ReForm: Static and Dynamic Resource-Aware DNN Reconfiguration Framework for Mobile Device

Zirui Xu†, Fuxun Yu†, Chenchen Liu‡, Xiang Chen†
†George Mason University, Fairfax, Virginia, {zxu21, fyu2, xchen26}@gmu.edu
‡Clarkson University, Potsdam, New York, chliu@clarkson.edu

ABSTRACT

Although the Deep Neural Network (DNN) technique has been widely applied in various applications, the DNN-based applications are still too computationally intensive for the resource-constrained mobile devices. Many works have been proposed to optimize the DNN computation performance, but most of them are limited in an algorithmic perspective, ignoring certain computing issues in practical deployment. To achieve the comprehensive DNN performance enhancement in practice, the expected DNN optimization works should closely cooperate with specific hardware and system constraints (*i.e.* computation capacity, energy cost, memory occupancy, and inference latency). Therefore, in this work, we propose *ReForm* – a resource-aware DNN optimization framework. Through thorough mobile DNN computing analysis and innovative model reconfiguration schemes (*i.e.* ADMM based static model fine-tuning, dynamically selective computing), *ReForm* can efficiently and effectively reconfigure a pre-trained DNN model for practical mobile deployment with regards to various static and dynamic computation resource constraints. Experiments show that *ReForm* has ∼3.5× faster optimization speed than state-of-the-art resource-aware optimization method. Also, *ReForm* can effective reconfigure a DNN model to different mobile devices with distinct resource constraints. Moreover, *ReForm* achieves satisfying computation cost reduction with ignorable accuracy drop in both static and dynamic computing scenarios (at most 18% workload, 16.23% latency, 48.63% memory, and 21.5% energy enhancement).

ACM Reference Format:
Zirui Xu†, Fuxun Yu†, Chenchen Liu‡, Xiang Chen†. 2019. ReForm: Static and Dynamic Resource-Aware DNN Reconfiguration Framework for Mobile Device . In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3324696

1 INTRODUCTION

In the past few years, the Deep Neural Network (DNN) technique has been widely applied in various cognitive applications, such as image classification [14], voice recognition [22], *etc.* Although effective and popular, the DNN-based applications are still too computationally intensive for resource-constrained platforms, especially the mobile devices. Therefore, many works have been proposed to optimize the DNN computation performance leveraging novel model designs [20], parameter compression [11], *etc.*

However, most optimization works are driven merely by a theoretical algorithm perspective, ignoring specific hardware and system constraints associated with practical computing scenarios. Therefore, many "algorithm-oriented" works fail to achieve comprehensive performance enhancement. For example, [27] shows that, although the advanced DNN model of MobileNetV1 [10] achieves 19% computation workload reduction on the Pixel-1 Android smartphone, its practical inference latency gains 29%. Such contradictory performance changes are caused by inconsistent optimization targets, where the aggressive structural compression fragments the computation process and introduces considerable latency.

To achieve the comprehensive performance enhancement in practical deployment, many recent DNN design and optimization works have taken into account of various hardware and system resource constraints, such as computation capacity, energy cost, memory occupancy, inference latency, *etc.* [5, 25, 27]. For example, Wang *et al.* [24] formulated energy loss in addition to the accuracy loss, which guides the DNN training for certain energy budgets. These works are referred as "resource-aware" DNN optimization.

Generally, these "resource-aware" DNN optimization works have several critical challenges: (1) The adaptability for inconsistent resource constraints in various computing scenarios. Since different computing scenarios have distinct specifications, the optimization works need to be capable to identify and adapt to different constraint requirements. (2) The comprehensiveness for multiple resource constraints. To achieve comprehensive optimization, the optimization works are expected to handle multiple constraints simultaneously. (3) The reconfigurability for dynamic resource constraint change. During the practical deployment, especially on the multi-task system like mobile devices, the constraints may be continuously changed by various applications. Therefore the optimization work is expected be dynamically configurable for real-time requirements. However, most of emerging "resource-aware" DNN optimization work can't fulfill all the challenges simultaneously.

To tackle these challenges, in this paper, we propose *ReForm* – a resource-aware DNN optimization framework, which can comprehensively enhance DNN models' computation performance on mobile devices. Through innovative model reconfiguration schemes, *ReForm* can efficiently and effectively optimize mobile DNN models with regards to various static and dynamic resource constraints.

