CEModule: A Computation Efficient Module for Lightweight Convolutional Neural Networks

Yu Liang, Maozhen Li, Changjun Jiang, and Guanjun Liu, *Senior Member, IEEE*

***Abstract*—Lightweight convolutional neural networks (CNNs) rely heavily on the design of lightweight convolutional modules (LCMs). For a LCM, lightweight design based on repetitive feature maps (LoR) is currently one of the most effective approaches. A LoR mainly involves an extraction of feature maps from convolutional layers (CE) and feature map regeneration through cheap operations (RO). However, existing LoR approaches carry out lightweight improvements only from the aspect of RO, but ignore the problems of poor generalization, low stability and high computation workload incurred in the CE part. To alleviate these problems, this paper introduces the concept of key features from a CNN model interpretation perspective. Subsequently, it presents a novel LCM, namely CEModule, focusing on the CE part. CEModule increases the number of key features to maintain a high level of accuracy in classification. In the meantime, CEModule employs a group convolution strategy to reduce floating-point operations (FLOPs) incurred in the training process. Finally, this paper brings forth a dynamic adaptation algorithm (α-DAM) to enhance generalization of CEModule enabled lightweight CNN models including the developed CENet in dealing with datasets of different scales. Comparing with the state-of-the-art results, CEModule reduces FLOPs by up to 54% on CIFAR-10 while maintaining a similar level of accuracy in classification. On ImageNet, CENet increases accuracy by 1.2% following the same FLOPs and training strategies.**

***Index Terms*—Lightweight, convolutional neural networks, neural network interpretation, hyper-parameter optimization, feature map regeneration, automated machine learning.**

I. Introduction

THE past few years have seen a growing interest in employing deep learning techniques such as convolutional neural networks (CNNs) in mobile and embedded systems. However, a general development trend of deep neural networks is to build deeper and more complex networks [1]–[4]. These complex models require a large number of parameters and floating-point operations (FLOPs) while satisfying a certain level of accuracy in classification tasks, which is not conducive to deploying the corresponding CNN models on mobile and embedded terminals. How to further reduce the computational complexity of CNNs has gradually become one of the important issues that need to be addressed during model design. Although a number of works such as pruning [5]–[7] and knowledge distillation [8]–[10] have been proposed in making CNN models lightweight, they mainly follow a post-hoc approach to optimizing a model structure only after the training process is completed. In 2016, the Google team first proposed the concept of lightweight CNN networks [11], making it possible to directly train deep neural networks on mobile terminals.

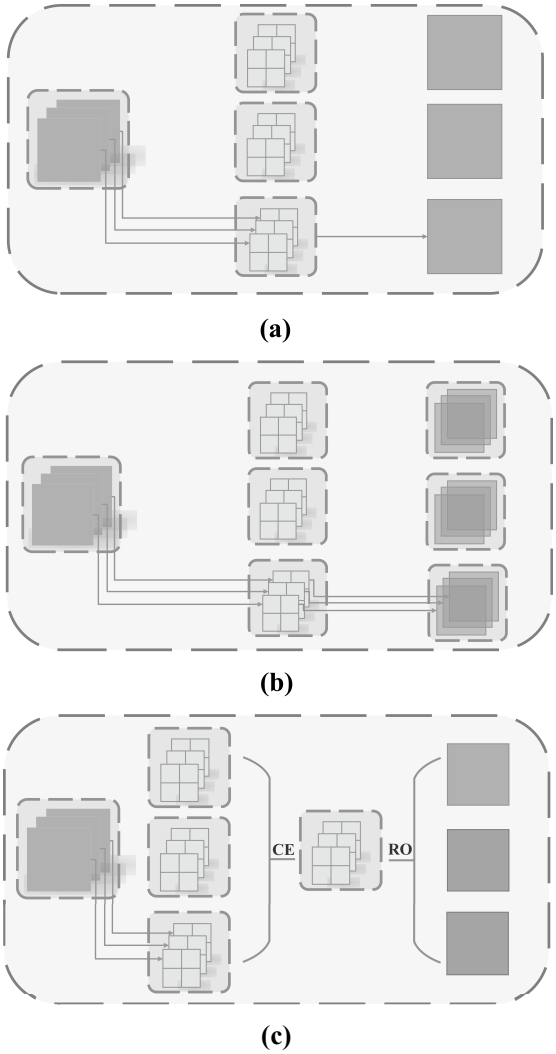


Fig. 1. Types of convolution: (a) normal convolution, (b) depthwise convolution, (c) LoR convolution.

The basic idea of a lightweight CNN is to reduce the computational complexity of convolution operations. Therefore, the key to a lightweight CNN is the design of lightweight convolutional modules (LCMs). Existing LCMs mainly follow depthwise convolution and lightweight design based on

Yu Liang, Changjun Jiang, Guanjun Liu are with the Key Laboratory of Ministry of Education on Embedded System and Service Computing and also with the Department of Computer Science and Technology, Tongji University, Shanghai 201804, China(1910660@tongji.edu.cn; cjjiang@tongji.edu.cn; liuguanjun@tongji.edu.cn).

Maozhen Li is with the Department of Electronic and Electrical Engineering, Brunel University London, Uxbridge, UB8 3PH, UK. He is also associated with the Key Laboratory at Tongji University (maozhen.li@brunel.ac.uk).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI10.1109/TNNLS.2021.3133127, IEEE Transactions on Neural Networks and Learning Systems

repetitive feature maps (LoR). Depthwise convolution is a widely used lightweight operation of LCMs which exists in most popular lightweight CNNs [11]–[15]. As shown in Fig. 1(a), each kernel in a normal convolution operates on all the input channels simultaneously. Different from the normal convolution, each kernel in a depthwise convolution operates only on one channel as shown in Fig. 1(b).

LoR is a new idea introduced in 2020, and the most recent work is GhostNet [16] which represents one of the best lightweight works. As shown in Fig. 1(c), a LoR mainly involves an extraction of feature maps from convolutional layers (CE) and feature map regeneration through cheap operations (RO). The main function of CE is to extract feature maps from the original convolutional layer to provide a basis for RO operations, which make the convolutional layer lightweight through low computation cost operations such as linear transformations [17].

Although LoR has achieved encouraging results, there is still room for improvement. Cheap linear transformations have become an optimal solution for RO. However, current work on CE in GhostModule only extracts a small number of feature maps with an aim to achieve low FLOPs but a high level of accuracy cannot be always maintained which leads to poor stability. In addition, existing lightweight CNN models do not effectively scale in dealing with new datasets resulting in poor generalizability in application scenarios.

To address these limitations, this paper introduces the concept of key features from a CNN model interpretation perspective which forms the basis of the proposed LoR. A relationship between key features and feature maps is established with an aim to extract sufficient key features to maintain a high level of accuracy. In the meantime, a lightweight strategy is employed to reduce FLOPs in computation. Considering the width multiplier (α) [11] that has an impact on the overall computation complexity of a CNN model, we scale a CNN model through a dynamic adaptation of the value of α. Specifically, the main contributions of the paper are summarized as follows:

(1) Based on the concept of key features in feature extraction, this paper presents a CNN model performance approximation approach. The approximation turns an unexplainable feature extraction process of a black box CNN model into an explainable process in the form of key features which can be analyzed through an underlying interpretation method.
(2) Based on the established relationship between key features and feature maps, this paper presents a LCM module (i.e. CEModule). The novelty of CEModule lies in two-fold. On one hand, CEModule employs sufficient feature maps with an aim to extract a large number of key features to achieve a high level of accuracy. On the other hand, it builds on a group convolution strategy to reduce FLOPs in computation. A new lightweight CNN model namely CENet is subsequently developed to potentially meet the needs in deploying deep neural networks on mobile and embedded terminals.
(3) To improve model generalizability, this paper introduces α-DAM, a method that dynamically adapts the value of the hyperparameter α to make a CEModule enabled CNN model scalable to the size of an input dataset.
(4) Comparing with the state-of-the-art results, CEModule reduces FLOPs by up to 54% on CIFAR-10 dataset while maintaining a similar level of accuracy. On ImageNet dataset, CENet increases the accuracy by 1.2% following the same FLOPs and training strategies.

