Journal of Systems Architecture 143 (2023) 102978

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

CoAxNN: Optimizing on-device deep learning with conditional approximate neural networks

Guangli Li [a,b,1], Xiu Ma [c,d,1], Qiuchu Yu [a,b], Lei Liu [c,d], Huaxiao Liu [c,d], Xueying Wang [a,b,*]

[a] State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[b] University of Chinese Academy of Sciences, Beijing, China
[c] College of Computer Science and Technology, Jilin University, Changchun, China
[d] MOE Key Laboratory of Symbolic Computation and Knowledge Engineering, Jilin University, Changchun, China

ARTICLE INFO

Keywords:
On-device deep learning
Efficient neural networks
Model approximation and optimization

ABSTRACT

While deep neural networks have achieved superior performance in a variety of intelligent applications, the increasing computational complexity makes them difficult to be deployed on resource-constrained devices. To improve the performance of on-device inference, prior studies have explored various approximate strategies, such as neural network pruning, to optimize models based on different principles. However, when combining these approximate strategies, a large parameter space needs to be explored. Meanwhile, different configuration parameters may interfere with each other, damaging the performance optimization effect. In this paper, we propose a novel model optimization framework, CoAxNN, which effectively combines different approximate strategies, to facilitate on-device deep learning via model approximation. Based on the principles of different approximate optimizations, our approach constructs the design space and automatically finds reasonable configurations through genetic algorithm-based design space exploration. By combining the strengths of different approximation methods, CoAxNN enables efficient conditional inference for models at runtime. We evaluate our approach by leveraging state-of-the-art neural networks on a representative intelligent edge platform, Jetson AGX Orin. The experimental results demonstrate the effectiveness of CoAxNN, which achieves up to 1.53× speedup while reducing energy by up to 34.61%, with trivial accuracy loss on CIFAR-10/100 and CINIC-10 datasets.

1. Introduction

Convolutional neural networks (CNNs) have achieved remarkable success in various intelligent tasks such as image classification [1]. To pursue superior performance on complex intelligent tasks, CNNs are becoming wider and deeper, leading to tremendous computational costs and expensive energy consumption for model execution. Recently, on-device deep learning has been a mainstay due to its immeasurable potential for privacy protection and real-time response. However, it is hard to deploy complicated neural network models on edge devices due to the limited resources.

Many efforts have been made to enable efficient on-device deep learning via model approximation. For instance, pruning-based strategies [2] compress a neural network model by reducing redundant neurons and connections and quantization-based methods [3] improve the efficiency of model execution by leveraging low-precision computations. In addition to these model compression techniques, emerging staging-based approximate strategies, such as early exiting, improve model performance by conditional execution at runtime.

While these methods optimize the deep neural network models from different directions, we found that it is still a challenging problem to effectively combine them (as described in Section 2.4). To achieve efficient on-device inference, it is needed to take full advantage of the superiority of different optimization strategies. Different approximate strategies, based on distinct principles, have their own configuration parameters. When combining different strategies, the configuration parameters of different strategies may affect each other, influencing the optimization effect of the model, and even leading to poor optimization results. As such, this paper aims to address the following challenging problem: *How to design an efficient model optimization framework to make*

* Corresponding author at: State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.
E-mail addresses: liguangli@ict.ac.cn (G. Li), maxiu18@mails.jlu.edu.cn (X. Ma), yuqiuchu19@mails.ucas.ac.cn (Q. Yu), liulei@jlu.edu.cn (L. Liu), liuhuaxiao@jlu.edu.cn (H. Liu), wangxueying@ict.ac.cn (X. Wang).
[1] Guangli Li and Xiu Ma contributed equally to this work.

https://doi.org/10.1016/j.sysarc.2023.102978
Received 24 April 2023; Received in revised form 18 July 2023; Accepted 23 August 2023
Available online 25 August 2023

full use of various model approximate strategies, so as to optimize on-device deep learning while meeting accuracy requirements?

In this paper, we present a novel neural network optimization framework, CoAxNN (Conditional Approximate Neural Networks), which effectively combines staging-based and pruning-based approximate strategies, for efficient on-device deep learning. The staging-based approximate strategy optimizes the model structure as multiple stages with different complexities by attaching multiple exit branches, whereas the pruning-based approximate strategy compresses the model parameters according to the importance of filters. CoAxNN takes account of both their optimization principles and automatically searches for reasonable configuration parameters to construct a compressed multi-stage neural network model, thus taking full advantage of the superiority of different approximate strategies to achieve efficient model optimization. The optimization techniques, including pruning and staging, have been studied individually in the past; the key novelty of our work is to provide an effective and efficient mechanism to combine them, so as to optimize the neural network performance with a reasonable configuration, for a given task and a platform.

The main contributions of this paper are as follows:

- We present a novel neural network optimization framework, namely CoAxNN, which effectively combines staging-based and pruning-based approximate strategies, thereby improving actual performance while meeting accuracy requirements, for efficient on-device model inference.
- According to the principles of staging-based and pruning-based approximate strategies, our framework constructs the design space, and automatically searches for reasonable configuration parameters, including the number of stages, the position of stages, the threshold of stages, and the pruning rate, so as to make full use of the advantages of both to achieve efficient model optimization.
- We validate the effectiveness of CoAxNN by optimizing state-of-the-art deep neural networks on a commercial edge device, Jetson AGX Orin, in terms of prediction accuracy, execution latency, and energy consumption, and experimental results show that CoAxNN can significantly improve the performance of model inference with trivial accuracy loss.

The rest of the paper is organized as follows. The background and motivation are introduced in Section 2. The details of our optimization framework are described in Section 3. The experimental evaluation is conducted in Section 4. A discussion is given in Section 5. The conclusion is presented in Section 6.

2. Background and motivation

2.1. Pruning-based approximation

Neural network pruning, one of the most representative model compression techniques, approximates the original neural network model by reducing redundant neurons or connections making less contribution to model performance. Most previous works on pruning-based approximation can be roughly divided into two categories: unstructured pruning and structured pruning.

Prior works on weight pruning [4,5] achieve high non-structured sparsity of pruned models by removing single parameters in a filter. Guo et al. [4] and Hal et al. [5] used magnitude-based pruning methods, which eliminate weights with the smallest magnitude. Guo et al. [4] proposed dynamic network surgery to reduce the network complexity by making on-the-fly connection pruning. Hal et al. [5] pruned low-weight connections to reduce the storage and computation demands by an order of magnitude. Some pruning research groups utilize first-order or second-order derivatives of the loss function with respect to the weights [6,7]. Hassibi et al. [6] proposed Optimal Brain Damage (OBD), which uses all second-order derivatives of the loss function to prune single non-essential weights. Optimal Brain Surgeon (OBS) [7] have optimized the OBD method, which considers the condition that the Hessian matrix is non-diagonal. These approaches show attractive theoretical performance improvement but are difficult to be supported by existing software and hardware. Unstructured sparse models require specific matrix multiplication calculations and storage formats, which can hardly leverage existing high-efficiency BLAS libraries.

Unlike the early efforts on unstructured pruning that may cause irregular calculation patterns, structured pruning reduces redundant computations on unimportant filters or channels to produce a structured sparse model. The corresponding feature maps can be deleted as the filters are pruned. Therefore, much recent work has focused on filter pruning methods. SFP [2] and ASFP [8] dynamically pruned the filters in a soft manner, which zeroizes the unimportant filters and keeps updating them in the training stage. Li et al. [9] presented a fusion-catalyzed filter pruning approach, which simultaneously optimizes the parametric and non-parametric operators. Luo et al. [10] pruned filters based on statistics information computed from its next layer. The filters of different layers may have different influences on model prediction. Li et al. [11] proposed a flexible-rate filter pruning approach, FlexPruner, which automatically selects the number of filters to be pruned for each layer. Plochaet et al. [12] introduced a hardware-aware pruning method with the goal of decreasing the inference time for FPGA deep learning accelerators, adaptively pruning the neural network based on the size of the systolic array used to calculate the convolutions. To preserve the robustness at a high sparsity ratio in structured pruning, Zhuang et al. [13] proposed an effective filter importance criterion to evaluate the importance of filters by estimating their contribution to the adversarial training loss. Besides, some researchers have found the value of network pruning in discovering the network architecture [14,15]. Liu et al. [14] demonstrated that in some cases pruning can be useful as an architecture search paradigm. Li et al. [15] proposed a random architecture search to find a good architecture given a pre-defined model by channel pruning. Li et al. [16] proposed an end-to-end channel pruning method to search out the desired sub-network automatically and efficiently, which learns per-layer sparsity through depth-wise binary convolution. Ding et al. [17] presented a neural architecture search with pruning method, which derives the most potent model by removing trivial and redundant edges from the whole neural network topology. The structured sparse model can be perfectly supported by existing libraries to achieve a realistic acceleration. In this paper, we adopt filter pruning to realize practical performance improvement for neural network models.

2.2. Staging-based approximation

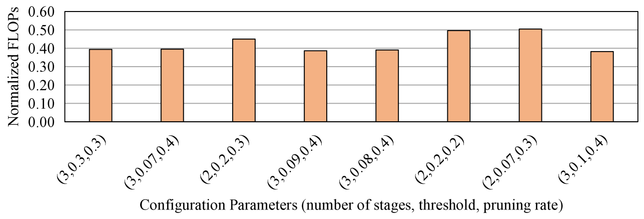
Prior studies [18,19] found that the difficulty of classifying an image in real-world scenarios is diverse. The easy samples can be classified with low effort, and difficult samples consume more computation efforts for prediction. Staging-based approximate strategies, such as early exiting [18] and layer skipping [20], emerge as a prominent important technique for separating the classification of easy and hard inputs. The original neural network uses a fixed computation process for the prediction of all samples. Staging-based approximate strategies perform adaptive computing for samples according to conditions at run-time. Teerapittayanon et al. [18] demonstrated that the deep neural network with additional side branch classifiers can both improve accuracy and significantly reduce the inference time of the network. Panda et al. [19] proposed Conditional Deep Learning cascading a linear network for each convolutional layer and monitoring the output of the linear network to decide whether classification can be terminated at the current stage or not. Fang et al. [21] presented an input-adaptive framework for video analytics, which adopts an architecture search-based scheme to find the optimal architecture for each early exit branch. Wang et al. [22] designed dynamic layer-skipping mechanisms, which suppress unnecessary costs for easy samples and halt inference for all samples to meet resource constraints for the inference of more complicated CNN backbones. Figurnov et al. [23] studied early termination in each residual unit of ResNets. Farhadi et al. [23] implemented an early-exiting method on the FPGA platform using partial reconfiguration to reduce the amount of needed computation. Jayakodi et al. [24] used Bayesian Optimization to configure the early exit neural networks to trade off accuracy and energy. To reduce unnecessary intermediate calculations in the inference process of Branchynet, Liang et al. [25] directly determined the exit position of the sample in the multi-branch network according to the difficulty of the sample without intermediate trial errors. Jo et al. [26] proposed a low-cost early exit network, which significantly improves energy efficiencies by reducing the parameters used in inference with efficient branch structures. In this paper, we achieve a multi-stage approximate model by early exiting to accelerate model inference for input samples in real-world scenarios.