Specifically, we have the following contributions in this work:

- We identify and formulate the major computation resource constraints for DNN computation on mobile devices.
- We propose a static DNN reconfiguration scheme to fine-tune a pre-trained DNN model to a specific mobile device's

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3324696

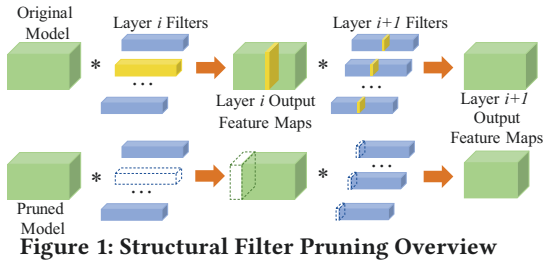

Figure 1: Structural Filter Pruning Overview

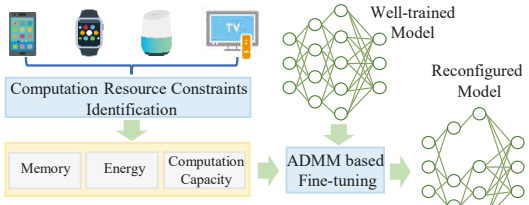

Figure 2: Scheme Overview of Static DNN Reconfiguration

default computation resource. We formulate the fine-tuning process as an ADMM [2] optimization problem, which can retain optimal accuracy performance under multiple configurable constraints with satisfying efficiency.

- We propose a dynamic DNN reconfiguration scheme to adapt a DNN model into real-time mobile computing scenarios. By evaluating individual model component's resource consumption and accuracy impact, the proposed scheme selectively compute the model components to balance the accuracy performance and real-time constraints without model retraining.
- We implemented *ReForm* on multiple types of mobile devices, and quantitatively evaluated its performance with practical mobile dynamic computing scenarios.

The experiment results show that *ReForm* has optimal efficiency with ~3.5× faster speed than state-of-the-art resource-aware optimization method. Also, *ReForm* can effectively reconfigure a DNN model to different mobile devices with distinct resource constraints. Moreover *ReForm* achieves satisfying cost reduction with ignorable accuracy drop in both static and dynamic computing scenarios.

2 PRELIMINARY

2.1 DNN Computation Optimization

Considering the intensive DNN computation cost, many works have been proposed for DNN computation optimization. Specifically, the optimization works can be categorized into several major approaches: (1) Novel Model Design, such as ShuffleNet [20], SqueezeNet [12], Xception [3]; (2) Parameter Compression, such as filter pruning [11, 18, 21] and weight sparsity [7, 8]; (3) Approximate Calculation, such as low rank [13] and quantization [4].

These "algorithm-oriented" works mainly take the model parameters and computing mechanism as the optimization targets. For example, the filter pruning is considered as one of the most effective optimization methods, which eliminates the insignificant filters to reduce the major DNN computation workload – filter convolution process. Fig. 1 illustrates the filter pruning process in the i^{th} convolutional layer. By pruning the insignificant filters, the correspondingly feature maps are also eliminated in the i^{th} layer's output. As the i^{th} layer's output feature maps are the inputs of the $(i+1)^{th}$ layer, all the filters in the $(i+1)^{th}$ layer therefore have less computation workload.

In this work, we adopt the filter pruning as the major tool for reconfiguration, expecting to take the advantages of both "algorithm-oriented" and "resource-aware" optimization approaches.

2.2 Resource-Aware DNN Optimization

As aforementioned, the "algorithm-oriented" works can't achieve comprehensive optimization result. While, many "resource-aware" optimization works have emerged by taking into account of practical computation resource constraints: (1) For the computation capacity constraint, Gordan *et al.* [6] evaluated the computation workload of each DNN component and dynamically reconfigured the model for different computing scenarios. (2) For the energy constraint, Yang *et al.* [26] identified the DNN layer-wise energy consumption and implemented corresponding filter pruning for energy-constrained system. (3) For the memory constraint, Liu *et al.* [19] leveraged the reinforcement learning to optimize the DNN model to meet certain memory budget.

However, most existing"resource-aware" DNN optimization works only address single constraint in a specific computing scenario, which can't fulfill the comprehensive optimization expectation.

2.3 Mobile DNN Computation Constraints

Although the mobile devices are considered as the most promising platform for DNN computing, the "resource aware" optimization works for mobile DNN computing have even more critical requirements than the general challenges as aforementioned:

(1) Compared to general computing platforms, the computation constraints of mobile devices have significant diversity with regards to default device configurations, computing scenarios, and real-time requirements. In other words, the computation resource constraints on mobile devices have greater complexity. Therefore, the mobile optimization works should be designed with more flexibility and effectiveness. (2) The mobile computing scenarios have distinct dynamics. When deployed on mobile devices, DNN based applications will be affected by much more real-time issues. For example, with starting a new application or closing an existing one, the available system computation resource will be dynamically changed. Conventionally, the optimization methods of DNN models replies on heavy model component analysis and hours of model retraining [6, 9]. Considering the real-time requirement, the mobile optimization works are expected with certain efficiency.

Motivated by these challenges, we propose our "resource-aware" optimization framework for mobile DNN computing.

