurons as $N_1, \ldots, N_i, \ldots, N_m$.
for all N_i in $N_1, N_2, \ldots, N_m$ **do**
    Neuron group pruning
    if $(E_A(t) - \sum E_B^n T) > E$, $M_A(t) > M$, $T_{Max} > T$ **then**
        Stop neuron group pruning.
        for all neuron in N_i **do**
            Pruned neuron retrieving.
        end for
    end if
end for
return : Mod_r or stop

DiReCtX will first identify the current computation scenario and corresponding computation resource constraints including T_{Max}, $M_A(t)$, and $(E_A(t) - \sum E_B^n T)$ through system *APIs*. According to the specific application scenario, the primary computation resource constraint will be also determined. Next, *DiReCtX* calculates the pruning priority PI for each neuron and groups all neurons into m groups, namely, $N_1, N_2, \ldots,$ and N_m. The proposed resource-aware reconfiguration scheme with neuron group pruning method is applied to optimize the original model Model_o in the CNN-based application. Groups N_1 to N_m are pruned consecutively until the reconfigured CNN model's resource consumptions meet all three constraints. Once the resource-aware reconfiguration with neuron group pruning finished, pruned neuron retrieving strategy is further executed to compensate the prediction accuracy by recovering those unnecessary pruned neurons.

IX. Experiment and Evaluation

In this section, we comprehensively evaluate the effectiveness of the proposed *DiReCtX* framework from four perspectives.

1) The accurateness of computation consumption profiling models for reconfigured CNN model.
2) The preciseness of inference accuracy profiling models.
3) The performance of the proposed CNN reconfiguration scheme in terms of energy cost, memory usage, and computation time reduction.
4) The adaptability to different mobile application scenarios.

The performance is evaluated with the state-of-the-art smartphone, and the experiment setup is shown as following.

A. Experimental Setup

1) Testing Platform: As aforementioned in Section IV, a set of Android smartphones—Google Nexus 5×, are used as the test platform. On these smartphones, we implement *DiReCtX* with Tensorflow Lite [39] and integrate it into an image classification application with Android Studio [43]. The integrated application will be tested with other mobile applications to mimic various practical computation scenarios.

2) CNN Models and Dataset: In the experiment, three CNN models are used as our testing targets, namely, *LeNet*, *CaffeNet*, and *VGG-13*. For *LeNet*, we train it on handwriting dataset *MNIST* and the baseline accuracy is 98.89%. For *CaffeNet* and *VGG-13*, we adopt ten class image classification dataset *CIFAR-10* and the accuracy baselines of two models are 90.57% and 91.00%, separately. We also apply *VGG-13* with larger dataset *CIFAR-100* and *ImageNet-10* to evaluate *DiReCtX*'s reconfiguration ability in Section IX-D.

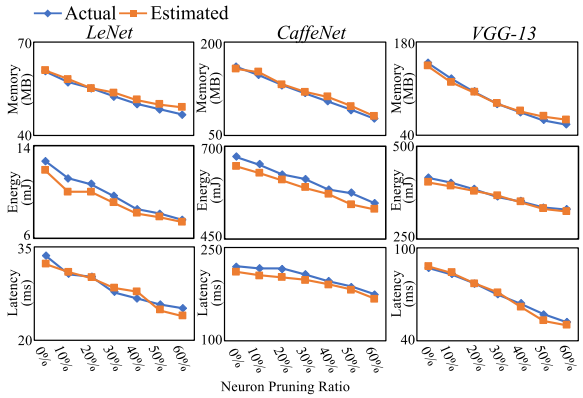


Fig. 11. Computation consumption models evaluation.

B. CNN Computation Consumption Models Evaluation

We first evaluate the accurateness of proposed mobile CNN computation consumption models, by comparing estimated results with realistic measurements. The realistic measurements are based on the 1000 times tests for *LeNet*, *CaffeNet*, and *VGG-13* and we calculate the average values.