2.3. Design space exploration

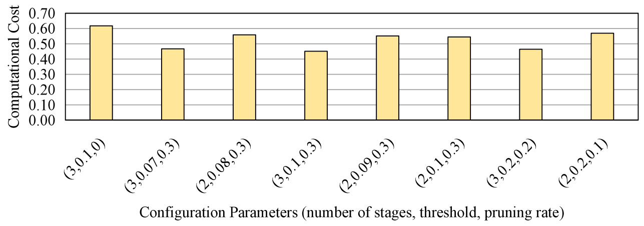
Design space exploration (DSE) is a systematic analysis method, which searches for the optimal solutions in a large design space according to the requirements. For example, in the staging-based approximate strategy, deciding whether or not an exit branch should be inserted at some position in the middle of the neural network model, and how the thresholds for each exit point should be set can be seen as a DSE problem. Panda et al. [19] and Teerapittayanon et al. [18] empirically set the location and threshold for each exit in the conditional neural network model. Jayakodi et al. [24] found the best thresholds via Bayesian Optimization for the specified trade-off between accuracy and energy consumption of inference. Park et al. [27] systematically determined the locations and thresholds of exit branches by genetic algorithm. Park et al. [28] integrated the once-for-all technique and BPNet, which consider architectures of base network and exit branches simultaneously in the same search process. Besides, the fine-grained filter pruning, that is assign reasonable pruning rates for different layers, can also be considered as a classic DSE problem. Li et al. [11] proposed a flexible-rate filter pruning method, which selects the filters to be pruned with a greedy-based strategy. He et al. [29] sampled design space using reinforcement learning, which performs customizing pruning for each layer, thus improving model compression. Qian et al. [30] proposed a hierarchical threshold pruning method, which considers the filter importance within relatively redundant layers instead of all layers, achieving layerwise pruning for a better network structure. In this paper, we regard the configuration parameters of staging-based and pruning-based approximate strategies as the whole design space and employ a genetic algorithm(GA)-based DSE to automatically find the (near-)optimal configuration to effectively combine them, achieving efficient on-device inference. In the future, we will consider setting reasonable pruning rates for different layers.

2.4. Motivation

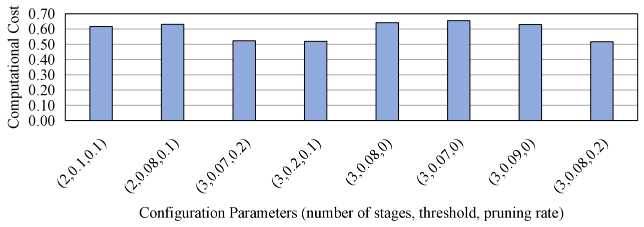
The pruning-based approximate strategy focuses on compressing the model, which reduces the computation costs by deleting unimportant parameters in the model, so how to set the pruning rate needs to be considered. The staging-based approximate strategy concentrates on improving the execution speed of the model, which allows the inference of most simple samples to terminate with a good prediction in the earlier stage by attaching multiple exits in the original model. How to place the exits and how to set a threshold for each exit should be considered for the design of a staging-based approximate strategy. Combining different approximate strategies will involve more configuration parameters and the approximate strategies may affect each other, which potentially influences the effect of the model optimization.



(a) The optimization effect of different configuration parameters under the accuracy of 98.1%.



(b) The optimization effect of different configuration parameters under the accuracy of 98.7%.



(c) The optimization effect of different configuration parameters under the accuracy of 98.8%.

Fig. 1. The optimization effect for ResNet-56 using different configuration parameters under the specified accuracy requirement.

Fig. 1 shows the optimization effect of the ResNet-56 using different configuration parameters under the specified requirements of accuracy on the CIFAR-10 dataset, where the triples (x, y, z) represent the number of stages, stage threshold, and pruning rate, respectively. Fig. 1(a), (b), and (c) show the computational costs (normalized to the computational cost of the baseline model) of various optimization configurations when the accuracy is 98.1%, 98.7%, and 98.8% (normalized to the accuracy of the baseline model). In practice, a certain error can be allowed in model accuracy (±0.001), for example, 98.09%, and 98.12% both meet the requirement of 98.1%. The relationship between the number of stages and the computational cost is not regular, which will be affected by the stage threshold and pruning rate, for example, in Fig. 1(c), the computational cost of (3,0.08,0) with more stages is larger than (2,0.1,0.1), the computational cost of (2,0.1,0.1) with fewer stages is larger than (3,0.2,0.1). Besides, affected by the staging-based optimization, the computational costs of the optimized model at a high pruning rate may be larger than that at low pruning rates, for example, in Fig. 1(b), the configuration of (2,0.08,0.3) with a pruning rate of 0.3 has more computational costs than (3,0.2,0.2) with a pruning rate of 0.2. In Fig. 1(a), we can observe from the partial experimental results that at the accuracy requirement of 98.1%, the computation of the optimized models using three stages is less than that of the model using two stages. But this law does not apply to the optimization effect of other accuracy requirements such as 98.7% and 98.8%. It is observed that the optimization effects of different configuration parameters are distinct and irregular under the specified accuracy requirement, and thus it is difficult to find an optimal model. This example shows that

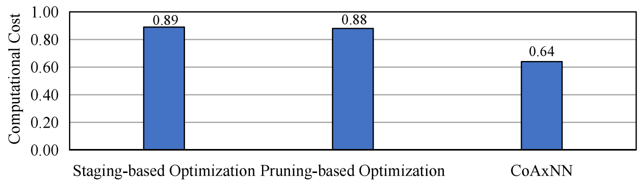


Fig. 2. The optimization effect of staging-based strategy, pruning-based strategy, and CoAxNN for ResNet-56 on the CIFAR-10.

it is challenging to combine different approximate strategies to achieve efficient optimization for neural network models.

In this paper, for a specified accuracy requirement, we focus on combining the principles of different approximate strategies to construct a design space and automatically search for reasonable configuration parameters, giving full play to the advantages of different approximate strategies to achieve efficient optimization for neural network models. As shown in Fig. 2, at the accuracy requirement of 99.6%, for the staging-based optimization strategy, the two stages are used and the threshold is set to 0.07 for each stage, the normalized computational cost is 0.89. For the pruning-based optimization, the pruning rate is set to 0.1, and the normalized computational cost is 0.89. CoAxNN effectively combines pruning-based and staging-based strategies, whose computational cost is 0.64, greatly improving the computational performance.

3. Methodology

3.1. Overview

In this paper, we propose an efficient optimization framework for neural network models, CoAxNN, which automatically searches for reasonable configuration parameters through a GA-based DSE. CoAxNN effectively combines staging-based with pruning-based approximate strategies to make full use of the superiority of both, thereby improving the actual performance while meeting the accuracy requirements for neural network models.

The overview of the CoAxNN is shown in Fig. 3. First, for the original deep neural network model, CoAxNN performs staging-based and pruning-based approximate strategies according to the genes of the chromosome for each individual, which generates a compressed multi-stage model. According to the availability of stages in the gene, CoAxNN attaches exit branches to the original model to build a multi-stage conditional activation model. According to the threshold of each stage, CoAxNN predicts input samples, having distinct difficulties, by multiple stages with different computational complexities, with the entropy-aware activation manner. The obtained multi-stage model is compressed by removing unimportant filters, thereby further reducing computational costs. Next, CoAxNN evaluates the fitness of the corresponding individual according to the accuracy and latency of the compressed multi-stage model and sorts the individuals according to their fitness. Then, the chromosome pool is updated, generating the next generation of individuals. After the evolution of multiple generations, which repeat the above steps, we can obtain several individuals that have optimal performance.

3.2. Staging-based approximate optimization

In general, executing a neural network model is a one-staged approach, which processes all the inputs in the same manner, i.e., starting from the input operator and performing it operator by operator until the final exit operator. Prior studies [19] found that classification difficulty varies widely across inputs in real-world scenarios. Different computational complexities need to be considered when predicting inputs. Most of the input samples can be correctly classified by employing a part of a neural network, without the computation effort of the entire neural network. Early exiting strategy often comes into play, which allows simple inputs to exit early with a good prediction by the addition of multiple exit points. By leveraging the early exiting strategy, CoAxNN achieves a staging-based approximation to give an early exiting opportunity for simple inputs. We denote a neural network model as $\mathcal{N} = \{f_1, f_2, \dots, f_m\}$ which consists of m operators. In CoAxNN, a multi-stage model, $\mathcal{N}^*$, can be formalized as follows:

$$\mathcal{N}^* = \bigcup_{i=0}^{\tau} \mathcal{S}_i \tag{1}$$

where τ is the number of stages.

The $\mathcal{S}_i$ is an approximate model with the staging-based strategy, which can be formalized as:

$$\mathcal{S}_i = \begin{cases} \mathcal{M}_i + \mathcal{B}_i + \mathcal{C}_i, \ 1 \leqslant i < \tau \\ \mathcal{N}, \ i = \tau \end{cases} \tag{2}$$

where $\mathcal{M}_i = \{f_1, f_2, \dots, f_{p_i}\}$ represents a part of the original neural network with p_i operators, $\mathcal{B}_i = \{f_1^*, f_2^*, \dots, f_{b_i}^*\}$ represents an additional exit branch with b_i operators, and $\mathcal{C}_i = \{c_i, \varepsilon_i\}$ represents an exit checker, containing a threshold ε_i and a conditional activation operator c_i using threshold ε_i. Especially, $\mathcal{S}_\tau = \mathcal{N}$ denotes the original (main) neural network model.