3 STATIC DNN RECONFIGURATION WITH ADMM BASED FINE-TUNING

In this work, we propose *ReForm* – a resource-aware DNN optimization framework, which can comprehensively enhance DNN models' computation performance on mobile devices. In *ReForm*, two DNN optimization schemes are proposed to optimize the DNN computation performance with regards to static and dynamic computing scenarios, respectively.

In the static scheme, the reconfiguration is focused on fine-tuning a pre-trained DNN model to adapt to a specific mobile device's default computation resource configuration. Fig. 2 shows the overview of the proposed scheme: The scheme first identifies and formulates specific device's computation resource constraints in terms of memory, energy, and computation capacity. Then, a pre-trained DNN

model is reconfigured via an ADMM based fine-tuning process to meet all the constraints and enhance the computation performance.

3.1 Computation Resource Constraints Identification and Formulation

In mobile device, there are various computation resource constraints, which can be formulated into mathematical expression and be easily inserted into the optimization objective function. In our scheme, we focus on three typical constraints, including computation capacity, memory occupancy and energy consumption.

3.1.1 Computation Capacity Constraint. Usually, the computation capacity C required by a DNN computation is represented as the total number of MACs (Multiply-Accumulate Operations), which can be modeled as:

$$C = \sum_{i=1}^{L} \sum_{j=1}^{n_i} r_i^j s_i^j n_{i-1} h_i^j w_i^j \tag{1}$$

where r_i^j and s_i^j represent j^{th} filter's kernel size in i^{th} layer, h_i^j and w_i^j denote the corresponding height and width of output feature map, L is the total layer number and n_i is the filter numbers in i^{th} layer. According to the computation unit's specification, the total computation cost C has a upper budget bound B_C, which denotes its maximum capability.

3.1.2 Memory Occupancy Constraint. We then calculate the memory for running a DNN using the total number of bits associated with weights and the feature maps as:

$$M = B_f \sum_{i=1}^{L} \sum_{j=1}^{n_i} r_i^j s_i^j n_{i-1} + B_a \sum_{i=1}^{L} \sum_{j=1}^{n_i} h_i^j w_i^j, \tag{2}$$

where M is the total memory cost. B_f and B_a are data bandwidth which usually equals to 32 bits in the hardware platforms. We set different memory cost budget B_M during the DNN reconfiguration process according to specific hardware platforms.

3.1.3 Energy Consumption Constraint. We then formulate the total energy consumption E and its constraint. Usually, the total energy consumption in a DNN includes two main parts: computation energy cost E_c and memory access energy cost E_m. According to the formulation of computation capacity, the former one can be represented as total cost of all MACs in the DNN, *i.e.*, $E_c = \varepsilon_c C$. Whereas the latter one is depended on the stored weights and feature maps. In this paper, according to [19], we assume that all the weights are stored in the Cache while all the feature maps are stored in the DRAM. Therefore, the total energy consumption is:

$$\begin{aligned} E =& E_c + E_m = \varepsilon_c \sum_{i=1}^{L} \sum_{j=1}^{n_i} r_i^j s_i^j n_{i-1} h_i^j w_i^j + \\ & \varepsilon_f B_f \sum_{i=1}^{L} \sum_{j=1}^{n_i} r_i^j s_i^j n_{i-1} + \varepsilon_a B_a \sum_{i=1}^{L} \sum_{j=1}^{n_i} h_i^j w_i^j, \end{aligned} \tag{3}$$

where ε_c represents the energy consumption for each MAC operation. ε_f and ε_a denote the energy cost per bit when accessing the Cache and DRAM memory, respectively. Denote B_E as the available energy budget which can be allocated to DNN during executing.

3.2 Fine-tuning Process Formulation

After identifying the potential computation resource constraints for specific device, we leverage the filter pruning technique to realize DNN reconfiguration. Firstly, we multiply each output feature map with a gate F_i^j for filter selection, where j means the jth output feature map in ith layer. The original value of gate F_i^j equals to 1. During the fine-tuning process, we will leverage the lasso regularization to force the F_i^j approach to 0 and we remove output feature map whose gate F_i^j value below a given threshold value. Therefore, for DNN model fine-tuning to specific hardware platform, we aim to solve the following optimization problem by embedding all potential resource constraints:

$$\min_{F \in \{0,1\}, \theta} Loss(F, \theta) + \lambda R(F), s.t., C_m^{con}(F) \le b_m, \tag{4}$$

where $Loss(F, \theta)$ is used to maintain the model accuracy, F denotes the set of all F_i^j. $R(\cdot)$ is a sparse regularization term to achieve the filter pruning based model regulation, which usually denotes as norm-1 value: $\|F\|_1$. $C_m^{con}(F)$ represents the mth type of computation resource constraint which mentioned above and b_m is its corresponding budget. Since we want to speed up the optimization process, we change $\le$ to $=$ in our reconfiguration scheme. Therefore, the objective function could be interpreted as minimizing both accuracy loss and filter numbers in the network but approximating to the given budget at the same time. In the next step, we will introduce how to use ADMM algorithm to optimize the objective function we formulated above.

