F2Zip: Finetuning-Free Model Compression for Scenario-Adaptive Embedded Vision

Puhan Luo
University of Science and Technology of China
Hefei, China
luopuhan@mail.ustc.edu.cn

Jiahui Hou
University of Science and Technology of China
Hefei, China
jhhou@ustc.edu.cn

Mu Yuan
University of Science and Technology of China
Hefei, China
ym0813@mail.ustc.edu.cn

Guangyu Wu
University of Science and Technology of China
Hefei, China
gywu9908@163.com

Yunhao Yao
University of Science and Technology of China
Hefei, China
sa21011190@mail.ustc.edu.cn

Xiang-Yang Li
University of Science and Technology of China
Hefei, China
xiangyangli@ustc.edu.cn

ABSTRACT

With the development of the Internet of Things and artificial intelligence, the deployment and inference of intelligent models have gradually raised concerns. To reduce the huge computation and storage overhead of modern deep neural networks, many studies use model pruning techniques to reduce the model size and computational cost. However, existing pruning techniques usually require model fine-tuning, which incurs high additional overhead, making them difficult to apply to real-world scenarios. In this work, we focus on vision model compression and present F2Zip, a scenario-adaptive finetuning-free pruning framework for embedded devices. First, we propose a scenario complexity measurement that quantifies scenario changes with pixel-level entropy. By analyzing the scenario complexity, F2Zip adaptively evaluates the importance of different channels and layers of the model using only a small amount (tens) of unlabeled data. Then we design a multi-constraint knapsack solver to prune scenario-unrelated redundant channels. We implemented and deployed F2Zip in surveillance scenarios and tested different models on videos collected from both public and real-world sources. Experimental results show that F2Zip is free of model fine-tuning in various scenarios. F2Zip reduces the end-to-end deployment time by 89.8% and reduces energy cost by 79.5%, which shows that F2Zip is computationally friendly for embedded devices. Without fine-tuning and any accuracy degradation, F2Zip achieves up to 50.2% parameter reduction, outperforming baseline methods by 35.1%.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; • **Computing methodologies → Machine learning**.

KEYWORDS

Model Compression, CNN Inference, Finetuning-Free

ACM Reference Format:
Puhan Luo, Jiahui Hou, Mu Yuan, Guangyu Wu, Yunhao Yao, and Xiang-Yang Li. 2024. F2Zip: Finetuning-Free Model Compression for Scenario-Adaptive Embedded Vision. In *ACM Conference on Embedded Networked Sensor Systems (SenSys '24), November 4–7, 2024, Hangzhou, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3666025.3699319

1 INTRODUCTION

Nowadays, the widespread smart surveillance cameras have become one of the most important IoT sensors [1]. These cameras deployed in cities and buildings have enabled many intelligent applications [14], such as crowd counting, behavior analysis, etc. In the traditional computing paradigm [17], the data collected by the camera is transmitted to the cloud, and the results are sent back to the local device after the model inference. However, due to user privacy, latency requirements, unstable network connections, etc., deploying convolutional neural networks (CNNs) to embedded devices (e.g. smart cameras) has become a growing demand [5].

However, the scarce computation and storage resources of embedded devices make it hard to deploy large pre-trained models. Existing works have studied optimizing the computational efficiency of models, including model pruning [12, 13, 24, 33, 41], parameter quantization [7, 21], knowledge distillation [44], input filtering [16, 42, 43], and hardware-level optimization [4]. Model pruning techniques remove redundant model parameters that hardly contribute to inference accuracy. Comparing to other techniques, model pruning does not need to train the model from scratch, and can directly reduce the size and resource occupancy of the model, thereby reducing the inference latency. It is widely used in resource-efficient deep learning applications on embedded platform. Thus, we focus on the optimization of model pruning techniques in this work.

Motivations: finetuning is not always available and has high overhead. Most existing pruning approaches follow a three-stage process, including parameter importance evaluation, pruning, and finetuning (some works combine pruning and fine-tuning into a single stage) [4, 10]. The finetuning step involves collecting labeled data and training models, which leads to a large amount of

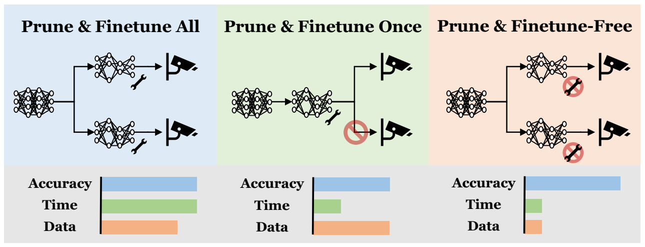

Figure 1: 1)Left: prune and finetune for every device, costing too much; 2)Middle: prune and finetune only one model, hardly satisfied for all devices. 3)Right: prune and finetune-free, efficient and diverse.

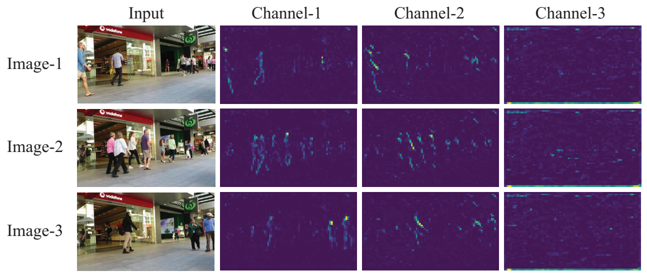

Figure 2: Different channels extract different features.

computational overhead. On the one hand, in the large-scale deployment scenario of smart cameras, the labeled data in practical applications is hard to collect [39], which may make fine-tuning unavailable. Even if fine-tuning is available, performing the fine-tuning process for each camera individually (Fig. 1-Left) would incur huge computational and time costs [2] even using high-end GPUs. On the other hand, different cameras may have various computation and storage limits. Moreover, the scenarios of these cameras are also different, which makes the same vision model have different inference accuracy. Training a single model using all the data (Fig. 1-Middle) cannot meet the diverse accuracy and resource requirements [28]. Though there may be some ways to avoid fine-tuning the model for each device in practice, when the number of devices and models to be deployed is large, the fine-tuning overhead grows sublinearly and still incurs high costs. Therefore, we attempt to develop a finetune-free model pruning algorithm (Fig. 1-Right), thereby speeding up the end-to-end process of model compression. It should be noted that finetune-free compression does not mean that we must remove fine-tuning. This is a capability of the approach that we attempt to realize, but not a requirement. Finetune-free compression means that we may have a chance to achieve comparable accuracy to the original model even pruning without fine-tuning. The opportunity to achieve this ambitious goal comes from our experimental observation that the data distribution in camera-specific scenarios is much narrower than that in general scenarios for model compression. The narrower data distribution enables performing finetuning-free model pruning using fewer resources and achieving comparable accuracy.

Insight: Scenario-specific and requirement-aware channel pruning. For compressing vision models, the key is to distinguish important and unimportant features. The distribution of camera-specific images is very narrow, making the important features concentrated in certain specific CNN channels and layers. Several studies have shown that different channels of the same layer of CNNs are often designed to extract features of different patterns. As shown in Fig. 2, for the object detection task, channel-1 and channel-2 extract more features about people. Conversely, channel-3 only extracts some features about the image background. Removing channel-3 (i.e., background features) usually has little impact on the accuracy. Thus, it is important to determine the contribution of each channel by analyzing whether the features extracted by each channel are relevant or irrelevant to the task. However, only considering the importance of intra-layer channel features for pruning is suboptimal. Because the depth of the CNN layer will also affect the channel features and computational overhead (e.g., the number of parameters and floating-point operations). Given the heterogeneous computation and storage resources in smart cameras, tailoring the pruning rate for each layer allows for a better accuracy-overhead trade-off of vision models.

F2Zip: Based on the aforementioned insights, we introduce F2Zip, a finetuning-free model pruning framework for scenario-adaptive embedded vision. Our framework consists of the following three modules: (1) Scenario complexity evaluation module: We define scenario complexity as the changes in the images captured by a camera device and propose a measurement method named pixel vector entropy to calculate the scenario complexity. (2) Channel importance evaluation module: We propose to evaluate the channel's importance by jointly considering three factors, including the dynamic feature map, static model parameters, and scenario complexity. (3) Layer sparsity allocation module: Given the channel importance scores, we formalize the pruning task as a single-objective multi-constraint optimization problem. We design a knapsack-based solver to decide the pruning rate of channels in each layer.

The contributions of this paper are summarized as follows:

• We find opportunities for finetuning-free model pruning in embedded vision tasks: narrower distribution of scenario-specific data. We then propose a pixel vector entropy method to measure scenario complexity, guiding model compression by quantifying the narrowness of the distribution.

• We propose F2Zip, the first finetuning-free model compression framework for scenario-adaptive vision tasks deployed on embedded devices. F2Zip can adaptively evaluate channel and layer importance according to the scenario complexity, and propose pruning strategies, using only a few unlabeled data. All computation of F2Zip can be done on embedded devices, avoiding privacy and bandwidth issues caused by data transmission.

