OCAP: On-Device Class-Aware Pruning for Personalized Edge DNN Models

Ye-Da Ma[a], Zhi-Chao Zhao[b,1], Di Liu[c], Zhenli He[a,*] and Wei Zhou[a]

[a] *School of Software, Yunnan University, China*
[b] *School of Computing, Chongqing University, China*
[c] *The Department of Computer Science, Norwegian University of Science and Technology, Norway*

ARTICLE INFO

Keywords:
Class-Aware Pruning
Edge Systems
Deep Neural Networks

Abstract

In this paper, we propose a new on-device class-aware pruning method for edge systems, namely OCAP. The motivation behind is that Deep Neural Network (DNN) models are usually trained with a large dataset so that they can learn more diverse features and be generalized to accurately predict numerous classes. Some works reveal that some features (channels) are only related to some classes. And edge systems are usually implemented in a specific environment, where classes the system detects are limited. As a result, implementing a general-trained model for a specific edge environment leads to unnecessary redundancy. Meanwhile, transferring some data and models to the cloud for personalization will cause privacy issues. Thus, we may have an on-device class-aware pruning method to remove the channels which are irrelevant for the classes the edge system observes mostly, thereby reducing the model's Floating Point Operations (FLOPs), memory footprint, latency, improving energy efficiency and keeping a relatively high accuracy for the observed classes while protecting the in-situ data privacy. OCAP proposes a novel class-aware pruning method based on the intermediate activation of input images to identify the class-irrelevant channels. Moreover, we propose a method based on KL-divergence to select diverse and representative data for effectively fine-tuning the pruned model. The experimental results show the effectiveness and efficiency of OCAP. In comparison with state-of-the-art class-aware pruning methods, OCAP has better accuracy and higher compression ratio. Additionally, we evaluate OCAP on Nvidia Jetson Nano, Nvidia Jetson TX2 and Nvidia Jetson AGX Xavier in terms of efficiency, where the experimental results demonstrate the applicability of OCAP on edge systems. The code is available at `https://github.com/mzd2222/OCAP`.

1. Introduction

Deep Neural Networks (DNNs) have achieved great success in computer vision, such as target detection [40, 36, 39], image classification [11, 34, 17], and image segmentation [35, 27]. Recently DNNs have been increasingly implemented on resource-constraint devices, like embedded systems and edge systems, to process data locally and reduce communication overhead [19, 25], thereby paving the way of ubiquitous Artificial Intelligence (AI). However, DNNs are computation-intensive and memory-hungry, and are becoming more deeper and wider. As a consequence, adopting DNN models on edge systems encounters two issues, poor performance and high power consumption, which undermine the applicability of edge AI systems.

To reduce the complexity of DNN models and improve the performance of DNN models on edge systems, some novel and efficient DNN architectures are proposed, like MobileNet [34], ShuffleNet [29] and GhostNet [8], while others strive to compress complex models [10, 26, 28, 24, 16]. Among several model compression techniques, model pruning is a promising and widely-used technique to reduce the complexity of DNN models. Since the seminar work of model pruning, Deep Compression [10], was proposed, a huge amount of efforts were made towards more effective and efficient pruning methods [24, 20, 1]. However, the majority of pruning works tend to reduce the redundancy of the complex and over-parameterized DNN models for all classes [26], but ignore that there is **class redundancy** when implementing the model upon a specific environment, i.e., the number of classes the model is able to predict is more than it needs in its adoption context.

Class redundancy is due to that when training the over-parameterized DNN models for better accuracy, we usually use a huge amount of data with numerous classes to learn various diverse features. For example, the widely-known ImageNet ILSVRC2012 [3] has 1.2M training images of 1k classes and Google's private dataset JFT has 303M images of 18k classes [14]. And vendors usually train a large and general model that can recognize many classes to meet the diverse needs of different users. Nevertheless, when adopting DNNs in some specific contexts, the system only needs to recognize a limited number of classes. For instance, an intelligent camera implemented in a wild park is expected to monitor wild animals, but rarely detects classes

This work was supported in part by the National Natural Science Foundation of China under Grant 62262070, in part by Yunnan Applied Basic Research Projects 202101AT070182, 202201AT070156 and 202301AT070194.

*Corresponding author

`mzd@mail.ynu.edu.cn` (Y. Ma); `zhaozhichao@cqu.edu.cn` (Z. Zhao); `di.liu@ntnu.no` (D. Liu); `hezl@ynu.edu.cn` (Z. He); `zwei@ynu.edu.cn` (W. Zhou)

ORCID(s):

[1] This work was done when ZC Zhao was an undergraduate at Yunnan University.

such as cars, football helmet[2]. Previous works [33, 13] have found that some features of DNN models are only related to certain classes, and removing these redundant classes and the features pertaining to these classes do not affect the accuracy of other classes. From our experiments on Section 4, removing irrelevant features can even improve the accuracy. Therefore, removing redundant classes and the classes-related features enable us to further reduce the complexity of models and improve the efficiency of DNN models on edge systems .

Some works consider the class-aware pruning [33, 13]. These methods feature a design-time method, i.e., the predicted classes are known in prior, and then the model is tailored for users at design time on a powerful server. However, the design-time methods suffer from two problems. First, as data privacy has gradually grown to a major concern for the digital world, data privacy is an issue of the design-time methods where models or sensitive data are prone to be hacked or leaked during the transmission. Second, there are some classes which can not be determined in advance until the system operation. Therefore, on-device machine learning which can protect local data and provide a flexible way to update models has become an emerging trend [42, 5]. Considering the limits of design-time methods and merits of on-device machine learning, it would be beneficial to have an on-device method for the class-aware pruning, so that the user can personalize the model in-situ to reduce the inference and memory overhead while protecting private data. It also can be used as the complement for other design-time pruning methods to optimize the model according to the run-time track.

A few methods implement run-time pruning. Lin *et al.* [23] proposed a run-time pruning method, but this approach prunes models for all classes instead of using class-aware pruning. Moreover, it is based on complex reinforcement learning which is inapplicable to resource-limited edge systems. A complex pruning method drains battery quickly and thus shortens the operational time. To this end, we, in this paper, propose an On-device Class-Aware Pruning (OCAP) method for DNN models on edge systems. Different from the prior class-aware pruning methods [33, 13] which feature an offline method and rely on a complex pre-processing procedure, OCAP exploits the intermediate activation of inference data to effectively and efficiently prune the model on the device at run-time. Our detailed contributions are as follows:

- We propose OCAP, an on-device class-aware pruning method, which prunes the DNN model according to the objects the model observes at run-time. The novel class-aware pruning method directly uses the intermediate activation of the input images observed at run-time to prune the model in a class-aware fashion;

- The pruned models need a fine-tuning method to retain the accuracy, which usually needs a huge amount of data. However, edge devices are subject to limited memory and computing resources, so we cannot store a lot of data. Thus, we propose a novel and data-efficient method based on KL-divergence to select diverse and representative data from the input data for effectively and efficiently fine-tuning the pruned model to retain the accuracy;

- We extensively evaluate the proposed method in terms of accuracy, compression ratio, and latency using different DNN models with different datasets. OCAP can increase the model accuracy by up to 20% when only a few classes are remained and it also can reduce the inference latency by more than 50%. Then we evaluate the online pruning overhead of OCAP on several edge systems to demonstrate its applicability for resource limited systems. In addition, we conduct a detailed ablation study for OCAP.

We have open-sourced the code at `https://github.com/mzd2222/OCAP`. The reminder of this paper is organized as follows: Section 2 discusses some related work. Section 3 presents the details of OCAP. Section 4 shows the experimental results and Section 5 conducts the ablation study of OCAP. Section 6 concludes this paper.

2. Related Work

Model pruning has been widely studied in recent years. Han *et al.* [10] proposed *Deep Compression*, the seminal work of DNN pruning, to remove the irrelevant weights and quantize weights to significantly reduce the model size with negligible accuracy loss. However, Deep Compression is an unstructured pruning method, i.e., such pruning method generates sparse and irregular patterns within the pruned model. As a consequence, the pruned model cannot be boosted without specialized hardware and software supports [9].

Now, the majority of pruning methods exploit *structured pruning*, i.e., instead of removing individual weights within the kernels of a model, structured pruning removes irrelevant or redundant channels/filters of each layer [22]. Structured pruning reserves the regular pattern of DNN models, so its pruning result can directly translate to the speed-up of the compressed model on off-the-shelf devices. In the past years, structured pruning receives more attention, such as [22, 26, 2, 21, 6]. [22] introduces a criterion based on the weights of convolution kernels. It uses the ℓ_2-norm of each channel's weights as the importance of the channel for structured pruning within a layer. While [26] uses the γ of each BN-layer (the scaling parameter of each layer) as the criterion to guide its structured pruning. Based on the criterion of [22], [2] proposes a global criterion which adds a learnable parameter to the importance of each layer and sorts channels globally to prune. However, these pruning methods target to identify and eliminate the redundant channels for all classes and are **inapplicable for class-aware pruning**. On the other hand, OCAP is a class-aware pruning method which is based on the empirical observation

[2]The 560th class in ImageNet is football helmet

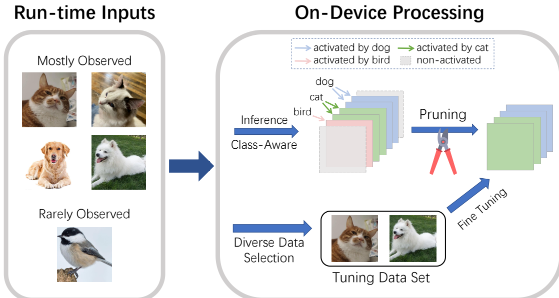


Figure 1: The overview of OCAP. A pet detection camera at home is considered in the figure. In this scenario, only cats and dogs need to be detected. Therefore, after class-aware, OCAP prunes the irrelevant channels (pink and gray channels in the figure) to cats and dogs, then uses the selected data to fine-tune the model after pruning.

of redundancy classes. Additionally, our experiments have shown that class-aware pruning can significantly improve accuracy while reducing model size, particularly in scenarios with a small number of classes. To know more about the normal model pruning, we refer interested readers to some survey papers [25, 4].

As indicated in [33], most of DNN applications on mobile devices only detect a limited number of classes during its operation, in some cases only a couple of classes. Thus, channels which are irrelevant to the classes of interest can be removed to further reduce the model complexity, thereby improving latency and energy efficiency. [33] and [13] are the two works close to OCAP. [33] exploits grid search and k-means to group the filters relevant to each classes, and then selects the filters according to the classes of interest. Similarly, [13] calculates a firing rate for each filter when inferring different classes, and then prunes the network according to the classes of interest and a predefined threshold. OCAP differs from them in two ways: 1) both are **offline methods with the classes of interest known in prior**, whereas OCAP considers a practical situation where the classes of interest are only known until its execution and the users may not or cannot share their interests with the server due to privacy concerns. Moreover, our method can be considered and used as a complement for the design-time method; 2) both approaches **have a complicated pre-processing** that cannot be deployed on resource-limited edge systems to conduct class-aware pruning at run-time. In addition, they only evaluate their approaches on some old models, AlexNet ([33]) and VGG ([33, 13]), which cannot represent the state-of-the-art DNN models, like ResNet [11] and MobileNet [34].

