electronics

MDPI

Article

Cloud–Edge Collaborative Inference with Network Pruning

Mingran Li [1], Xuejun Zhang [1,2,3,*], Jiasheng Guo [1] and Feng Li [1]

[1] School of Computer, Electronics and Information, Guangxi University, Nanning 530004, China
[2] Guangxi Key Laboratory of Multimedia Communications and Network Technology, Nanning 530004, China
[3] Guangxi Big White & Little Black Robots Co., Ltd., Nanning 530007, China
* Correspondence: xjzhang@gxu.edu.cn

Abstract: With the increase in model parameters, deep neural networks (DNNs) have achieved remarkable performance in computer vision, but larger DNNs create a bottleneck for deploying DNNs on resource-constrained edge devices. The cloud–edge collaborative inference based on network pruning provides a solution for the deployment of DNNs on edge devices. However, the pruning methods adopted by existing frameworks are locally effective, and the compressed models are over-sparse. In this paper, we design a cloud–edge collaborative inference framework based on network pruning to make full use of the limited computing resources on edge devices. In our framework, we propose a sparsity-aware feature bias minimization pruning method to reduce the feature bias that happens during network pruning and prevent the pruned model from being over-sparse. To further reduce the inference latency, we consider the difference in computing resources between edge devices and the cloud, then design a task-oriented asymmetric feature coding to reduce the communication overhead of transmitting intermediate data. With comprehensive experiments, our framework can reduce end-to-end latency by 82% to 84% with less than 1% accuracy loss, compared to the cloud–edge collaborative inference framework with traditional methods, and our framework has the lowest end-to-end latency and accuracy loss compared to other frameworks.

Keywords: collaborative intelligence; network pruning; edge computing; cloud–edge collaborative computing

Citation: Li, M.; Zhang, X.; Guo, J.; Li, F. Cloud–Edge Collaborative Inference with Network Pruning. *Electronics* **2023**, *12*, 3598. https://doi.org/10.3390/electronics12173598

Academic Editor: Palden Lama

Received: 6 August 2023
Revised: 22 August 2023
Accepted: 24 August 2023
Published: 25 August 2023

1. Introduction

Over recent years, DNNs have achieved state-of-the-art performance in a wide range of applications, including image classification, semantic segmentation, and object detection [1]. With the continuous improvement of model capacity, DNNs require an increasing amount of computing and storage resources, which prohibits the full deployment of DNNs on resource-constrained edge devices. Currently, three main approaches [2] exist to solve this problem: device-based, edge–server-based, and cloud–edge-based, as shown in Figure 1. In the device-based approach, computing tasks are offloaded from the cloud center to the end devices; the structure of this approach is shown in Figure 1a. This will generate plenty of communication overhead [3]. Due to the limited computing resources of the end devices, the model accuracy is low and the result is unsatisfactory. As Figure 1b shows, edge–server-based is to deploy the model on the edge server [4], this approach incurs significant communication overhead and cannot support latency-sensitive applications. Figure 1c is the structure of the cloud–edge-based approach; it aims to divide the model according to the computing tasks and the computing resources of the edge nodes. The initial part of the model is deployed on edge nodes, and the remaining parts operate in the cloud. This strategy makes efficient use of the edge nodes' computing and achieves far superior latency performance.