• We implement and deploy F2Zip on embedded devices, testing different models on both public and our collected datasets. Without finetuning and accuracy loss, F2Zip achieves up to 50.2% parameter reduction, outperforming the existing baseline methods by 35.1%. F2Zip reduces the end-to-end model deployment time by 89.8% and reduces energy cost by 79.5%.

2 BACKGROUND AND CHALLENGE

2.1 Preliminaries of Model Pruning

Modern convolutional neural networks are composed of multiple layers or blocks with different configurations. These layers are supposed to extract varying features of input data. The main model function can be defined as Convolution-Batchnorm-Activation:

$$\mathcal{X}_{l+1} = Act(BatchNorm(\mathcal{W}_l * \mathcal{X}_l)), \tag{1}$$

where $\mathcal{X}_l \in \mathbb{R}^{b \times n_l \times w_l \times h_l}$ is the input of layer i, b is batch size, n_l is number of input channels and h_l, w_l are the height and width of layer i input feature maps. The convolutional weights parameters can be denoted as $\mathcal{W}_l \in \mathbb{R}^{n_l \times n_{l+1} \times k_l \times k_l}$, where the kernel size k_l. Furthermore, the $\mathcal{W}_l$ can be decomposed into $W_l^1, ...W_l^i, ...W_l^{n_{l+1}} \in \mathbb{R}^{n_l \times k_l \times k_l}$, where weights W_l^i extract the input features and output the feature map X_{l+1}^i.

A CNN model often uses different channels W_l^i to extract features of different patterns from the input data. According to the theory of model pruning, most pre-trained models are over-parameterized, which means that the importance of model parameters varies greatly and there may be many redundant parameters (even negative parameters). Consequently, some existing studies have explored structured pruning techniques to eliminate specific channels, i.e. removing X_{l+1}^i. Because the vast majority of the number of parameters and computation of CNN is from the convolution operator, we mainly consider the convolution calculation in this work. We use Params and FLOPs to represent the number of parameters and the amount of computation, defined as:

$$Params = \sum_l n_l \times n_{l+1} \times k_l \times k_l, \tag{2}$$

$$FLOPs = \sum_l b \times n_l \times n_{l+1} \times k_l \times k_l \times h_l \times w_l \times 2. \tag{3}$$

If we can reduce the number of channels (i.e., n_l and n_{l+1}) with structured pruning, the model inference latency and memory access overhead can be decreased accordingly.

2.2 Opportunity in Embedded System

The vision tasks for general scenarios and embedded system scenarios are quite different, including narrower data distributions and limited computational resources. For the general case, we train a single model that has strong generalization ability on a large-scale dataset for general scenarios. That is, an AI model with good performance, should work well in most general cases. Thus, classical pruning techniques assume the pre-trained model is highly over-parameterized. The loss of accuracy caused by removing unimportant parameters can be recovered by fine-tuning. Existing pruning methods study how to maintain generalization ability and accuracy after model pruning and finetuning. On the contrary, in embedded system scenarios, taking large-scale video streaming systems as an example, the number of cameras is often large. However, the view of each camera is limited, only capturing a small region. The shooting background of video frames is unchangeable. Our goal is to prune a large number of different models that are lightweight and scenario-specific. However, existing pruning methods rely on finetuning, leading to heavy data labeling costs and training costs. Besides, it is also difficult to finetune one model and adapt it to different scenarios and resource constraints of camera devices.

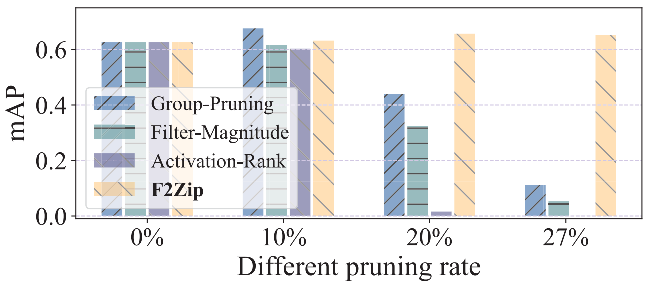

Figure 3: test on MOT20 dataset after pruning YOLOv7 model with different pruning rates and methods.

Therefore, it is desirable to design a pruning framework that only utilizes unlabeled data and does not need finetuning and adapting to various scenarios and device computational resources. Through experiments shown in Fig. 3, we find that in embedded vision tasks, a pruned model also has sufficient generalization ability for the narrower scenario data distribution, even without finetuning. It validates our conjecture that while finetuning-free pruning almost always loses accuracy in general scenarios, it is still possible to maintain accuracy in specific scenarios. Therefore, scenario-adaptive finetuning-free pruning can be considered as a conditional computing method, which dramatically reduces the costs of model deployment for embedded vision tasks.

2.3 Challenges

Although there is an opportunity to achieve finetuning-free pruning by exploiting the characteristics of the embedded system, based on our experimental and empirical analysis, there are still two key challenges to achieving our goal on embedded devices.

1) High sensitivity between the number of parameters and model accuracy. It's challenging to maintain model accuracy while removing a high proportion of parameters, especially when we only exploit some unlabeled data. The existing approaches include neuron merging [15], parameter magnitude evaluation [23], activation rank evaluation [25], group pruning [6], etc. However, most of these methods are designed for general scenarios and are highly dependent on finetuning to recover accuracy. Our experimental results show that when applied to modern vision tasks, these methods are only able to maintain accuracy under low-proportion pruning (10%). As shown in Fig. 3, the mAP performance of existing methods drops sharply to below 0.1 at a pruning rate of 27%. How to find parameters that do not affect the accuracy is still challenging.

2) Diverse scenarios and computational resource requirements. Although the data distribution perceived by each scenario is much narrower than that in the model pre-training stage, there are also differences between these scenarios. To analyze the complexity of the scenario and adjust the pruning strategy become a problem. Meanwhile, different embedded devices have different computing and storage resources, so how to prune the model so that it can

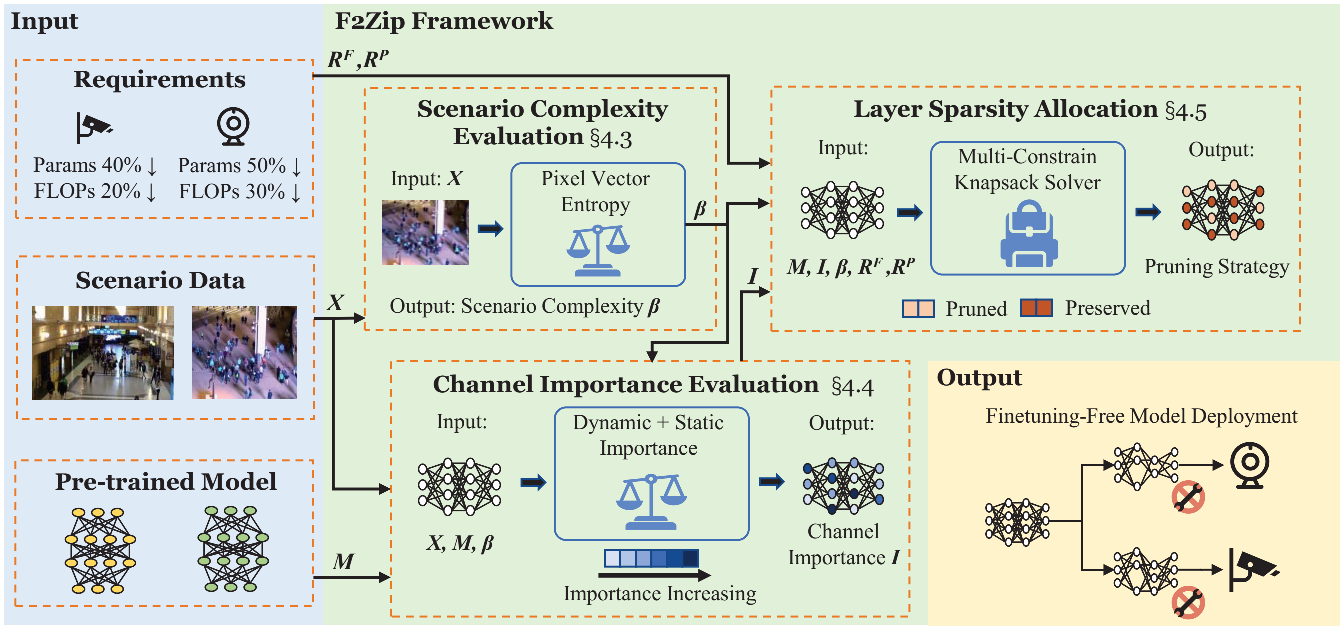

Figure 4: Overview of F2Zip framework. Our framework uses only a few unlabeled data, evaluates the channel importance and layer importance of a given model, and generates a pruning strategy that matches the requirements.

maintain accuracy while meeting different requirements also needs to be further studied.