3. OCAP

In this section, we present the details of OCAP. Fig. 1 shows the overview of OCAP. The whole procedure is completed on the device upon which the DNN model is implemented. When the edge DNN system starts to operate in its context and observes some target images, it generates the predicted results. At the same time, the model is pruned according to the observed images. Meanwhile, the observed images form a diverse dataset for fine-tuning. More details will be discussed below.

OCAP conducts an on-device pruning and targets the resource-limited edge systems. Edge systems usually have limited processing units and memory is shared among CPU, GPU and accelerators. Hence, OCAP should have low overhead and does not consume a lot of memory. As we have discussed in Section 2, although the unstructured pruning can greatly compress the model, the pruned model cannot reduce its latency due to the irregular pattern [10]. Therefore, in OCAP, we deploy the structured pruning (pruning channels) that can boost the execution of the compressed model on off-the-shelf platforms. OCAP consists of two steps:

1) **class-aware pruning**: it selects the irrelevant channels and prunes the model according to the images obtained by the system during its operation;

2) **fine-tuning procedure**: it fine-tunes the pruned model from class-aware pruning with the selected images to retain the accuracy after the pruning.

We proceed to the details of these two parts below.

3.1. Class-Aware Pruning

To have an effective and efficient pruning, we need to first determine how to select the pruned channels. In OCAP, our goal is to reserve the channels which are relevant for the classes the model mostly detects or observes and to prune those which are irrelevant to the classes of interest. And then, the model can be compressed and the latency, FLOPs, memory footprint, and energy consumption of the system can be improved over its execution. For edge systems, such reduction may have a significantly accumulative benefit over long operation time.

A successful channel pruning relies on a good criterion and pruning algorithm to select and prune the redundant channels. The existing pruning works usually use the reconstruction error [28] or gradient-based methods [30] to identify which channels can be removed without affecting the accuracy. However, since they need to evaluate the effect of removing each channel, these approaches suffer from high computational overhead and are not suitable for on-device operation, especially on resource-limited edge systems. Therefore, we need to design a simple yet effective method for our on-device pruning.

3.1.1. Class Relevance Masks

OCAP is based on the fact that different classes activate different channels within a model to make the accurate prediction [41]. Thus, some works like [41] exploit this observation and attention mechanism [38] to magnify the large (important) activation and suppress the small (unimportant) activation to further improve the model accuracy. We are inspired by this and propose our class-aware pruning method that uses the inputs the model observes at run-time to efficiently determine the pruned channels.

Many pruning methods use different and specific criteria for determining channel importance [22, 24, 12, 2], and use a binary mask to mark whether a channel should be pruned. Since we consider a class-aware pruning, we define the importance of each channel from the lens of predicted classes and propose *Class Relevance Value* (CRV), as shown in Eq. (1), to compute the class relevance masks.

$$CRV_{i,j} = ||LeackyRELU(\boldsymbol{x}_{i,j})||_2 \tag{1}$$

$$LeakyRELU(x) = \begin{cases} x & x \geq 0 \\ -px & x < 0 \end{cases} \tag{2}$$

where $CRV_{i,j}$ is the CRV of the i^{th} channel of the j^{th} layer. $\boldsymbol{x}_{i,j} \in \mathbb{R}^{H_j \times W_j}$ represents the activation values of the i^{th} channel of the j^{th} layer, and $||\cdot||_2$ indicates all values in the matrix are squared and then summed up, i.e., the ℓ_2-norm, and $LeakyRELU$ is a pre-processing function to process the activation values, as shown in Eq. (2).

We design the CRV based on the following observations. Most of the modern DNN models stack several layers to form a block, including convolutional layer, batch normalization (BN) layer [18], activation layer, and pooling layer. While the convolutional (conv) layer extracts features from its input and the activation layer (e.g., $ReLU$) filters the output from the conv layer, BN layer normalizes inputs for fast and stable training performance and the pooling layer downsamples inputs to reduce computation. As [15] points out that low activation value may imply the less importance of that channel, we exploit this feature to identify the class-irrelevant features. We compute the ℓ_2-norm of BN outputs to evaluate the importance of a channel. Before computing the ℓ_2-norm of the BN layer, a pre-processing function, $LeakyRELU$, is applied to the BN output so that it can avoid that ℓ_2-norm makes the negative value as important as the positive value but still can reserve some information from the negative value. The negative slope p in Eq. (2) adjusts the importance of the negative value. And the ablation study of $LeakyRELU$ and parameter p is given in Section 5.2.

Channel pruning are usually divided into global pruning [2] and layer-based pruning [22, 26]. OCAP adopts a layer-based pruning method. This is because class-aware pruning usually prunes a lot of channels, especially if there are only few classes reserved, and the use of global pruning for OCAP will cause layer collapse phenomenon (pruning a whole layer) [37], resulting in a sharp decrease of the final accuracy. In addition, global pruning usually requires the introduction of additional parameters and is more complex than the layer-based pruning. Therefore, global pruning is not suitable for an on-device pruning method.

Since we use a layer-based pruning method, after calculating the $CRVs$ of channels, OCAP sorts the $CRVs$ within layers, and get threshold T_j for each layer according to a pruning ratio. Then the pruning is conducted by a binary channel mask $M_{i,j}$:

$$M_{i,j} = \begin{cases} 1 & CRV_{i,j} \geq T_j \\ 0 & CRV_{i,j} < T_j \end{cases} \tag{3}$$

where $M_{i,j}$ denotes the mask of the i^{th} channel at the j^{th} layer, and $M_{i,j} = 1$ means the channel will be reserved and $M_{i,j} = 0$ indicates to remove the channel. $\mathcal{M}$ denotes the mask set of all channels in a DNN model, but excluding some earlier layers. As shown in literature [41], the earlier layers are to extract the general features of inputs and pruning these layers may significantly degrade the accuracy, so we do not prune the earlier layers. Moreover, threshold T_j in OCAP is not a fixed value and it depends on the reserved number of classes and the target pruning ratio, and we will introduce T_j in detail later.

In addition, we need to decide how many images to be reserved for each class to calculate $CRVs$. To determine the number of images or design a good method to do so, we first empirically evaluate the impact of the number of images on $CRVs$, where we use different number of images with the same reserved classes to calculate the masks of each channel, as shown in Fig. 2. For each subfigure named x-y_z (e.g. ResNet-101_6), x-y denotes the model name and depth of the target model, and z denotes the layer index of the model. For different layers of different models, we have found that the masks generated for a single layer is always the same or just changed slightly, regardless of how much data the model has used. It suggests that the masks of DNNs can be accurately estimated using only a small portion of the input images, thus, we can use a small number of images to calculate the masks to raise efficiency for our on-device method. Our experiments in Fig. 3 show that using an extremely small number of images to calculate the mask (e.g. one) leads to a slight decrease in accuracy, around 1%. On the other hand, increasing the number of images beyond a certain threshold, 20 (for ResNet and VGG) or 30 (for MobileNetV2), has no effect on the final accuracy. Therefore, to strike a balance between accuracy and efficiency in OCAP, we set the number of images used to compute the mask to 20 (for ResNet and VGG) and 30 (for MobileNetV2) for each class.

3.1.2. Pruning Strategies

With the method to determine the irrelevant channels, in this section, we present two proposed pruning strategies which can be used when having different goals.

Fixed Ratio Pruning Strategy: For resource-limited devices, we may have a target compression ratio to control the resource utilization. Therefore, it is necessary for the pruning algorithm to accurately reach the target compression ratio. To this case, we present a Fixed Ratio Pruning Strategy called OCAP-FR, as shown in Algorithm 1. The core of this algorithm is to precisely adjust the pruning ratio of each layer to the target compression ratio P, so that the entire model can accurately reach the target compression ratio. For each image, we first use the pruning ratio P to

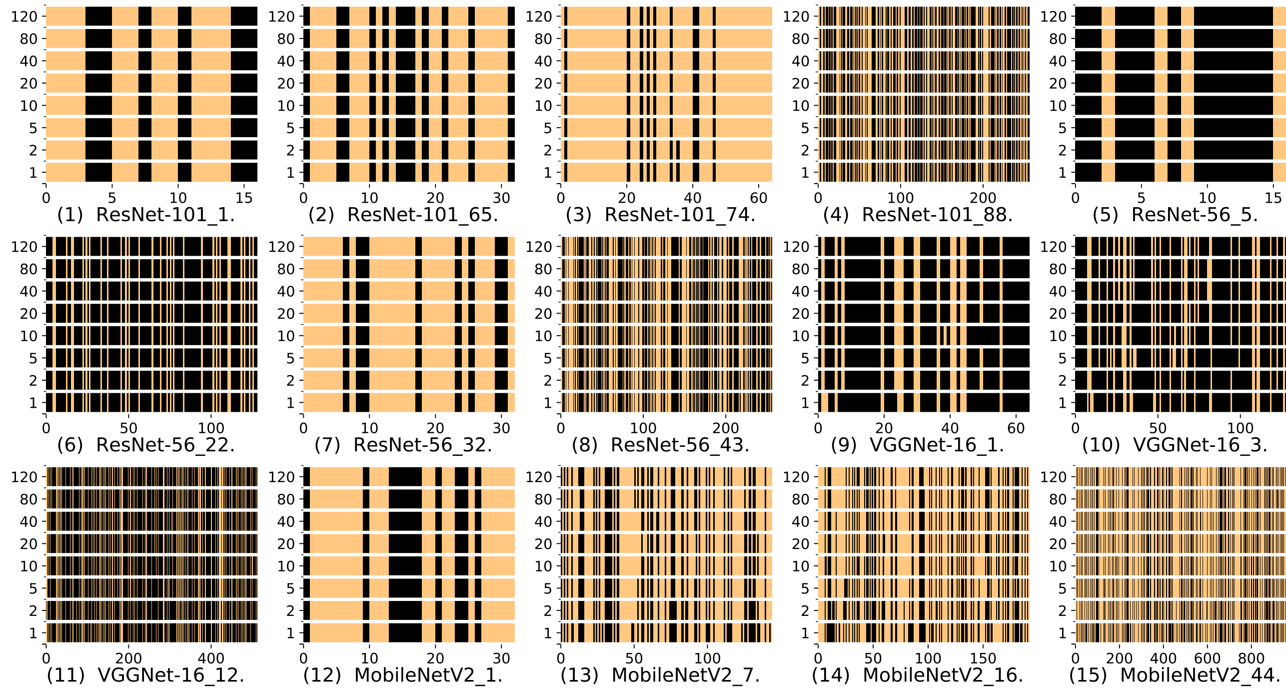


Figure 2: The layer masks from different convolutional layers and architectures on CIFAR-10 with saving the first 5 classes. For each subfigure, the x-axis represents the index of channels in the current layer, and the y-axis represents the number of images for each reserved class to calculate the masks. The different colors denote different masks (True or False). As shown in each subfigure, the mask of each channel is almost the same (the same color) or just changed slightly, regardless the images numbers. Thus, using small number of images can calculate the masks effectively and efficiently. This also justifies the applicability of CRV.