The remainder of this work is organized as follows. Section II reviews related work on lightweight CNN models. Section III introduces the concept of key features based on which the specific structure of CEModule is presented. It also introduces α-DAM to dynamically adapt the scale of a CNN model based on the size of an input dataset. Section IV conducts comprehensive experiments and validates the performance of both CEModule and CENet in comparison with state-of-the-art results. Section V concludes the paper and points out some future work.

II. Related work

This section reviews related work from the aspects of lightweight CNN module design and hyperparameter optimization.

A. Lightweight CNN Networks

A lightweight CNN network refers to a deep neural network that can be deployed on mobile and embedded terminals. The first work on lightweight networks was MobileNet_V1 [11], which is based on depthwise separable convolution, a way of decomposing standard convolution into depthwise convolution and pointwise convolution. This decomposition can effectively reduce the amount of calculations and the size of the model. Compared with MobileNet_V1, MobileNet_V2 [12] introduces two changes, i.e., inverted residuals and linear bottlenecks. These two new changes greatly improve the accuracy of the model. MobileNet_V3 [13] further employs automated machine learning [18], [19] to search for better lightweight models. Xception [20] mainly draws on depthwise separable convolution to replace the original convolution operation in Inception_V3 [21]. ShuffleNet_V1 [14] proposes a ShuffleNet unit that includes two operations: pointwise group convolution and channel shuffle. ShuffleNet_V2 [15] further considers the actual speed of the target hardware in the compact model design. Although these works have their characteristics, the main ideas on lightweight CNN networks are inseparable from the depthwise convolution operation.

Unlike the aforementioned works that mainly rely on depthwise convolution operations, GhostNet is the latest lightweight CNN structure presented at CVPR 2020. GhostNet provides a LCM module, namely GhostModule, with an aim to generate feature maps through cheap linear transformation operations. GhostModule selects a portion of the original feature maps and based on which it applies a series of linear transformations to regenerate Ghost feature maps at a low cost in computation. However, GhostModule is not stable in maintaining a high level of accuracy due to the quality of the selected feature maps varies in terms of the number of key features.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI10.1109/TNNLS.2021.3133127, IEEE Transactions on Neural Networks and Learning Systems

B. Hyper-parameter Optimization

Hyper-parameter optimization (HPO) refers to a type of methods that does not rely on manual parameter adjustments [22]–[24] instead employs certain algorithms to find an optimal or a near optimal hyperparameter setting in deep learning. The essence to hyperparameter optimization methods is to generate multiple sets of hyperparameters, and adjust them according to the obtained evaluation indicators. Traditional hyper-parameter optimization methods are mainly based on brute force search and heuristic search. With the development of deep learning, these hyper-parameter optimization methods are gradually integrated into automated machine learning (AutoML) [19].

AutoML has powerful generalization and learning functions for a given dataset and task. For AutoML, network architecture search (NAS) [18], [25] is the most important part. Generally speaking, NAS first defines a search space, then finds a candidate network structure through search strategies. In addition to searching for a pure network structure, many recent works [26]–[28] combine NAS and HPO to form a better network.

The width multiplier α is a key hyperparameter in the field of lightweight models and it has an impact on the overall computation complexity of a CNN model. However, it has been a challenge in dynamically setting the value of α. The existing NAS and HPO methods mainly use the brute force search strategy in adaptation of α which is undoubtedly not an efficient way. To solve this problem, the proposed α-DAM establishes a heuristic relationship between an input dataset and a CNN model to adapt α to make a CNN model scalable.

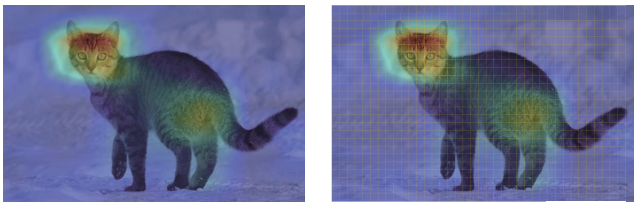
A lightweight CNN model relies heavily on the design of a lightweight CNN module. However, in view of different application backgrounds and resource constraints, a lightweight CNN model can be further optimized through hyperparameter optimization methods. As a result, a combination of hyperparameter optimization with lightweight CNN module design would generate an optimal solution for lightweight CNN models.

III. Methodology

In this section, we first introduce the concept of key features from the perspective of CNN model interpretation based on which we approximate the performance of a CNN model. By analyzing the relationship between feature maps and key features, we then present the specific structure of CEModule. Finally, α-DAM is introduced and the flexibility in incorporating α-DAM in a CNN model is discussed.

A. Model Approximation

Key features are valuable contents learned by CNN during a training process. Due to the black-box nature of CNN in training, key features cannot be directly expressed through rigorous mathematical reasoning. However, as we have reviewed in [29], many local interpretation methods [30]–[33] can indirectly show the valuable learning results of a CNN. A local interpretation method checks individual predictions trying to figure out how a CNN model makes a decision [34], [35]. Following [32], we can learn that feature maps play a role in extracting image features. The works presented in [30], [31], [33] calculate the scores of feature maps to explore what a CNN has learned. Fig. 2 shows an output generated by Grad-CAM [33], a local interpretation method. The output image is reproduced from a pre-trained ResNet [36]. The test image is taken from Wikimedia Commons, a global resource sharing website. The red part of a heat map on Fig. 2(a) represents the most valuable areas that the pre-trained ResNet has learned.



(a) (b)

Fig. 2. A reproduced CNN model interpretation output: (a) key feature areas, (b) quantified key features in squares.

Although a local interpretation method can show the key areas learned during a CNN training process, the interpretation is usually represented by a fuzzy area, which is not conducive to a further analysis of the training process in CNN. To depict the role of feature maps in a training process more clearly, interpretation can be visualized into multiple areas such as small squares based on approximation rules as shown in Fig. 2(b), and each area approximately represents a feature. Among these features, a feature that covers a critical area is referred to as a key feature.

For a target category c in the training process, each feature map extracts part of the features in the input image. The feature maps that contain key features can get a much higher score than other feature maps without key features. The higher the cumulative score of feature maps, the higher the probability that the classification result is c. Therefore, the feature extraction process of a CNN model can be approximated as a process of extracting m key features as shown in Eq. (1).

$$\Psi(I) \approx \Psi_r(\sum_{x=1}^{m} x) \tag{1}$$

where

- I is an input.
- r is an approximation rule.
- Ψ represents the feature extraction process of a CNN model.
- m is the number of key features.
- x is a key feature.

The core of model approximation lies in the selection of approximation rules which can be set independently according to different needs. The smaller the feature size obtained based on approximation rules is, the more accurate the model approximation produces. We mainly introduce two approximation rules in this work which are based on small squares and image segmentation respectively. The approximation rule based on image segmentation refers to the use of image segmentation technology to determine features. The approximation rule

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI10.1109/TNNLS.2021.3133127, IEEE Transactions on Neural Networks and Learning Systems

based on small squares is to visualize interpretation into a large number of small square pixel matrices to improve the flexibility of approximate feature representation.

As shown in Fig. 3(a), an input image is transformed into a number of feature maps through a CNN where n_t is the total number of feature maps in the entire model, and each feature map can extract random features. For a well performed CNN, the ideal situation would be to have all the key features extracted from the generated feature maps.

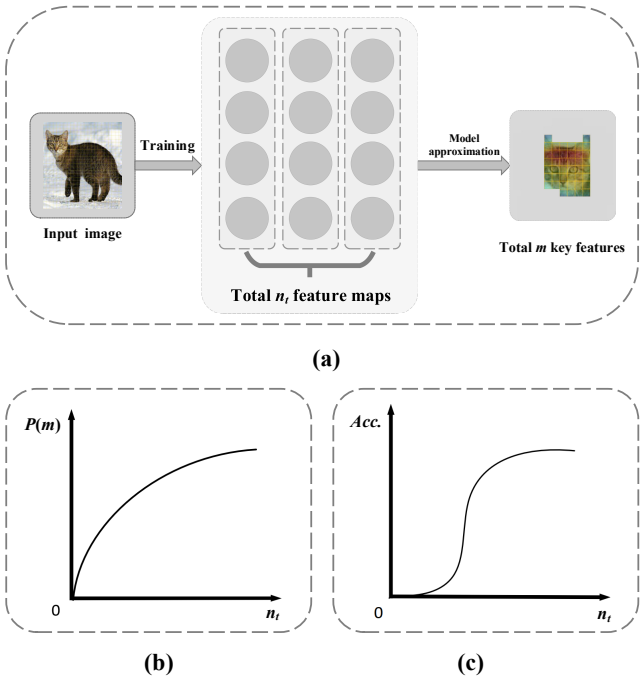


Fig. 3. The relationship between feature maps and key features.