It is non-trivial to design a staging-based approximate strategy for adaptive conditional inference of a multi-stage model, and the following factors need to be considered:

- **Number of $\mathcal{B}_i$.** A multi-stage model with arbitrary exits can be built by stage availability. However, fewer exits cannot cover the diverse difficulty of classification of input samples, whereas too many exits increase the latency of hard samples that do not exit early.
- **Selection of Attached Position (p_i) for $\mathcal{B}_i$.** The exits at a more former position cannot provide satisfactory accuracy, while redundant computation may be involved at a more latter exit. Besides, attached exit branches may also interfere with a variety of computational graph optimization methods, such as operator fusion and memory reuse, provided by the deep learning frameworks, increasing operation counts, data movement, and other system overheads.
- **Confidence Threshold (ε_i) of $\mathcal{B}_i$.** The confidence threshold is used to determine whether the prediction result of stage $\mathcal{S}_i$ has sufficient confidence. With a higher threshold, complex samples may finish predictions from the previous exits with lower accuracy, and using a lower threshold, simple samples may use more complex computations to complete inference, due to cannot end from the previous classifiers, incurring additional computational overheads.
- **Structure Design for $\mathcal{B}_i$.** The structure of each exit branch ($\mathcal{B}_i$) is not identical. Each $\mathcal{B}_i$ consists of several operators ($\{f_1^*, f_2^*, \dots, f_{b_i-1}^*\}$) used for feature extraction and a linear classifier $f_{b_i}^*$. Feature extraction operators receive the intermediate feature map from f_{p_i} and extract more high-level features in the form required by a subsequent linear classifier. The configuration and complexity of the intermediate feature maps for different depths of the main neural networks are varying, making the design of $\mathcal{B}_i$ arduous. The $f_{b_i}^*$ operator is used to produce classification results based on the output of $f_{b_i-1}^*$, and the number of input feature maps for $f_{b_i}^*$ is different at each $\mathcal{B}_i$.

To effectively utilize the early-exiting method to build an approximate multi-stage model, our approach carefully designs principles for each module.

- **Setting of Number (τ) and Attached Position (p_i) of $\mathcal{B}_i$.** The number (τ) and the position (p_i) of the exit branches ($\mathcal{B}_i$) are two factors that will affect each other. Some unnecessary exits may be inserted, having little improvement in accuracy but leading to non-negligible computational overheads, when the number of exit branches is large. When the position of the exit branch is not

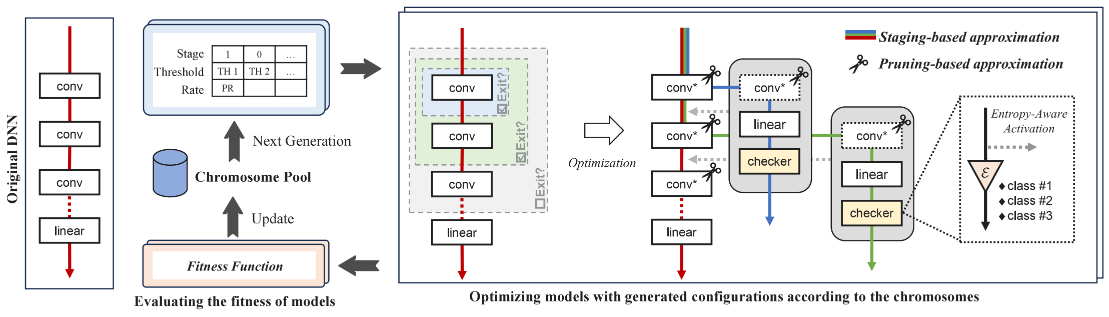


Fig. 3. Overview of CoAxNN.

reasonable, and cannot distinguish the difficulty of the sample, it is difficult to increase the number of exit branches to reduce the computational cost while meeting the model accuracy requirement. To address this problem, CoAxNN puts the availability of each stage into the design space of the GA, and each available stage corresponds to a new exit branch. The availability of the stage can control the number and position of exit branches at the same time. Besides, to introduce fewer new model structures and preserve the existing graph optimizations, CoAxNN chooses to attach exit branches $\mathcal{B}_i$ at the end of the group of building blocks. It is noted that CoAxNN does not attach the exit branch after the last group of building blocks, as there is already an existing original exit for the original backbone.

- **Structure Design for $\mathcal{B}_i$.** We introduce a feature extractor and a linear classifier for each exit branch $\mathcal{B}_i$. The structure of the feature extractor is designed with the building block as granularity. This design not only retains the original neural network structure but also provides more opportunities for system-level optimizations. Generally, operators for feature extraction also contain non-linear activation operators such as rectified linear units, and normalization operators such as batch normalization. Besides, prior studies [24] revealed that the output feature maps of operators at shallow depths of a neural network have a relatively large height and width, which results in a large number of input feature maps being passed to the linear classifier of former exits, thus leading to a long latency for easy samples that exit early. As such, in CoAxNN, we add an extra pooling operator after the last feature extraction operator of shallow $\mathcal{B}_i$.
- **Confidence Measure in C_i.** The C_i takes a threshold checking step, which determines whether an input returns from the current exit or continues to the next exit according to the prediction result of $\mathcal{S}_i$. A reliable C_i should have the ability to identify whether the classification results are sufficiently confident. There are various methods [31], including maximum probability, entropy, and margin, for the design of C_i. Prior work [24] has demonstrated the performance of the aforementioned three confidence types is almost identical. CoAxNN chooses to use the entropy of predicted probability as the entropy-aware activation operator (c_i) to evaluate the confidence of the prediction result for the input sample (x) of the ith stage classifier, as follows:

$$\text{entropy}(\hat{y}^i) = \sum_{c=1}^{C} \hat{y}^i(c) \log \hat{y}^i(c) \tag{3}$$

where $\hat{y}^i$ is the probability distribution of the output of the linear classifier $f^*_{b_i}$ on different classification labels, calculated by the softmax operator, and C is the number of classes. An entropy threshold ϵ_i is used to decide whether an input returns the prediction of the current exit or activates the latter operators. A higher confidence value implies that the input sample that arrived at the current exit is hard and needs to be processed by a more complex stage to complete accurate classification.

3.3. Pruning-based approximate optimization

In addition to the staging-based approximate strategy that provides adaptive computing based on conditional activation at runtime, CoAxNN also integrates a pruning-based approximate strategy to compress model size. The neural network pruning technique has been widely studied by researchers, which can be broadly categorized as structured and unstructured pruning. Structured pruning such as filter pruning has higher computational efficiency than unstructured pruning [32]. Especially, filter pruning is employed, which not only deletes redundant computations of unimportant filters but also leads to the removal of corresponding feature maps, providing realistic performance improvements. In CoAxNN, we utilize the filter pruning method to compress the multi-stage model and quantify the importance of each filter in a convolutional operator based on the ℓ_2-norm:

$$\|\mathcal{W}_r\|_2 = \sqrt{\sum_{t=1}^{k}\sum_{i=1}^{m}\sum_{j=1}^{n} w_{t,i,j}^2} \tag{4}$$

where $\mathcal{W}_r$ indicates the rth filter in a convolutional operator, $w_{t,i,j}$ denotes an element of $\mathcal{W}_r$ that resides in the ith row and jth column in the tth channel, k denotes the input channels, m denotes the height of filters, and n denotes the width of filters. The filters with smaller ℓ_2-norm will be given higher priority to be pruned than those of higher ℓ_2-norm. To keep the model capacity and minimize the loss of accuracy as much as possible, we utilize a dynamic pruning scheme [2] for staging-based approximate CNNs, which zeroes the pruned filters and keeps updating them in the re-training process.

3.4. Training of CoAxNN

Joint training trains all classifiers in a neural network model at the same time, which is widely used in the training process of neural network models with exit branches [18,27]. It defines a loss function for each classifier and minimizes the weighted sum of loss functions for all classifiers during training. Therefore, each classifier provides regularization for others to alleviate the overfitting of the model. CoAxNN utilizes joint training optimization to train the backbone neural network and exit branches at the same time and minimize the weighted sum of the cross-entropy loss functions of all stages, denoted as follows:

$$\mathcal{L}_{\text{joint}} = \sum_{i=1}^{\tau} \lambda_i \mathcal{L}_{\text{CE}}(y, \bar{y}^i) \tag{5}$$

where λ_i represents the weight of the loss function of the ith stage, y is the real classification of x which is shared by all stages, $\bar{y}^i$ is the output of linear classifier $f^*_{b_i}$ of the ith stage, and the cross-entropy loss function $\mathcal{L}_{\text{CE}}$ is calculated as follows:

$$\mathcal{L}_{\text{CE}}(y, \bar{y}^i) = -\sum_{c=1}^{C} y(c) \log \frac{e^{\bar{y}^i(c)}}{\sum_{j=1}^{C} e^{\bar{y}^i(j)}} \tag{6}$$

The training process of CoAxNN is summarized in Algorithm 1. It is given training dataset $\mathcal{X}$, training epochs $epoch_{max}$, batch size ρ, original deep neural network model $\mathcal{N}$, number of stages τ, the weights for loss functions of all stages λ, and the chromosome pool P. First, based on the genes on the chromosome, CoAxNN performs a staging-based optimization strategy, which approximates the original neural network model as a multi-stage conditional activation model by attaching exit branches (Lines 1–11). Then, the generated multi-stage model is initialized randomly (Lines 12–13). Next, the model is compressed and tuned according to the training data $\mathcal{X}$ and the gene of the pruning rate ($P[p].pruning\_rate$) on the chromosome in $epoch_{max}$ epochs (Lines 14–34). In each epoch, CoAxNN calculates the loss function according to Eq. (5) and updates the weights by the traditional backpropagation algorithm (Lines 15–26). Besides, for each convolutional operator in the approximate multi-stage model, CoAxNN obtains the number of filters (t) and calculates the ℓ_2-norm of each filter according to Eq. (4), then the dynamic pruning scheme is used to prune $\lfloor t \times P[p].pruning\_rate \rfloor$ filters with low ℓ_2-norm (Lines 27–33). The pruned filters can be updated once it is found to be important at any time, thus maintaining the learning ability of the model. In the pruning process of each epoch, CoAxNN will reorder the importance of filters for each convolutional operator, and select the filters to be pruned. Finally, the trained model $\mathcal{N}'$ is obtained (Line 36).