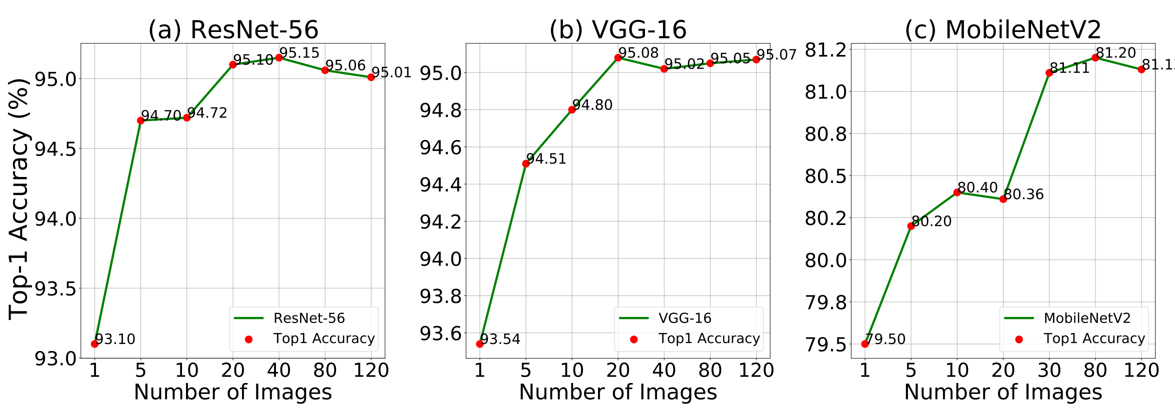


Figure 3: The relationship between the accuracy and the number of images for each reserved class to calculate the masks. The experiment is conducted on CIFAR10 with 5 classes remained.

calculate the activation state (i.e., True or False) of each channel (line 6-8), where CT_1 first sorts $CRVs$ within layers, then uses P to obtain the threshold of $CRVs$ of each layer, and CM_1 uses threshold T_1 and $CRVs$ to obtain the mask of current image. After calculating masks for all images (line 4-10), we merge the masks layer-by-layer (line 11-16). For each layer, the number of images that activates the channel within the layer is used as the score (NT) for the channel (line 12). Then we use P to compute the threshold T_2 for the current layer according to the sorted NT (line 13). Using P again within each layer can precisely control the compression ratio of each layer, so that the pruned model can reach the target compression ratio. Afterwards, CM_2 uses threshold T_2 and NT to obtain the mask of the current layer (line 14). After calculating the mask of each layer, the calculated masks are used for pruning.

As OCAP-FR uses the fixed ratio, the actual pruning rate is the same for each layer, but this is not optimal in terms of accuracy. We conduct some experiments to reveal the impact of the fixed ratio, as shown in Fig. 4. We use Layer-x-y to represent the x layer with the number of y channels. In this experiment, we input 256 images into the

Algorithm 1: OCAP-FR

Input: Original model $\mathcal{N}$, input data $\mathcal{D}$, remained classes C, the number of layers $\mathcal{L}$, the pruning ratio function P from Eq. (4)
Output: The pruned model $\hat{\mathcal{N}}$
1 Initialize CRV with $\mathbf{0}$ /* CRV vector includes the CRV value of each channel in each layer */
2 Initialize M_a with [] /* It includes the mask of each image. */
3 Initialize $\hat{\mathcal{M}}$ with [] /* It includes the mask of each layer. */
4 **for** $D \in \mathcal{D}$ **do**
  /* each image D. */
5 $O \leftarrow \mathcal{N}(D)$
6 Compute CRV for current image D using Eq. (1)
7 $T_1 \leftarrow CT_1(CRV, P(C))$ /* Compute the threshold vector for each layer */
8 $\mathcal{M}_1 \leftarrow CM_1(CRV, T_1)$ /* Obtain the mask for current image */
9 Add $\mathcal{M}_1$ to M_a
10 **end**
11 **for** $l := 1$ *to* $\mathcal{L}$ **do**
12 Count the number of $True$ (NT) for every channel in current layer l using M_a
13 $T_2 \leftarrow CT_2(NT, P(C))$ /* Compute the threshold vector for current layer l according to the sort of NT */
14 $\mathcal{M}_2 \leftarrow CM_2(NT, T_2)$ /* Obtain the mask for current layer l */
15 Add $\mathcal{M}_2$ to $\hat{\mathcal{M}}$
16 **end**
17 **return** $\hat{\mathcal{N}} \leftarrow prune(\mathcal{N}, \hat{\mathcal{M}})$

model and count the activation times for each channel layer by layer. For instance, if a channel is activated by 20 out of 256 images (i.e., $M_{i,j} = 1$), its activation count is set to 20. After counting the activation times for each channel, we plot the channel activation times for each layer, as shown in Fig. 4, where the x-axis represents the channel activation count, and the y-axis represents the frequency (i.e., the number of channels with this activation count), e.g., Layer-1-16 in Fig. 4 (b), indicates that the first layer of ResNet-56 has 16 channels, of which there are 7 channels with activation times of 0, and 9 channels with activation times of 256. Based on the experiment results, we further classify the layers into two types:

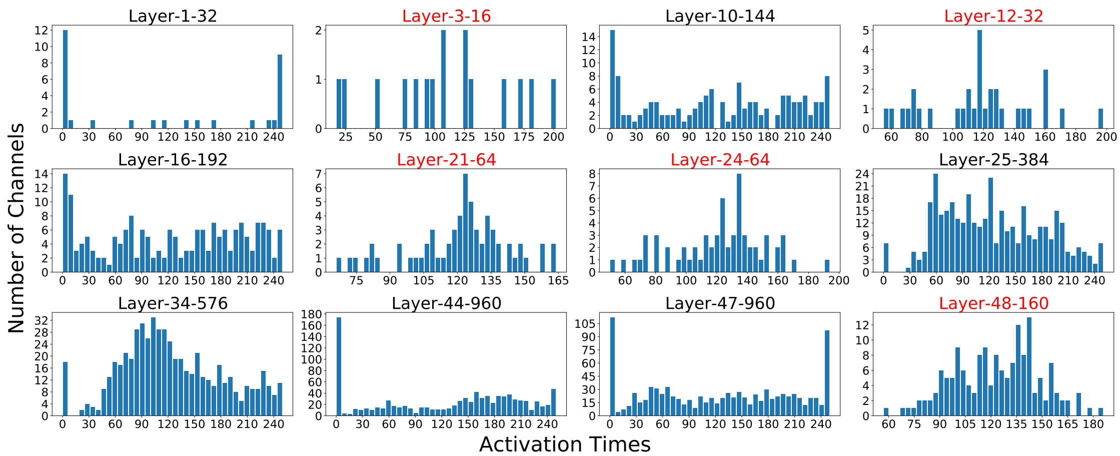
1) **bottleneck layers** [31]: All channels within these layers are used, i.e., activated ($M_{i,j} = 1$) by images at least once. We highlight bottleneck layers using red color in Fig. 4, such as Layer-21 of MobileNetV2, Layer-27 of ResNet-56 and Layer-5 of VGG-16. All the channels in these layers contain important information and cannot be pruned.
2) **non-bottleneck layers**: Some channels in these layers are not used at all, i.e., no image activates ($M_{i,j} = 0$) these channels, such as Layer-10 of MobileNetV2, Layer-16 of ResNet-56 and Layer-12 of VGG-16 in Fig. 4. It is not difficult to envision that the zero-activated channels in these layers can be pruned.

We empirically find that for all three models, there are some bottleneck layers and non-bottleneck layers. If we use the fixed ratio for all layers, some important channels in bottleneck layers will be pruned, resulting in a decrease in pruned model accuracy. Moreover, for different non-bottleneck layers, the number of zero-activated channels that can be pruned are different as well.

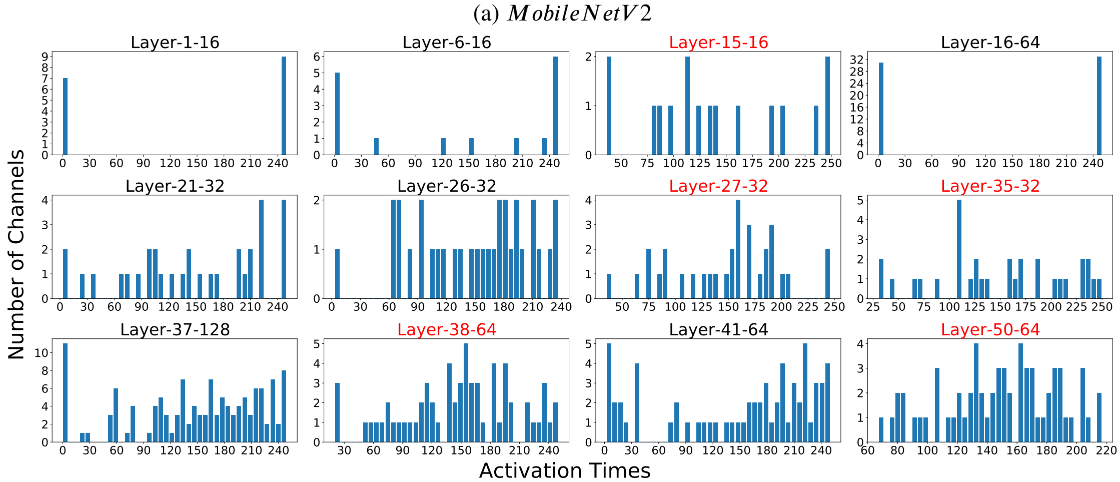
Accuracy Best Pruning Strategy: Based on our previous observations, to set different pruning ratio for different layers, we propose a novel class-aware pruning strategy for better accuracy, called Accuracy Best Pruning Strategy (OCAP-AB), which can automatically adjust the pruning ratio and decide whether to prune the current layer and how many channels should be pruned. The pseudo code of OCAP-AB is given in Algorithm 2. We first calculate the mask for an image using P like OCAP-FR (line 5-7), then merge the current mask $\mathcal{M}$ to the final mask $\hat{\mathcal{M}}$ channel-by-channel by using *or* ($\bigvee$) operator (line 8). It means that for each channel in the final mask, if there is one image that activates the channel, this channel will be activated (reserved) in the final mask. Different from OCAP-FR, it is not easy to control the compression ratio of the model pruned by Algorithm 2 to accurately reach at the target compression ratio, because the masks of all channels are merged by the $\bigvee$ operator. Experiments in Section 5.1 show that OCAP-AB outperforms OCAP-FR in terms of accuracy, but has higher pruning overhead.