3.3 Fine-tuning Process Optimization with ADMM-based Algorithm

Although the formulated DNN fine-tuning process with Eq. 4 is flexible and comprehensive, it will be prohibitively difficult to solve via directly stochastic gradient descent method, since the constraints $C_m^{con}(F)$ could be complex, non-differentiable, and non-convex. Therefore, we explore the ADMM algorithm to decompose the original optimization problem down into several easier-to-solve sub-problems. Before applying ADMM, to simplify computation process, we first put the constraints $C_m^{con}(F) = b_m$ into the Eq. 4 as norm-2 term:

$$\min_{F \in \{0,1\}, \theta} Loss(F, \theta) + \lambda_1 R(F) + \lambda_2 \|C_m^{con}(F) - b_m\|_2^2. \tag{5}$$

Leveraging the ADMM algorithm, we further introduce a simple auxiliary variable Z to replace F in the equation terms of sparse regularization and hardware constraints. Then, the augmented Lagrange function of Eq. 5 will be formulated as:

$$\begin{aligned} L(F, Z, u) =& Loss(F, \theta) + \lambda_1 R(Z) + \lambda_2 \|C_m^{con}(Z) - b_m\|_2^2 \\ & + u^T (F - Z) + \frac{\rho}{2} \|F - Z\|_2^2, \end{aligned} \tag{6}$$

where u is a Lagrange Multiplier. Then by defining $u = \rho s$, we derive the scaled form of ADMM and get:

$$\begin{aligned} L(F, Z, s) =& Loss(F, \theta) + \lambda_1 R(Z) + \lambda_2 \|C_m^{con}(Z) - b_m\|_2^2 \\ & + \frac{\rho}{2} \|F - Z + s\|_2^2 - \frac{\rho}{2} \|s\|_2^2. \end{aligned} \tag{7}$$

We could use ADMM to solve the problem Eq. 7 through decomposition and iteratively solving subproblems in the kth iteration:

$$F^{k+1} = \arg\min_F L(F^k, Z^k, s^k), \tag{8}$$

$$Z^{k+1} = \arg\min_Z L(F^{k+1}, Z^k, s^k), \tag{9}$$

$$s^{k+1} = s^k + F^{k+1} - Z^{k+1}. \tag{10}$$

In every sub-problem, we only optimize the targeted variable and fix the other variables with values taken from last iterations. For example, in Eq. 8, we fix Z and u but optimize F according to the Eq. 7. Therefore, F and Z are updated iteratively and in an alternating way. ADMM converges when the difference between F and Z is smaller than a given threshold ϵ.

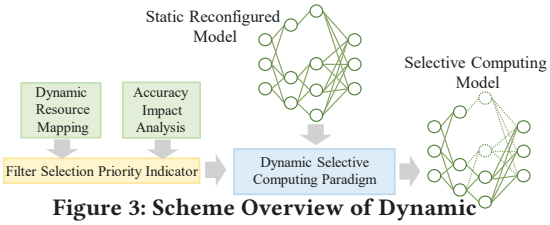

Figure 3: Scheme Overview of Dynamic DNN Reconfiguration

By using proposed static DNN reconfiguration scheme, we can optimize a DNN model under all potential computation resource constraints for specific platforms with high optimization efficiency.

4 DYNAMIC DNN RECONFIGURATION WITH SELECTIVE COMPUTING

Although the static DNN reconfiguration scheme can customize DNN models for static platform requirements, dynamic computation resource constraints might still be introduced by various real-time mobile applications. Therefore, in this section, we propose a dynamic DNN model reconfiguration scheme to adapt DNN model to dynamic computation resource by selectively computing filters in the network.

Fig. 3 shows the overview of proposed dynamic DNN model reconfiguration scheme. Firstly, we determine the filter computing priority by identifying a filter selection priority indicator. This indicator can be derived by conducting filter resource mapping and filter accuracy impact analysis. Then, with selection priority indicator obtained, we further propose our dynamic selective computing paradigm to dynamically reconfigure the DNN model generated from the static reconfiguration. By doing this, the DNN model can be optimized for all dynamic computation resource constraints.