Algorithm 1: CoAxNN training

```
Input: training data: $\mathcal{X}$, training epoch: $epoch_{max}$, batch size: $\rho$,
       original model backbone: $\mathcal{N}$, the number of stages: $\tau$, the
       weights for loss functions: $\lambda$, chromosomes: $P$
Output: trained models: $\mathcal{N}'$
1  for $p = 1 \rightarrow P.size()$ do
       // Generate model structure
2      $\mathcal{N}'[p] = \mathcal{N}$;
3      for $i = 1 \rightarrow \tau - 1$ do
4          if P[p][i] is available then
5              Construct $\mathcal{M}_i$ from $\mathcal{N}$;
6              Construct $\mathcal{B}_i$ according to $\mathcal{M}_i$;
7              Construct $C_i$ according to $P[p][i].threshold$;
8              $\mathcal{S}_i = \mathcal{M}_i + \mathcal{B}_i + C_i$;
9              $\mathcal{N}'[p] = \mathcal{N}'[p] \cup \mathcal{S}_i$;
10         end
11     end
       // Tune and prune model parameters
12     $\mathcal{N}'[p]$ = LoadModel($\mathcal{N}'[p]$, $inital\_weights$);
13     $train\_batches$ = make_batch($\mathcal{X}$, $\rho$);
14     for $epoch = 1 \rightarrow epoch_{max}$ do
15         foreach $(input, target) \in train\_batches$ do
16             $output = \mathcal{N}'[p]$.forward($input$);
17             $weighted\_loss \leftarrow 0$;
18             for $i = 1 \rightarrow \tau$ do
19                 if P[p][i] is available then
20                     $loss$ = CrossEntropy($output[i]$, $target$);
21                     $weighted\_loss$ += $\lambda[i] \times loss$;
22                 end
23             end
24             $weighted\_loss = weighted\_loss / sum(\lambda)$;
25             $\mathcal{N}'[p]$.backward($weighted\_loss$);
26         end
27         foreach $f \in \mathcal{N}'[p]$ do
28             if $f.type == CONV$ then
29                 $t \leftarrow$ the filters number of $f$;
30                 Calculate the $\ell_2$-norm for the filters;
31                 Zeroize the lowest filters $\lfloor t \times P[p].pruning\_rate \rfloor$ filters;
32             end
33         end
34     end
35 end
36 return $\mathcal{N}'$;
```

3.5. GA-based design space exploration

To effectively combine the staging-based with the pruning-based approximate strategies, the design space of CoAxNN includes the number of stages, the position of the stage, the threshold of the stage, and the pruning rate, which is a very large search space. When the number of stages is τ, the search space for determining which stage is available is 2^τ, the search space for thresholds is Q^τ where Q is the number of candidate thresholds, and the search space for pruning rate is R, which indicates the number of candidate pruning rates. The parameter configurations are searched independently, making the search space as large as $2^\tau \times Q^\tau \times R$. It is laborious to explore the large parameter space by brute force search. CoAxNN adopts the genetic algorithm for the design space exploration. Genetic algorithm [33] is inspired by biological evolution based on Charles Darwin's theory of natural selection, which is often used to find the (near-)optimal solution in a large search space. In CoAxNN, the number of genes on each chromosome is $2\times(\tau-1)+1$. For the first $\tau-1$ stages, CoAxNN uses two genes, one for whether the stage is available, and the other for the threshold of the stage. In addition, CoAxNN also uses a gene to represent the pruning ratio. The fitness of the single individual is represented by a 2-tuple $(accuracy, latency)$. GA-based DSE aims to increase accuracy and reduce latency, finding the (near-)optimal solutions for model performance.

Algorithm 2 shows that how accuracy and latency are evaluated for individuals. It is given the test dataset $\mathcal{X}$, the number of stages τ, and the chromosome set P. For each individual, CoAxNN obtains the model net configured with the corresponding gene (Line 4). Then, the test dataset $\mathcal{X}$ is predicted by the model, and the result of prediction $output$ is got (Line 6). For each input sample, CoAxNN traverses all available stages and calculates the confidence e of corresponding output at this stage according to Eq. (3) (Lines 7–12). If the confidence (e) is less than the confidence threshold (ε_i) of this stage, the prediction is ended, and the accuracy of the sample at this stage is added to the accuracy score ($\delta[p]$) of the current individual (p) (Lines 13–16). The `accuracy` function returns 1 if the prediction is correct, and 0 otherwise. When the sample does not exit from the first $\tau-1$ stages, it must be exited from the τth stage. Therefore, in the τth stage, the accuracy is directly added to the accuracy score ($\delta[p]$) (Lines 17–19). The evaluation of latency is similar to accuracy. CoAxNN evaluates the latency score ($\mu[p]$) using a similar manner as the accuracy score ($\delta[p]$), which accumulates the latency of the backbone neural network and exit branches until the end of the prediction (Line 8–10). For the latency, we test the original network with all possible exit branches attached on the target edge devices. The execution time of all operators is recorded. Finally, the average accuracy score (δ) and average latency score (μ) for all individuals are obtained (Lines 23–26).

GA-based DSE gets the (near-)optimal solutions about the goal of accuracy and latency. Users choose the (near-)optimal solution among them according to their requirements. If the accuracy requirement is high, the model with the least computation cost is selected under a trivial accuracy loss. If a certain accuracy loss can be tolerated, the model with greatly less computation cost is selected. Finally, unavailable branches and unimportant filters are removed to obtain an optimized neural network model.

4. Evaluation

4.1. Experimental setting

Evaluation Platforms. We conduct optimization with PaddlePaddle,[2] an open-sourced deep learning framework, for neural network models on a server with Intel Xeon CPUs and an Nvidia A100 GPU. We evaluate

[2] https://www.paddlepaddle.org.cn/en.

Algorithm 2: Performance Collection

```
Input: test data: $\mathcal{X}$, the number of stages: $\tau$, chromosomes: $P$
Output: accuracy for each configuration of neural network
        models: $\delta$, latency for each configuration of neural
        network models: $\mu$
for $p = 1 \rightarrow P.size()$ do
    $\delta[p] \leftarrow 0$;
    $\mu[p] \leftarrow 0$;
    $net = \text{getModel}(P[p])$;
    foreach $(input, target) \in \mathcal{X}$ do
        $output = net.\text{forward}(input)$;
        for $i = 1 \rightarrow \tau$ do
            $\mu[p]$ += computeLatency($\mathcal{M}_i$);
            if $P[p][i]$ is available then
                $\mu[p]$ += computeLatency($\bigcup_{j=1}^{i} \mathcal{B}_j$);
                if $i \neq \tau$ then
                    $e \leftarrow$ Compute entropy of $output[i]$;
                    if $e < \varepsilon_i$ then
                        $\delta[p]$ += accuracy($output[i]$, $target$);
                        break;
                    end
                else
                    $\delta[p]$ += accuracy($output[i]$, $target$);
                end
            end
        end
    end
    $\delta[p] = \delta[p]/\mathcal{X}.size()$;
    $\mu[p] = \mu[p]/\mathcal{X}.size()$;
end
return $(\delta, \mu)$;
```

the realistic speedup and energy consumption of optimized models on a representative intelligent edge platform, Jetson AGX Orin, integrated with Ampere GPUs and Arm Cortex CPUs. For the genetic algorithm, we adopted the OpenGA [34] and the NSGA-III [35].

Benchmark Datasets and Models. We demonstrate the effectiveness of our proposed method on the CIFAR [36] dataset and the CINIC-10 [37] dataset. The CIFAR dataset, which consists of 50,000 images for training and 10,000 images for testing, contains two datasets: CIFAR-10 and CIFAR-100. The CIFAR-10 and CIFAR-100 datasets are categorized into 10 and 100 classes, respectively. CINIC-10 consisting of 27 000 images is split into three equal-sized train, validation, and test subsets and is categorized into 10 classes. We adopt the state-of-the-art residual neural network (ResNet) [1], which has less redundancy and is more challenging to be compressed and accelerated than conventional model structures, as model architectures. ResNet-20/32/56/110 models are evaluated for the CIFAR-10 dataset, ResNet-56/110 models are evaluated for the CIFAR-100 dataset and ResNet-18/50 models are evaluated for the CINIC-10 dataset.

Hyper-parameters Setting. For staging-based approximation, we attach exit branches after each residual block by default for building a multi-stage model, and the weight of the loss function of each stage is set to 1.0 by default. For pruning-based approximation, we follow the same data argumentation strategies and scheduling settings as [1].

4.2. GA-based design space exploration

The GA-based DSE, taking increasing accuracy and reducing latency as the goal, evaluates and sorts the solutions in the design space, and obtains the (near-)optimal solutions about accuracy and latency after multiple generations of individuals. After the process of survival of the fittest for multiple generations of individuals, the (near-)optimal solutions about accuracy and latency are obtained.

Fig. 4 shows solutions, obtained by GA-based DSE, for ResNet-20, ResNet-32, ResNet-56, and ResNet-110 on the CIFAR-10 dataset. The x-axis and y-axis represent the normalized top-1 accuracy and latency, normalized to the top-1 accuracy and latency of the corresponding baseline model, respectively. The data, marked by the green dot, are the design points of the brute-force algorithm, and the data, marked by the red triangle, are the (near-)optimal results found by CoAxNN. The optimal solutions found by brute force are plotted by the boundary of the green and red regions. It can be observed that the (near-)optimal solutions searched by CoAxNN are close to this boundary, which demonstrates the effectiveness of CoAxNN. Therefore, CoAxNN can search for the model having the least computational cost in most cases and meeting the accuracy requirements by GA-based DSE.