Algorithm 2: OCAP-AB

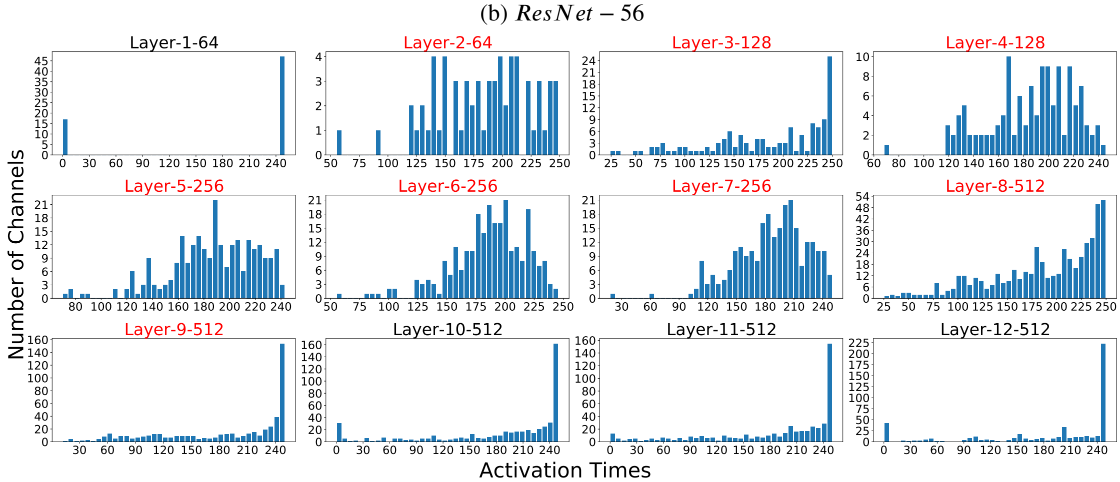
Input: Original model $\mathcal{N}$, input data $\mathcal{D}$, remained classes C, the pruning ratio function P from Eq. (4)
Output: The pruned model $\hat{\mathcal{N}}$
1 Initialize CRV with $\mathbf{0}$ /* CRV vector includes the CRV value of each channel in each layer */
2 Initialize $\hat{\mathcal{M}}$
3 **for** $D \in \mathcal{D}$ **do**
  /* each image D. */
4 $O \leftarrow \mathcal{N}(D)$
5 Compute CRV for current image D using Eq. (1)
6 $T \leftarrow CT(CRV, P(C))$ /* Compute the threshold vector for each layer */
7 $\mathcal{M} \leftarrow CM(CRV, T)$ /* Obtain the mask for current image */
8 $\hat{\mathcal{M}} \leftarrow \hat{\mathcal{M}} \bigvee \mathcal{M}$ /* Using *or* ($\bigvee$) to merge masks from different images */
9 **end**
10 **return** $\hat{\mathcal{N}} \leftarrow prune(\mathcal{N}, \hat{\mathcal{M}})$



(a) $MobileNetV2$



(b) $ResNet-56$



(c) $VGG-16$

Figure 4: The activation times distribution diagram of layers of different models. The experiment is conducted on CIFAR-10 with 5 classes remained. For each subfigure, the x-axis represents the number of times that the channel is activated, and the y-axis represents the number of channels corresponding to activation times.

3.1.3. Adaptive Pruning Ratio

In OCAP, we use a layer-based pruning, i.e., the channels within each layer are sorted in terms of $CRVs$ and are pruned according to a given pruning ratio. Different from other pruning methods which only have one fixed pruning ratio, our pruning ratio is dependent on the number of classes remained. It is not difficult to envision the more classes the model remains, the more channels it should keep. In OCAP, we deploy a simple linear function to formulate the relationship between the number of remained classes C

Model (Original accuracy)	α	β	Accuracy of different number of reserved classes		
			20%	50%	80%
VGG-16 (94%)	-0.25	0.90	99.2%	95.7%	93.7%
ResNet-56 (94%)	-0.51	0.85	98.0%	95.4%	93.9%
MobileNetV2 (88%)	-0.75	0.78	95.5%	83.6%	80.9%

Table 1
The experimentally obtained values of α and β, and their corresponding accuracy on CIFAR10.

and the pruning ratio P, as shown in Eq. (4).

$$P = \alpha C + \beta \tag{4}$$

where α and β are two constants that are related to the model and its training dataset and can be determined at design-time. Fig. 5 shows the estimated lines for three models VGG, ResNet and MobileNetV2 with CIFAR10. Table 1 presents the α and β values obtained through multiple experiments, along with their corresponding accuracy of CIFAR10. VGG is known to be a redundant model, so we can prune more channels. Whereas MobileNetV2 is a compact model, so the pruning ratio is smaller than others.

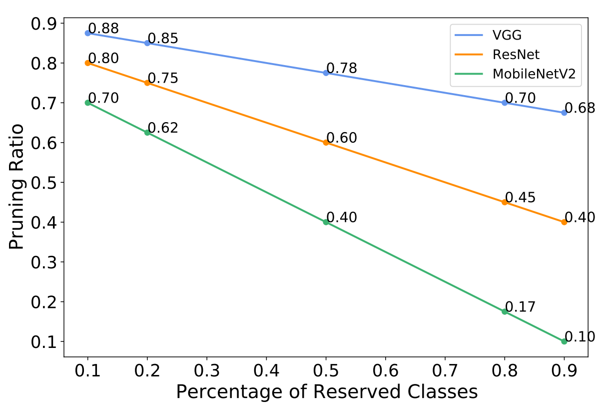


Figure 5: The relationship between the number of remained classes and pruning ratio for three models.

3.1.4. Pruning Procedure

Class-aware pruning is conducted as follows: we store the inputs and the feature maps before the fully connected layer obtained at run-time to form the fine-tuning dataset later. If a pruning signal is triggered, we calculate the pruning masks according to the stored inputs and the pruning ratio obtained from Eq. (4). Then we prune the model according to the pruning masks and the prune ratio. Various ways can trigger the pruning procedure, such as the model has accumulated a sufficient number of inputs, the system has operated for a certain time, etc. Note that besides convolutional layers, we also can prune the classifier (fully connected layers) to further reduce the model complexity.

3.2. Fine-Tuning Procedure

After the pruning, OCAP only reserves the channels which are relevant for the remained classes. However, the pruning changes the structure and weights of the original DNN model, so the pruned model needs to be fine-tuned to retain its accuracy. If only a couple of classes are remained, the fine-tuning procedure can be skipped. When more classes are remained, the fine-tuning procedure is a necessary step to retain the accuracy.

Since OCAP is expected to execute on edge devices, the fine-tuning of OCAP should be simple and uses as few data as possible. All existing pruning methods need a fine-tuning process, where the pruned model is retrained with all training data [10, 28]. Nevertheless, the whole training data is too large to store for resource-limited systems. For example, ImageNet [3] has 1.2M images of 1000 classes, needs more than 100GB space to store all training data, and a class contains 1300 images, needs about 150MB space to store. In our setting, we do not know in prior which classes are preferred or will be remained. Then, preserving all training data is not practical and results in high memory overhead. Thus, in OCAP, we strive to use the images the systems obtain at run-time to form a small fine-tuning dataset[3]. The advantage of this method is that we do not have to keep all training data on device, but only need to reserve the data the model infers. Hence, it can significantly reduce the memory occupation.

Fine-tuning data selection plays a pivotal role in retaining the competitive accuracy of the pruned model. Here, we need to answer two questions: 1) *what kinds of inputs should we save for each class?* and 2) *how many inputs should we reserve for each class?*

3.2.1. Data Selection

When adding data for the fine-tuning procedure, we need to select diverse data for each class. This is because our pruning masks are determined by the inputs' activation. If the most of data is similar, e.g., the same color, the same angle, etc, the model may be unable to predict the same class with different characteristics. In OCAP, we employ KL-divergence to facilitate the selection of diverse data. KL-divergence measures the similarity of two statistic distributions. Some works exploit KL-divergence to measure the similarity between images and use this feature for image retrieval [7].

[3]Here, we have to consider that users may join to provide a correct label for the new input or we only use the prediction images with high prediction confidence.

As shown in Algorithm 3, we design our data selection mechanism as follows: Each class has one memory of size N. The images are directly added to the class memory if the class memory is not full. For each image, we use the histogram of features to compute a KL-divergence score K_c shown in Eq. (5). This KL-divergence score can be deemed as the similarity measurement between the input image and the existing images for this class. The larger the KL-divergence score is, the more different the input image is from the existing images.

$$K_c = \begin{cases} \frac{1}{N_c} \sum_{k=0}^{N_c} KL(P||Q_k) & \text{if } N_c < \frac{N}{d} \\ \frac{d}{N} \sum_{k=0}^{\frac{N}{d}} KL(P||Q_k) & \text{if } N_c \geq \frac{N}{d} \end{cases} \tag{5}$$

$$KL(P||Q) = \sum P(\boldsymbol{f_s}) \log \frac{P(\boldsymbol{f_s})}{Q(\boldsymbol{f_e})} \tag{6}$$

where $\boldsymbol{f_s}$ represents the feature of the current image x_s, and $\boldsymbol{f_e}$ represents the feature of a randomly selected image x_e from the class memory. P and Q represent the histogram distribution of $\boldsymbol{f_s}$ and $\boldsymbol{f_e}$ respectively. N_c denotes the number of images in the class memory. To reduce the computation overhead, we do not compute the KL-divergence of the input image with all existing images. Instead, we consider two cases: 1) if $N_c < \frac{N}{d}$, we compute the KL-divergence of the input with all existing images; 2) if $N_c \geq \frac{N}{d}$, we randomly select $\frac{N}{d}$ images from the class memory to compute the KL-divergence score. Then, if the class memory is full and a new image for this class is coming, we compute the KL-divergence score and remove the image with the lowest KL-divergence score to have a diverse dataset. Note that d ($d \geq 1$) represents a divided parameter for efficient computing and is changeable. If the underlying hardware is more capable and has a large memory, we can increase $\frac{N}{d}$ to have more data to calculate K_c more accurately. And confidence threshold T is used to select images with higher confidence, indicating that the model is confident in correctly classifying these images. In OCAP, we set $T = 0.9$.