4.1 Resource Aware Filter Significance

4.1.1 Dynamic Resource Mapping. Since filters in same layer has identical resource consumption, based on the neural network structure and computation mechanism, we can formulate the resource consumption for any filter in the ith layer with regard to memory M_i, energy E_i, and latency L_i:

$$M_i = B(r_i s_i n_{i-1} + h_i w_i + r_{i+1} s_{i+1} n_{i+1}), \tag{11}$$

$$E_i = \varepsilon_c (r_i s_i n_{i-1} h_i w_i + r_{i+1} s_{i+1} n_{i+1} h_{i+1} w_{i+1}) + \varepsilon_w B(r_i s_i n_{i-1} + r_{i+1} s_{i+1} n_{i+1}) + \varepsilon_a B h_i w_i, \tag{12}$$

$$L_i = (r_i s_i n_{i-1} h_i w_i + r_{i+1} s_{i+1} n_{i+1} h_{i+1} w_{i+1})/p, \tag{13}$$

where $r_i s_i$ and $h_i w_i$ represent the calculated sizes of the filter and feature map. n_{i-1} is the number of the output feature maps in the $(i-1)th$ layer, B is the data bandwidth (usually 32-bit). p means the processor's average computation capability in terms of MACs.

Based on these formulations, a preliminary resource-mapping analysis for *VGG-13* [23] is shown in Fig. 4. (a), (b), and (c) represent each filter's corresponding energy consumption, memory occupancy and inference latency, respectively. We can find that all 13 layers in *VGG-13* have distinct resource consumption preferences. For energy consumption and inference latency, stop one filter computation in $2nd$ layer can lead to largest energy and latency reduction. On the contrary, stopping filter's computation in last 4 layers will cause larger memory reduction.

4.1.2 Accuracy Impact Analysis. To obtain the consumption-accuracy trade-off, we need to further investigate each filter's accuracy impact. Since the accuracy impact differs for different layers,

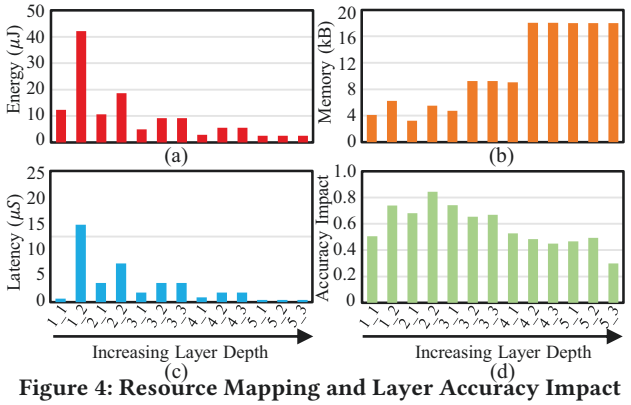

Figure 4: Resource Mapping and Layer Accuracy Impact

we need to divide the analysis into two steps: comparing layer's accuracy impact firstly and then measuring the filter's accuracy impact in each layer.

1. *Layer Accuracy Impact Analysis.* The first step aims to reveal layer's accuracy impact. For each layer, the impact can be measured by the model's accuracy drop when a certain portion of filters are gradually stop computing in this layer (empirically, we adopt 20% in each time). Larger accuracy drop indicates the layer's bigger impact (denoted as LI) to the classification results. For example, Fig. 4 (d) shows the LI distribution of all layers in *VGG-13*. We can find that $2nd$ to $7th$ layers have relatively larger LI values, which indicate higher accuracy impact.

2. *Filter Accuracy Impact Analysis.* The second step aims to determine filter's accuracy impact in each layer. In here, we introduce contribution index (CI) to indicate the filter's accuracy impact, which is defined by each filter's total differential impact to the network's final loss value Z:

$$Z(A_i^j + \delta) = Z(A_i^j) + \frac{\partial Z(A_i^j)}{\partial A_i^j} * \Delta, \tag{14}$$

$$CI_i^j = \sum |\frac{\partial Z(A_i^j)}{\partial A_i^j}|, \tag{15}$$

where A_i^j denotes the activation of jth filter in ith layer and $Z(A_i^j)$ means its corresponding final loss value. The coefficients matrix $\frac{\partial Z(A_i^j)}{\partial A_i^j}$ represents filter's contribution to the nth task. We use average L1-norm of the coefficients as CI, which is the filter's average contribution to final accuracy. With higher CI, the filter has more impact to the network accuracy.

4.1.3 Selection Priority Indicator. Based on the analysis above, we can evaluate the consumption-accuracy trade-off for each filter, which can be used as the priority indicator for selective computing:

$$PI_i^j = \frac{LI_i \times norm(CI_i^j)}{\alpha_M norm(M_i) + \alpha_E norm(E_i) + \alpha_L norm(L_i)}, \tag{16}$$

where $LI_i \times norm(CI_i^j)$ is the comprehensive accuracy impact for jth filter in ith layer, $norm(M_i)$, $norm(E_i)$, and $norm(L_i)$ are respectively the normalized memory, energy, and latency consumption. α_M, α_E, and α_L are the consumption weights determined by practical constraints. The filters with higher PI_i^j values are supposed to have higher accuracy impact and less resource consumption, which will be favored by selective computing.