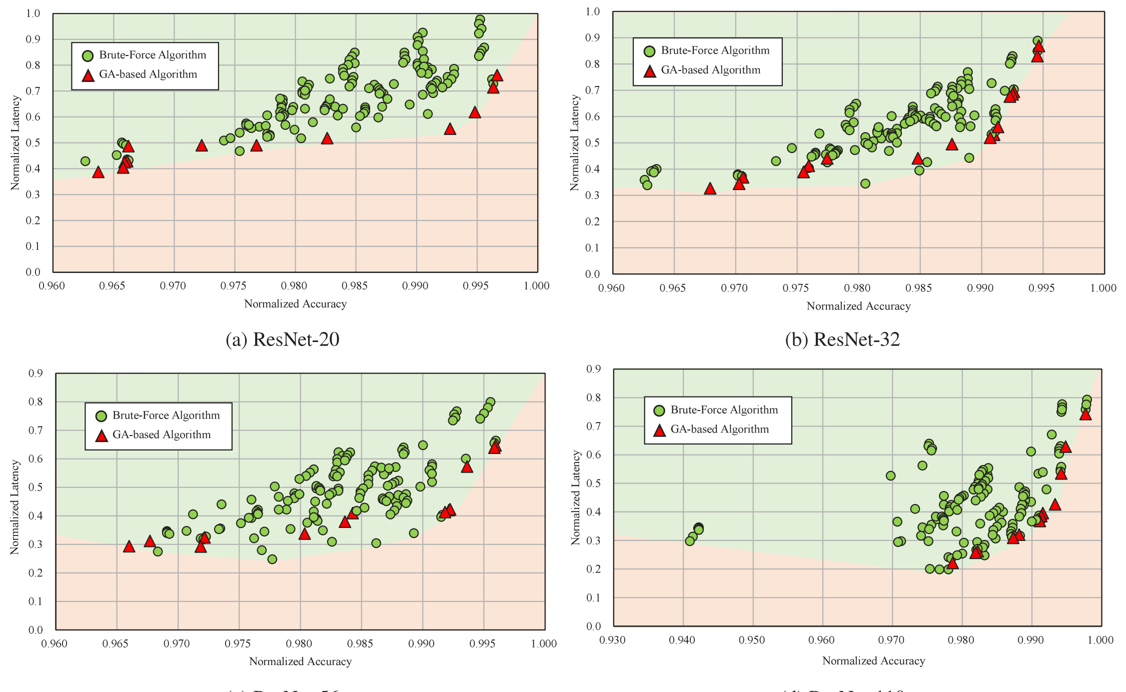
4.3. Performance of optimized models

We compare CoAxNN with state-of-the-art optimization methods such as ASRFP [38]. For the sake of fairness, the accuracy numbers are directly cited from their original papers. Different hyperparameters, such as learning rate, are used by distinct optimization methods, so the accuracy of the baseline model may be slightly different. Therefore, both the accuracy of the baseline model and the optimized model are shown in our experimental results, and "ACC. Drop" is used to represent the accuracy dropping of the model after optimization. A smaller number of "ACC. Drop" is better, and a negative number indicates the optimized model has higher accuracy than the baseline model. This is because model optimization has a regularization effect, which can reduce the overfitting of neural network models [2,18]. To avoid interference, we run each experiment three times and report the mean and standard deviation (mean ±std) of accuracy. Besides, we employ FLOPs to quantify the computational costs of neural network models.

4.3.1. ResNets on CIFAR-10

Table 1 shows the accuracy and FLOPs of ResNet-20/32/56/110 on the CIFAR-10 dataset. CoAxNN reduces the computational complexity of the original neural network model while meeting the accuracy requirements. The optimized ResNet-20, ResNet-32, ResNet-56, and ResNet-110 by CoAxNN achieves the FLOPs reduction from 4.06E7, 6.89E7, 1.25E8, 2.53E8 (refer to Table 2) to 3.00E7, 4.89E7, 8.06E7, 1.63E8, reduced by 25.94%, 28.93%, 35.76%, 35.57% in computational complexity, with the accuracy loss of 0.67%, 0.84%, 0.74%, and 0.63%, respectively. Moreover, CoAxNN can exploit less computation to achieve top-1 accuracy that is comparable to other state-of-the-art model optimization methods. For example, ResNet-20 optimized by SFP demands the computational complexity of 2.43E7 FLOPs while reducing the top-1 accuracy by 1.37%. The optimized ResNet-20 by CoAxNN consumes less computation cost, i.e., 2.27E7 FLOPs, drops by 1.39% in top-1 accuracy. CoAxNN reduces the computational cost of ResNet-32 to 3.44E7 FLOPs with a 1.58% accuracy drop. MIL spends more computations (4.70E7 FLOPs), reducing the top-1 accuracy by 1.59%. The compressed ResNet-56 by SFP achieves the FLOPs reduction of 52.60% and the accuracy loss of 1.33%. CoAxNN decreases the computational cost of ResNet-56 by 54.88% with a 1.22% accuracy drop. The optimized ResNet-110 by GAL reduces FLOPs by 48.50% with a 0.81% drop in top-1 accuracy. CoAxNN achieves a similar accuracy loss (0.88%) while reducing the computational complexity by 62.09%. For original neural network models, CoAxNN automatically searches for a reasonable configuration to effectively optimize the computational complexity while meeting the accuracy requirements. For the same accuracy requirement, CoAxNN reduces more computations than existing methods, achieving less resource consumption.

We also analyze the FLOPs and the percentage of predicted images for different stages of optimized ResNet-20, ResNet-32, ResNet-56,



(a) ResNet-20 (b) ResNet-32

(c) ResNet-56 (d) ResNet-110

Fig. 4. The solutions with GA-based DSE on the CIFAR-10 dataset. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Performance of optimized neural network models on CIFAR-10 (see [39–43]).

Model	Method	Top-1 Acc. Baseline (%)	Top-1 Acc. Accelerated (%)	Top-1 Acc. Drop (%)	#FLOPs	FLOPs ↓ (%)
ResNet-20	MIL [39]	92.49	91.43	1.06	2.61E7	36.00
	SFP [2]	92.20	90.83	1.37	2.43E7	42.20
	FPGM [40]	92.20	91.09	1.11	2.43E7	42.20
	TAS [41]	92.88	90.97	1.91	2.19E7	46.20
	CoAxNN (0.67%)	92.68	92.01 (±0.43)	**0.67**	**3.00E7**	**25.94**
	CoAxNN (1.39%)	92.68	91.29 (±0.26)	**1.39**	**2.27E7**	**44.02**
ResNet-32	MIL [39]	92.33	90.74	1.59	4.70E7	31.20
	SFP [2]	92.63	90.08	2.55	4.03E7	41.50
	TAS [41]	93.89	91.48	2.41	4.08E7	41.00
	CoAxNN (0.84%)	93.56	92.72 (±0.13)	**0.84**	**4.89E7**	**28.93**
	CoAxNN (1.58%)	93.56	91.98 (±0.41)	**1.58**	**3.44E7**	**49.98**
ResNet-56	SFP [2]	93.59	92.26	1.33	5.94E7	52.60
	ASFP [8]	93.59	92.44	1.15	5.94E7	52.60
	CP [42]	92.8	90.9	1.90	–	50.00
	AMC [29]	92.8	91.9	0.90	6.29E7	50.00
	CoAxNN (0.74%)	94.15	93.41 (±0.05)	**0.74**	**8.06E7**	**35.76**
	CoAxNN (1.22%)	94.15	92.93 (±0.25)	**1.22**	**5.66E7**	**54.88**
ResNet-110	SFP [2]	93.68	92.90	0.78	1.21E8	52.30
	ASRFP [38]	94.33	93.69	0.67	1.21E8	52.30
	TAS [41]	94.97	94.33	0.64	1.19E8	53.00
	GAL [43]	93.26	92.74	0.81	–	48.50
	CoAxNN (0.63%)	94.42	93.79 (±0.36)	**0.63**	**1.63E8**	**35.57**
	CoAxNN (0.88%)	94.42	93.54 (±0.17)	**0.88**	**9.59E7**	**62.09**

and ResNet-110, with an accuracy loss of 0.67%, 0.84%, 0.74%, and 0.63%, respectively, as shown in the Table 2. Weighted average FLOPs (“Avg. #FLOPs”) are computed by exit percentage and exit FLOPs for each stage (e.g., 3.00E7 = 58.71% × 1.93E7 + 41.29% × 4.53E7), which indicates the average model performance on the entire dataset. CoAxNN employs distinct stages for different neural network models. The two stages are used for ResNet-20, and three stages are used for more complex ResNet-32, ResNet-56, and ResNet-110. The neural network prediction finished at earlier stages costs less computational effort. Simple images, making up most of the dataset, are predicted by the first few stages, which reduces the computational complexity while ensuring accuracy.

Besides, we show the configurations of the optimized models searched by the GA-based DSE, as shown in Table 3. The pruning rate, the number of stages, and the position and the threshold for each stage are reported. For the optimized ResNet-20, the pruning rate is 0, i.e., no pruning is performed, the number of stages is two, the position of the first stage is the end of the fourth residual block, the corresponding threshold is 0.3, and the second stage refers to the backbone neural network with no confidence threshold since images must be exited from

Table 2
Analysis of optimized models on CIFAR-10.

Model (Acc.Drop)	Stages	CoAxNN			Baseline	FLOPs ↓ (%)
		Percentage	#FLOPs	Avg. #FLOPs	#FLOPs	
ResNet-20 (0.67%)	$\mathcal{S}_1$	58.71%	1.93E7	3.00E7	4.06E7	25.94
	$\mathcal{S}_2$	41.29%	4.53E7			
ResNet-32 (0.84%)	$\mathcal{S}_1$	41.72%	2.88E7	4.89E7	6.89E7	28.93
	$\mathcal{S}_2$	38.78%	5.59E7			
	$\mathcal{S}_3$	19.50%	7.83E7			
ResNet-56 (0.74%)	$\mathcal{S}_1$	44.81%	4.76E7	8.06E7	1.25E8	35.76
	$\mathcal{S}_2$	36.79%	9.36E7			
	$\mathcal{S}_3$	18.40%	1.35E8			
ResNet-110 (0.63%)	$\mathcal{S}_1$	42.66%	9.01E7	1.63E8	2.53E8	35.57
	$\mathcal{S}_2$	30.93%	1.79E8			
	$\mathcal{S}_3$	26.41%	2.62E8			

Table 3
Configurations optimized by GA-based DSE for CIFAR-10.

Model (Acc.Drop)	Configurations			
ResNet-20 (0.67%)	Rate	0		
	Stage	1	2	3
	Position	4	–	–
	Threshold	0.3	–	–
ResNet-32 (0.84%)	Rate	0		
	Stage	1	2	3
	Position	6	11	–
	Threshold	0.09	0.1	–
ResNet-56 (0.74%)	Rate	0		
	Stage	1	2	3
	Position	10	19	–
	Threshold	0.07	0.08	–
ResNet-110 (0.63%)	Rate	0		
	Stage	1	2	3
	Position	19	37	–
	Threshold	0.07	0.015	–

the last stage. Although ResNet-32, ResNet-56, and ResNet-110 are all optimized into three stages with a pruning rate of 0, the position and threshold of each stage are different. For the optimized ResNet-32, the threshold is 0.09, 0.1 for each stage where the position is the end of the 6, 11th residual block of the backbone network, respectively. For the optimized ResNet-56, the position of three stages is the end of the 10, 19th residual block with the threshold of 0.07, 0.08. The optimized ResNet-110 uses three-stage with the threshold of 0.07, 0.017, where the position is the end of the 19, 37th residual block.