Fig. 3(b) and Fig. 3(c) show the impact of feature maps on key features and accuracy respectively, where $P(m)$ in Fig. 3(b) represents the probability of obtaining all the key features. If a CNN is expected to learn all the key features, it means that n_t must be large enough. As shown in Fig. 3(b), an increase of feature maps pushes $P(m)$ higher gradually. Similarly, as $P(m)$ increases, the corresponding accuracy level will go up. However, as shown in Fig. 3(c), when the number of feature maps initially increases, the accuracy of a CNN model increases at an accelerated rate. As the model learns a sufficient number of key features, the accuracy level will gradually increase until the CNN model is saturated. It is worth noting that the relationships depicted in Fig. 3(b) and Fig. 3(c) are in line with the experimental results presented in Section IV-B.

To summarize, the number of feature maps directly determines the number of key features to be extracted for model performance approximation. Therefore, fewer feature maps would have a higher probability of losing some key features leading to a low accuracy level of a CNN model.

B. The Structure of CEModule

As a representative work of LoR, GhostNet observes that feature maps have high repetitiveness which is reflected through a visualization of multiple feature maps. Therefore, the GhostModule in GhostNet performs a cheap linear transformation on these repetitive feature maps which reduces the overall computation complexity. GhostModule mainly contains CE and RO, and these two operations are in a progressive relationship. A CE operation extracts a number of feature maps in a normal convolutional layer as the basis of RO. We set the number of feature maps required for CE to n' and the number of feature maps in the normal convolutional layer to n where n is usually divisible by n'. In order to keep the spatial size of the feature maps consistent with that of the normal convolutional layer, GhostModule supplements the missing $(n - n')$ feature maps through RO operations.

Following the work of GhostNet, we formally define the required FLOPs of GhostModule in computation.
Let

- k be the size of a convolution kernel.
- c be the number of input channels, i.e. the number of intput feature maps.
- d be the kernel size of a linear transformation in RO which is usually equal to k.
- $H \times W$ be the size of a new feature map generated by a convolution layer.

For convolutional layers with bias, if a multiplication-accumulation operation is counted as two FLOPs, the total FLOPs incurred by GhostModule can be formally expressed as:

$$\begin{aligned}\Phi_{ghost} &= \Phi^{'}_{ghost} + \Phi^{''}_{ghost} \\ &= 2 \cdot c \cdot H \cdot W \cdot n' \cdot k^2 + 2 \cdot (n - n') \cdot H \cdot W \cdot d^2\end{aligned} \tag{2}$$

where

- Φ_{ghost} represents the FLOPs of GhostModule in computation.
- Φ'_{ghost} represents the consumed FLOPs on CE operations.
- Φ''_{ghost} represents the consumed FLOPs on RO operations.
- (n-n') is the number of feature maps to be generated through RO.

It should be pointed out that the ideal situation considered by GhostNet may not always exist. GhostModule only extracts n' feature maps in the process of CE, and n' could be much smaller than n. Therefore, some key features may not be obtained. On one hand, it is not the case that sufficient key features can always be obtained from feature maps generated through linear transformations as GhostModule does. On the other hand, even if a single GhostModule operation has a low probability in losing key features, when considering all the convolution operations in a CNN that are replaced with GhostModule operations, the probability of the entire CNN in losing key features would increase significantly, which leads to an unstable CNN in maintaining a high level of accuracy.

RO performs a linear transformation on the output of CE, and RO will also change accordingly when CE changes. In view of the limitations of GhostModule, we believe that

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI10.1109/TNNLS.2021.3133127, IEEE Transactions on Neural Networks and Learning Systems

increasing the number of feature maps required by CE is the most direct and effective way to achieve a higher probability in extracting all the key features to enhance CNN stability in maintaining a high level of accuracy. Inevitably, the increase of the number of feature maps will directly cause high FLOPs. The key of CEModule is to extract sufficient key features through the process of CE but without increasing FLOPs.

To generate more feature maps without incurring a higher cost on FLOPs, CEModule employs group convolution, an effective lightweight strategy that has been widely used in CNNs. Group convolution can effectively reduce the amount of calculations of CNNs [4], [14], [15], [20]. However, GhostModule cannot directly incorporate group convolution. Unlike traditional convolution that can be directly transformed to group convolution, the RO part in GhostModule performs a linear transformation which cannot directly incorporate group convolution. In addition, adding the number of feature maps in the CE part will also increase the FLOPs of the RO part in GhostModule. In order to reduce the FLOPs required but in the meantime to increase the number of convolution kernels, CEModule incorporates group convolution only in the CE part. Fig. 4 shows the difference between GhostModule and CEModule. As shown in Fig. 4(b), CEModule first expands the number of feature maps in the original convolutional layer with an expansion ratio of g'. CEModule then performs a group convolution operation on the extracted feature maps to reduce the amount of calculation, and g is the number of groups in CEModule. Therefore, the FLOPs incurred by CEModule can be expressed as:

$$\begin{aligned}\Phi_{ce} &= \Phi_{ce}^{'} + \Phi_{ce}^{''} \\ &= \frac{2 \cdot n' \cdot g^{'}}{g} \cdot H \cdot W \cdot c \cdot k^2 + 2 \cdot g' \cdot (n - n') \cdot H \cdot W \cdot d^2\end{aligned} \tag{3}$$

where

- Φ_{ce} represents the FLOPs of CEModule.
- $\Phi_{ce}^{'}$ represents the consumed FLOPs on CE operations.
- $\Phi_{ce}^{''}$ represents the consumed FLOPs on RO operations.

The value of c is the number of feature maps in the previous layer which is usually set to 64∼1024 [3], [11]–[13], [16], [36], [37]. Comparing with g, g' and $\frac{n-n'}{n'}$ which are set less than 8, c is a significantly large value.

According to Eq. (2) and Eq. (3), we have:

$$\begin{aligned}\frac{\Phi_{ce}}{\Phi_{ghost}} &= \frac{\frac{n' \cdot g'}{g} \cdot H \cdot W \cdot c \cdot k^2 + g' \cdot (n - n') \cdot H \cdot W \cdot d^2}{n' \cdot H \cdot W \cdot c \cdot k^2 + (n - n') \cdot H \cdot W \cdot d^2} \\ &= \frac{\frac{n' \cdot g'}{g} \cdot c + g' \cdot (n - n')}{n' \cdot c + (n - n')}\end{aligned} \tag{4}$$

where

- $g \in N^*$ and c $\gg g$.
- $g' \in N^*$ and c $\gg g'$.
- $\frac{n-n'}{n'} \in R^+$ and c $\gg \frac{n-n'}{n'}$.

The theoretical value of g only needs to satisfy the condition that $g \in N^*$, and g' is a hyperparameter closely related to g. To replace GhostModule with CEModule without increasing the extra FLOPs, we aim to satisfy $\Phi_{ce} = \Phi_{ghost}$, then the value of g' can be calculated with Eq. (5).

$$g' = \frac{g \cdot c \cdot n' + g \cdot (n - n')}{n' \cdot c + g \cdot (n - n')} < g \tag{5}$$

C. α-DAM

In this section, we present α-DAM, a method that adapts α according to an input image dataset to better initialize a CNN model. The factors that affect α are mainly the sizes of input images.

The key to adapt α is to establish a mapping between the size of an input image and the number of feature maps. For an input image, corner or edge detection methods [38], [39] are normally employed to determine the number of corner points or edges. For feature maps, the most representative layer which is normally the first layer of a CNN model is selected. In fact, for the existing mainstream CNN models such as ResNet, VGGNet [37], MobileNet [11]–[13] and GhostNet, the number of feature maps in all subsequent convolutional layers is proportional to the number of feature maps in the first layer.