Fig. 11 illustrates comparison results with respect to three CNN models and three kinds of computation consumptions (energy, memory, and time). The figures in each column represent different models while the figures in each row represent different kinds of computation consumptions. According to the accuracy analysis in Section V, the inference accuracy will drop dramatically after neuron pruning ratio exceeds 60%. Therefore, each CNN model is pruned with the proposed resource-aware reconfiguration scheme from 0% to 60% (0%, 10%, 20%, 30%, 40%, 50%, 60%, separately). According to Fig. 11, we can find that the following.

1) For memory estimation model, it has accuracies of 95.4% ∼ 99.9% for *LeNet*, 93.8% ∼ 99.3% for *CaffeNet*, and 87.7% ∼ 99.1% for *VGG-13*, averaging 96.0%.
2) The energy estimation model can achieve 87.1%∼97.1% estimation accuracy for *LeNet*, 94.3% ∼ 97.4% estimation accuracy for *CaffeNet* and 96.1% ∼ 98.7% estimation accuracy for *VGG-13*, with an average rate of 93.4%.
3) For latency estimation model, it has estimation accuracies of 94.1% ∼ 98.9% for *LeNet*, 93% ∼ 97% for *CaffeNet*, and 92.5% ∼ 99.7% for *VGG-13*, averaging 94.5%.

Overall, three CNN model computation consumption estimation models have an average estimation accuracy of 94.6%, demonstrating the optimal performance of our computation consumption estimation models.

C. Reconfigured CNN Accuracy Profiling Evaluation

We further evaluate the accurateness of the proposed reconfigured CNN accuracy profiling. The estimated CNN profiling results are compared with realistic measurements for CNN

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
RECONFIGURED CNN ACCURACY PROFILING EVALUATION

LeNet	0%	10%	20%	30%	40%	50%	60%
Actual	98.0%	97.9%	97.1%	93.9%	90.2%	80.8%	63.3%
Profiled	98%	97.0%	95.5%	91.1%	87.3%	79.5%	63.0%
CaffeNet	0%	10%	20%	30%	40%	50%	60%
Actual	90.6%	90.3%	89.3%	88.0%	84.0%	74.2%	55.6%
Profiled	90.6%	90.2%	88.1%	85.2%	80.8%	70.0%	56.2%
VGG-13	0%	10%	20%	30%	40%	50%	60%
Actual	91.0%	91.0%	89.1%	87.8%	85.0%	76.2%	55.6%
Profiled	91.0%	89.1%	86.2%	82.4%	78.7%	75.1%	56.0%

models which are reconfigured by our proposed resource-aware reconfiguration method. Table IV illustrates the comparison results with respect to *LeNet*, *CaffeNet*, and *VGG-13*. According to our previous analysis, the reconfigured model's accuracies under different computation resource constraint priorities only have slight difference. Moreover, since we aim to evaluate the accurateness of our proposed CNN accuracy profiling model, it is unnecessary to take resource-awareness into consideration. Therefore, for fair and simplicity, we set three computation resource constraints with equal priorities in this experiment. In Section IX-E, we will evaluate *DirectX*'s reconfiguration performance with different resource constraint priorities. Moreover, we set n as 60 for each CNN model. According to Table IV, we can find that the following.

1) For *LeNet*, the total number of neurons in the first 60% is 180. We divide them into 10 groups with 18 neurons in each group. Thus, the original accuracy loss curve can be converted to the combination of ten linear lines. The profiling accurateness are from 96.7% $\sim$ 100%, averaging 98.4%.
2) For *CaffeNet*, we divided the first 60% neurons in 20 groups with around 50 neurons in each group. The profiling results achieve accurateness from 94.6% $\sim$ 100%, with an average rate of 97.8%.
3) For *VGG-13*, the total number of neurons in the first 60% is 3000 and those neurons can be divided into 30 groups. Therefore, the original accuracy loss curve can be approximated to the combination of 30 linear lines. The profiling results can achieve accurateness from 92.3% $\sim$ 100%, averaging 95.2%.

Overall, profiling results on three CNN models have an average accurateness of 97.1%, demonstrating the good estimation performance of our reconfigured CNN profiling analysis.