4.2 Dynamic Selective Computing Paradigm

Since the resource constraints are dynamic during the DNN reconfiguration, we further propose the DNN dynamic selective computing algorithm to optimize the network without retraining. The

Algorithm 1 DNN Dynamic Selective Computing Algorithm

Input: 1)Reconfigured DNN model after fine-tuning process; 2)total computation cost C_m^{total}, real-time resource budget b_r;
Initialize the Selection Priority Index PI_l
while $C_m^{total} - \sum D_j^i > b_r$ **do**
    Masking the filter computing with least PI_l;
    Regard filter with sub-least PI_l value as least one
end while
Return Reconfigured DNN

algorithm is shown in Algorithm. 1. During DNN-based applications executing, the system consistently obtains the available resource b_r that can be allocated to DNN. Once any DNN computation costs C_m^{total} exceeds the available budget b_r, the filter with least PI_i^j value in current status will be masked for computing in a filter pruning manner. Then the DNN total computation cost C_m^{total} is updated and the filter with sub-least PI_i^j value will be updated as the least one in next masking status. The system iteratively executes the masking process until C_m^{total} below b_r. By applying this algorithm, a DNN can be dynamically reconfigured to meet any resource constraints introduced by real-time applications. Since all PI_i^j values are determined by pre-analysis, no further computation cost will be introduced. Also, to ensure the real-time performance, no model retraining is utilized. Although slight accuracy drop is inevitable, the consumption-accuracy trade-off is highly manageable based on thorough trade-off analysis.

5 EXPERIMENT

In this section, we conduct comprehensive evaluations to demonstrate the effectiveness of the proposed framework through three perspectives: optimization efficiency, static reconfiguration and dynamic reconfiguration.

5.1 Experiment Setup

We implement the static DNN reconfiguration scheme in *ReForm* in Tensorflow [1] environment. The dynamic DNN reconfiguration scheme is implemented with Tensorflow Lite.

To evaluate the performance of the proposed *ReForm*, two well-known models are considered: *LeNet* [17] and *VGG-13* [23]. The corresponding datasets are MNIST [16] and CIFAR-10 [15]. The original accuracy is 97% for *LeNet* and 90% for *VGG-13*. We evaluate *ReForm* on 4 commercial off-the-shelf mobile platforms, including 3 smartphones and 1 smart home device, which are Nexus 4, Honor 8, Redmi 3S and Xiaomi Box. These 4 platforms are equipped with different hardware configurations in perspectives of processors, DRAM size, battery capacity.

5.2 Experiment Evaluation

5.2.1 ReForm Optimization Efficiency Evaluation. We first evaluate *ReForm*'s static DNN reconfiguration efficiency and compare its performance with *NetAdapt* [27], which is one of the state-of-the-art resource-aware DNN optimization methods. It should be notified that, for simplicity, the original *NetAdapt* only consider inference latency constraint. Since latency is not a constraint during DNN static reconfiguration, we reproduce their method with memory occupancy consideration in our experiment. We reconfigure the *LeNet* model on MNIST and *VGG-13* model on CIRFAR-10 by utilizing both methods. The memory constraints are set as 12.4*MB* and 31*MB* respectively. During the evaluation, we set λ_1 and λ_2 with values from 10^{-2} to 10^2.

Tab. 1 shows the static DNN reconfiguration efficiency evaluation by comparing the optimization time cost. The notes below the table are the original baselines for *LeNet* and *VGG-13*. It is observed that both methods can keep original accuracy for *LeNet* on MNIST, and *ReForm* needs 3 minutes to finish the reconfiguration. Meanwhile, the time consumed by *NetAdapt* is 10 minutes. For *VGG-13* on CIFAR-10, the proposed *ReForm* needs 19 minutes while the *NetAdapt* needs 72 minutes. Therefore, *ReForm* has 3.3× and 3.8× speed-up than *NetAdapt*, indicating a better optimization efficiency.

5.2.2 ReForm Static Reconfiguration Evaluation. Our proposed framework's static reconfiguration is further evaluated by comparing the *VGG-13* reconfiguration results of both *ReForm* and *NetAdapt* on 4 mobile platforms mentioned above with different computation, energy and memory budgets. The accuracy drop is constrained within 1.5%. *NetAdapt* is evaluated under individual computation, energy, and memory constraints that is shown as the first three histograms in Fig. 5, and *ReForm* is evaluated under all constraints which is demonstrated as last histogram. The original *VGG-13* computation costs are used as baseline, which are 317.04*M*(Million MACs) for computation capacity, 62*MB* for memory occupancy, and 33.06$m\mathcal{J}$ for energy cost.