Therefore, α-DAM is initialized with a corner (or edge) detection method as well as the number of feature maps in the first layer to establish a mapping relationship. We set the total number of corner points or edges obtained from an individual image as $\gamma(I)$. Considering the size variations of images in an input dataset, we set h as the expected value of $\gamma(I)$:

$$h = \frac{\sum_{i=1}^{w} \gamma(I)}{w} \tag{6}$$

where

- w is the number of images in an input dataset.
- I is an image in the dataset.

We set the number of feature maps in the first layer to h'. To establish the corresponding relationship between h and h', we define a criterion point as $(h,\ h')$ which can be obtained through a process of pre-training an existing model on a specific dataset. After getting multiple criterion points, we can fit a function based on these criterion points through a linear regression as shown in Fig. 5(a). Generally speaking, h is much larger than h' and each black point represents a criterion point.

For a dataset D consisting of $(h,\ h')$ criterion points, the linear regression function aims to generate $f(h) = \eta \cdot h + b$ which satisfies $f(h) \simeq h'$. We can calculate the solution of η and b through the following least square method:

$$(\eta^*, b^*) = \underset{(\eta, b)}{argmin} \sum_{i=1}^{D} (h_i' - \eta \cdot h_i - b)^2 \tag{7}$$

where

- η is the slope rate.
- b is a bias.
- (η^*, b^*) represents the solution of η and b.
- h_i' represents the value of h' at the i-th point in D.
- h_i represents the value of h at the i-th point in D.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI10.1109/TNNLS.2021.3133127, IEEE Transactions on Neural Networks and Learning Systems

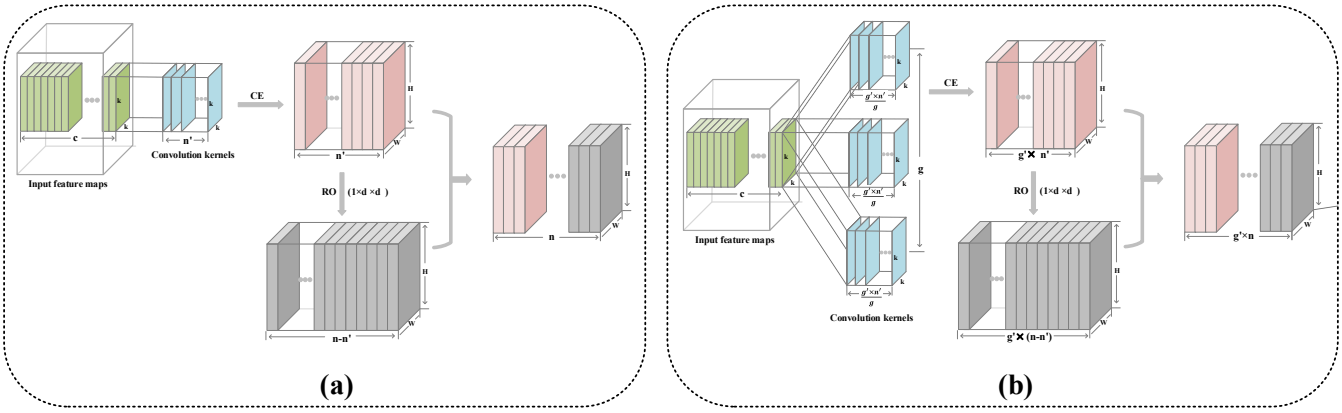


Fig. 4. The structure difference between GhostModule and CEModule: (a) GhostModule, (b) CEModule.

The core idea of α-DAM is to generate the corresponding regression equations based on criterion points. However, if the model size is too small, the model will cause under-fitting. To solve this problem, we set a low bound h_{min} for the regression equation to avoid under-fitting as shown in Fig. 5(b).

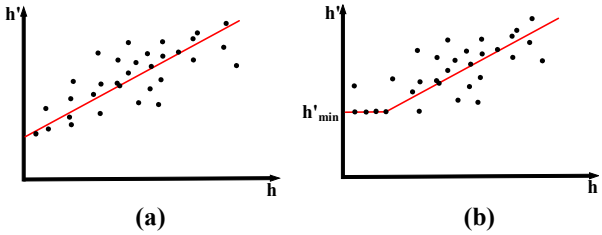


Fig. 5. The relationship between h and h'.

The mapping strategy is reflected with a ReLU function in Eq. (8).

$$h' = max(h'_{min}, f(h)) \tag{8}$$

where

- h'_{min} is the minimum value of h'.

α-DAM is flexible when applying to the underlying CNN models. The value of α is adapted by $\frac{h'}{h'_{org}}$ where h'_{org} represents the number of feature maps in the first layer of the CNN model. And the low bound is usually adapted based on the specific type of CNN.

IV. Experimental Results

This section validates the performance of both CEModule and CENet through a set of experiments. It also analyzes the effectiveness of α-DAM.

A. Exprerimental Setups

This section presents the experimental settings. First it briefly introduces three widely used datasets for the experiments conducted in this work.

1) Datasets: The MNIST dataset [40] has 60,000 training images and 10,000 test images. Each sample is a 28 × 28 pixel grayscale handwritten digital image. The CIFAR-10 dataset [41] has 10 categories composed of 60,000 32×32 color images, each of which has 6,000 images. The ImageNet dataset [42] has more than 14 million images, covering more than 20,000 categories.

2) Performance Metrics: We evaluated the performance from the aspects of accuracy and FLOPs. For classification problems, we have true positive (TP), false positive (FP), true negative (TN) and false negative (FN) and the corresponding accuracy can be calculated with Eq. (9):

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{9}$$

In CNN, the convolution operations account for the largest portion of the total FLOPs. Therefore, we take a convolutional operation as an example to depict the FLOPs incurred.

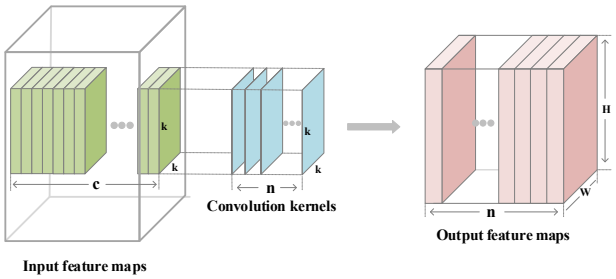


Fig. 6. A normal convolution operation.

As shown in Fig. 6 where the parameters are defined in Section III-B, the total FLOPs incurred in a convolutional operation equal to the FLOPs spent on a single pixel multiplied by the number of pixels in the output feature maps. For a single pixel, the convolution operation performs $(k \cdot k \cdot c)$ multiplication and $(k \cdot k \cdot c - 1)$ accumulation operations. The total number of pixels in the output feature maps is $(H \cdot W \cdot n)$. Therefore, the FLOPs spent on a convolution operation can be expressed as follows without considering bias:

$$(2 \cdot k \cdot k \cdot c - 1) \cdot (H \cdot W \cdot n) \tag{10}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI10.1109/TNNLS.2021.3133127, IEEE Transactions on Neural Networks and Learning Systems

When bias is considered, the FLOPs incurred in a convolutional operation can be expressed as:

$$(2 \cdot k \cdot k \cdot c) \cdot (H \cdot W \cdot n) \tag{11}$$

3) Parameter Settings: All the experiments were conducted using the PyTorch [43] framework in Python 3.6 and a single NVIDIA Tesla V100 GPU (16G RAM). To ensure fairness in comparison, all the experiments were conducted following the same training settings. CEModule and GhostModule were compared on VGGNet [37] and ResNet [36] respectively. CENet was evaluated in comparison with another 6 CNN models using the official training file of PyTorch [44] as a basis. The total number of epochs was set to 120, the weight decay was $4e^{-5}$, the batch size was 512 due to the limit of the GPU RAM and the learning rate was 0.2.