4.3.2. ResNets on CIFAR-100

We evaluate CoAxNN on the CIFAR-100 dataset by ResNet-56 and ResNet-110, as shown in Table 4. Similarly, CoAxNN outperforms other state-of-the-art methods. For example, the computational complexity of optimized ResNet-110 by ASFP is 1.82E8 FLOPs, reduced by 28.20% compared to the original neural network model, leading to a 1.48% drop in top-1 accuracy. CoAxNN consumes 1.69E8 FLOPs, achieving a higher computation reduction of 33.34% and a lower accuracy loss of 1.30%. Although GHFP achieves a lower accuracy drop of 1.10%, it uses a higher computational complexity of 1.82E8 FLOPs. These results demonstrate the effectiveness of CoAxNN.

In addition, Table 6 shows the configurations of the optimized models with the accuracy loss of 0.98% and 1.30%, searched by CoAxNN, on the CIFAR-100 dataset. Despite the optimized ResNet-56 employing a three-stage and deactivating pruning-based strategy, which is as same as CIFAR-10, the thresholds are distinct. The optimized ResNet-56 uses three-stage with the threshold of 0.7 and 0.065. Besides, The optimized ResNet-110 adopts three-stage with a pruning rate of 0.1.

We also study the FLOPs and the percentage of predicted images of the optimized model at each stage on CIFAR-100, as shown in Table 5. CoAxNN uses three stages for the ResNet-56 and the ResNet-110 as same as CIFAR-10. But, since the CIFAR-100 is more complex, more complex models are required, leading to a smaller percentage of images predicted at $\mathcal{S}_1$ and $\mathcal{S}_2$ than CIFAR-10. For example, for ResNet-56 on CIFAR-10, the percentages of predicted images by $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ are 44.81%, 36.79%, and 18.40%, respectively. For ResNet-56 on CIFAR-100, the percentages of predicted images by $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ are 29.67%, 32.85%, and 37.48%, respectively. For both CIFAR-10 and CIFAR-100, most of the images on the whole dataset are predicted by the first few stages with less computation. On the CIFAR-100, CoAxNN reduces the FLOPs by 23.93% and 33.34%, with an accuracy drop of 0.98% and 1.30%, for ResNet-56 and ResNet-110, respectively.

4.3.3. ResNets on CINIC-10

We utilize the CINIC-10 dataset, which consists of images from both CIFAR and ImageNet [46], avoiding the time-consuming process of model training on the entire ImageNet dataset, to facilitate experiments for complicated image classification scenarios. We evaluate CoAxNN on the CINIC-10 dataset by ResNet-18 and ResNet-50 models that are in line with the model structures on the ImageNet dataset.

Table 7 shows the accuracy and computational cost of optimized models. For ResNet-18, when the FLOPs are reduced from 5.49E8 (i.e., the computational cost of the original ResNet-18, refer to Table 8) to 2.21E8, reduced by 59.80%, the top-1 accuracy is dropped by 1.01%. If the accuracy requirement is higher, CoAxNN can achieve 0.50% accuracy loss while reducing the computational complexity by 43.71% for the ResNet-18. ResNet-50 with a large number of computations is improved by 0.10% in top-1 accuracy, and the corresponding FLOPs is reduced from 1.18E9 (i.e., the computational cost of the original ResNet-50, refer to Table 8) to 4.63E8, reduced by 60.75% in computational complexity. We compare CoAxNN with state-of-the-art model optimization methods, FPC [47] and CCPrune [48]. FPC reduces the computational complexity by 40.48% (7.76E8 FLOPs) while increasing the top-1 accuracy by 1.14% for the ResNet-50 model. CCPrune increases the top-1 accuracy of the ResNet-50 model by 0.23% with a computational complexity of 7.44E8 FLOPs. CoAxNN reduces the computational complexity by 49.73% (5.93E8 FLOPs) with a 0.38% improvement in top-1 accuracy. By effectively combining stage-based with pruning-based approximate strategies, CoAxNN achieves better performance than existing methods.

Moreover, we analyze the FLOPs and predicted images at each stage for the optimized ResNet-18 and ResNet-50 with a 1.01% and −0.10% accuracy drop respectively, as shown in Table 8. For the CINIC-10 dataset, both the ResNet-18 and the ResNet-50 use four stages. More than 80% of the images are finished in the previous two stages, and less than 10% of images are predicted in the last stage.

Table 9 shows the configurations of the ResNet-18 and the ResNet-50. ResNet-18 uses four-stage with thresholds of 0.23, 0.2, and 0.4, whose position is the end of the 3, 5, and 7th residual block, and the pruning rate is 0.3. When the sample does not exit from the first few stages, it must be exited from the last stage. Therefore, the last stage

Table 4
Performance of optimized neural network models on CIFAR-100 (see [44,45]).

Model	Method	Top-1 Acc. Baseline (%)	Top-1 Acc. Accelerated (%)	Top-1 Acc. Drop (%)	#FLOPs	FLOPs ↓ (%)
ResNet-56	MIL [39]	71.33	68.37	2.96	7.63E7	39.30
	CoAxNN (0.98%)	72.75	71.77 (±0.28)	**0.98**	**9.55E7**	**23.93**
	CoAxNN (2.36%)	72.75	70.39 (±0.11)	**2.36**	**7.46E7**	**40.53**
ResNet-110	MIL [39]	72.79	70.78	2.01	1.73E8	31.30
	SFP [2]	74.14	71.28	2.86	1.21E8	52.30
	ASFP [8]	74.39	72.91	1.48	1.82E8	28.20
	ASRFP [38]	74.39	73.02	1.37	1.82E8	28.20
	GHFP [44]	74.39	73.29	1.10	1.82E8	28.20
	AHSG-HT [45]	74.46	72.74	1.72	–	29.30
	CoAxNN (1.30%)	74.17	72.87 (±0.19)	**1.30**	**1.69E8**	**33.34**
	CoAxNN (3.42%)	74.17	70.75 (±0.38)	**3.42**	**1.15E8**	**54.47**

Table 5
Analysis of optimized models on CIFAR-100.

Model (Acc.Drop)	Stages	CoAxNN			Baseline	FLOPs ↓ (%)
		Percentage	#FLOPs	Avg. #FLOPs	#FLOPs	
ResNet-56 (0.98%)	S_1	29.67%	4.76E7	9.55E7	1.25E8	23.93
	S_2	32.85%	9.36E7			
	S_3	37.48%	1.35E8			
ResNet-110 (1.30%)	S_1	27.60%	8.23E7	1.69E8	2.53E8	33.34
	S_2	30.18%	1.59E8			
	S_3	42.22%	2.32E8			

Table 6
Configurations optimized by GA-based DSE for CIFAR-100.

Model (Acc.Drop)	Configurations			
ResNet-56 (0.98%)	Rate	0		
	Stage	1	2	3
	Position	10	19	–
	Threshold	0.7	0.65	–
ResNet-110 (1.30%)	Rate	0.1		
	Stage	1	2	3
	Position	19	37	–
	Threshold	0.73	0.62	–

has no threshold value. The ResNet-34 uses four-stage with thresholds of 0.08, 0.09, and 0.09, whose position is the end of the 4, 8, and 14th residual block, and the pruning rate is 0.2.

Summary. As shown in Tables 1, 4, and 7, CoAxNN, which automatically finds (near)-optimal configurations for effectively combining staging-based and pruning-based approximate strategies, is comparable to the state-of-the-art methods. The staging-based approximate strategies perform adaptive inference for inputs according to conditions at run-time. The inference of simple input can be terminated with a good prediction confidence in the earlier stage, thereby avoiding remaining layerwise computations, so that the overall computation cost can be significantly reduced. However, the number of model parameters is still too large to be deployed on mobile devices. The pruning-based approximate strategies remove the unimportant weights or filters to gain a thinner model. However, the pruning method lacks the ability to configure the neural network dynamically, which will miss the opportunities to optimize the model inference. Based on these previous mentioned optimization principles, CoAxNN automatically finds (near-)optimal configurations by GA-based DSE, making full use of the advantages of both, thus achieving efficient model optimization.

4.4. *Realistic performance of on-device inference*

To demonstrate the realistic speedup and energy savings of our approximate compressed multi-stage models, we evaluate the performance of models on a representative intelligent edge device, Jetson AGX Orin.

For the measurement of inference latency, on the one hand, we pre-execute each neural network model 10 times to warm up the machine, and then repeat the single-batch inference 100 times to record the average execution time to reduce the interference, such as system initialization. On the other hand, after executing all the operators on the device we insert synchronous instructions to obtain timestamps, thus avoiding inaccurate measurements for inference time. Table 10 depicts the inference latency for the optimized ResNet-20, ResNet-32, ResNet-56, and ResNet-110 by CoAxNN, respectively dropped by 0.67%, 0.84%, 0.74%, and 0.63% in top-1 accuracy on the CIFAR-10 dataset. The results show that CoAxNN can accelerate ResNet-20, ResNet-32, ResNet-56, and ResNet-110 models by 1.33×, 1.34×, 1.53×, and 1.51×, respectively. In general, the larger models can obtain a more significant speedup.

To analyze the energy consumption of optimized models, we use the `jetson-stats`[3] to monitor the power of the system. We perform 10 000 times single-batch inference for ResNet-20, ResNet-32, ResNet-56, and ResNet-110 on Jetson AGX Orin, and the instantaneous powers are obtained to multiply the average inference time per image to compute the energy consumption of models. Table 11 shows the energy consumption for ResNet-20, ResNet-32, ResNet-56, and ResNet-110 with the accuracy loss of 0.67%, 0.84%, 0.74%, and 0.63% on the CIFAR-10 dataset. CoAxNN reduces the energy consumption of ResNet-20, ResNet-32, ResNet-56, and ResNet-110 by 25.17%, 25.68%, 34.61%, and 33.81%, respectively. The experimental results show that the optimized models by CoAxNN can be improved in terms of energy consumption, and the more complex neural network models can save more energy.