Additionally, to further enhance the efficiency of the data selection, we use a parameter called N_R, i.e. the max replacements number to constrain the image replacement times for every remained class. For class i, after the class memory for i is full, we increment n_i by 1 for every image exchange. If n_i is equal to N_R, we stop to select data for class i. After n of every class reaches to the threshold value N_R, the data selection process is done.

3.2.2. The Number of Images

The class memory size N is another important factor for the fine-tuning procedure. It is not difficult to envision that the larger memory size may allow us to store more data, thereby improving the fine-tuning performance. Fig. 6 shows the relationship between the accuracy and the number of the remained images for each class. Ideally, we expect to have as many data as possible for each class. Nevertheless, it will be an issue for resource-limited edge systems in terms of training cost and memory overhead. The experimental results show that OCAP can retain a good accuracy without reserving too many images. Therefore, in OCAP, to trade off between the accuracy and fine-tuning cost, we set $N = 128$ for ResNet and VGG, and $N = 256$ for MobileNetV2 with 50 fine-tuning epochs. This decision is based on the complexity of the implemented model. Meanwhile, a better solution can be proposed further considering the target hardware and the number of the remained classes.

4. Experiments

4.1. Experimental Setting

We extensively evaluate the effectiveness and efficiency of OCAP on CIFAR-10/CIFAR-100 and ImageNet using different models, ResNet-56 [11], VGG-16 and MobileNetV2 [34]. For all experiments, we select images from the corresponding training dataset as the models' input, so that we would not have data leakage on the validation data. The experiments are conducted on five types of devices, a PC with Nvidia RTX2080-Ti, a PC with Nvidia RTX2060-Super and three low-power edge systems, Nvidia Jetson Nano, Nvidia Jetson TX2 and Nvidia Jetson AGX Xavier. And all the experimental results are obtained by randomly selecting different reserved classes three times and then take the average of Top-1 accuracy, time overhead and latency. We implement OCAP using pytorch [32]. We compare OCAP to CAPTOR in [33] and CAPNN in [13]. Since they have not yet open-sourced their code and we cannot reproduce the same results, we directly compare our results to the numbers reported in their papers. In OCAP, we fine-tune the pruned models for 50 epochs. The initial learning rate is set to 1e-3 with learning rate decay after 20 and 40 epochs. We use an SGD optimizer with the momentum 0.9 and the weight-decay 5e-4. And we set $d = 4$ in Eq. (5). In addition, except for special instructions, all the experiments are conducted using OCAP-AB. In addition, for CIFAR10 and CIFAR100, we use the pruning ratios shown in Figure 5. For ImageNet, a fixed pruning ratio 0.9 is used. And since OCAP-AB automatically adjusts the pruning rate, we introduce another parameter, related FLOPs Ratio, to reflect the compression ratio and the complexity of the pruned model, as shown in Eq. (7).

$$\text{FLOPs Ratio} = \frac{F_p}{F_o} \tag{7}$$

where F_p and F_o represent the FLOPs of the pruned model and the original model, respectively.

4.2. Experiments of CIFAR10

Fig. 7 shows the experimental results for CIFAR10. In this experiment, we compare OCAP to CAPTOR and CAPNN, which only show the experimental results of VGG-16 on CIFAR10. The line Original Model in the figure represents the accuracy of each original model reserving the

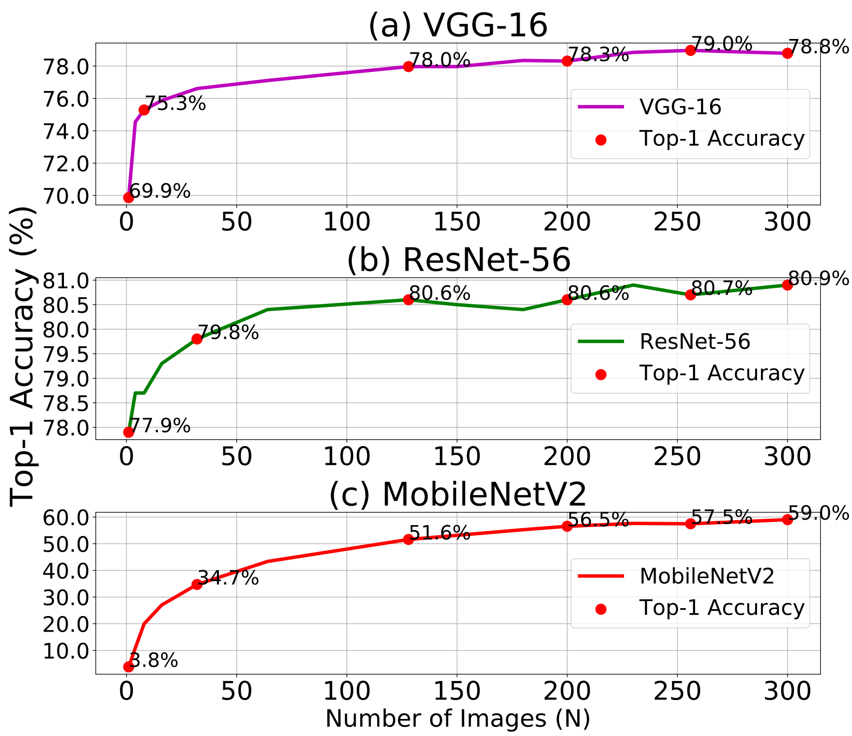


Figure 6: The relationship between the accuracy and the number of images for each class memory. The experiment is conducted on CIFAR100 with 50 classes remained.

same random classes as OCAP model. In other words, we test the well trained model using the reserved classes.

We can see that in both ResNet-56 and VGG-16, OCAP can improve the model accuracy with a lower FLOPs ratio no matter how many classes are remained. Especially for ResNet-56 and VGG-16 with two remained classes, OCAP can achieve 98% accuracy with 43% relative FLOPs ratio and 99% accuracy with 42% relative FLOPs ratio respectively. And for MobileNetV2, when only reserving a small number of classes for the models, OCAP can improve the model accuracy. For example, MobileNetV2 can achieve 96% accuracy with 53% relative FLOPs ratio when reserving two classes. As the number of reserved classes increases, the accuracy gradually drops. That's probably because that VGG and ResNet are both large and redundant models, and they can be pruned more. However, MobileNetV2 is a compact model which has fewer channels and uses depth-wise separable convolution, thus it is hard to prune while guaranteeing the accuracy. In the worst case for MobileNetV2, the accuracy drops by 5% with 35% FLOPs. We think this is acceptable since OCAP is an on-device approach and it trades off the accuracy for efficiency. Comparing to CAPTOR and CAPNN, our method can achieve better accuracy and lower FLOPs ratio no matter how many classes remained.

4.3. Experiments of CIFAR100

Fig. 8 shows the experimental results of CIFAR100. We can see that for CIFAR100 results, the trend is similar to that of CIFAR10. Since CAPTOR and CAPNN do not conduct experiments on CIFAR100, we compare the OCAP results with the original model.

For ResNet-56 and VGG-16, OCAP can improve accuracy while compressing the model, regardless of how many classes are reserved. In particular reserving a small number of classes for the models, OCAP can compress ResNet by 40% with 15% accuracy increase and VGG by 30% with 10% accuracy increase. For MobileNetV2, when less than 40% of the classes are reserved, OCAP can improve accuracy while compressing the model. With the increasing number of reserved classes, the accuracy gradually drops. The accuracy loss is up to 5% (the original 63% to 58% of the pruned model) with 0.78 FLOPs ratio when 80% classes are reserved.

4.4. Experiments of ImageNet

We also evaluate OCAP on VGG with ImageNet as CAPNN does. For ImageNet, it is more challenging to conduct OCAP on resource limited devices due to the limited RAM and computing units. We evaluate OCAP with 2-10 remained classes while CAPNN reports the 2-5 classes remained results of VGG on ImageNet. As CAPNN does not mention the compression ratio or FLOPs, we only compare the accuracy reported in their paper. Furthermore, we follow the same setting in CAPNN and randomly select 10 classes from ImageNet to conduct this experiments. We directly use the pre-trained VGG-16 model from Pytorch model zoo and the accuracy for the original model is 75%.

Fig. 9 plots the experimental results. We can see that, with 2 reserved classes, OCAP can compress the FLOPs

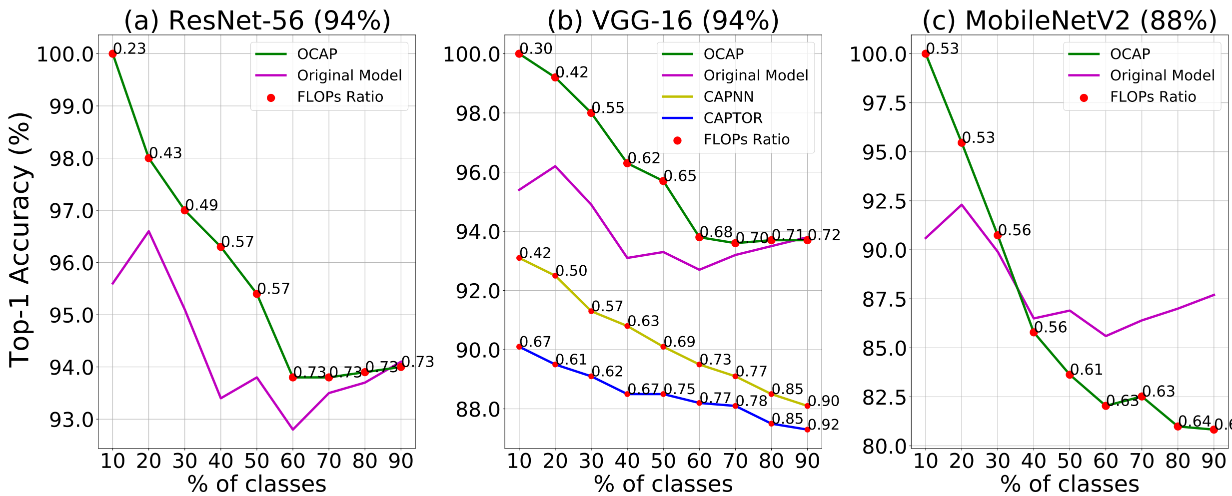


Figure 7: Experimental results of CIFAR10. The results are obtained by randomly selecting different classes several times and then take the average of Top-1 accuracy. The original accuracy of each model is indicated in parentheses.