D. DiReCtX Reconfiguration Effectiveness

As aforementioned, our method can achieve comparable accuracy performance with previous gradient-based method. At the same time, it is designed to optimize entire network *with respect to* computation resource consumption. Therefore, in this section, we evaluate *DiReCtX*'s model reconfiguration effectiveness by comparing with two state-of-the-art methods, *NetAdapt* and global ℓ_1-norm [11], [42].

Fig. 12 first shows the comparison results of three network (*LeNet*, *CaffeNet*, and *VGG-13*) in terms of energy, memory, and time consumption. A_1, A_2, and A_3 represent 1%, 3%, and 5% accuracy loss separately while $A_{\max}$ indicates the maximum accuracy loss for the given CNN-based application

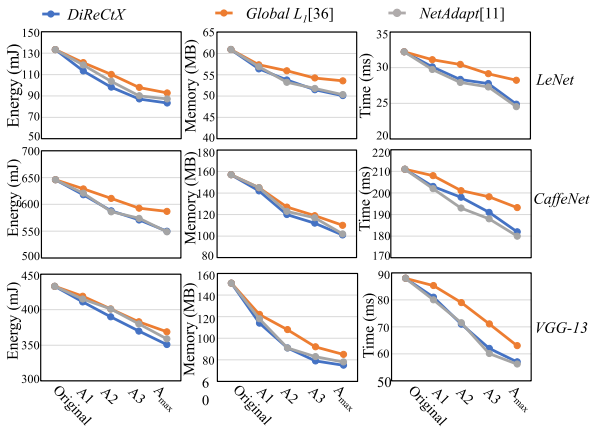


Fig. 12. Reconfiguration performance comparison for *LeNet*, *CaffeNet*, and *VGG-13*.

during its practical using. We assume $A_{\max}$ equals to 10% in this experiment.

According to the figure, both *DiReCtX* and *NetAdapt* can always achieve better computation resource reduction performance than global ℓ_1-norm method. Since *NetAdapt* only focuses on computation time optimization, it can achieve at most 3% extra computation time reduction compared with *DiReCtX*. However, compared to *NetAdapt*, *DiReCtX* can reconfigure CNN models targets multiple resource consumption optimization, which can achieve better reduction performance in all of three resource types. In that case, it can achieve better optimization effectiveness on energy cost and memory usage than *NetAdapt*.

As mentioned, we extend the reconfiguration effectiveness evaluation to larger datasets, including *CIFAR-100* and *ImageNet-10*. The baseline accuracies of *VGG-13* on these two datasets are 71.60%, and 93.70%, separately.

According to Table V, we can find that: For *VGG-13* on *CIFAR-100*-based mobile application, the original energy, memory and time costs are 448 mJ, 151 MB, and 88 ms, respectively. In terms of the energy cost, a significant energy reduction of 13.85% is introduced (i.e., from 448 mJ to 373 mJ) with an accepted maximum accuracy loss $A_{\max}$ of 10%. In terms of the memory usage, reconfiguration with 1%, 3%, and 5% accuracy loss can let the application's memory usage decrease to 128 MB, 115 MB, and 111 MB, respectively. Furthermore, *DiReCtX* can achieve at most 37.08% memory reduction with an maximum accuracy loss of 10%. In terms of the time cost, reconfiguring the model with 1%, 3%, and 5% accuracy loss can reduce the application's computation cost to 84 ms, 80 ms, and 78 ms, respectively. When getting maximum accuracy loss of 10%, *DiReCtX* can achieve 21.59% computation time reduction without any retraining. For *VGG-13* on *ImageNet-10*-based mobile application, the original energy, memory and time costs are 7150 mJ, 1120 MB, and 3600 ms, respectively. In the condition of the 10% maximum accuracy loss, *DiReCtX* can significantly reconfigure *VGG-13* with 32.39% energy cost, 31.69% memory usage, and 44.44% time cost reduction.