3 FRAMEWORK DESIGN

3.1 Formalization

The model pruning problem can be formulated as an optimization problem that aims to maximize the accuracy of the pruned model without fine-tuning while satisfying the pruning requirements:

$$\begin{aligned} \max_{M} \quad & Acc(M_p), \\ \text{subject to} \quad & \frac{Cost_i(M_p)}{Cost_i(M)} \leqslant R_i, \quad i = 1, \ldots, r, \end{aligned} \tag{4}$$

where $Acc(\cdot)$ is the testing accuracy evaluation function, and $Cost_i(\cdot)$ is the different cost of models (e.g., Params, FLOPs in Eq. 2 and Eq. 3, etc.). Constraints $\frac{Cost_i(M_p)}{Cost_i(M)} \leqslant R_i$ ensure that the pruned model satisfies all the pruning requirements so that the model can be deployed on a specific device. Assuming that the total number of channels in the model is N, the time complexity of the optimization problem by traversal search is $O(2^N)$ because each channel has two states: pruned or preserved. For modern convolutional neural networks, N is usually about 10^4, and the time cost to find the optimal solution is not acceptable. Even if using reinforcement learning methods to speed up, the cost of training agents is still high [9]. Therefore, we aim to find an approximate solution to this problem in $O(N)$ complexity.

3.2 Framework Overview

As depicted in Fig. 4, to implement finetuning-free model pruning for embedded vision tasks, F2Zip consists of the following modules:

- *Scenario Complexity Evaluation Module*: given inputs scenario data, this module calculates scenario complexity β based on pixel vector entropy (§3.3).
- *Channel Importance Evaluation Module*: this module evaluates the importance I of each channel by combining dynamic feature map importance and static parameter importance according to scenario complexity β (§3.4).
- *Layer Sparsity Allocation*: this module translates the importance and cost of a channel into the value and constraints of a single-objective multi-constraint knapsack problem, which is solved using the multi-constraint dynamic programming algorithm (§3.5).

3.3 Scenario Complexity Evaluation

Evaluating scenario complexity is a prerequisite for designing a scenario-adaptive pruning method. As mentioned earlier, although the perceived data in embedded system scenarios has a narrower distribution than in general scenarios, the degree of narrowness will affect our optimization space (for example, a camera deployed in a high-traffic area may also have a wide data distribution containing various objects). So first we need to evaluate the scenario dataset complexity.

Many existing works have studied the dataset complexity, which is used to estimate the learning difficulty of a dataset [11, 22]. However, most of the existing works focus on classification problems and do not pay attention to the change of datasets. For example, evaluating the complexity of a single image based on gray image entropy and then averaging it does not consider the information interaction between different images [32]. It is not suitable for the problem we are concerned about. In our scenario, we essentially focus on the change degree of the whole dataset (e.g. the fixed proportion of the background, the density of moving objects, etc.), and we need to pay attention to the global information of the dataset. So we propose a method based on pixel vector entropy to evaluate scenario complexity. Given a scenario dataset $\mathcal{X} \in \mathbb{R}^{b \times c_0 \times w_0 \times h_0}$ with total images number b, channels number c_0 and size $w_0 \times h_0$, we extract the pixels of all images at the position (c, w, h) and flatten them into vectors with size b. Then the pixel vector entropy can be defined as:

$$H_{c,w,h} = \sum_{i=0}^{n-1} -p(i|c,w,h) \log p(i|c,w,h), \tag{5}$$

where i represents the n kinds of different pixel values. $H_{c,w,h}$ reflects the amount of information of the dataset $\mathcal{X}$ at the position (c, w, h). After averaging and normalizing different pixel positions, we can calculate the scenario complexity, defined as:

$$\beta = \frac{\sum_{c=0}^{c_0} \sum_{w=0}^{w_0} \sum_{h=0}^{h_0} H_{c,w,h}}{c_0 \cdot w_0 \cdot h_0 \cdot max\_entropy}, \tag{6}$$

where $\beta \in [0, 1]$. The maximum entropy depends on the number of pixel values. Here we give some comprehensible examples. When the images captured by the camera never change, which is the simplest scenario. Since the scenario dataset actually has only one image and the data distribution is very narrow, then the scenario complexity is 0. Another example is that the scenario for a camera deployed on a busy street is usually more complex than the scenario for a camera deployed in a warehouse. The scenario complexity will directly affect the design of pruning strategies in subsequent modules.

3.4 Channel Importance Evaluation

In this module, based on the weights parameters and channel feature map outputs, we propose a hybrid strategy of dynamic importance and static importance to evaluate the importance of each channel under the control of scenario complexity.

Dynamic Importance: As mentioned in Sec. 1 and Fig. 2, different channels in the same layer of a CNN are often designed to extract different features (e.g. background features and object features). However, background features usually have little effect on the accuracy of detection models (even have a negative effect). For a specific scenario, we consider that the channels that tend to extract background features should have lower importance, while the channels that tend to extract object features have higher importance. Therefore, we need a method to judge whether a channel extracts background features or object features. In the embedded vision scenario, there are many similar or the same backgrounds perceived by the smart camera, but the objects are often moving and changing. Based on this observation, we evaluate the importance of the channel by analyzing the changes in the channel feature maps for different input images. Specifically, for a feature map $X \in \mathbb{R}^{b \times w_l \times h_l}$, we have:

$$I_{l,c}^{D} = \frac{\sum_{w=0}^{w_l} \sum_{h=0}^{h_l} \sum_{i=0}^{b} (X_{i,w,h} - \frac{\sum_{i=0}^{b} X_{i,w,h}}{b})^2}{b \cdot w_l \cdot h_l}. \tag{7}$$

Algorithm 1: Channel Importance

```
Input: scenario dataset $\mathcal{X}$, pre-trained model $M$, scenario
       complexity $\beta$.
Output: Channel importance $I$
1  Get the minibatch $\mathcal{X}_1$ from $\mathcal{X}$
2  Get the number of layers of model $L$
3  for $l \in \{1, ..., L\}$ do
4      $\mathcal{Y} = Forward_l(\mathcal{X}_l)$
5      get the parameter weights of $i$-th layer $\mathcal{W}_i$
6      calculate the importance factor $\alpha = \frac{\sqrt{\beta}}{T}$
7      for $c \in \{1, ..., n_l\}$ do
8          get the $c$-th feature map $Y^c$ from $\mathcal{Y}$
9          calculate $I_{l,c}^D$ of $Y^c$ by Eq. 7
10         get the $c$-th parameters $W_l^c$ from $\mathcal{W}_i$
11         calculate $I_{l,c}^S$ of $W_l^c$ by Eq. 8
12         $I_{l,c} = (1 - \alpha) \cdot I_{l,c}^D + \alpha \cdot I_{l,c}^S$
13     end
14     $I_l \leftarrow \{I_{l,1}, ..., I_{l,n_l}\}$, $\mathcal{X}_{l+1} \leftarrow \mathcal{Y}$
15 end
16 $I \leftarrow \{I_1, ..., I_L\}$
17 return $I$
```

Since $I_{l,c}^D$ depends on the input data $\mathcal{X}$, in other words, changes dynamically as the scenario changes, we call it dynamic importance.

Static Importance: Dynamic importance can reflect the tendency of a channel to extract features. However, evaluating importance only considering the data input may lead to a pruned model that has too poor generalization ability and can only fit the data used for evaluation (we do not use fine-tuning to recover accuracy). To retain some pre-trained model generalization ability, we also consider the importance of the model parameters. In summary, we want to keep channels that can not only extract object features but also have good generalization ability. There are many existing techniques for measuring the importance of model parameters, here we used the parameters norm to define the parameter importance. For a parameter weight W, the static importance can be defined as:

$$I_{l,c}^S = \|W_l^c\|_1, \tag{8}$$

where the importance $I_{l,c}^S$ comes from the pre-trained model parameters and does not change from different scenarios, hence it is static importance.