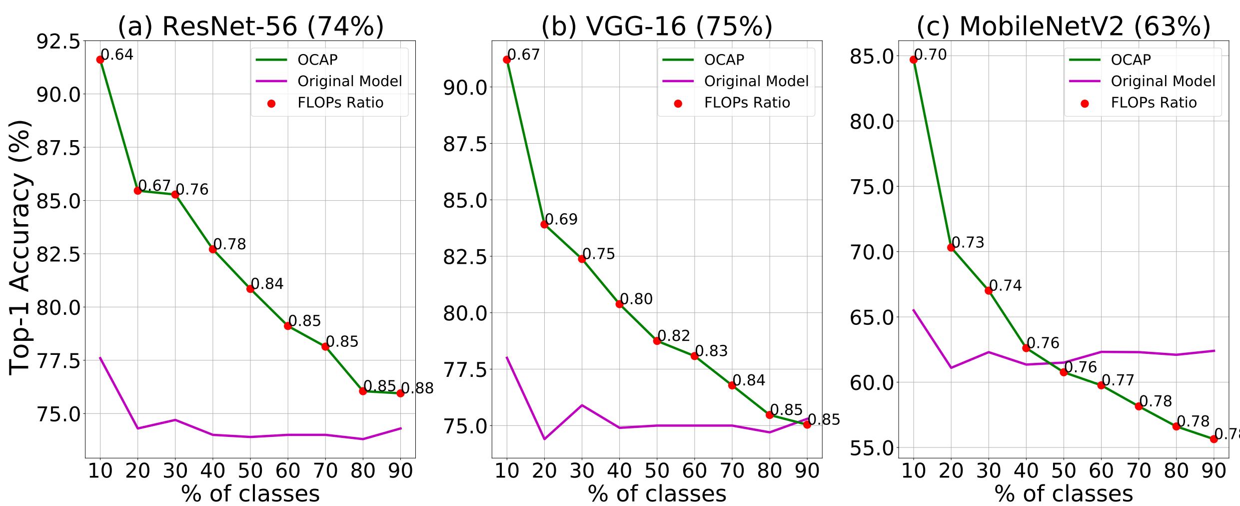


Figure 8: Experimental results of CIFAR100. The experimental setting is the same as that of CIFAR10.

of the original model by 65% under the current setting and at the same time significantly increases accuracy (75% to 98%) comparing to the original model. With the increasing number of reserved classes, OCAP can still achieve a considerable compression ratio while maintaining the accuracy. OCAP always outperforms CAPNN in terms of accuracy.

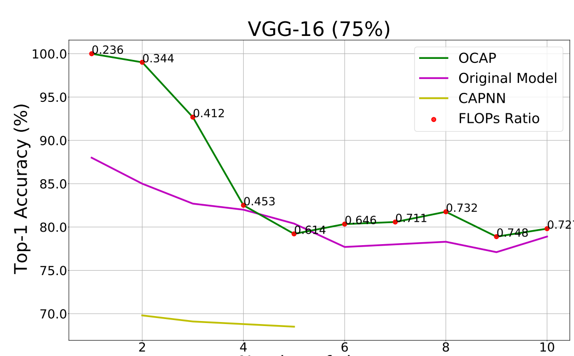


Figure 9: The experimental results for ImageNet.

4.5. Time Overhead and Inference Time on Different Devices

OCAP is an online method for edge systems, so we evaluate the pruning overhead of three parts (including pruning time overhead, data selection time overhead and fine-tuning time overhead) of OCAP on different devices (including one powerful device and three resource-limited devices as shown in Table 2) to show the benefit of class-aware pruning in terms of latency. The results are shown in Table 3, Table 4 and Table 5.

Table 3 shows the results of different reserved classes for CIFAR10. As the pruning procedure of MobileNetV2 with the deep-wise separable convolutional layers is much more complex than VGG and ResNet with the normal convolutional layers, the pruning overhead of VGG and

Algorithm 3: Data Selection

Input: Input image set $\mathcal{X}$, class memory size N, reserved classes C, the maximum number of image replacements N_R, divided ratio d, confidence threshold T

Output: dataset $\mathcal{D}$

Initialize K_c list K with [] for each reserved class
Initialize image set D with [] for each reserved class
Initialize the number of images N_c with 0 for each reserved class
Initialize the number of image replacements N_r with 0 for each reserved class
Initialize total dataset $\mathcal{D}$ with []
for $x_s \in \mathcal{X}$, **do**
    $out, f_s \leftarrow Get(x_s)$ /* Get model output and feature map of current image */
    $l_s \leftarrow Argmax(out)$ /* Predicted class of x_s */
    if $l_s \in C$ *and* $Max(out) > T$ **then**
        if $N_c < N$ **then**
            Compute K_c for x_s by using f_s, d and Eq. (5)
            /* Add image and label to the corresponding image set D */
            Add x_s, l_s to D
            /* Add K_c to the corresponding K_c set K */
            Add K_c to K
            $N_c \leftarrow N_c + 1$
        end
        if $N_c = N$ *and* $N_r < N_R$ **then**
            Compute K_c for x_s by using f_s, d and Eq. (5)
            if $k_c > min(K)$ **then**
                Replace the image, label in D with the smallest K_c by x_s, l_s
                Replace the smallest K_c in K with the new K_c of x_s
                $N_r \leftarrow N_r + 1$
            end
        end
    end
end
Add all image sets D to $\mathcal{D}$
return $\mathcal{D}$

ResNet is much smaller, 13.51s and 13.99s respectively with 20% reserved classes on Jetson Nano, while the pruning time of MobileNetV2 is 52.70s with 20% reserved classes on Jetson Nano. On the other hand, compared to ResNet and VGG, MobileNetV2 is a lightweight model and easier to train, so the fine-tuning overhead of MobileNetV2 is always smaller than ResNet and VGG. For instance, on Jetson Nano, the fine-tuning time overhead of VGG-16 and ResNet-56 is 891.75s and 1099.0s respectively with 20% reserved classes, while the fine-tuning time overhead of MobileNetV2 is 734.04s with 20% reserved classes.

All models can significantly benefit from the class-aware pruning in terms of the inference time and the inference time can be reduced by more than half with 20% reserved classes. For example, for VGG-16 with 20% classes reserved, the inference time is reduced by almost 50% on Jetson Nano (277ms to 142ms) after pruning with an acceptable time overhead (907s) which demonstrates the effectiveness of OCAP. The system can benefit from the inference time reduction over time. With the increase of reserved classes, the inference time of pruned models grows accordingly.

Table 4 shows the results of different reserved classes for CIFAR100. As CIFAR100 is more complicated than CIFAR10, the time overhead increases significantly. In CIFAR100, the pruning time of VGG-16 with 20% reserved classes (20 classes reserved) is 99.62s in Jetson TX2, while that in CIFAR10 is 9.95s. We think it is still acceptable due to OCAP is an one-time pruning scheme on resources limited devices and the benefit of inference time reduction is considerable in the long run.

Table 5 shows the results of VGG-16 with 2 and 3 reserved classes of ImageNet on different devices. For ImageNet, it is more challenging to conduct OCAP due to the limited RAM and computing units. Storing intermediate features for data selection consumes a lot of memory. So we cannot conduct pruning for ImageNet on Jetson Nano with a limited RAM of 4 GB and we increase the swap space of Nvidia Jetson TX2 to cope with the memory consumption. We can see that with 2 and 3 reserved classes, both Jetson TX2 and Jetson AGX Xavier can achieve more than 50% inference time reduction and RTX 2060 Super can achieve more than 40% inference time reduction. Due to the using of swap space, the time overhead of Nvidia Jetson TX2 is relative high (1261s for 2 reserved classes and 2773s for 3 reserved classes). Some efficient training methods may help us to improve the pruning efficiency and reduce memory consumption for large inputs, like low-precise training and early stopping [43]. We leave it for our future consideration.

5. Ablation Study

5.1. Different Pruning Strategies

To compare the two pruning strategies introduced in Section 3.1.2, we test the accuracy and model compression ratio of OCAP-FR and OCAP-AB under the same configuration. And we set a target FLOPs ratio (0.5 for VGG-16 and ResNet-56, 0.6 for MobileNetV2) to evaluate the two algorithms. Table 6 shows the experimental results of two different strategies. We can see that for all three models with different reserved classes, OCAP-AB always achieves a better accuracy, whereas OCAP-FR has lower pruning time overhead. For instance, for VGG-16 with 50% of classes

	Nvidia Jetson Nano	Nvidia Jetson TX2	Nvidia Jetson AGX Xavier	GeForce RTX 2060 Super
AI Performance	0.47 TFLOPs	1.33 TFLOPs	32 TFLOPs	57 TFLOPs
GPU	NVIDIA Maxwell architecture with 128 CUDA cores	NVIDIA Pascal architecture with 256 CUDA cores	NVIDIA Volta architecture with 512 CUDA cores and 64 Tensor cores	NVIDIA Turing architecture with 2176 CUDA cores
CPU	4-core ARM A57 @ 1.43 GHz	2-core Denver 2 64-bit CPU and 4-core Arm® Cortex®-A57 MPCore processor	8-core NVIDIA Carmel Armv8.2 64-bit CPU 8MB L2 + 4MB L3	8-core Intel Core i7-10700F @ 2.9GHz
Memory	4GB 64-bit LPDDR4 25.6GB/s	8 GB 128-bit LPDDR4 59.7GB/s	32GB 256-bit LPDDR4x 136.5GB/s	8GB 256-bit LPDDR6 448GB/s
Storage	microSD	32 GB eMMC 5.1	32GB eMMC 5.1	512G SN730 NVMe SSD
Max Power	10W	15W	30W	200W

Table 2
The detailed specifications of experimental devices.