We also evaluate *DiReCtX* reconfiguration effectiveness on *MobileNetV1* and *ResNet-18*, two widely used CNN models for mobile computing scenarios. For *MobileNetV1*, the accuracy baseline on *CIFAR-10* is 86%. The original energy, memory and time costs are 265 mJ, 35 MB, and 18 ms, respectively. With 10% maximum accepted accuracy loss, *DiReCtX*

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

TABLE V
DiReCtX'S RECONFIGURATION PERFORMANCE FOR *VGG-13* ON *CIFAR-100* AND *ImageNet-10*

CIFAR-100	Energy(*mJ*)	Memory(*MB*)	Computation Time(*ms*)
Original	448	151	88
A_1	431	128	84
A_2	411	115	80
A_3	386	111	78
A_{max}	372	95	69

ImageNet-10	Energy(*mJ*)	Memory(*MB*)	Computation Time (*ms*)
Original	7150	1120	3600
A_1	6740	1032	3100
A_2	6140	941	2700
A_3	5630	850	2500
A_{max}	4850	765	2100

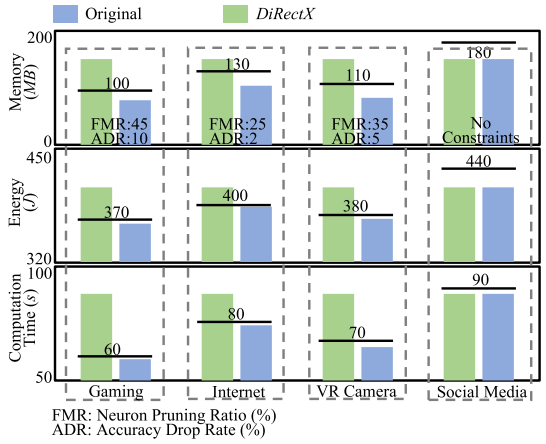


Fig. 13. Mobile computation optimization under different application scenarios with *VGG-13*.

can achieve 23% energy reduction, 35% memory decrease and 28% time saving. For *ResNet-50*, the accuracy baseline on *CIFAR-10* is 93%. Due to the shortcut connections, the output channel numbers of each block need to be fixed. Therefore, we conduct the neuron pruning by following the same setting mentioned in [9]. Finally, our method achieves at most 20% energy reduction, 29% memory decrease, and 22% time saving the maximum accuracy decrease.

The above results prove that *DiReCtX* can effectively reconfigure the CNN model in the given CNN-based application with an acceptable accuracy loss. Compared with other state-of-the-art methods, *DiReCtX* can achieve better overall computation consumption reduction performance.

E. Mobile Computation Scenario Adaptability

The adaptability of the proposed *DiReCtX* is also evaluated in four mobile computation scenarios featured with popular mobile applications. In Fig. 13, we investigate the dynamic resource-aware CNN reconfiguration with *DiReCtX* for adapting different computation resource budgets. The experiment results are obtained based on 1000 times *VGG-13* task executions with a battery capacity of 5400 J.

From Fig. 13, we can observe that different mobile computation scenarios have distinct energy, memory, and time budgets. Some scenarios can not afford the computation resource for the original *VGG-13* execution, such as *Gaming*, *Internet*, and *VR* applications. Taking the energy cost of *Gaming* as an example, *VGG-13* execution requires 448 J for 1000 times execution, while the mobile platform needs approximate 4952 J to support the entire system. In such case, the energy budget for *VGG-13* is only 370 J. Meanwhile, the memory and computation time budgets are 100 MB and 60 s, respectively. Therefore, the primary computation resource constraint is energy. With the proposed *DiReCtX* framework, *VGG-13* is automatically reconfigured to a scheme with *FMR* (Neuron Pruning Ratio) of 45% and 9.8% accuracy loss based on the precise computation consumptions estimation. Hence, *VGG-13* related applications can be well balanced with an acceptable accuracy performance and manageable resources. For other two computation scenarios, such as Internet and VR Camera, we set memory and computation time as primary constraints and *DiReCtX* can also reconfigure *VGG-13* to adapt corresponding computation resource constrains.

Above results demonstrate that *DiReCtX* can effectively adapt the CNN configuration to dynamic mobile scenarios.