B. Basic Validations

To validate the relationship of feature maps with accuracy and key features as plotted in Fig. 3(b) and Fig. 3(c) respectively, we constructed TestNet (TN), a testing CNN model whose structure is shown in Table I. By varying the number of feature maps, we modified TestNet into 8 models which are represented by TN-x (x is between 1 and 8). For each TN-x model, we set the number of feature maps in the first layer h' to 2^x.

TABLE I
THE STRUCTURE OF TESTNET.

Layer	Input	Output	Kernel size	Pooling
Conv1	3	h'	5	Maxpooling
Conv2	h'	h'	5	
Fc1	h'	120	-	-
Fc2	120	84	-	-
Fc3	84	10	-	-

The training loss reflects the ability of a CNN model in extraction of key features. When a CNN model extracts more key features, the loss value will become lower. To validate the relationship of key features and feature maps as shown in Fig. 3(b), Fig. 7 presents a series of observations on the training loss of TN-x under different training epochs. It can be observed that as the number of feature maps increases, the model loss gradually becomes lower. It is worth noting that when the number of feature maps reaches 64 (i.e. the case of x being 6), increasing the number of feature maps does not have much impact on training loss.

Table II shows that accuracy increases at an accelerated rate initially with more feature maps. However, as the model learns a sufficient number of key features, the accuracy level gradually increases until the model is saturated which confirms Fig. 3(c).

C. CEModule Validation

In this section, we first introduce the specific strategies in applying CEModule to both VGGNet and ResNet. It is worth noting that all the following experiments were conducted on the CIFAR10 dataset.

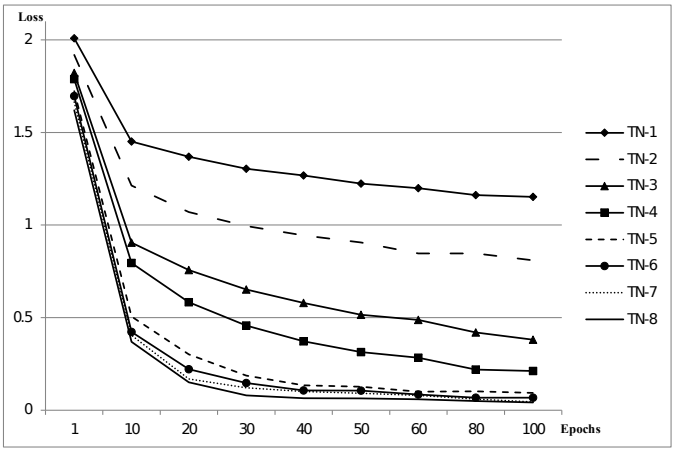


Fig. 7. Training loss of TN-x models.

TABLE II
ACCURACY OF TN-x MODELS.

Models	h'	Acc. (Epoch=1)	Acc. (Epoch=5)	Acc. (Epoch=10)	Acc. (Epoch=20)
TN-1	2	17.2%	44.8%	48.6%	48.8%
TN-2	4	21.4%	46.9%	57.0%	56.9%
TN-3	8	31.6%	52.7%	64.3%	62.7%
TN-4	16	34.7%	53.5%	62.7%	63.5%
TN-5	32	37.7%	58.3%	68.1%	68.3%
TN-6	64	38.3%	60.3%	68.8%	70.3%
TN-7	128	39.8%	61.9%	71.6%	71.9%
TN-8	256	39.5%	61.7%	72.0%	71.7%

1) BNlayer or No_BNlayer: CEModule replaces the normal convolution layers in a CNN model to make the model lightweight. However, compared to a traditional convolutional layer, CE and RO need to be calculated separately during the training process in CEModule which could cause under-fitting due to a small number of parameters in the CE part. To reduce over-fitting caused by lightweight operations, CEModule employs a batch normalization (BN) layer [45] as shown in Fig. 8 and the results are shown in Table III. In CEModule, batch normalization not only alleviates the possible gradient dispersion phenomenon, it also speeds up training and convergence. Here C refers to the concat operation which is mainly used to merge channels. Through the concat operation, we can superimpose the number of feature maps in a horizontal or vertical space.

In Table III, CE_ResNet56 is a lightweight ResNet56 model using CEModule. *No_BNlayer* refers to the CEModule without a BN layer. It can be observed that having a BN layer increases the accuracy by 8.68% in CE_ResNet56.

2) All_CEModule or First_Layer Retaining: CEModule also retains the first convolutional layer for performance improvement. As described in Section III-B, a LoR method like CEModule is heavily dependent on the number of key features extracted in the CE part. When the first convolutional layer is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI10.1109/TNNLS.2021.3133127, IEEE Transactions on Neural Networks and Learning Systems

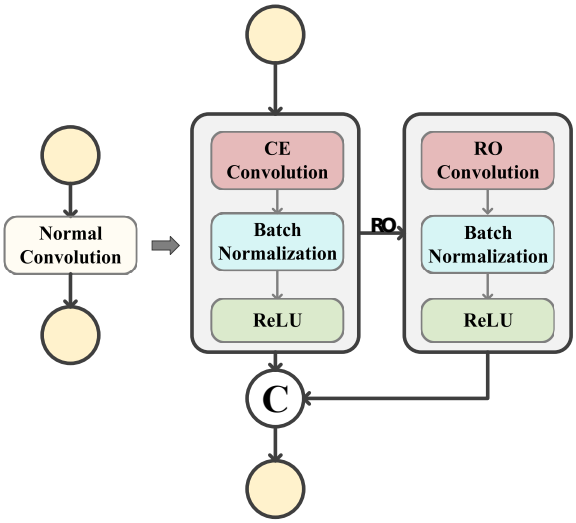


Fig. 8. The implementation of CEModule.

TABLE III
ACCURACY OF CE_RESNET56 WITH BN LAYER.

Models	Acc.
CE_ResNet56(No_BNlayer)	82.39 %
CE_ResNet56	**91.07%**

retained, CEModule obtains more information from multiple feature maps, thereby reducing the probability of losing key features and increasing the level of accuracy. Due to the characteristics of LoR methods, retaining other convolutional layers cannot have the same effect.

In a set of experiments as shown in Table IV, we replaced all the convolutional layers in an original model with CEModule, and subsequently retained the first convolutional layer.

TABLE IV
THE IMPACT OF RETAINING THE FIRST-LAYER ON BOTH RESNET56 AND VGG16.

Models	Acc.
CE_ResNet56(All_CEModule)	90.46 %
CE_ResNet56(First_Layer)	**91.07%**
CE_VGG16(All_CEModule)	91.50 %
CE_VGG16(First_layer)	**92.37%**

CE_VGG16 is a lightweight VGG16 model using CEModule. *All_CEModule* replaces all the convolutional layers with CEModule. *First_Layer* retains the first convolutional layer. It can be observed from Table IV that retaining the first layer increases the accuracy by 0.61% on CE_ResNet56 and 0.87% on CE_VGG16 respectively.

3) The Size of Convolution Kernel: The size of a convolutional kernel also affects FLOPs and accuracy of a CEModule enabled CNN model. Following the views of [46], [47], a small kernel size like 3 is usually considered as a reasonable choice for a normal convolutional layer. We conducted a number of tests on the CE_VGG16 with the size of a kernel setting to 3, 5, and 7 respectively. As shown in Table V, the CE_VGG16 using a kernel size of 3 reduces nearly 70% of FLOPs in comparison with the original VGG16, while maintaining a similar level of accuracy. As a result, CEModule does not have an additional impact on the setting of kernel size.

TABLE V
THE IMPACT OF KERNEL SIZE ON CE_VGG16.

Models	FLOPs	Acc.
VGG16	333.35M	92.82%
CE_VGG16(Kernel_Size=3)	**101.15M**	**92.37%**
CE_VGG16(Kernel_Size=5)	242.71M	89.49%
CE_VGG16(Kernel_Size=7)	455.05M	-

4) Groups and Channel Shuffle: The number of groups g is the most important parameter for CEModule which affects both FLOPs and accuracy. As the g value increases, the learning ability of the model also increases under the same FLOPs. However, as shown in Fig. 3, with an increase of feature maps, the model's ability to extract key features has an upper limit which means the linear increase in the number of groups does not bring up a linear increase in the classification accuracy, but can easily cause over-fitting. On the other hand, a larger value of g also increases the calculations of both g' and Φ''_{ce} when keeping the same FLOPs of CEModule. As a result, the FLOPs of Φ'_{ce} in CEModule decreases which damages the ability of the CE part in feature extraction. Under ~50M FLOPs in CE_ResNet56 and ~100M FLOPs in CE_VGG16, we validated the impact of the number of groups on both CE_VGG16 and CE_ResNet56. It can be observed from Table VI that setting g to 2 increases the accuracy at least by 0.34% on CE_VGG16 and 0.13% on CE_ResNet56 respectively.