Model , % of classes	Device	Pruning	Data Selection	Fine-tuning	Inference Time
VGG-16 , 20%	Jetson Nano	13.51s	1.78s	891.75s	277ms → 142ms
	Jetson TX2	9.95s	1.77s	535.60s	111ms → 51ms
	Jetson AGX Xavier	8.86s	1.89s	435.66s	37ms → 16ms
	RTX 2060 Super	1.99s	0.30s	136.99s	11ms → 6ms
VGG-16 , 50%	Jetson Nano	33.59s	4.87s	1648.83s	277ms → 187ms
	Jetson TX2	25.82s	4.56s	1010.97s	111ms → 71ms
	Jetson AGX Xavier	22.36s	4.60s	644.33s	37ms → 23ms
	RTX 2060 Super	4.82s	0.81s	219.68s	11ms → 7ms
VGG-16 , 80%	Jetson Nano	54.17s	7.93s	2324.64s	277ms → 197ms
	Jetson TX2	42.44s	7.25s	1742.94s	111ms → 76ms
	Jetson AGX Xavier	35.49s	7.49s	794.58s	37ms → 25ms
	RTX 2060 Super	7.63s	1.32s	278.33s	11ms → 7ms
ResNet-56 , 20%	Jetson Nano	13.99s	1.63s	1099.00s	417ms → 246ms
	Jetson TX2	10.07s	1.39s	685.36s	176ms → 100ms
	Jetson AGX Xavier	9.35s	1.74s	471.77s	57ms → 41ms
	RTX 2060 Super	2.21s	0.29s	153.92s	21ms → 16ms
ResNet-56 , 50%	Jetson Nano	34.72s	4.25s	2157.93s	417ms → 272ms
	Jetson TX2	25.31s	3.90s	1263.93s	176ms → 124ms
	Jetson AGX Xavier	23.56s	4.83s	714.66s	57ms → 42ms
	RTX 2060 Super	5.39s	0.77s	271.76s	21ms → 17ms
ResNet-56 , 80%	Jetson TX2	40.79s	5.70s	1846.25s	176ms → 133ms
	Jetson AGX Xavier	37.31s	8.10s	968.45s	57ms → 45ms
	RTX 2060 Super	8.48s	1.23s	357.82s	21ms → 18ms
MobileNetV2 , 20%	Jetson Nano	52.70s	5.94s	734.04s	105ms → 55ms
	Jetson TX2	37.64s	5.48s	529.52s	17ms → 11ms
	Jetson AGX Xavier	17.62s	2.24s	260.11s	18ms → 11ms
	RTX 2060 Super	7.52s	1.42s	157.84s	6ms → 5ms
MobileNetV2 , 50%	Jetson Nano	122.87s	15.10s	1179.47s	105ms → 65ms
	Jetson TX2	69.21s	11.36s	765.43s	17ms → 11ms
	Jetson AGX Xavier	43.75s	5.43s	374.54s	18ms → 11ms
	RTX 2060 Super	17.90s	4.20s	250.49s	6ms → 5ms
MobileNetV2 , 80%	Jetson Nano	208.48s	24.54s	1607.68s	105ms → 70ms
	Jetson TX2	101.83s	13.96s	782.00s	17ms → 11ms
	Jetson AGX Xavier	69.83s	8.74s	462.54s	18ms → 11ms
	RTX 2060 Super	28.63s	6.01s	432.81s	5ms → 5ms

Table 3
The time experiments on CIFAR10 on different devices with diverse reserved classed. All the experiments are run multiple times, then we take the average values. Pruning presents the time overhead of pruning procedure including calculating masks, pre-pruning and real-pruning. Data Selection is the time overhead of data selection procedure including calculating the image's K_c and choosing fine-tuning images. And Fine-tuning is the time overhead of fine-tuning procedure with 50 epochs. Inference Time is the average inference time of the original model and pruned model with batch size 100.

remained, OCAP-AB achieves a 4.2% higher accuracy than OCAP-FR with 50% fewer parameters. It demonstrates the effectiveness of OCAP-AB in terms of accuracy. On the other hand, we can see that OCAP-FR can always reach the target FLOPs ratio accurately, while OCAP-AB can only approach the target FLOPs ratio approximately. This is the advantage of OCAP-FR, especially on edge devices with strict resource constraints. Meanwhile, as using the *or* ($\bigvee$) operator to merge masks from different images channel-by-channel, the pruning time overhead, especially the time

Model , % of classes	Device	Pruning	Data Selection	Fine-tuning	Inference Time
VGG-16 , 10%	Jetson Nano	49.74s	9.72s	2112.38s	276ms → 163ms
VGG-16 , 20%	Jetson TX2 Jetson AGX Xavier RTX 2060 Super	99.62s 46.75s 18.63s	18.55s 9.36s 3.55s	1978.01s 813.67s 691.32s	111ms → 85ms 37ms → 28ms 11ms → 8ms
VGG-16 , 50%	Jetson TX2 Jetson AGX Xavier RTX 2060 Super	190.57s 115.62s 48.07s	54.38s 27.08s 10.27s	4263.76s 1698.92s 1339.06s	111ms → 98ms 37ms → 32ms 11ms → 9ms
VGG-16 , 80%	Jetson AGX Xavier RTX 2060 Super	187.63s 79.57s	49.91s 18.66s	2439.94s 2040.00s	37ms → 34ms 11ms → 10ms
ResNet-56 , 20%	Jetson TX2 Jetson AGX Xavier RTX 2060 Super	106.71s 45.00s 20.33s	15.24s 8.89s 3.36s	2249.86s 984.75s 791.56s	175ms → 125ms 57ms → 44ms 23ms → 18ms
ResNet-56 , 50%	Jetson AGX Xavier RTX 2060 Super	115.62s 55.82s	27.05s 10.73s	2055.45s 1622.80s	57ms → 48ms 23ms → 20ms
ResNet-56 , 80%	Jetson AGX Xavier RTX 2060 Super	189.42s 68.33s	49.00s 20.60s	2892.79s 2086.31s	57ms → 49ms 23ms → 20ms
MobileNetV2 , 10%	Jetson Nano	262.68s	31.74s	2402.13s	105ms → 70ms
MobileNetV2 , 20%	Jetson TX2 Jetson AGX Xavier RTX 2060 Super	170.88s 172.51s 72.19s	34.75s 26.61s 15.60s	2822.97s 1182.33s 746.84s	45ms → 31ms 18ms → 12ms 6ms → 5ms
MobileNetV2 , 50%	Jetson TX2 Jetson AGX Xavier RTX 2060 Super	1055.38s 424.59s 181.78s	156.15s 78.98s 41.41s	3699.86s 3283.74s 1598.19s	45ms → 32ms 18ms → 12ms 6ms → 6ms
MobileNetV2 , 80%	Jetson TX2 Jetson AGX Xavier RTX 2060 Super	1632.13s 667.43s 284.90s	261.50s 102.47s 64.81s	5222.86s 2329.35s 2086.20s	45ms → 33ms 18ms → 13ms 6ms → 6ms

Table 4
The time experiments on CIFAR100 on different devices with diverse reserved classed. The experimental settings are the same as Table 3.

Model, Number of classes	Device	Pruning	Data Selection	Fine-tuning	Inference Time
VGG-16, 2	Jetson TX2 Jetson AGX Xavier RTX 2060 Super	265.05s 12.75s 4.87s	22.39s 2.75s 1.56s	974.72s 292.88s 785.12s	681ms → 325ms 344ms → 104ms 66ms → 37ms
VGG-16, 3	Jetson TX2 Jetson AGX Xavier RTX 2060 Super	532.94s 13.40s 6.07s	32.35s 3.19s 2.27s	2209.36s 419.91s 1039.06s	680ms → 336ms 344ms → 120ms 66ms → 41ms

Table 5
The time experiments on ImageNet on different devices with diverse reserved classed.

Model , % of classes	Algorithm	Accuracy	FLOPs Ratio	Parameters Ratio	Remained Filters Ratio	Pruning Time Overhead
VGG-16 , 50%	OCAP-FR OCAP-AB	90.49% **94.70**%	0.50 0.53	0.42 0.21	0.66 0.46	**0.42s** 4.64s
VGG-16 , 80%	OCAP-FR OCAP-AB	87.35% **91.99**%	0.50 0.54	0.42 0.21	0.65 0.45	**0.63s** 7.38s
ResNet-56 , 50%	OCAP-FR OCAP-AB	87.93% **94.40**%	0.50 0.54	0.50 0.70	0.61 0.47	**0.83s** 3.90s
ResNet-56 , 80%	OCAP-FR OCAP-AB	83.38% **89.00**%	0.50 0.52	0.51 0.68	0.61 0.45	**1.13s** 6.18s
MobileNetV2 , 50%	OCAP-FR OCAP-AB	79.69% **83.63**%	0.60 0.61	0.42 0.41	0.45 0.46	**0.68s** 18.81s
MobileNetV2 , 80%	OCAP-FR OCAP-AB	77.97% **80.99**%	0.60 0.64	0.40 0.45	0.44 0.49	**1.16s** 30.11s

Table 6
The experimental results of two different pruning strategies. The experiment is conducted on CIFAR10 with 5 and 8 classed remained. For VGG-16 and ResNet-56 with OCAP-FR, we manually skip first three layers to get a better accuracy. And we only test and compare the pruning time overhead not data selection time overhead nor fine-tuning time overhead, since different pruning strategies only affect the time overhead of computing the mask, which is included in the pruning time overhead.

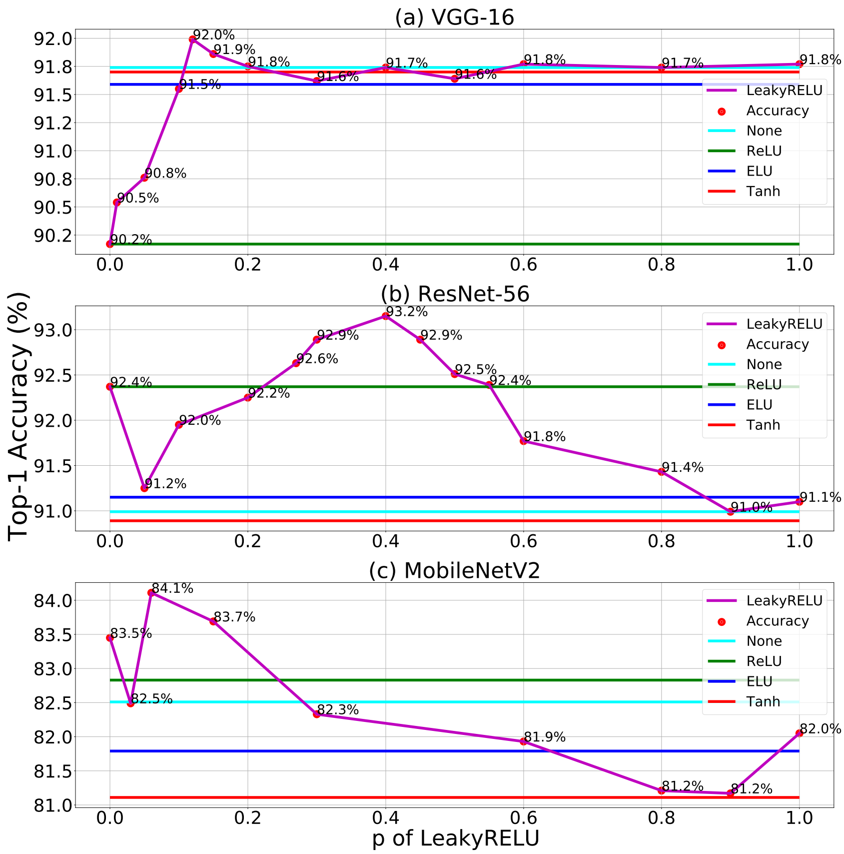


Figure 10: The accuracy of pruned models using different pre-processing functions on CIFAR-10, with saving the first five classes. For each subfigure. the x-axis represents the negative slope p of $LeakyRELU$, and the y-axis represents the Top-1 accuracy of pruned model using the corresponding pre-processing function. As shown in each subfigure, the $LeakyRELU$ with a special p can always have the best accuracy for each model.

overhead of calculating masks of OCAP-AB is much more than that of OCAP-FR. If the target system is sensitive to pruning time overhead and system resources utilization, it is suitable to use OCAP-FR. If there are no such restrictions, OCAP-AB should be the first choice.