Fig. 5 indicates that both *ReForm* and *NetAdapt* can reconfigure the original network under the given constraints. However, the reconfigured network may still exceed the other constraints as *NetAdapt* only consider one constrain during reconfiguration. For example, *NetAdapt* can reduce the memory below to memory budget 45*MB* when it used to reconfigure network on Nexus 4, while the energy and computation capacity are still larger than the given budgets. On the contrary, *ReForm* can optimize *VGG-13*'s resource consumption since it takes all computation resource constraints into consideration during the reconfiguration process. Take Nexus 4 as an example, the given budget are 270*M*, 45*MB*, and 26$m\mathcal{J}$. After reconfiguration, the proposed framework reduce computation capacity from 317.04*M* to 259.46*M*, memory occupancy from 62*MB* to 43.25*MB*, and energy consumption from 33.06$m\mathcal{J}$ to 25.96$m\mathcal{J}$.

Therefore, compared with *NetAdapt*, our proposed framework can optimize the network under all computation resource constraints with neglect accuracy drop and can achieve at most 18% capacity, 30% memory, and 21.5% energy reduction.

5.2.3 ReForm Dynamic Reconfiguration Evaluation. The dynamic reconfiguration of the proposed framework is also evaluated on Honor 8 and its specific *VGG-13* model. The model is obtained from the above static reconfiguration and runs in an image recognition application. According to our measurement, the memory occupancy of this VGG-based application includes model size, application native data size, camera graphic size and other overhead which is 2.5

Table 1: DNN Optimization Efficiency Comparison

		Accuracy	Memory	Time Cost
LeNet	NetAdapt [27]	97%	12.38MB	10min
	ReForm	97%	12.35MB	3min
VGG-13	NetAdapt [27]	90%	30.67MB	72min
	ReForm	90%	30.3M	19min

∗Original *LeNet* Computation Cost Baseline: Capacity:42.77M Memory: 12.6MB Energy:3.56mJ
∗Original *VGG-13* Computation Cost Baseline: Capacity:317.04M Memory: 62MB Energy:33.06mJ

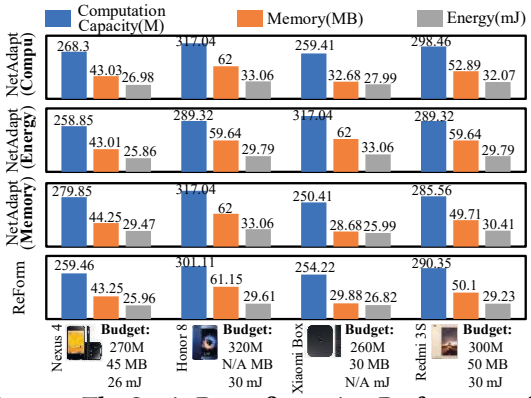

Figure 5: The Static Reconfiguration Performance for Various Mobile Platforms

times as model size. Three sets of representative mobile applications are considered as the background computing scenarios which includes gaming, Internet and VR camera. In addition, we add VR camera (charging) to show a more clear comparison, meaning that the mobile device is charging during VR scenario. The experiment results are obtained based on 10000 times task executions.

Fig. 6 shows *ReForm* adaptation result in the four computing scenarios. With different applications running, the resource budgets left for model execution are different (denoted by black lines). And some scenarios can not afford the computation resource for the VGG-based application execution, such as gaming and VR applications. For example, in VR camera scenario, *ReForm* first examines the energy budget and finds the original VGG-based task's energy consumption exceeds the budget. In such case, *ReForm* dynamically masks 19.8% filters with 1.3% accuracy drop and reduces model consumption to meet the energy constraint. On the contrary, in VR camera(charging) scenario, *ReForm* finds there is no constraint for energy consumption because of charging. It further examines the memory budget and reduce the memory occupancy until meets the given budget with only 0.8% accuracy drop.

Therefore, the DNN related applications can be well balanced with acceptable accuracy performance and manageable resources under the dynamic reconfiguration of *ReForm*.

6 CONCLUSION

In this paper, we propose a resource-aware DNN reconfiguration framework *ReForm* to solve the challenges of deploying DNNs in mobile platforms. Through innovative model reconfiguration schemes, *ReForm* can efficiently and effectively optimize mobile DNN models with regards to various static and dynamic computation resource constraints. The experiment results show that *ReForm* has optimal efficiency with ~3.5× faster speed than state-of-the-art resource-aware optimization method. Also, *ReForm* can effective reconfigure a DNN model to different mobile devices with distinct resource constraints. Moreover, *ReForm* achieves satisfying computation cost reduction with ignorable accuracy drop in both static and dynamic computing scenarios (at most 18% workload, 16.23% latency, 48.63% memory, and 21.5% energy enhancement). In summary, our work can comprehensively optimize DNN models to various constraints simultaneously and provide optimal performance enhancement in various computing scenarios.