Finally, we use the scenario complexity to adjust the ratio of different importance. Intuitively, if the scenario complexity is low, then the generalization ability requirements can be reduced accordingly. On the contrary, if the scenario complexity is high (i.e. closer to a general scenario), then we need to preserve more of the model generalization ability. Combining the above design, we give Algorithm 1. First, we obtain the feature maps and parameters in the process of forward propagation. Then, we use the scenario complexity to calculate the importance ratio $\alpha = \frac{\sqrt{\beta}}{T}$. Finally, we calculate two kinds of importance, getting the total importance of

Algorithm 2: Knapsack Initialize

```
Input: Model $M$, channel importance $I$, scenario complexity $\beta$
Output: values vector $v$, weights vector $w^F, w^P \in \mathbf{R}^t$
1  Total items $t = 0$
2  for $l \in \{1, ..., L\}$ do
3      Calculate preserved lower bounds
       $\eta = k(1 - (1 - \beta)\sqrt{\frac{l}{L}}) + b$
4      Sort $I_l$ by decreasing order
5      Group $I_{l,\lfloor n_l \cdot \eta \rfloor}, ...I_{l,n_l}$ into $G$ groups
6      for $g \in \{1, ..., G\}$ do
7          $v_t = \sum I_{l,y}, w_t^F = FLOPs(W_l^y), w_t^P = Params(W_l^y)$,
           where $y$ in group $g$
8          $t = t + 1$
9      end
10 end
```

Algorithm 3: Multi-Constrain Knapsack Solver

```
Input: values vector $v$, weights vector $w^F, w^P \in \mathbf{R}^t$,
       requirements $R^F, R^P$
Output: the preserved channel set
1  Set $C^F, C^P$
2  $\gamma^F = \frac{(1-R^F)\cdot FLOPs(M)}{C^F}, \gamma^P = \frac{(1-R^P)\cdot Params(M)}{C^P}$
3  Initiallize the DP matrix $A = \mathbf{0}^{t*C^F*C^P}$
4  for $i \in \{1, ..., t\}$ do
5      for $c^F \in \{1, ..., C^F\}$ do
6          for $c^P \in \{1, ..., C^P\}$ do
7              if $w_i^F > \gamma^F \cdot c^F$ or $w_i^P > \gamma^P \cdot c^P$ then
8                  $A_{i,c^F,c^P} = A_{i-1,c^F,c^P}$
9              else
10                 $A_{i,c^F,c^P} = max(A_{i-1,c^F,c^P}, v_{i-1} + A_{i-1,c^F-\lfloor w_{i-1}^F/\gamma^F \rfloor, c^P-\lfloor w_{i-1}^P/\gamma^P \rfloor})$
11             end
12         end
13     end
14 end
15 $i, c^F, c^P = t, C^F, C^P$
16 while $i, c^F, c^P > 0$ do
17     if $A_{l,c^F,c^P} != A_{l-1,c^F,c^P}$ then
18         $i - 1$ is preserved
19         $c^F = c^F - \lfloor w_{i-1}^F/\gamma^F \rfloor$
20         $c^P = c^P - \lfloor w_{i-1}^P/\gamma^P \rfloor$
21         $i = i - 1$
22     end
23 end
```

each channel. We will discuss the effect of different T on the results in Sec. 4.

3.5 Layer Sparsity Allocation

In this module, we need to remove the channels of each layer according to the channel importance. A naive idea is to uniformly remove the channels with the least importance for each layer according to a given pruning rate. However, uniform pruning does not consider both the computational and storage constraints of devices in specific scenarios. Meanwhile, we have demonstrated through pre-experiments that this idea is feasible only under lower pruning rate constraints in Fig. 3.

Some studies have pointed out that layers of different depths of the model extract feature at different levels of the images. In shallow layers, the model extracts more general features, while deep layers extract more task-related features. It means that for a specific scenario, we can remove more parameters in deep layers without affecting accuracy. Inspired by Shen et al. [35], we further transform the layer sparsity allocation problem into a single objective and multiple constraints knapsack problem. Each channel is treated as an item with value $I_{l,c}$, number of parameters $Params(W_l^c)$, and computation amount $FLOPs(W_l^c)$. The knapsack constraint is the total $Params(M_p)$ and $FLOPs(M_p)$ of the pruned model. The objective is to maximize the total value of the items that are loaded into knapsack, i.e., maximize the total importance of preserved channels. The R^F and R^P corresponding to the $FLOPs(M)$ and $Params(M)$ are usually set according to the system-level metrics. For example, given the computing performance and memory bandwidth of the device, we can estimate the approximate inference latency of the model under a given FLOPs according to the Roofline model. Thus, We can set R^F corresponding to different latency requirement.

We use the dynamic programming algorithm to solve the multi-constrains knapsack problem and optimize it for the actual problem in our work. *1) Scale the constraints and cost proportionally.* The size of the dynamic programming transition matrix depends on the values of the constraints. Since the FLOPs and the number of parameters of the model is a large number, it is impossible to construct the original transition matrix. So we scale the constraints and cost proportionally, using the rounding operation for an approximation of the solution. *2) Channel grouping.* We group the channels of the same layer according to their importance, sum the value and cost of each group of channels as a new item, and update the problem to solve the knapsack problem of the new item to further reduce the time complexity. *3) Control the upper bounds of pruned channels by scenario complexity.* The unrestricted knapsack algorithm may cause all channels of a layer to not be loaded into the knapsack, which will cause the pruned model to fail to inference properly. Therefore we need to control the proportion of channels that enter the knapsack algorithm. We adjust the upper bounds of pruned channels using scenario complexity (e.g. allowing more deep channels to be removed for low-complexity scenarios).

Based on the above design, we propose Alg. 2 and Alg. 3. In Alg. 2, we calculate the total number of channels to be allocated at each layer, and after channel grouping, the corresponding channel importance, FLOPs, and the number of parameters are transformed into the value v and cost w^F, w^P of the knapsack problem. In Algorithm 3, we design a dynamic programming solver for the multi-constrained knapsack problem. By scaling the constraints, we can find an approximate solution that satisfies the constraints

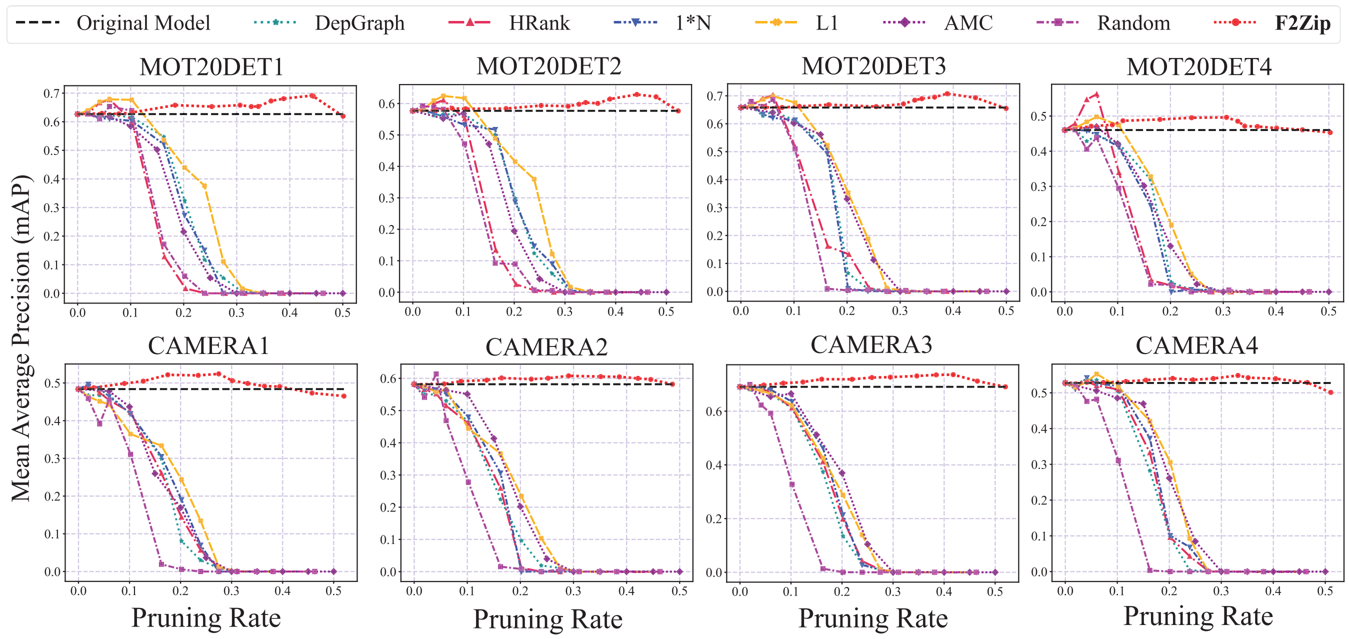

Figure 5: Effect of pruning rate on accuracy in different scenarios and methods.

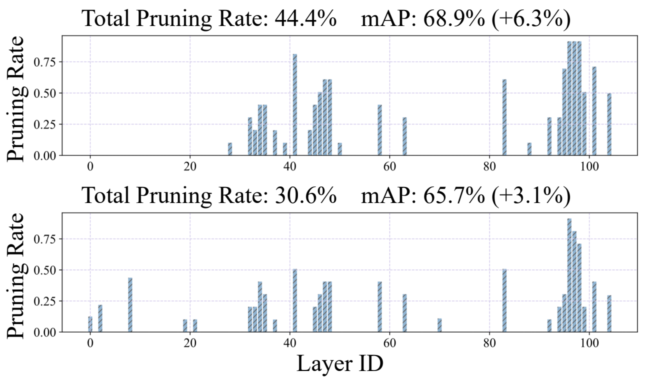

Figure 6: Pruning rate of different layers.

as much as possible in a relatively fast time. We will also discuss the influence of the involved parameters in Sec. 4.