X. CONCLUSION

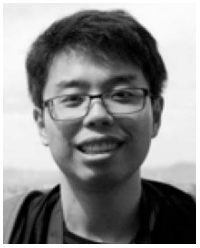
In this article, we proposed *DiReCtX*, a resource-aware CNN reconfiguration framework, which adapts various CNN models to dynamic mobile computation scenarios. *DiReCtX* is composed of a set of CNN profiling models for computation consumption and inference accuracy estimation, a resource-aware CNN reconfiguration scheme, and two post-reconfiguration optimization strategies for real-time mobile implementation. With these technical efforts, *DiReCtX* can dynamically reconfigure a CNN model with manageable resource consumption and accuracy tradeoffs. It can be well implemented on state-of-the-art smartphones for CNN computation optimization.

REFERENCES

[1] F. Wang *et al.*, "Residual attention network for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, 2017, pp. 3156–3164.
[2] K. Alex, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.
[3] G. Ross, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, Santiago, Chile, 2015, pp. 1440–1448.
[4] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 577–585.
[5] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The microsoft 2017 conversational speech recognition system," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Calgary, AB, Canada, 2018, pp. 5934–5938.
[6] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2014.
[7] Z. Xu, Z. Qin, F. Yu, C. Liu, and X. Chen, "DiReCt: Resource-aware dynamic model reconfiguration for convolutional neural network in mobile systems," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2018, pp. 1–6.
[8] W. Meng, Z. Gu, M. Zhang, and Z. Wu, "Two-bit networks for deep learning on resource-constrained embedded devices," 2017. [Online]. Available: arXiv:1602.07360.
[9] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2016. [Online]. Available: arXiv:1608.08710.
[10] Z. Xu, F. Yu, C. Liu, and X. Chen, "ReForm: Static and dynamic resource-aware DNN reconfiguration framework for mobile device," in *Proc. 56th Annu. Design Autom. Conf.*, Las Vegas, NV, USA, 2019, p. 183.
[11] T. Yang *et al.*, "NetAdapt: Platform-aware neural network adaptation for mobile applications," *Energy*, vol. 41, p. 46, Oct. 2018.
[12] Y. Wang, T. Nguyen, Y. Zhao, Z. Wang, Y. Lin, and R. Baraniuk, "EnergyNet: Energy-efficient dynamic inference," in *Proc. 32nd Conf. Neural Inf. Process. Syst.*, 2018.

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.