Acknowledgment: This work was supported in part by NSF CNS-1717775.

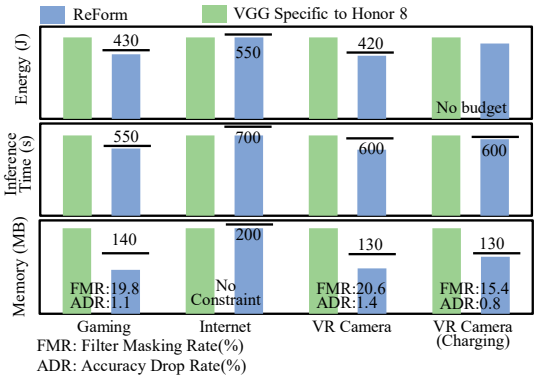

Figure 6: The Reconfiguration Performance for Various Mobile computing Scenarios

REFERENCES

[1] Martín Abadi and *et al.* 2016. Tensorflow: A System for Large-scale Machine Learning. In *OSDI*, Vol. 16. 265–283.

[2] Stephen Boyd. 2011. Alternating direction method of multipliers. In *Talk at NIPS workshop on optimization and machine learning.*

[3] François Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv preprint* (2017), 1610–02357.

[4] Matthieu Courbariaux and *et al.* 2016. Binarized neural networks: Training Deep Neural Networks with Weights and Activations Constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[5] Biyi Fang and *et al.* 2018. NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision. In *Proc. of MobiCom.*

[6] Ariel Gordon and *et al.* 2017. MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks. *arXiv preprint arXiv:1711.06798* (2017).

[7] Song Han and *et al.* 2015. Deep compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149* (2015).

[8] Song. Han and *et al.* 2015. Learning Both Weights and Connections for Efficient Neural network. In *Advances in neural information processing systems.* 1135–1143.

[9] Yihui He and *et al.* 2018. Amc: Automl for Model Compression and Acceleration on Mobile Devices. In *Proc. of ECCV.*

[10] Andrew G Howard and *et al.* 2017. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861* (2017).

[11] Hengyuan Hu and *et al.* 2016. Network trimming: A data-driven Neuron Pruning Approach Towards Efficient Deep Architectures. *arXiv preprint arXiv:1607.03250.*

[12] Forrest N Iandola and *et al.* 2016. Squeezenet: Alexnet-level Accuracy with 50x Fewer Parameters and Less 0.5mb Model Size. *arXiv preprint arXiv:1602.07360.*

[13] Max Jaderberg and *et al.* 2014. Speeding up Convolutional Neural Networks with Low Rank Expansions. *arXiv preprint arXiv:1405.3866* (2014).

[14] Alex Krizhevsky and *et al.* 2012. Imagenet Classification with Deep Convolutional Neural Networks. In *Proc. of NIPS.*

[15] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning Multiple Layers of Features from Tiny Images. (2009).

[16] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST Handwritten Digit Database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist.*

[17] Yann LeCun and *et al.* 2015. LeNet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet* (2015), 20.

[18] Hao Li and *et al.* 2016. Pruning Filters for Efficient Convnets. *arXiv preprint arXiv:1608.08710* (2016).

[19] Sicong Liu and *et al.* 2018. On-Demand Deep Model Compression for Mobile Devices: A Usage-Driven Model Selection Framework. (2018).

[20] Ningning Ma and *et al.* 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv preprint arXiv:1807.11164* (2018).

[21] Adam Polyak and Lior Wolf. 2015. Channel-level Acceleration of Deep Face Representations. *IEEE Access* 3 (2015), 2163–2175.

[22] Changhao Shan and *et al.* 2018. Attention-based End-to-end Speech Recognition on Voice Search. In *Proc. of ICASSP.*

[23] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-scale Image Recognition. *arXiv preprint arXiv:1409.1556* (2014).

[24] Yue Wang and *et al.* 2018. EnergyNet: Energy-Efficient Dynamic Inference.

[25] Zirui Xu and *et al.* 2018. DiReCt: Resource-Aware Dynamic Model Reconfiguration for Convolutional Neural Network in Mobile Systems. In *Proc. of ISLPED.*

[26] Tien-Ju Yang and *et al.* 2016. Designing Energy-efficient Convolutional Neural Networks using Energy-aware Pruning. *arXiv preprint arXiv:1611.05128* (2016).

[27] Tien-Ju Yang and *et al.* 2018. Netadapt: Platform-aware Neural Network Adaptation for Mobile Applications. *Energy* 41 (2018), 46.