4 EVALUATION

We evaluate the F2Zip framework on various video inference tasks using real embedded vision systems. Our highlights are as follows:

- F2Zip outperforms strong model compression baselines by 35.1% pruning rate and achieves an average 50.2% pruning rate without fine-tuning and accuracy descent.
- F2Zip reduces the end-to-end model deployment time by 89.8% and reduces the energy cost by 79.5% in real systems.
- F2Zip shows effectiveness under different conditions, including pruning rates, task models, and various scenarios datasets. Ablation experiments show the effectiveness of each module in our design. Meanwhile, F2Zip can also be well combined with other complementary approaches.

4.1 Experiments Setup

Datasets and devices: We selected datasets for the most common object detection tasks in embedded vision, including the public dataset *MOT20DET* [3] and the dataset *CAMERA* captured by our cameras deployed in real systems. When collecting the *CAMERA* dataset, all cameras were installed in public areas in the lab, and we obtained proper authorization to conduct experiments. All the collected data will be only used for scientific research. The dataset *CAMERA* includes 112 videos from different scenarios in one week, totaling 56 hours. It has 80 categories to be used in object detection tasks the same as the COCO dataset [27]. We extract some parts of video frames and divide the training set and the validation set to 4:1. Note that we only extract part of the training set to calculate the scenario complexity and channel importance and do not involve real training in F2Zip (because it is a finetuning-free method). We use the common embedded devices Jetson Orin and Jetson Nano as the computing platforms for the cameras.

Baselines: In order to show the generality of our method, we selected some advanced object detection models, including YOLOv7-L, YOLOv7-X [38]. We also use some classical models, including SSD-VGG [29], and EfficientDet-d6 [37]. We choose six channel pruning methods to compare as the baseline: (1) DepGraph [6]. Use dependency graphs to comprehensively group coupled parameters for pruning. (2) HRank [25]. Calculate the rank of feature maps to evaluate the importance for pruning. (3) $1 \times N$ [26]. Remove $1 \times N$ consecutive output kernels with the same input channel index. (4) L1 [23]. Calculate the L1-Norm of each convolution filter for pruning. (5) AMC [9]. Use reinforcement learning agents to search

Table 1: Pruning results on different models.

Model	Dataset	Δ mAP	Δ FLOPs	Δ Params
YOLOv7-X	Random	+0.3%	-7.65%	-7.89%
	L1	-0.3%	-14.12%	-14.96%
	AMC	-0.2%	-13.40%	-14.75%
	HRank	+0.2%	-10.24%	-10.28%
	1*N	+0.3%	-12.05%	-12.44%
	DepGraph	+0.3%	-14.55%	-14.81%
	F2Zip	**+0.1%**	**-26.31%**	**-51.80%**
SSD-VGG	Random	+0.1%	-6.04%	-6.52%
	L1	+0.0%	-12.46%	-13.01%
	AMC	-0.1%	-11.13%	-11.62%
	HRank	-0.4%	-9.86%	-10.25%
	1*N	+0.3%	-11.93%	-12.30%
	DepGraph	-0.4%	-13.18%	-13.69%
	F2Zip	**+0.2%**	**-24.41%**	**-48.93%**
EfficientDet-d6	Random	-0.2%	-6.38%	-6.75%
	L1	+0.0%	-12.78%	-13.23%
	AMC	+0.1%	-12.39%	-13.01%
	HRank	-0.1%	-8.50%	-8.82%
	1*N	-0.2%	-12.29%	-12.53%
	DepGraph	+0.3%	-13.10%	-13.38%
	F2Zip	**-0.2%**	**-21.27%**	**-49.13%**

for the optimal pruning rate of each layer. (6) Random. Randomly select a set of channels to prune. (7) MSM [34]. Use the matrix similarity to guide model pruning. (8) SNIP [20]. Use the connection sensitivity to prune the network at initialization (PaI).

4.2 Overall Performance

In this section, we compare F2Zip with strong baselines under different pruning rates and various models. We also evaluate the deployment cost and inference latency of different methods on different devices. We choose two aspects to demonstrate the superiority of F2Zip. First, by controlling the same deployment cost and pruned model accuracy, F2Zip achieves higher pruning rates. Second, by controlling the same pruning rates and pruned model accuracy, F2Zip has a lower deployment cost.

Different pruning rate. We pruned the YOLOv7-L model on eight scenarios datasets respectively and reported the accuracy of the model without fine-tuning after pruning under different methods and different pruning rates. As shown in Fig. 5, when the pruning rate is low, all methods can almost maintain the accuracy. When the pruning rate exceeds 10%, the baseline method starts to show a non-negligible loss of accuracy. When the pruning rate exceeds 30%, the accuracy of all baseline methods is close to 0, and the model pruned by F2Zip can even be higher than the original model, which reflects a great advantage of F2Zip. We think that it is because pre-trained models are usually over-parameterized for specific scenarios, and existing methods lack a scenario-based parameter importance analysis. F2Zip can accurately find the parameters that are redundant for a specific scenario, and thus is able to remove 50% of the parameters while still maintaining accuracy. In addition, we also found that in some cases, increasing the pruning rate actually improved the accuracy, which indicates that there are indeed a large number of features and their corresponding channel parameters that negatively affect the model prediction accuracy. F2Zip can accurately remove these channel parameters that interfere with the prediction of the model. Meanwhile, we found that the classic L1-norm method outperforms some of the recent techniques surprisingly. This indicates that the advantage of the baseline method is highly dependent on fine-tuning to recover accuracy. In all scenarios, F2Zip achieves 45% parameter reduction on average without accuracy drop and has a strong generalization ability for different scenarios.

Table 2: Average end-to-end deployment time cost of different methods (hours per model).

Method	Data Collecting and Labeling	Pruning	Finetuning	Total
AMC	3.69	1.0	0.83	5.52
HRank		0.3	1.12	5.11
Random		0.01	1.36	5.06
L1		0.02	1.23	4.94
MSM		0.2	1.02	4.91
1*N		0.1	0.99	4.78
SNIP		1.02		4.71
DepGraph		0.02	0.82	4.53
F2Zip (Orin)	**0.16**	**0.3**	**0**	**0.46(-89.8%)**
F2Zip (Nano)	**0.16**	**0.55**	**0**	**0.71**

To further analyze the pruning strategy implemented by our method, we selected two results for visualization. As shown in Fig. 6, our method shows a large difference between layers, compared to the common uniform pruning. For channels of shallow layers, we almost do not remove them, and most of the removed channels are concentrated in the middle and deep layers, which is consistent with our ideas about the role of features. However, it is surprising that the pruning rate of the last few layers reaches 90%, which indicates that the parameter redundancy is higher than expected in certain scenarios.

Different models. To verify the universality of our method, we tested more detection models, including models with different sizes of the same structure (YOLOv7-X) and models with different architectures (SSD-VGG and EfficientDet-d6). We adjust the accuracy of the pruned model to be close to the original model, and statistics the maximum possible pruning rate, Table 1 shows the average results of different methods on the eight datasets. Since existing methods are highly dependent on fine-tuning to recovery accuracy, they can only achieve 7% - 15% pruning rate without fine-tuning. Our method significantly outperforms the baseline on all three models, achieving an average of 23.99% computation reduction and 49.95% parameters reduction. The difference between the results on different models is mainly due to the design of different models for specific architectures (e.g. detection heads are generally not pruned).

End-to-end deployment cost. We compare the pruning deployment end-to-end time and energy cost between the baseline method and our method. We divide the deployment time into three

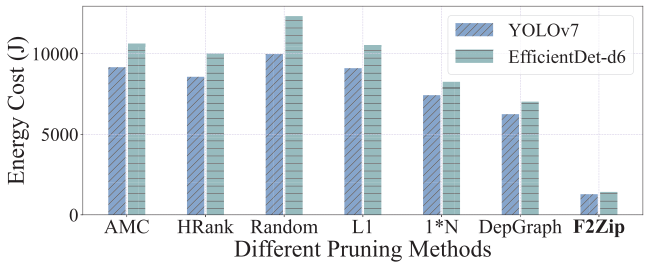

Figure 7: Average energy costs of different pruning methods to deploy one model on the device.

parts, namely data collecting and labeling, pruning, and fine-tuning. For the first part, we calculated the total time by referring to the average annotation time (19s per image) reported in a fast annotation work [18] and the average training dataset size (about 700 images) of our dataset. For the pruning and fine-tuning parts, we tested the actual time costs on the Jetson Orin and Jetson Nano (all baseline methods cannot be finetuned on Nano due to out of memory), guaranteeing similar accuracy. As shown in Table 2, we show the average deployment time of different methods (single model for single scenario). Although our method spends more time on the pruning phase, it saves a lot of time for the data labeling and fine-tuning phase (we only need dozens of images to obtain the feature maps). Compared with the baseline, our method saves 89.8% end-to-end time overhead. Meanwhile, F2Zip is the only method that can realize end-to-end model pruning and deployment on weak devices like Jetson Nano.