In addition to g, the shuffle operation [14] is also an important operation that affects the performance of group convolution. For a normal group convolution layer, shuffle mainly solves the difficulty in channel interaction caused by a large value of g. For CEModule, because channel interaction is enhanced through a linear transformation in the RO part, we do not need to add additional shuffle operations to improve the performance of the module when g is 2. We further tested whether the shuffle operation has a significant effect on CEModule. It can be observed from Table VI that channel shuffle does not have much effect on both CE_VGG16 and CE_ResNet56 when g is set to 2.

TABLE VI
THE IMPACT OF GROUPS.

Models	Acc.
CE_VGG16(g=2)	**92.37%**
CE_VGG16(g=4)	91.87%
CE_VGG16(g=2&shuffle)	92.03%
CE_VGG16(g=4&shuffle)	91.86%
CE_VGG16(g=8&shuffle)	88.68%
CE_ResNet56(g=2)	**91.07%**
CE_ResNet56(g=2&shuffle)	90.94%
CE_ResNet56(g=3&shuffle)	90.30%

Last but not least, a large value of g will increase the com-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI10.1109/TNNLS.2021.3133127, IEEE Transactions on Neural Networks and Learning Systems

putation latency of CEModule. The latency in the lightweight model refers to the time required to perform a training process in single-threaded mode with a batch size of one. As the scale of the model becomes larger, the latency increases accordingly. We conducted multiple latency experiments with different g values (2, 4, 8) under different model scales. Fig. 9 shows the impact of the groups on computation latency, with the case of g being 2 again performing the best.

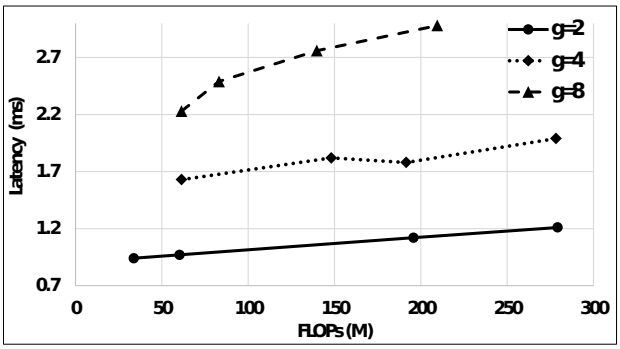


Fig. 9. Computation latency of CEMoule with different groups.

5) Comparison with GhostModule: Based on the best settings of CEModule parameters, we conducted another set of experiments in comparison with GhostModule. First, we incorporated CEModule and GhostModule into VGG16 and ResNet56 models respectively. The comparison results are shown in Table VII.

TABLE VII
A COMPARISON OF CEMODULE WITH GHOSTMODULE.

Models	FLOPs	h'	Acc.
VGG16	333.35M	64	92.82%
Ghost_VGG16	179.01M	64	92.26%
CE_VGG16	**101.15M**	80	**92.37%**
ResNet56	125M	16	93.2%
Ghost_ResNet56	62.28M	16	90.68%
CE_ResNet56	**50.92M**	20	**91.07%**

It can be observed that both GhostModule and CEModule lighten the original VGG16 and ResNet56 models in reducing FLOPs significantly while having a similar level of accuracy. Notably, CE_VGG16 reduces the amount of FLOPs by 54% compared to Ghost_VGG16 and increase the level of accuracy by 0.11%. Compared with Ghost_ResNet56, CE_ResNet56 still achieves higher accuracy but with lower FLOPs.

CEModule has a better generalization ability than GhostModule on small CNN models. For validation, we designed two more lightweight models, one is a small model of ~30M FLOPs and the other is a tiny model of ~10M FLOPs. Table VIII and Table IX show the experimental results confirming CEModule is far more scalable than GhostModule on small models.

We finally tested the stability of both CEModule and GhostModule. We employed TN, LeNet and AlexNet as the baseline models and deleted the fully connected layer to highlight the feature extraction capabilities of CEModule and GhostModule.

TABLE VIII
SMALL MODEL PERFORMANCE.

Models	FLOPs	h'	Acc.
Ghost_ResNet56	35.78M	12	89.22%
CE_ResNet56	**33.12M**	16	**90.15%**
Ghost_VGG16	34.05M	20	88.30%
CE_VGG16	**33.56M**	26	**89.67%**

TABLE IX
TINY MODEL PERFORMANCE.

Models	FLOPs	h'	Acc.
Ghost_VGG16	12.02M	8	86.89%
CE_VGG16	**11.54M**	10	**87.59%**

The FLOPs of the three baseline models were set to 820K, 900K and 1.5M. The stability is mainly determined by the median of accuracy, and the interval between the maximum accuracy and the minimum accuracy. We conducted 60 sets of tests on the three baseline models. Fig. 10 shows the stability of CEModule in maintaining a higher level of accuracy in comparison with GhostModule.

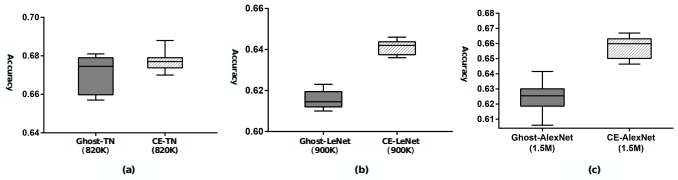


Fig. 10. A comparison of stability between CEModule and GhostModule.

D. CENet on ImageNet Dateset

Following the works of [12], [13], [16], [48], we have also designed CENet, a new lightweight CNN model based on CEModule and validated it on ImageNet. CENet is mainly composed of CE_bottleneck as shown in Fig. 11. A normal CE_bottleneck contains two CEModules with the corresponding BN layer and ReLU layer. The dashed area in Fig. 11 indicates the additional changes. Here '+' refers to the common add operation in skip connection. The function of add operation is mainly to increase the amount of information in each feature map to improve the classification performance, but the horizontal or vertical dimension itself does not increase. When stride equals to 2, we need to introduce an additional depthwise layer [12], [16] to further reduce FLOPs. Squeeze & excitation layer [48] is also included to improve the accuracy of CENet as shown in Table X.

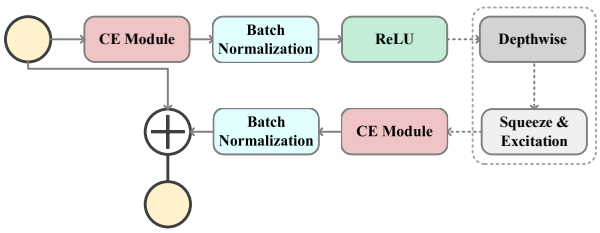


Fig. 11. The implementation of CE_bottleneck.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI10.1109/TNNLS.2021.3133127, IEEE Transactions on Neural Networks and Learning Systems

The structure of CENet basically follows the framework on AutoML search [13] and replaces the corresponding bottleneck block with CE_bottleneck. In Table X, I_c is the number of input channels, O_c is the number of output channels, SE represents the case of adding a squeeze and excitation (SE) layer to CE_Bottleneck, and I is the size of input feature maps, k is the kernel size of CE_Bottleneck, and s is the number of strides.

TABLE X
THE STRUCTURE OF CENET.