5.2. Pre-processing Functions

In our CRV computing, we use $LeakyRELU$, which is one of the most used activation functions. In this section, we justify the usage of $LeakyRELU$. To evaluate the effectiveness of the activation functions used to pre-process the output feature maps of BN layer before calculating the $CRV_{i,j}$ in Eq. (1), we use different pre-processing activation functions, including $ReLU$, $LeakyRELU$ with different negative slope p, ELU, $Tanh$ and $Sigmoid$. Meanwhile, we fix the reserved classes to 5 and make all hyper-parameters the same. Furthermore, for regulating the model compression ratio easily, we use OCAP-FR and adjust the pruning ratio to make sure that the pruned models which are produced by different pre-processing functions have the same FLOPs and parameters.

The experiment results are shown in Fig. 10. For all the models, the pre-processing function $LeakyRELU$ with a special negative slope p (for VGG-16 $p = 0.12$, for ResNet-56 $p = 0.40$,and for MobileNetV2 $p = 0.06$), is able to achieve the best accuracy, compared to other functions. It means that the negative value does reserve some information and the information is beneficial for pruning procedure. In addition, the importance of the negative value varies from model to model.

5.3. The Effectiveness Of Data Selection

To demonstrate the effectiveness of data selection in the fine-tuning process, we evaluate the impact of data selection on different datasets and models, as shown in Table 7. We take the randomly selected dataset ($N_R = 0$) as the baseline in the table. As shown in the table, data selection can achieve an average accuracy improvement of 0.5% compared to the baseline. Especially when the dataset is more complex (like CIFAR100) and the number of reserved images is small (e.g., $N = 16$), data selection can obtain the maximum accuracy improvement. For example, ResNet-56 can achieve the maximum accuracy improvement of 2.42% when CIFAR100 with 20 classes reserved and $N_R = 2N$. We see that data selection process can effectively improve the accuracy of the model after fine-tuning. These results show the importance of data selection.

Dataset	Model , Number of classes	N	N_R	Top-1 Accuracy	Δ-Accuracy
CIFAR10	VGG-16 , 5	16	0 (Baseline)	95.25%	-
			$0.5N$	**95.61%**	**+0.36%**
			$1N$	95.57%	+0.32%
			$2N$	95.49%	+0.24%
		64	0 (Baseline)	95.35%	-
			$0.5N$	95.49%	+0.14%
			$1N$	**95.65%**	**+0.30%**
			$2N$	95.49%	+0.14%
		128	0 (Baseline)	95.52%	-
			$0.5N$	**95.63%**	**+0.11%**
			$1N$	95.55%	+0.03%
			$2N$	95.62%	+0.10%
	ResNet-56 , 5	16	0 (Baseline)	93.87%	-
			$0.5N$	93.99%	+0.12%
			$1N$	**94.29%**	**+0.42%**
			$2N$	93.95%	+0.08%
		64	0 (Baseline)	94.53%	-
			$0.5N$	94.60%	+0.07%
			$1N$	**94.63%**	**+0.10%**
			$2N$	94.61%	+0.08%
		128	0 (Baseline)	94.67%	-
			$0.5N$	**94.85%**	**+0.18%**
			$1N$	94.79%	+0.12%
			$2N$	94.82%	+0.15%
CIFAR100	VGG-16 , 20	16	0 (Baseline)	82.46%	-
			$0.5N$	82.58%	+0.12%
			$1N$	**82.96%**	**+0.50%**
			$2N$	82.66%	+0.20%
		64	0 (Baseline)	83.58%	-
			$0.5N$	83.71%	+0.13%
			$1N$	83.96%	+0.38%
			$2N$	**84.04%**	**+0.46%**
		128	0 (Baseline)	83.96%	-
			$0.5N$	**84.46%**	**+0.50%**
			$1N$	84.31%	+0.35%
			$2N$	84.21%	+0.25%
	ResNet-56 , 20	16	0 (Baseline)	78.56%	-
			$0.5N$	78.84%	+0.28%
			$1N$	80.38%	+1.82%
			$2N$	**80.98%**	**+2.42%**
		64	0 (Baseline)	83.33%	-
			$0.5N$	83.41%	+0.08%
			$1N$	**83.84%**	**+0.51%**
			$2N$	83.56%	+0.23%
		128	0 (Baseline)	83.72%	-
			$0.5N$	83.93%	+0.21%
			$1N$	83.94%	+0.22%
			$2N$	**83.96%**	**+0.24%**

Table 7
The ablation study of data selection. The experiment evaluates the influence of data selection on the pruned model accuracy under different datasets and different models. The N represents the number of images for each class memory, and N_R represents the maximum replacements number of constraining the image replacement times for every reserved class. $N_R = 0$ means no redundant data selection, that is, the fine-tuning images are randomly selected. And we take $N_R = 0$ as the baseline. The experimental results show that data selection can effectively improve the pruned model accuracy.

6. Conclusion

In this paper, we propose OCAP, an on-device class-aware pruning method. The main target of OCAP is to support an on-device pruning which can maximally protect the privacy and reduce the model complexity on edge systems. The experiments show that OCAP demonstrates its effectiveness and efficiency over the original models comparing with the state of the arts. The experimental results show that class-aware pruning can improve the accuracy and compress the model especially when only few classes are remained. And it will not significantly degrade the accuracy even when a large number of classes are remained. To further improve the pruning efficiency, we may need to design low-cost training libraries for edge systems. In OCAP, we target the image classification models that are a backbone for many other computer vision tasks, such as image detection and image segmentation. Thus, in the future, we plan to extend OCAP to these computer vision tasks. However, we can envision that the extension is not trivial and may need substantial modifications on the current OCAP methods.

References

[1] Linhang Cai, Zhulin An, Chuanguang Yang, and Yongjun Xu. Softer pruning, incremental regularization. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 224–230. IEEE, 2021.

[2] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In

Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 1518–1528, 2020.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[4] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[5] Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. On-device machine learning: An algorithms and learning theory perspective. *arXiv preprint arXiv:1911.00623*, 2019.

[6] Chinthaka Gamanayake, Lahiru Jayasinghe, Benny Kai Kiat Ng, and Chau Yuen. Cluster pruning: An efficient filter pruning method for edge ai vision applications. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):802–816, 2020.

[7] Jacob Goldberger, Shiri Gordon, Hayit Greenspan, et al. An efficient image similarity measure based on approximations of kl-divergence between two gaussian mixtures. In *ICCV*, volume 3, pages 487–493, 2003.

[8] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1580–1589, 2020.

[9] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[10] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi 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*, pages 4340–4349, 2019.

[13] Maedeh Hemmat, Joshua San Miguel, and Azadeh Davoodi. Cap'nn: Class-aware personalized neural network inference. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[15] Hengyuan Hu et al. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[16] Shuo Huai, Lei Zhang, Di Liu, Weichen Liu, and Ravi Subramaniam. Zerobn: Learning compact neural networks for latency-critical edge systems. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 151–156. IEEE, 2021.

[17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[20] Guangli Li, Xiu Ma, Xueying Wang, Lei Liu, Jingling Xue, and Xiaobing Feng. Fusion-catalyzed pruning for optimizing deep learning on intelligent edge devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3614–3626, 2020.

[21] Guangli Li, Xiu Ma, Xueying Wang, Hengshan Yue, Jiansong Li, Lei Liu, Xiaobing Feng, and Jingling Xue. Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning. *Journal of Systems Architecture*, 124:102431, 2022.

[22] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[23] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. *Advances in neural information processing systems*, 30, 2017.

[24] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1529–1538, 2020.

[25] Di Liu, Hao Kong, Xiangzhong Luo, Weichen Liu, and Ravi Subramaniam. Bringing ai to edge: From deep learning's perspective. *Neurocomputing*, 485:297–320, 2022.

[26] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[27] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[28] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[29] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[30] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272, 2019.

[31] Manuel Nonnenmacher, Thomas Pfeil, Ingo Steinwart, and David Reeb. Sosp: Efficiently capturing global correlations by second-order structured pruning. *arXiv preprint arXiv:2110.11395*, 2021.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[33] Zhuwei Qin, Fuxun Yu, Chenchen Liu, and Xiang Chen. Captor: A class adaptive filter pruning framework for convolutional neural networks in mobile applications. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 444–449, 2019.

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[35] Pedro Savarese, Sunnie SY Kim, Michael Maire, Greg Shakhnarovich, and David McAllester. Information-theoretic segmentation by inpainting error maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4029–4039, 2021.

[36] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, et al. Sparse r-cnn: End-to-end object detection with learnable proposals. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14454–14463, 2021.

[37] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33:6377–6389, 2020.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[39] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 13029–13038, 2021.

[40] Jianfeng Wang, Lin Song, Zeming Li, Hongbin Sun, Jian Sun, and Nanning Zheng. End-to-end object detection with fully convolutional network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15849–15858, 2021.

[41] Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. Eca-net: Efficient channel attention for deep convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11534–11542, 2020.

[42] Ze-Han Wang, Zhenli He, Hui Fang, Yi-Xiong Huang, Ying Sun, Yu Yang, Zhi-Yuan Zhang, and Di Liu. Efficient on-device incremental learning by weight freezing. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 538–543. IEEE, 2022.

[43] Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

Ma-Ye Da is currently a graduate student at School of Software, Yunnan University, China. He obtained his bachelor degree from School of Software, Yunnan University. His research interests include model compression and pruning on edge devices.

Zhi-Chao Zhao obtained his bachelor's degree in information security from School of Software, Yunnan University, China. He is currently pursuing his master's degree at the School of Computer Science, Chongqing University, China. His research interests include model pruning, adversarial learning.

Di Liu received his BEng and MEng degrees from Northwestern Polytechnical University, Xi'an, China, and the PhD degree from Leiden University, Leiden, The Netherlands. He was an assistant professor at Yunnan University, China and a research fellow at Nanyang Technological University, Singapore. He is currently an associate professor with the Department of Computer Science, Norwegian University of Science and Technology, Norway. His research interests include the fields of edge systems, machine learning on embedded systems and cyber-physical systems.

Zhenli He received the M.S. degree in Software Engineering and the Ph.D. degree in Systems Analysis and Integration from Yunnan University (YNU), Kunming, China, in 2012 and 2015, respectively. He was a postdoctoral researcher at Hunan University (HNU), Changsha, China, from 2019 to 2021. He is currently an Associate Professor with the School of Software, Yunnan University, Kunming, China. His current research interests include edge computing, energy-efficient computing, heterogeneous computing, and machine learning.

Wei Zhou received his PhD degree from the Chinese Academy of Science. Now he is a full professor at School of Software, Yunnan University. His current research interests include distributed data intensive computing and bioinformatics.