Additionally, we also test the average energy costs of different pruning methods to deploy one model on the device. We control the deployed model inference accuracy and pruning rate to be the same. As shown in Fig. 7, our method reduces the energy costs by 79.5% compared to the baseline. Existing methods consume a lot of energy due to the need for finetuning. However, the energy costs of our method mainly come from one feedforward of the model and the knapsack problem solver. When switching models of different sizes, the increase in energy costs of our method comes only from the linear increase in solution time caused by the increase in the total amount of items in the knapsack problem. However, the finetuning-based methods are also affected by the unstable model convergence time, and the increased energy costs is more difficult to predict.

Combination with complementary technologies. In addition to model pruning, parameter quantization, and acceleration frameworks are two other commonly employed techniques for speeding up inference. To explore the effect of combining our approach with other complementary optimization techniques, we test the actual inference acceleration effect of F2Zip on Jetson Orin. We control the model accuracy to be close after pruning. As shown in Table 3, our method can be well combined with the existing techniques to obtain a higher speedup ratio. When using TensorRT+FP16 inference, F2Zip can improve the inference speed of the compressed model by 15.2%. When using TensorRT+INT8 inference, F2Zip can improve the inference speed of the compressed model by 21.3%. After using the combination of F2Zip+TensorRT+INT8 together, the pruned model can achieve an inference speed of up to 2.36 × of the original model.

Table 3: Average inference latency of different methods combination.

Model	Method	Latency (ms)
YOLOv7	PyTorch	65.2
YOLOv7	TensorRT+FP16	45.5
YOLOv7	TensorRT+INT8	35.1
YOLOv7 + DepGraph	PyTorch	61.9
YOLOv7 + DepGraph	TensorRT+FP16	39.3
YOLOv7 + DepGraph	TensorRT+INT8	30.0
YOLOv7 + F2Zip	PyTorch	61.5
YOLOv7 + F2Zip	TensorRT+FP16	38.6
YOLOv7 + F2Zip	TensorRT+INT8	27.6

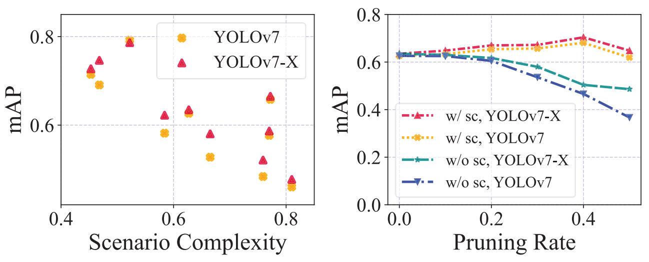

Figure 8: Effect of scenario complexity (sc) with different datasets, model, and pruning rate.

4.3 Micro-Benchmarks

In this section, we explore the impacts of some variables and parameters involved in the F2Zip framework.

Scenario complexity. In order to verify the rationality of our proposed scenario complexity evaluation method, we explore the relationship between scenario complexity and inference accuracy. As shown in Fig. 8, we evaluate the complexity of 10 scenario datasets and test the accuracy under different models. In general, dataset complexity and test accuracy show a negative correlation, which is consistent with our assumption. This means that complex scenarios require more parameters and a higher generalization ability model to achieve the same accuracy compared with simple scenarios. We also find that using scenario complexity improves 6% mAP on average compared to not using it, as shown in Fig. 8. It shows the effectiveness of the scenario-adaptive pruning strategy designed by F2Zip.

Data requirements. In both the scenario complexity evaluation module and the channel importance evaluation module, we use the scenario dataset. Here we investigate the highest pruning rate that F2Zip can achieve without accuracy descent when using datasets of different sizes. As shown in Fig. 9, F2Zip removes only a few parameters like the baseline when the dataset size is small. This is because too little data cannot reflect the data distribution and complexity of the scenario. We cannot accurately evaluate the importance of each channel, and some important channels may be removed. When we

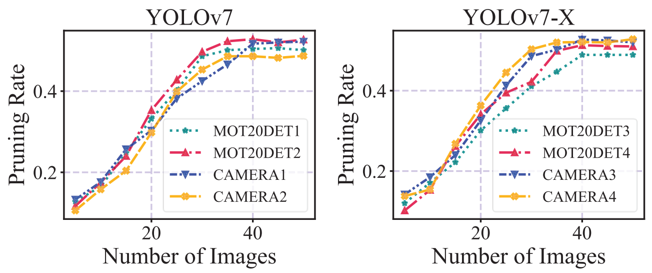

Figure 9: Effect of number of images in different models and scenarios.

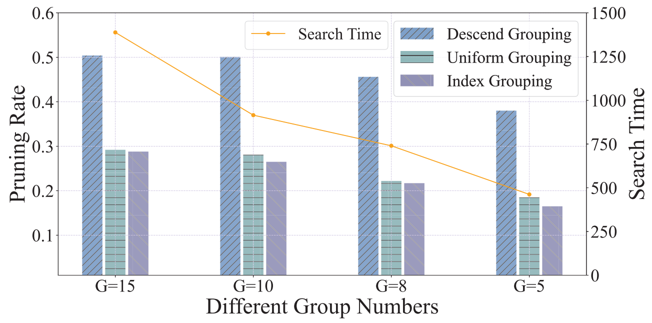

Figure 10: Effect of grouping strategies and numbers on pruning rate and search time.

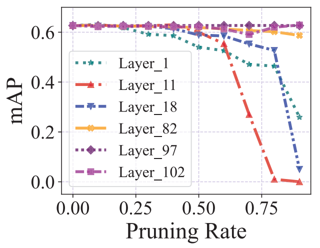

Figure 11: Effect of using different pruning rates on accuracy under different layers.

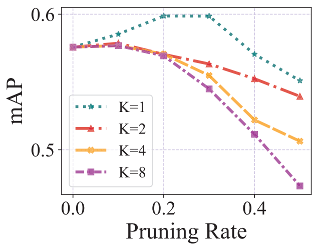

Figure 12: Effect of data distribution with different narrowness on accuracy.

collect more images, the correctness of F2Zip in finding redundant features also increases gradually so that it can remove a higher proportion of parameters. Based on our comprehensive evaluations across diverse scenarios and models, we determine that F2Zip only requires a dataset of about 40 unlabeled images as input to achieve significant pruning rates. Therefore, F2Zip can significantly reduce deployment time compared to finetuning-based approaches that require large amounts of labeled data.

Grouping strategy. In algorithm 2, when initializing the knapsack problem, we first group the channels to reduce the solving scale of the problem and then reduce the end-to-end deployment cost. However, different grouping strategies and the number of groups will affect the final result. For the grouping strategy, we explore three possible designs. *1) Descending grouping.* Consider the importance within the group and aggregate the channels for extracting similar features. For each layer of channels, we select successive $\lfloor n_l \cdot \eta / N \rfloor$ channels to be grouped in descending order of importance. *2) Uniform grouping.* Consider the diversity within the group and aggregate the channels for extracting different features. For each layer of channels, we select one channel for every $\lfloor n_l \cdot \eta / N \rfloor$ into groups in descending order of importance. *3) Index grouping.* Grouping directly in the natural order of the original model.

As shown in Fig. 10, we compared the maximum pruning rate without accuracy drop for different numbers of groups. The descending grouping strategy can achieve the highest pruning rate. It indicates that aggregating similar feature channels can remove redundancy to the greatest extent and preserve channels that extract the object features. Additionally, we also found that as the number of groups decreased, the search time decreased, and the pruning rate also decreased. This is because the number of groupings determines the number of available values for the pruning ratio of each layer. The smaller number of groupings means more coarse-grained pruning rate choices, resulting in worse solutions. Based on the above conclusions, we set the number of groups to 10, which achieves a better trade-off between the pruning rate and search time.

Algorithm parameters. We discuss the influence of the parameters involved in the algorithm, including T for controlling the importance weights in Alg. 1 and k and b for controlling the upper bounds of pruned channels in Alg. 2. As shown in Fig. 15, the effect of different T is mainly reflected in the degree of accuracy reduction at a high pruning rate, because T changes the weights of dynamic importance and static importance. The setting of T mainly depends on the magnitude of the weight norm and the variance of the feature map. The value of b affects the upper bound of the pruning rate for channels in all layers, while k affects the decreasing rate of the pruning rate when the layers become deeper. When the value of b is too large, the algorithm gradually degenerates to uniform pruning when the pruning rate is high, because too many channels are forced to be retained, so the accuracy will show a large drop. The effect of k is mainly reflected in the slight decrease in accuracy when the pruning rate is increased, which may be caused by removing a part of shallow channels. But the effect of changing k is far less than that of changing b. Choosing too high b value should be avoided in practical systems.