Operators	I_c	O_c	SE	I	k	s
Conv2d	3	20	-	$224^2 \times 3$	3	-
CE_Bottleneck	20	20	-	$112^2 \times 20$	3	1
CE_Bottleneck	60	30	-	$112^2 \times 20$	3	2
CE_Bottleneck	90	30	-	$56^2 \times 30$	3	1
CE_Bottleneck	90	50	SE	$56^2 \times 30$	5	2
CE_Bottleneck	150	50	SE	$28^2 \times 50$	5	1
CE_Bottleneck	300	100	-	$28^2 \times 50$	3	2
CE_Bottleneck	250	100	-	$14^2 \times 100$	3	1
CE_Bottleneck	230	100	-	$14^2 \times 100$	3	1
CE_Bottleneck	230	100	-	$14^2 \times 100$	3	1
CE_Bottleneck	600	140	SE	$14^2 \times 100$	3	1
CE_Bottleneck	840	140	SE	$14^2 \times 140$	3	1
CE_Bottleneck	840	200	SE	$14^2 \times 140$	5	2
CE_Bottleneck	1200	200	-	$7^2 \times 200$	5	1
CE_Bottleneck	1200	200	SE	$7^2 \times 200$	5	1
CE_Bottleneck	1200	200	-	$7^2 \times 200$	5	1
CE_Bottleneck	1200	200	SE	$7^2 \times 200$	5	1
Conv2d	-	1200	-	$7^2 \times 200$	1	-
Avg_Pool	-	-	-	$7^2 \times 1200$	7	-
Conv2d	-	1280	-	$1^2 \times 1200$	1	-
FC	-	1000	-	$1^2 \times 1280$	-	-

We selected the major works on lightweight CNN models [11]–[16] in the past three years for comparison. Since GhostNet and MobileNet_V3 are currently the best performing CNN models, we compared the performance of CENet, GhostNet and MobileNet_V3 using two training strategies. For fairness and reproducibility, we first followed Pytorch's official training strategy for ImageNet [44] and the recommended training parameter settings for CENet, GhostNet and MobileNet_V3. Specifically, the weight decay was $4e^{-5}$, the learning rate was 0.4, the batch size was 1024, the total number of epochs was 120 and the learning rate was reduced by 10 every 30 epochs. We evaluated the accuracy under similar FLOPs. It can be observed from Table XI that under ~150M FLOPs, CENet increases the accuracy more than 0.8% on Top-1 and 0.7% on Top-5 respectively.

TABLE XI
A COMPARISON WITH MOBILENET AND GHOSTNET.

Models	FLOPs	Top-1 Acc.	Top-5 Acc.
MobileNetV3 (L&0.75×) [13]	155M	67.9%	88.1%
GhostNet (1.0×) [16]	151M	69.5%	88.7%
CENet (1.0×)	151M	**70.3%**	**89.4%**

We then compared CENet, GhostNet and MobileNet following the cosine annealing training strategy in 360 epochs. Except CENet, GhostNet and MobileNet_V3, we also compared with another 4 models following their published results, and CENet performs best in accuracy while consuming similar FLOPs as shown in Table XII. It can be observed from Table XII that under ~150M FLOPs, CENet increases the accuracy more than 1.2% on Top-1 and 0.3% on Top-5 respectively.

TABLE XII
A COMPARISON WITH GHOSTNET, MOBILENET AND SHUFFLENET.

Models	FLOPs	Top-1 Acc.	Top-5 Acc.
MobileNetV1(0.5×) [11]	150M	63.3%	84.9%
MobileNetV2(0.6×) [12]	141M	66.7%	-
ShuffleNetV1(1.0×) [14]	138M	67.8%	87.7%
ShuffleNetV2(1.0×) [15]	146M	69.4%	88.9%
MobileNetV3 (L&0.75×) [13]	155M	70.5%	89.7%
GhostNet (1.0×) [16]	151M	71.8%	90.5%
CENet (1.0×)	151M	**73.0%**	**90.8%**

E. Evaluation of α-DAM

As presented in Section III-C, α-DAM follows a linear regression using multiple criterion points (h, h'). In this set of experiments, we evaluated α-DAM on both CE_ResNet and ResNet models. MNIST, CIFAR-10 and ImageNet datasets were employed as the basis of criterion points and FLOWER-5 dataset [49], a dataset with labelled 4242 images of flowers was selected for testing.

For h' in a criterion point (h, h'), the best value of h' normally can be obtained from a pre-training model through the tuning of parameter settings. Following the works [44], [50] in setting the value of h' on ResNet, the value of h' was set to 16 for CIFAR-10, 64 for ImageNet. It is worth noting that ResNet performs well on MNIST even when the value of h' is 2. However, a small h' could cause a large error in linear fitting. Having conducted multiple sets of low bound tests and mainstream CNN design reviews [4], [13], [36], [37], we found that when the convolution kernel of the first layer of the model is lower than 8, the instability of the model will be significantly improved. As a result, we set 8 as the minimum value of h' following Eq. (7).

Images in a dataset like ImageNet have varied numbers of corner or edge points as shown in Fig. 12. For fairness, the value of h in a criterion point (h, h') is calculated following Eq. (5) in this experiment.

In addition, the value of h is also affected by different thresholds or detection methods. However, due to the robustness of α-DAM in the adaptation process, the error between the predicted values generated by different thresholds and detection methods is relatively small. To validate the robustness of α-DAM, we employed three Harris detection methods. Table XIII shows three groups of h values using the Harris corner detection method [51] but with a threshold value of 0.04, 0.1 and 0.2 respectively.

To assess the impact of threshold values, we conducted a linear regression adaptation. Through the number of corner points h and the number of feature maps h', we can obtain

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI10.1109/TNNLS.2021.3133127, IEEE Transactions on Neural Networks and Learning Systems

Fig. 12. Examples of corner points extracted from ImageNet.

TABLE XIII
THREE SETS OF CORNER POINTS.

Datasets	Image size	h(0.04)	h(0.1)	h(0.2)
ImageNet	224×224	1196	706	230
CIFAR-10	32×32	202	120	36
MNIST	28×28	36	16	10
FLOWER-5	112×112	865	500	154

the criterion points as shown in Table XIV and fit three ReLU functions in Eq. (12).

TABLE XIV
THREE SETS OF CRITERION POINTS.

Threshold	Criterion Points (MNIST)	Criterion Points (CIFAR-10)	Criterion Points (ImageNet)
0.2	(10, 8)	(36, 16)	(230, 64)
0.1	(16, 8)	(120, 16)	(706, 64)
0.04	(36, 8)	(202, 16)	(1196, 64)

$$h' = \begin{cases} max(8, [0.049h + 6.1]) & threshold = 0.04 \\ max(8, [0.082h + 6.2]) & threshold = 0.1 \\ max(8, [0.248h + 7.1]) & threshold = 0.2 \end{cases} \quad (12)$$

where

- 8 is the value of h'_{min}.

Following the three ReLU functions in Eq. (12), the corresponding number of feature maps h' is 48, 47 and 45 respectively for the FLOWER-5 dataset. It can be concluded that, following the adaptation of α-DAM, the numbers of the generated feature maps are in a close range regardless of the underlying corner detection method that is employed.

For the FLOWER-5 dataset, we selected an average value of 46 as the final value of h'. Table XV shows the validation results of α-DAM. Following the existing mainstream works [11], [12], [36], [44], [50], the number of feature maps in the first layer of ResNet is usually set to a power of 2. We performed a number of tests using a power of 2 feature maps on FLOWER-5 dataset and conducted experiments on ResNet20 and ResNet32 respectively. It was found that when the number of feature maps was equal to 64, a similar level of accuracy can be maintained but with the least FLOPs.

Since α-DAM is based on the prediction results generated by the ResNet model in this experiment, for fairness, we employed ResNet as a baseline. In addition to ResNet, the prediction of α-DAM can also be used in the derivative model of ResNet. In order to test the diversity, we verified ResNet, ResNet32 and CE_ResNet20 respectively. Table XV shows a comparison between α-DAM and the optimal solution using 64 feature maps generated through a manually pre-trained ResNet model. It is observed that using the h' dynamically adapted by α-DAM requires even less FLOPs but maintains a similar level of accuracy.

TABLE XV
PERFORMANCE OF α-DAM.