[13] Y. LeCun *et al.*. (2015). *LeNet-5, Convolutional Neural Networks*. [Online]. Available: http://yann. lecun. com/exdb/lenet
[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: arXiv:1409.1556.
[15] *Nexus5 Performance*, Geekbench, Toronto, ON, Canada, 2017. [Online]. Available: https://browser.geekbench.com/
[16] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Proc. Int. Conf. Artif. Neural Netw.*, 2014, pp. 281–290.
[17] N. Ma, X. Zhang, H. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," 2018. [Online]. Available: arXiv:1807.11164.
[18] F. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and less 0.5mb model size," 2016. [Online]. Available: arXiv:1602.07360.
[19] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2017. [Online]. Available: arXiv:1610.02357.
[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015. [Online]. Available: arXiv:1510.00149.
[21] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014. [Online]. Available: arXiv:1405.3866.
[22] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1," 2016. [Online]. Available: arXiv:1602.02830.
[23] T. Yang, Y. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," 2016. [Online]. Available: arXiv:1611.05128.
[24] A. Gordon *et al.*, "MorphNet: Fast & simple resource-constrained structure learning of deep networks," 2017. [Online]. Available: arXiv:1711.06798.
[25] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proc. 16th Annu. Int. Conf. Mobile Syst. Appl. Service*, 2018, pp. 389–400.
[26] J. Luo and J. Wu, "An entropy-based pruning method for CNN compression," 2017. [Online]. Available: arXiv:1706.05791.
[27] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 153–168.
[28] *Neural Networks Usage at Mobile Development*, Dashdevs, Wilmington, DE, USA, 2019. [Online]. Available: https://codeburst.io/neural-networks-usage-at-mobile-development-e9de81d25f18
[29] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proc. 9th Int. Conf. Mobile Syst. Appl. Services*, 2011, pp. 335–348.
[30] M. Kim, J. Kong, and S. W. Chung, "Enhancing online power estimation accuracy for smartphones," *IEEE Trans. Consum. Electron.*, vol. 58, no. 2, pp. 333–339, May 2012.
[31] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 11, pp. 2072–2085, Nov. 2019.
[32] A. Ignatov *et al.*, "AI benchmark: All about deep learning on smartphones in 2019," 2019. [Online]. Available: arXiv:1910.06663.
[33] Y. H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. IEEE/ACM 43rd Annu. Int. Symp. Comput. Architect.*, Seoul, South Korea, 2016, pp. 367–379.
[34] S. Williams, A. Waterman, and D. A. Patterson, "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Rep. UCB/EECS-2008-134, 2009.
[35] S. Swanson and M. B. Taylor, "Greendroid: Exploring the next evolution in smartphone application processors," *IEEE Commun. Mag.*, vol. 49, no. 4, pp. 112–119, Apr. 2011.
[36] *Monsoon Power Monitor*, M. S. Inc., Attleboro, MA, USA, 2010.
[37] *WeTest Smartphone Analysis Tool*, Air T, Inc., Maiden, NC, USA, 2017. [Online]. Available: https://github.com/Tencent/WeTest-Assistant/
[38] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement.*, 2016, pp. 265–283.
[39] *Tensorflow Lite*, G. Inc., Mountain View, CA, USA, 2018. [Online]. Available: https://www.tensorflow.org/lite
[40] A. Carroll and G. Haiser, "An analysis of power consumption in a smartphone," in *Proc. USENIX Annu. Techn. Conf.*, 2010, p. 21.
[41] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016. [Online]. Available: arXiv:1611.06440.
[42] A. Salama, O. Ostapenko, M. Nabi, and T. Klein, "Pruning at a glance: A structured class-blind pruning technique for model compression," in *Proc. Conf. Adv. Neural Inf. Process. Syst.*, 2018.
[43] *Android Studio*, Android Studio, Mountain View, CA, USA, 2016.



Zirui Xu received the B.S. and M.S. degrees from Beijing Jiaotong University, Beijing, China, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA, under the supervision of Prof. X. Chen.

His current research interests include high performance mobile computing system, neural network model optimization, and mobile intelligent application robustness and security.

Fuxun Yu received the B.S. degree from the Harbin Institute of Technology, China, in 2017. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA, under the supervision of Prof. X. Chen.

His current research directions include deep learning security, adversarial attacks and defenses on neural network, high-performance deep neural network on mobile devices, and interpretability and explainability of deep learning.

Zhuwei Qin received the B.S. degree from the Tianjin University of Science and Technology, Tianjin, China, in 2014, and the M.S. degree from Oregon State University, Corvallis, OR, USA, in 2017. He is currently pursuing the Ph.D. degree with ECE Department, George Mason University, Fairfax, VA, USA, under the supervision of Prof. X. Chen.

His current research directions include deep neural network compression, interpretable deep neural network for mobile application.

Chenchen Liu received the M.S. degree from Peking University, Beijing, China, in 2013, and the Ph.D. degree from the ECE Department, University of Pittsburgh, Pittsburgh, PA, USA, in 2017.

She is currently an Assistant Professor with the Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore, MI, USA. Her current research interests include brain-inspired computing system and security, integrated circuits design, and emerging nonvolatile memory technologies.

Xiang Chen received the M.S./Ph.D. degrees from the ECE Department, University of Pittsburgh, Pittsburgh, PA, USA, in 2012 and 2016, respectively.

In 2016, he joined the Department of the Computer Engineering, George Mason University, Fairfax, VA, USA, as an Assistant Professor. His research interests are in the low-power mobile system, high-performance mobile computing, machine learning, and secure computing system.

Authorized licensed use limited to: George Mason University. Downloaded on January 02,2022 at 02:34:00 UTC from IEEE Xplore. Restrictions apply.