Different layers. In order to explore the sensitivity of different layers to pruning rate, we selected layers with different depths and observed the change of model inference accuracy with a single-layer pruning rate. As shown in Fig. 11, when the pruning rate is low, pruning a single layer has almost no effect on the inference accuracy of the model. However, when the pruning rate exceeds 50%, non-negligible accuracy loss starts to appear for shallow layer pruning. When the pruning rate reaches 90%, the accuracy of shallow layer pruning seriously decreases. However, the accuracy can still be maintained for deep layer pruning. This is consistent with our assumption on pruning rate allocation, since deep layers usually extract task-related features, which often have large redundancy, and only a small number of task-related features are needed to maintain accuracy in a specific scenario. However, the general

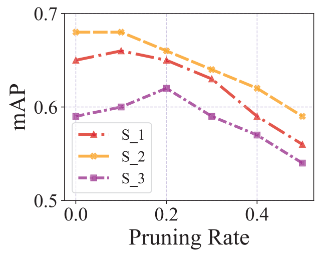

Figure 13: Effect of using different pruning rates on accuracy under static scenarios.

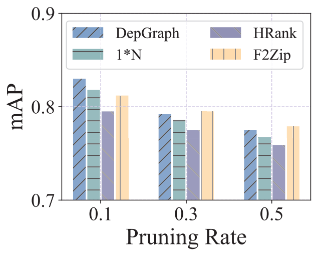

Figure 14: Effect of finetuning after pruning on accuracy.

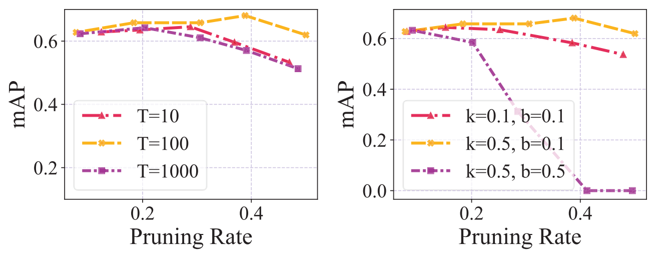

Figure 15: Effect of different parameters in F2Zip.

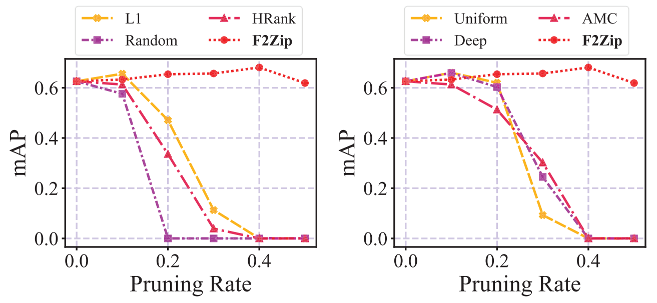

Figure 16: Ablations of channel importance evaluation and layer sparsity allocation module.

features extracted by the shallow layers usually have a great impact on the final accuracy.

Data distribution. We also explore the effect of data distribution with different narrowness on F2Zip pruning. To simulate the data distribution with different narrowness, we use the images from K data sources (i.e., K cameras) and combine them into a new scenario dataset. As shown in Fig. 12, we show the average accuracy for different data distributions with different pruning rates. We can find that the accuracy of F2Zip gradually decreases as the data distribution gradually widens. This is because the scenario becomes more complex and the model needs stronger generalization ability. Meanwhile, we also find that while the pruning rate increases, the influence of data distribution on accuracy becomes more obvious.

Static scenario. We explored the performance of F2Zip in some extreme cases (i.e., almost completely static scenarios). We construct a simulated static scenario dataset by adding some random noise to a single picture. As shown in Fig. 13, the three static scenarios show an average accuracy decrease of 7.6% when the pruning rate increases. We think that this is because the static scenarios lack enough dynamic features, so F2Zip can not judge the importance of the features well. In the future, we also intend to continue to improve our method to adapt to more scenarios.

Finetuning after pruning. As we explained in the motivation section, finetuning-free compression is a capability of F2Zip, but not a requirement. Thus, F2Zip and fine-tuning are not in conflict, but complementary. As shown in Fig. 14, we finetuned the models pruned by different methods for 20 epochs. Compared with baseline, F2Zip improves the accuracy by 0.6% on average. We believe that it is because the model with higher accuracy pruned by F2Zip is equivalent to providing a better initial state of the model for subsequent finetuning. Therefore, assuming that fine-tuning is necessary and available (e.g., online continuous learning after deployment). In this case, our method is also helpful for fine-tuning, which reduces the deployment overhead from another perspective (i.e., achieving higher accuracy at the same cost).

4.4 Ablation Studies

Channel importance evaluation module. We replace the channel importance evaluation module in F2Zip with some baseline methods, keeping the other modules in F2Zip unchanged, and study the impact of our proposed hybrid strategy on evaluating the importance of channels in specific scenarios. Fig. 16 shows that at a pruning rate of 20%, the existing importance evaluation methods all show an accuracy decrease of more than 15%. When the pruning rate exceeds 30%, all baseline methods are already unavailable. This indicates that our proposed hybrid strategy is significant for scenario-oriented channel importance evaluation.

Layer sparsity allocation module. We use a similar research approach for the layer sparsity allocation module. The *Uniform* method sets the same pruning rate for all the layers. The *Deep* method sets a higher pruning rate for the deeper layers. The *Shallow* method sets a higher pruning rate for the shallower layers. We find almost no accuracy loss of *Uniform* and *Deep* methods at a pruning rate of 20%. However, when the pruning rate exceeds 30%, serious accuracy loss begins to appear. This indicates that although a properly designed channel importance evaluation method can maintain accuracy under a low pruning rate, there is still different redundancy between layers. As the pruning rate increases, adaptively allocating the sparsity of each layer based on the scenario can remove redundant parameters more accurately.

5 RELATED WORK

Magnitude-based Pruning. One of the most important areas of model pruning is magnitude-based pruning, which measures the importance of model parameters in terms of model weights, activations, gradient magnitudes, and other related statistics. Lecun et al. [19] proposed to optimize brain damage. They calculate the second-order derivative of the loss function on different parameters to represent the contribution of the parameters. Lin et al. [25] proposed to use the rank of the feature maps to calculate the importance of different feature maps. Polyak et al. [31] proposed to use the variance of channels as the importance. Han et al. [8] designed a fine-grained pruning method by setting the weights smaller than a specific threshold to 0. Li et al. [23] calculated the L1-norm of each filter as the importance. Fang et al. [6] devised a group importance metric to eliminate entire sets of filters and associated elements across multiple layers.

Learning-based Pruning. Recently, numerous studies have delved into the automation of model pruning by employing techniques such as reinforcement learning [36]. For instance, the AMC framework [9] introduced the compression rate as the action space and utilized reinforcement learning to explore the optimal pruning strategy automatically. MetaPruning [30] employed meta-learning, training a meta-network to explore diverse pruning networks under varying constraints. Overall, learning-based techniques for network pruning often yield better model architecture compared to heuristic methods.

Hardware-based Pruning. Hardware-based pruning mainly considers the combination of hardware features and pruning algorithms to achieve higher speedup on embedded devices. Yang et al. [40] conducted experiments to record the latency of various layer configurations in hardware. This data was utilized to estimate the latency of the pruned model, contributing to the reduction of computational costs during the pruning process. Shen et al. [35] utilize a latency lookup table to track the potential for latency reduction and a global saliency score to measure the decrease in accuracy. Liu et al. [28] used DQN to realize a trade-off between accuracy and resource constraints according to the user commands.

Uniqueness. Unlike existing methods, we focus on the problem of model compression deployment cost in practical embedded vision tasks. Our proposed framework F2Zip mainly considers high-proportion model pruning without fine-tuning and accuracy drop. By jointly designing scenario-specific channel importance evaluation metrics and layer sparsity allocation methods, F2Zip can accurately remove useless parameters to the specific scenario. F2Zip enables the pruned model to quickly deploy in different scenarios and devices. It has great significance for the large-scale deployment of models in embedded vision systems.

6 DISCUSSION

Extension for other models In this work, we mainly study the pruning without fine-tuning for CNN models. We expect that F2Zip can also be applied to models with Transformer architecture, such as Swin-Transformer, DETR, etc. Some studies have pointed out that different heads in the multi-head attention module also have the nature of extracting different features, which is highly similar to the channel in the CNN model. Therefore, evaluating the redundancy of the head can be considered to achieve the pruning of the Transformer-based vision model.

Extension for other tasks and scenarios. We also consider how F2Zip can be extended to other different vision tasks. For example, we think it is possible to extend to semantic segmentation tasks. Many current model architectures support both object detection and instance segmentation tasks (e.g. YOLOv5, YOLOv7), which means that the importance of the intermediate features can be analyzed in a similar way, then enabling pruning without fine-tuning. Meanwhile, we also consider extending F2Zip to the scenario where the camera may be not stationary. We choose the fixed IP camera as an example because this is the most common embedded vision device and it is easy to introduce our idea. Moreover, in the future we consider extending our approach to the devices which can move in a small area. For example, a robot moving in a room is able to equally perceive a large number of images with similar backgrounds, the same as in our case setting. Possible challenges include how to design an appropriate complexity measurement for mobile scenarios.