Models	Acc. (α-DAM)	FLOPs (α-DAM)	Acc. (64)	FLOPs (64)
ResNet20	80.2%	**4.09G**	79.4%	7.91G
ResNet32	82.1%	**6.96G**	81.3%	13.46G
CE_ResNet20	81.9%	**1.05G**	82.4%	2.02G

V. CONCLUSION

In this paper, we have presented CEModule, a computation efficient module in making CNN models even more lightweight in comparison with the state-of-the-art results. CEModule enabled CNN models including the developed CENet can be potentially deployed on resource constrained mobile and embedded devices. CEModule builds on the concept of key features and group convolution to lighten the computation workload in training and in the meantime achieve a stable performance in maintaining a high level of accuracy in classification. The developed α-DAM further makes the CEModule enabled CNN models scalable in dynamically dealing with new datasets.

It is worth noting that a limitation of this research lies in the inability of the proposed α-DAM in adaptation to the most suitable scale when working on an unknown model. Neural architecture search (NAS), as a hyperparameter optimization technique, has been widely used in searching for unknown models. As a result, one immediate future work will be to combine α-DAM and NAS to further expand the search space of unknown models to accurately search for an optimal model according to different data scales. Another future work will be to deploy the developed CENet on mobile devices and to further optimize the performance of CENet in real life mobile applications.

ACKNOWLEDGMENT

This research is supported by the National Key R&D Project of China [grant number 2018YFB2100801], the Director Foundation Project of National Engineering Laboratory for Public Safety Risk Perception and Control by Big Data (PSRPC), the Fundamental Research Funds for the Central Universities, the China Electronics Technology Group Corporation (CETC) and the Shanghai Municipal Science and Technology Major Project (2021SHZDZX0100).

REFERENCES

[1] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[2] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016.

[3] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[6] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, pp. 1135–1143, 2015.

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

[8] H. Chen, Y. Wang, C. Xu, Z. Yang, C. Liu, B. Shi, C. Xu, C. Xu, and Q. Tian, "Data-free learning of student networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 3514–3522.

[9] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang, and M. Sugiyama, "Co-teaching: Robust training of deep neural networks with extremely noisy labels," in *Advances in neural information processing systems*, 2018, pp. 8527–8537.

[10] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[13] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1314–1324.

[14] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[15] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131.

[16] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "Ghostnet: More features from cheap operations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1580–1589.

[17] X. Li, S. Liu, J. Kautz, and M.-H. Yang, "Learning linear transformations for fast image and video style transfer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3809–3817.

[18] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[19] Q. Yao, M. Wang, Y. Chen, W. Dai, H. Yi-Qi, L. Yu-Feng, T. Wei-Wei, Y. Qiang, and Y. Yang, "Taking human out of learning applications: A survey on automated machine learning," *arXiv preprint arXiv:1810.13306*, 2018.

[20] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[22] E. Bochinski, T. Senst, and T. Sikora, "Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms," in *2017 IEEE international conference on image processing (ICIP)*. IEEE, 2017, pp. 3924–3928.

[23] R. Andonie and A.-C. Florea, "Weighted random search for cnn hyperparameter optimization," *International Journal of Computers Communications and Control*, 2020.

[24] W.-C. Yeh, Y.-P. Lin, Y.-C. Liang, and C.-M. Lai, "Convolution neural network hyperparameter optimization using simplified swarm optimization," *arXiv preprint arXiv:2103.03995*, 2021.

[25] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[26] X. Dong, M. Tan, A. W. Yu, D. Peng, B. Gabrys, and Q. V. Le, "Autohas: Differentiable hyper-parameter and architecture search," *arXiv preprint arXiv:2006.03656*, 2020.

[27] X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, K. Chen, Y. Tian, M. Yu, P. Vajda *et al.*, "Fbnetv3: Joint architecture-recipe search using neural acquisition function," *arXiv preprint arXiv:2006.02049*, 2020.

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

[29] Y. Liang, S. Li, C. Yan, M. Li, and C. Jiang, "Explaining the black-box model: A survey of local interpretation methods for deep neural networks," *Neurocomputing*, vol. 419, pp. 168–182, 2021.

[30] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[31] R. Fong, M. Patrick, and A. Vedaldi, "Understanding deep networks via extremal perturbations and smooth masks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 2950–2958.

[32] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[33] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[34] M. Du, N. Liu, and X. Hu, "Techniques for interpretable machine learning," *Communications of the ACM*, vol. 63, no. 1, pp. 68–77, 2019.

[35] C. Molnar, *Interpretable machine learning*. Lulu. com, 2020.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[38] P. R. Possa, S. A. Mahmoudi, N. Harb, C. Valderrama, and P. Manneback, "A multi-resolution fpga-based architecture for real-time edge and corner detection," *IEEE Transactions on Computers*, vol. 63, no. 10, pp. 2376–2388, 2013.

[39] P.-L. Shui and W.-C. Zhang, "Corner detection and classification using anisotropic directional derivative representations," *IEEE Transactions on Image Processing*, vol. 22, no. 8, pp. 3204–3218, 2013.

[40] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

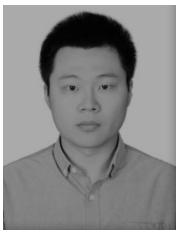
[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[44] Facebook, "Pytorch," https://github.com/pytorch/examples/tree/master/imagenet.

[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

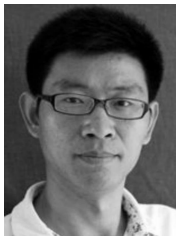
[46] X. Li, W. Wang, X. Hu, and J. Yang, "Selective kernel networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 510–519.
[47] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 558–567.
[48] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.
[49] A. Mamaev, "Flower-5, a dataset using for flowers recognition," https://www.kaggle.com/alxmamaev/flowers-recognition.
[50] Y. Idelbayev, "Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch," https://github.com/akamaster/pytorch_resnet_cifar10.
[51] P.-Y. Hsiao, C.-L. Lu, and L.-C. Fu, "Multilayered image processing for multiscale harris corner detection in digital realization," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 5, pp. 1799–1805, 2010.



Yu Liang received the Bachelor degree in Computer Science and Technology from Shandong Agricultural University, China in 2015. He received the Master degree in Software Engineering from Shandong University of Science and Technology, China in 2019. He is currently pursuing a Doctorate in the Department of Computer Science and Technology, Tongji University, Shanghai. His research interests include lightweight deep neural networks (DNNs), interpretation methods for DNNs and graph DNNs.

Maozhen Li is a Professor in the Department of Electronic and Computer Engineering, Brunel University London, UK. He received the PhD from the Institute of Software, Chinese Academy of Sciences in 1997. His main research interests include high performance computing, big data analytics and intelligent systems with applications to smart grid, smart manufacturing and smart cities. He has over 180 research publications in these areas including 4 books. He has served over 30 IEEE conferences and is on the editorial board of a number of journals. He is a Fellow of the British Computer Society (BCS) and the Institute of Engineering and Technology (IET).

Changjun Jiang is a Professor in the Department of Computer Science and Technology, Tongji University, Shanghai, China. He is also the Director of the Key Laboratory of the Ministry of Education on Embedded System and Service Computing, Tongji University. His research interests include concurrency theory, Petri nets, formal verification of software, cluster, machine learning, intelligent transportation systems, and service-oriented computing. He has published more than 300 papers in journals and conference proceedings in these areas. He has led over 30 research projects sponsored by the National Natural Science Foundation of China, the National High Technology Research and Development Program of China, and the National Basic Research Developing Program of China. He is an Academician of the Chinese Academy of Engineering, a Fellow of the Chinese Association for Artificial Intelligence (CAAI) and also a Fellow of the Institute of Engineering and Technology (IET).



Guanjun Liu (M'16, SM'19) received the Ph.D. degree in computer software and theory from Tongji University, Shanghai, China, in 2011. He was a Post-Doctoral Research Fellow with the Singapore University of Technology and Design, Singapore, from 2011 to 2013. He was a Post-Doctoral Research Fellow with the Humboldt University of Berlin, Berlin, Germany, from 2013 to 2014, supported by the Alexander von Humboldt Foundation. He is currently a Professor with the Department of Computer Science and Technology, Tongji University. He has authored over 100 articles and two books. His research interests include Petri net theory, model checking, machine learning, information security, real-time concurrent system and multi-agent system.