We also evaluate the realistic speedup and energy reduction of models optimized by existing filter pruning approaches [2,8,11]. Tables 12 and 13 show the execute latency and energy consumption of single-batch inference of optimized ResNet-20, ResNet-32, ResNet-56, and ResNet-110 by filter pruning with the accuracy loss of 2.32%, 1.12%, 0.23%, and 0.10%, on the CIFAR-10 dataset, respectively. Compared with the baseline models, the optimized models have higher execution latency and more energy consumption. Although filter pruning can reduce theoretical computation costs and memory footprint, the optimized models cannot obtain actual acceleration and energy reduction

[3] https://pypi.org/project/jetson-stats/.

Table 7
Performance of optimized neural network models on CINIC-10.

Model	Method	Top-1 Acc. Baseline (%)	Top-1 Acc. Accelerated (%)	Top-1 Acc. Drop (%)	#FLOPs	FLOPs ↓ (%)
ResNet-18	CoAxNN (0.50%)	87.57	87.07 (±0.29)	**0.50**	**3.09E8**	**43.71**
	CoAxNN (1.01%)	87.57	86.56 (±0.43)	**1.01**	**2.21E8**	**59.80**
ResNet-50	FPC [47]	86.63	87.77	−1.14	7.76E8	40.48
	CCPrune [48]	88.30	88.53	−0.23	7.44E8	–
	CoAxNN (−0.38%)	88.52	88.14 (±0.15)	**−0.38**	**5.93E8**	**49.73**
	CoAxNN (−0.10%)	88.52	88.62 (±0.34)	**−0.10**	**4.63E8**	**60.75**

Table 8
Analysis of optimized models on CINIC-10.

Model (Acc.Drop)	Stages	CoAxNN			Baseline	FLOPs ↓ (%)
		Percentage	#FLOPs	Avg. #FLOPs	#FLOPs	
ResNet-18 (1.01%)	S_1	50.86%	1.35E8	2.21E8	5.49E8	59.80
	S_2	30.96%	2.57E8			
	S_3	13.13%	3.79E8			
	S_4	5.05%	4.56E8			
ResNet-50 (−0.10%)	S_1	39.90%	2.11E8	4.63E8	1.18E9	60.75
	S_2	41.58%	4.91E8			
	S_3	9.27%	8.66E8			
	S_4	9.26%	1.02E9			

Table 9
Configurations optimized by GA-based DSE for CINIC-10.

Model (Acc.Drop)	Configurations				
ResNet-18 (1.01%)	Rate	0.3			
	Stage	1	2	3	4
	Position	3	5	7	–
	Threshold	0.23	0.2	0.4	–
ResNet-50 (−0.10%)	Rate	0.2			
	Stage	1	2	3	4
	Position	4	8	14	–
	Threshold	0.08	0.09	0.09	–

Table 10
Speedups of optimized models by CoAxNN on Jetson AGX Orin.

Model (Acc.Drop)	Latency (ms)		Speedup
	Baseline	CoAxNN	
ResNet-20 (0.67%)	6.26	4.69	1.33
ResNet-32 (0.84%)	9.55	7.11	1.34
ResNet-56 (0.74%)	16.89	11.05	1.53
ResNet-110 (0.63%)	32.33	21.4	1.51

Table 11
Energy reductions of optimized models by CoAxNN on Jetson AGX Orin.

Model (Acc.Drop)	Energy (mJ)		Reduction
	Baseline	CoAxNN	
ResNet-20 (0.67%)	27.89	20.87	25.17%
ResNet-32 (0.84%)	42.79	31.8	25.68%
ResNet-56 (0.74%)	76.57	50.07	34.61%
ResNet-110 (0.63%)	146.69	97.10	33.81%

Table 12
Speedups of optimized models by existing pruning approaches [2,8,11] on Jetson AGX Orin.

Model (Acc.Drop)	Latency (ms)		Speedup
	Baseline	Filter pruning	
ResNet-20 (2.32%)	6.26	8.70	0.72
ResNet-32 (1.12%)	9.55	13.73	0.70
ResNet-56 (0.23%)	16.89	22.51	0.75
ResNet-110 (0.10%)	32.33	42.59	0.76

Table 13
Energy reductions of optimized models by existing pruning approaches [2,8,11] on Jetson AGX Orin.

Model (Acc.Drop)	Energy (mJ)		Reduction
	Baseline	Filter pruning	
ResNet-20 (2.32%)	27.89	45.74	−63.99%
ResNet-32 (1.12%)	42.79	72.47	−69.36%
ResNet-56 (0.23%)	76.57	119.24	−55.73%
ResNet-110 (0.10%)	146.69	225.34	−53.61%

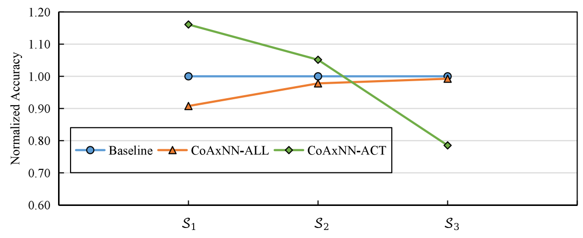


Fig. 5. Accuracy of the optimization model at different stages. "CoAxNN-ALL" and "CoAxNN-ACT" denote the accuracy of the model at each stage on the whole dataset and on the images that satisfy the activation condition of the corresponding stage, respectively.

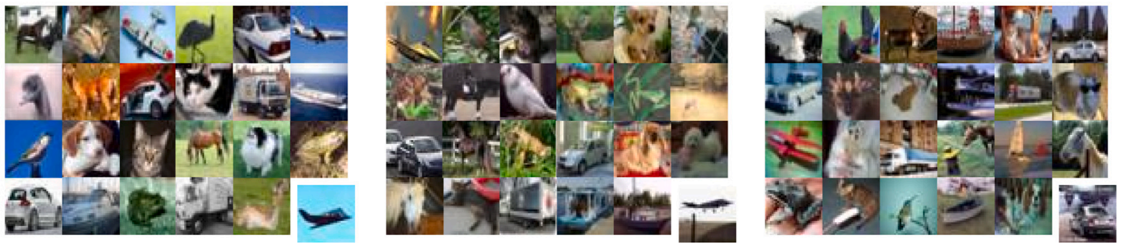

Images predicted at stage $\mathcal{S}_1$ Images predicted at stage $\mathcal{S}_2$ Images predicted at stage $\mathcal{S}_3$

Fig. 6. Example images predicated correctly at different stages.

Table 14
Overheads of GA-based DSE.

Model	GA time (s)	Training time (s)
ResNet-20	1.15	5472
ResNet-32	1.60	1813
ResNet-56	1.69	2720
ResNet-110	1.46	7712

on Jetson AGX Orin. Therefore, the critical motivation of CoAxNN is to find a satisfying optimization configuration for practical scenarios.

4.5. Ablation study

Accuracy of CoAxNN models at different stages. We study the accuracy of ResNet-56 optimized by CoAxNN at different stages, as shown in Fig. 5. In "CoAxNN-ALL", the accuracy of the model in the first few stages is lower than that of the baseline model. As the computational complexity of the model increases, the accuracy in the later stages gradually converges to that of the baseline model. CoAxNN separates the prediction of simple and complex images by conditional activation, allowing simple images to exit from the first few stages and complex images to exit from the latter stages. In "CoAxNN-ACT", the accuracy of the first few stages becomes higher and even exceeds that of the baseline model, which indicates that the first few stages have sufficient ability to classify simple images. Besides, since complex images are predicted by the later stages, the accuracy of the last stage of the optimization model is lower than that of the baseline model.

Visualization results at different stages. Fig. 6 depicts the predicted sample images for each stage of optimized ResNet-56 on CIFAR-10. The samples predicated at stage $\mathcal{S}_1$ are relatively "easy", which have a small number of objects and clear background, whereas the samples predicated at stage $\mathcal{S}_2$ and $\mathcal{S}_3$ are relatively "hard", which have various objects and complex background. CoAxNN can separate "easy" images consuming less effort from "hard" ones consuming more computation, significantly reducing computation costs for neural network models.

Overheads of GA-based DSE. We collect the latency of each operator of the neural network model on the edge device in the profiling phase beforehand to be used in GA-based search. We perform the model optimization processes, including model training and GA-based search, on a server with Intel Xeon CPUs and an Nvidia A100 GPU. The inference of optimized models is performed on edge devices such as Jetson AGX Orin. Table 14 shows the times for GA-based search and the time to train the model once during the model optimization. The GA-based DSE takes 1–2 s on the CPU platform, which is greatly less than model training (e.g., ResNet-20 takes 5472 s for training once). Therefore, the runtime overhead of the GA is negligible.

5. Discussion

Generality. CoAxNN is a generic framework for optimizing on-device deep learning via model approximation, which can be generalized to other intelligent tasks such as object detection [49]. In addition, more approximate strategies such as knowledge distillation [50] can be integrated into CoAxNN to further optimize neural network models.

Applicability. CoAxNN is system-independent, which not requires specific software implementations and hardware design support. The optimized models by CoAxNN can be directly deployed on the target platform, especially intelligent edge accelerators. Users can choose the (near)-optimal model according to the accuracy and performance requirements of intelligent tasks. Moreover, the time-consuming optimization process can be performed offline on high-performance servers, achieving efficient fine-tuning.

Limitations. Although CoAxNN shows the advantages of combining staging-based with pruning-based approximate strategies for model optimization, there is still room for further improvement. On one hand, the NSGA-III used in GA-based DSE cannot always find the optimal solutions for the goals of increasing accuracy and decreasing latency. We will explore other genetic algorithms such as NPGA [51] for multi-objective optimization. On the other hand, the fixed-rate filter pruning strategy is used in CoAxNN. Prior works [11] demonstrated that different layers have different sensitives for model accuracy. Setting different pruning ratios for different layers can potentially further improve the performance, which will be explored in future studies.

6. Conclusion

In this paper, we proposed an efficient optimization framework, CoAxNN, which effectively combines staging-based with pruning-based approximate strategies for efficient model inference on resource-constrained edge devices. Evaluation with state-of-the-art CNN models demonstrates the effectiveness of CoAxNN, which can significantly improve the performance with trivial accuracy loss. We plan to integrate more model approximate strategies into CoAxNN in future work.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by the National Key R&D Program of China (2021ZD0110101), the National Natural Science Foundation of China (62232015, 62302479), the China Postdoctoral Science Foundation (2023M733566), and the CCF-Baidu Open Fund, China.