Scalability. We consider how to apply F2Zip to large-scale and complex scenarios (e.g., city surveillance scenarios where lighting and weather conditions change over time). Using model switching is a possible solution. By designing a simple scenario change detection mechanism, F2Zip can quickly collect new unlabeled data, and obtain the new models. The models in the model pool will soon gradually cover different conditions of the same scenario. Additionally, we aim to reuse pruning strategies for similar scenarios (e.g. different views of the same area) to reduce deployment costs in large-scale settings.

Limitations. During the algorithm design and experiments, we also found some limitations where F2Zip can be improved. For example, we use some approximations to speed up the solution of the multi-constrained knapsack algorithm. However, it may result in the solution not fully satisfying all constraints. Therefore, in practice, we need to increase the requirement to eliminate the error caused by the approximation. In addition, we find that reducing the number of parameters and computation of the model is not fully proportional to the actual acceleration effect, which is caused by the hardware structure and system implementation. Thus, we also further consider the integration of hardware features to guide the pruning algorithm design of the model.

7 CONCLUSION

In this paper, we propose F2Zip, a finetuning-free model pruning framework for scenario-adaptive embedding vision. F2Zip evaluates scenario complexity, calculates channel importance based on dynamic feature maps and static weights, and finally uses a multi-constraints knapsack solver to allocate pruning rates to each layer. We validate F2Zip on public datasets and real-world scenarios. Extensive experiments show that F2Zip achieves up to 50.2% parameter reduction, outperforming the strong baseline methods by 35.1%, without finetuning and accuracy degradation. F2Zip shows effectiveness under different conditions. Experiments on real devices show that F2Zip can reduce the end-to-end deployment time by 89.8% and reduce the energy cost by 79.5%, which is friendly for embedded devices.

ACKNOWLEDGMENTS

We thank the Sensys reviewers for their constructive comments. The research is partially supported by National Key R&D Program of China under Grant No. 2021ZD0110400, Innovation Program for Quantum Science and Technology 2021ZD0302900 and China National Natural Science Foundation with No. U23A20308, 62132018, 62231015, "Pioneer" and " Leading Goose" R&D Program of Zhejiang, 2023C01029, and 2023C01143, and Fundamental Research Funds for the Central Universities.

REFERENCES

[1] 2023. Hikvision. https://www.hikvision.com/cn/products/Front-End-Product/Fixed-Camera/

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-All: Train One Network and Specialize it for Efficient Deployment. arXiv:1908.09791 [cs.LG]

[3] Patrick Dendorfer, Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, Ian Reid, Stefan Roth, Konrad Schindler, and Laura Leal-Taixé. 2020. Mot20: A benchmark for multi object tracking in crowded scenes. *arXiv preprint arXiv:2003.09003* (2020).

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

[5] Lachit Dutta and Swapna Bharali. 2021. Tinyml meets iot: A comprehensive survey. *Internet of Things* 16 (2021), 100461.

[6] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. DepGraph: Towards Any Structural Pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 16091–16101.

[7] Yunhui Guo. 2018. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752* (2018).

[8] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* 28 (2015).

[9] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*. 784–800.

[10] Yang He and Lingao Xiao. 2023. Structured pruning for deep convolutional neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence* (2023).

[11] Tin Kam Ho and M. Basu. 2002. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24, 3 (2002), 289–300. https://doi.org/10.1109/34.990132

[12] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250* (2016).

[13] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. 2018. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 709–718.

[14] Xiaoheng Jiang, Li Zhang, Mingliang Xu, Tianzhu Zhang, Pei Lv, Bing Zhou, Xin Yang, and Yanwei Pang. 2020. Attention Scaling for Crowd Counting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[15] Woojeong Kim, Suhyun Kim, Mincheol Park, and Geunseok Jeon. 2020. Neuron Merging: Compensating for Pruned Neurons. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 585–595. https://proceedings.neurips.cc/paper_files/paper/2020/file/0678ca2eae02d542cc931e81b74de122-Paper.pdf

[16] Rui Kong, Yuanchun Li, Yizhen Yuan, and Linghe Kong. 2023. ConvReLU++: Reference-based Lossless Acceleration of Conv-ReLU Operations on Mobile CPU. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. 503–515.

[17] Asif Ali Laghari, Kaishan Wu, Rashid Ali Laghari, Mureed Ali, and Abdullah Ayub Khan. 2021. A review and state of art of Internet of Things (IoT). *Archives of Computational Methods in Engineering* (2021), 1–19.

[18] Trung-Nghia Le, Akihiro Sugimoto, Shintaro Ono, and Hiroshi Kawasaki. 2020. Toward Interactive Self-Annotation For Video Object Bounding Box: Recurrent Self-Learning And Hierarchical Annotation Based Framework. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*.

[19] Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems* 2 (1989).

[20] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. 2019. SNIP: Single-shot Network Pruning based on Connection Sensitivity. arXiv:1810.02340 [cs.CV] https://arxiv.org/abs/1810.02340

[21] Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, and Rong Jin. 2018. Extremely low bit neural network: Squeeze the last bit out with admm. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[22] Guang Li, Ren Togo, Takahiro Ogawa, and Miki Haseyama. 2022. Dataset complexity assessment based on cumulative maximum scaled area under Laplacian spectrum. *Multimedia Tools and Applications* 81, 22 (April 2022), 32287–32303. https://doi.org/10.1007/s11042-022-13027-3

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

[24] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* 461 (2021), 370–403.

[25] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. 2020. HRank: Filter Pruning Using High-Rank Feature Map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[26] Mingbao Lin, Yuxin Zhang, Yuchao Li, Bohong Chen, Fei Chao, Mengdi Wang, Shen Li, Yonghong Tian, and Rongrong Ji. 2023. 1xN Pattern for Pruning Convolutional Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 4 (2023), 3999–4008. https://doi.org/10.1109/TPAMI.2022.3195774

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 740–755.

[28] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. 2018. On-Demand Deep Model Compression for Mobile Devices: A Usage-Driven Model Selection Framework *(MobiSys '18)*. Association for Computing Machinery, New York, NY, USA, 389–400. https://doi.org/10.1145/3210240.3210337

[29] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 21–37.

[30] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. 2019. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF international conference on computer vision*. 3296–3305.

[31] Adam Polyak and Lior Wolf. 2015. Channel-level acceleration of deep face representations. *IEEE Access* 3 (2015), 2163–2175.

[32] Ameet Annasaheb Rahane and Anbumani Subramanian. 2020. Measures of Complexity for Large Scale Image Datasets. In *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE. https://doi.org/10.1109/icaiic48513.2020.9065274

[33] Aruni RoyChowdhury, Prakhar Sharma, Erik Learned-Miller, and Aruni Roy. 2017. Reducing duplicate filters in deep neural networks. In *NIPS workshop on deep learning: Bridging theory and practice*, Vol. 1. 6.

[34] Mingwen Shao, Junhui Dai, Jiandong Kuang, and Deyu Meng. 2021. A dynamic CNN pruning method based on matrix similarity. *Signal, Image and Video Processing* 15 (2021), 381–389.

[35] Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose M. Alvarez. 2022. Structural Pruning via Latency-Saliency Knapsack. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 12894–12908. https://proceedings.neurips.cc/paper_files/paper/2022/file/5434be94e82c54327bb9dcaf7fca52b6-Paper-Conference.pdf

[36] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2820–2828.

[37] Mingxing Tan, Ruoming Pang, and Quoc V Le. 2020. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10781–10790.

[38] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. 2023. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 7464–7475.

[39] Bufang Yang, Lixing He, Neiwen Ling, Zhenyu Yan, Guoliang Xing, Xian Shuai, Xiaozhe Ren, and Xin Jiang. 2023. EdgeFM: Leveraging foundation model for open-set learning on the edge. *arXiv preprint arXiv:2311.10986* (2023).

[40] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[41] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 9194–9203.

[42] Mu Yuan, Lan Zhang, Fengxiang He, Xueting Tong, and Xiang-Yang Li. 2022. Infi: end-to-end learnable input filter for resource-efficient mobile-centric inference. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 228–241.

[43] Mu Yuan, Lan Zhang, Xuanke You, and Xiang-Yang Li. 2023. PacketGame: Multi-Stream Packet Gating for Concurrent Video Inference at Scale. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 724–737. https://doi.org/10.1145/3603269.3604825

[44] Mingyi Zhou, Yipeng Liu, Zhen Long, Longxi Chen, and Ce Zhu. 2019. Tensor rank learning in CP decomposition via convolutional neural network. *Signal Processing: Image Communication* 73 (2019), 12–21.