References

[1] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[2] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI), 2018, pp. 2234–2240.

[3] S.K. Esser, J.L. McKinstry, D. Bablani, R. Appuswamy, D.S. Modha, Learned step size quantization, in: International Conference on Learning Representations, 2020.

[4] Y. Guo, A. Yao, Y. Chen, Dynamic network surgery for efficient dnns, in: Advances in Neural Information Processing Systems, Vol. 29, 2016.

[5] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, Vol. 28, 2015.

[6] B. Hassibi, D. Stork, Second order derivatives for network pruning: Optimal brain surgeon, in: Advances in Neural Information Processing Systems, Vol. 5, 1992.

[7] B. Hassibi, D.G. Stork, G.J. Wolff, Optimal brain surgeon and general network pruning, in: IEEE International Conference on Neural Networks, IEEE, 1993, pp. 293–299.

[8] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, Y. Yang, Asymptotic soft filter pruning for deep convolutional neural networks, IEEE Trans. Cybern. 50 (8) (2019) 3594–3604.

[9] G. Li, X. Ma, X. Wang, L. Liu, J. Xue, X. Feng, Fusion-catalyzed pruning for optimizing deep learning on intelligent edge devices, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (11) (2020) 3614–3626.

[10] J.-H. Luo, J. Wu, W. Lin, Thinet: A filter level pruning method for deep neural network compression, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5058–5066.

[11] G. Li, X. Ma, X. Wang, H. Yue, J. Li, L. Liu, X. Feng, J. Xue, Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning, J. Syst. Archit. (2022) 102431.

[12] J. Plochaet, T. Goedemé, Hardware-aware pruning for FPGA deep learning accelerators, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 4481–4489.

[13] X. Zhuang, Y. Ge, B. Zheng, Q. Wang, Adversarial network pruning by filter robustness estimation, in: ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2023, pp. 1–5.

[14] Z. Liu, M. Sun, T. Zhou, G. Huang, T. Darrell, Rethinking the value of network pruning, in: International Conference on Learning Representations (ICLR), 2019.

[15] Y. Li, K. Adamczewski, W. Li, S. Gu, R. Timofte, L. Van Gool, Revisiting random channel pruning for neural network compression, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 191–201.

[16] Y. Li, P. Zhao, G. Yuan, X. Lin, Y. Wang, X. Chen, Pruning-as-search: Efficient neural architecture search via channel pruning and structural reparameterization, in: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, 2022, pp. 3236–3242.

[17] Y. Ding, Y. Wu, C. Huang, S. Tang, F. Wu, Y. Yang, W. Zhu, Y. Zhuang, NAP: Neural architecture search with pruning, Neurocomputing 477 (2022) 85–95.

[18] S. Teerapittayanon, B. McDanel, H.-T. Kung, Branchynet: Fast inference via early exiting from deep neural networks, in: 2016 23rd International Conference on Pattern Recognition (ICPR), IEEE, 2016, pp. 2464–2469.

[19] P. Panda, A. Sengupta, K. Roy, Conditional deep learning for energy-efficient and enhanced pattern recognition, in: 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2016, pp. 475–480.

[20] Y. Yang, D. Liu, H. Fang, Y.-X. Huang, Y. Sun, Z.-Y. Zhang, Once for all skip: efficient adaptive deep neural networks, in: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2022, pp. 568–571.

[21] B. Fang, X. Zeng, F. Zhang, H. Xu, M. Zhang, FlexDNN: Input-adaptive on-device deep learning for efficient mobile vision, in: 2020 IEEE/ACM Symposium on Edge Computing (SEC), IEEE, 2020, pp. 84–95.

[22] Y. Wang, J. Shen, T.-K. Hu, P. Xu, T. Nguyen, R. Baraniuk, Z. Wang, Y. Lin, Dual dynamic inference: Enabling more efficient, adaptive, and controllable deep inference, IEEE J. Sel. Top. Sign. Proces. 14 (4) (2020) 623–633.

[23] M. Figurnov, M.D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, R. Salakhutdinov, Spatially adaptive computation time for residual networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1039–1048.

[24] N.K. Jayakodi, A. Chatterjee, W. Choi, J.R. Doppa, P.P. Pande, Trading-off accuracy and energy of deep inference on embedded systems: A co-design approach, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37 (11) (2018) 2881–2893.

[25] Z. Liang, Y. Zhou, Dispense mode for inference to accelerate branchynet, in: 2022 IEEE International Conference on Image Processing (ICIP), IEEE, 2022, pp. 1246–1250.

[26] J. Jo, G. Kim, S. Kim, J. Park, LoCoExNet: Low-cost early exit network for energy efficient CNN accelerator design, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2023).

[27] K. Park, C. Oh, Y. Yi, Bpnet: branch-pruned conditional neural network for systematic time-accuracy tradeoff, in: 2020 57th ACM/IEEE Design Automation Conference (DAC), IEEE, 2020, pp. 1–6.

[28] G. Park, Y. Yi, Condnas: neural architecture search for conditional CNNs, Electronics 11 (7) (2022) 1101.

[29] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, S. Han, Amc: Automl for model compression and acceleration on mobile devices, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 784–800.

[30] Y. Qian, Z. He, Y. Wang, B. Wang, X. Ling, Z. Gu, H. Wang, S. Zeng, W. Swaileh, Hierarchical threshold pruning based on uniform response criterion, IEEE Trans. Neural Netw. Learn. Syst. (2023).

[31] K. Wang, D. Zhang, Y. Li, R. Zhang, L. Lin, Cost-effective active learning for deep image classification, IEEE Trans. Circuits Syst. Video Technol. 27 (12) (2016) 2591–2600.

[32] S. Anwar, K. Hwang, W. Sung, Structured pruning of deep convolutional neural networks, ACM J. Emerg. Technol. Comput. Syst. (JETC) 13 (3) (2017) 1–18.

[33] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications To Biology, Control, and Artificial Intelligence, MIT Press, 1992.

[34] A. Mohammadi, H. Asadi, S. Mohamed, K. Nelson, S. Nahavandi, OpenGA, a C++ genetic algorithm library, in: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2017, pp. 2051–2056.

[35] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints, IEEE Trans. Evol. Comput. 18 (4) (2013) 577–601.

[36] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images, 2009.

[37] L.N. Darlow, E.J. Crowley, A. Antoniou, A.J. Storkey, Cinic-10 is not imagenet or cifar-10, 2018, arXiv preprint arXiv:1810.03505.

[38] L. Cai, Z. An, C. Yang, Y. Xu, Softer pruning, incremental regularization, in: 2020 25th International Conference on Pattern Recognition (ICPR), IEEE, 2021, pp. 224–230.

[39] X. Dong, J. Huang, Y. Yang, S. Yan, More is less: A more complicated network with less inference complexity, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5840–5848.

[40] Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, Filter pruning via geometric median for deep convolutional neural networks acceleration, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4340–4349.

[41] X. Dong, Y. Yang, Network pruning via transformable architecture search, Adv. Neural Inf. Process. Syst. 32 (2019).

[42] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1389–1397.

[43] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, D. Doermann, Towards optimal structured cnn pruning via generative adversarial learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 2790–2799.

[44] L. Cai, Z. An, C. Yang, Y. Xu, Soft and hard filter pruning via dimension reduction, in: 2021 International Joint Conference on Neural Networks (IJCNN), IEEE, 2021, pp. 1–8.

[45] X. Yang, H. Lu, H. Shuai, X.-T. Yuan, Pruning convolutional neural networks via stochastic gradient hard thresholding, in: Pattern Recognition and Computer Vision: Second Chinese Conference, PRCV 2019, Xi'an, China, November 8–11, 2019, Proceedings, Part I, Springer, 2019, pp. 373–385.

[46] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252.

[47] Y. Chen, X. Wen, Y. Zhang, Q. He, FPC: Filter pruning via the contribution of output feature map for deep convolutional neural networks acceleration, Knowl.-Based Syst. 238 (2022) 107876.

[48] Y. Chen, X. Wen, Y. Zhang, W. Shi, CCPrune: Collaborative channel pruning for learning compact convolutional networks, Neurocomputing 451 (2021) 35–45.

[49] X. Chen, H. Ma, J. Wan, B. Li, T. Xia, Multi-view 3d object detection network for autonomous driving, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1907–1915.

[50] G. Hinton, O. Vinyals, J. Dean, et al., Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531 2 (7) (2015).

[51] J. Horn, N. Nafpliotis, D.E. Goldberg, Multiobjective optimization using the niched Pareto genetic algorithm, 1993.

Guangli Li received his B.S. and M.S. degrees from the College of Computer Science and Technology, Jilin University, Changchun, China, in 2015 and 2018, and received his Ph.D. degree from the University of Chinese Academy of Sciences, Beijing, China, in 2022. He is currently an Assistant Professor at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include deep learning compilers and neural network compression.

Xiu Ma received her B.S. degree in network and information security from the College of Computer Science and Technology, Jilin University, Changchun, China, in 2016. She is currently a Ph.D. student in computer software and theory of the College of Computer Science and Technology, Jilin University, Changchun, China. Her research interests include programming systems and computational intelligence.

Qiuchu Yu received his B.S. degree from the University of Chinese Academy of Sciences, Beijing, China, in 2023. He is currently pursuing his M.S. degree at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His major research interests include deep learning frameworks and compilers.

Lei Liu is a doctoral supervisor in the College of Computer Science and Technology, Jilin University, Changchun, China. He received his M.S. degree in computer science from Jilin University, Changchun, China, in 1985. The central themes of his research are programming language and its realization technology, software security and cloud computing, the semantic web and ontology engineering, knowledge representation and reasoning, etc.

Huaxiao Liu is an assistant professor in the College of Computer Science and Technology, Jilin University, Changchun, China. He received his Ph.D. in computer science from Jilin University, Changchun, China, in 2013. The central theme of his research is improving software quality, and his recent research concerns the software requirements engineering, software cybernetics and formal methods of software development. More specifically, he develops techniques to verify aspect-oriented requirements model based on ontology.

Xueying Wang received her B.E. degree in software engineering from Northeast Normal University, Changchun, China, in 2017. She is currently pursuing her Ph.D. degree at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. Her major research interests include deep learning, code generation and optimization, and computer architecture.