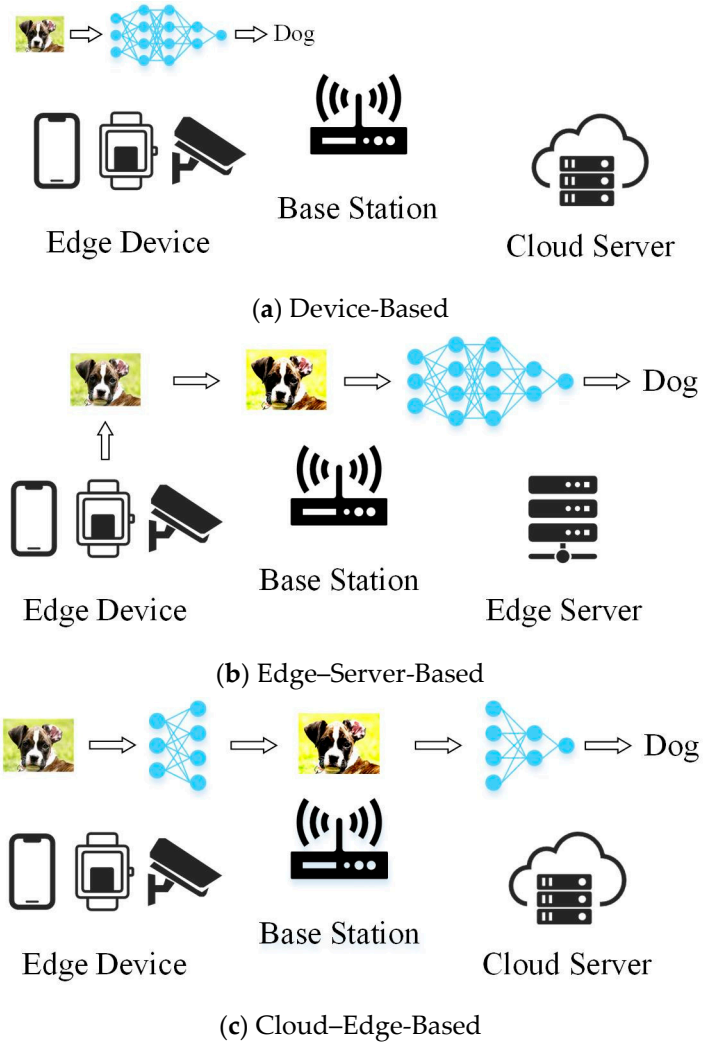


Figure 1. Three strategies for devices with limited resources.

The cloud–edge-based approach adopts model splitting to achieve partitioned deployment of the model. Models at the edge nodes are responsible for low-dimension feature extraction, after which intermediate features are transmitted to the cloud through wireless networks. Here, the cloud continues to perform the remaining computing tasks. Through cloud–edge collaborative computing, this method enables models to obtain a better trade-off between communication overhead and network latency. Recent studies have concentrated on DNN partition [5–7]. They attempt to split DNN into two components and then employ different strategies to optimize processing load distribution [8]. However, the direct partition of the model introduces the following issues: (1) the splitting model struggles to adapt to the limited computing resources of the edge nodes; and (2) the random selection of the partition point results in the size of the intermediate features generated by the model larger than the size of the model input features, thereby increasing the communication overhead.

One potential approach to address the first issue involves considering the optimization of the models deployed at the edge. An effective way is to use the neural network compression strategy to compress the model operating at the edge. This method ensures that the model deployed at the edge node is less sensitive to the edge nodes' resources. Commonly utilized methods for model compression include network pruning [9], parameter quantization [10], knowledge distillation [11], and tensor decomposition [12]. The second issue can be tackled based on the task and model features, selecting an appropriate partition point, or employing feature coding to compress and downscale the intermediate

data generated after the partition. Then the data will be transmitted to the cloud by wireless networks. The cloud will decode the received data. To reduce the loss of intermediate features in the encoding and decoding process, certain studies have constructed feature encoders based on DNNs, such as NECST [13]; these methods compress intermediate features within an acceptable accuracy loss. Early cloud–edge collaborative inference frameworks focused on enhancing the selection of partitioning points [14], and designing model-splitting strategies based on the edge nodes' computing resources to realize model partitioning deployment. However, this approach is limited by the extensive number of parameters in the neural network and only supports edge devices with sufficient computing resources. Subsequently, the work in [15] proposed a cloud–edge collaborative inference approach with a two-step pruning framework. This approach introduced a model pruning strategy based on model splitting to reduce the number of parameters in the neural network. However, this method conducts two pruning operations on the model, reducing the model generalization ability and resulting in a significant accuracy loss. Additionally, the communication overhead generated by intermediate feature transmission leads to a high inference latency. To achieve more efficient cloud–edge collaborative inference, the work in [16] designed a three-step framework that incorporates model splitting, model compression, and feature coding to achieve a better trade-off between communication and computation. However, the framework prunes the model by assigning the channel mask values to zero according to the given sparsity ratio. The channel pruning algorithm used for model compression follows a single criterion for choosing channels, focusing predominantly on local optimization. All channels in the model are compressed to achieve a sparse network structure, introducing extra computation. Moreover, important channels are over-regularized, and the model appeared to be underfitting. In addition, the structure of feature coding within the framework is symmetric, it is not an optimal design for the edge devices and the cloud due to the significant discrepancy in their computing resources. In summary, existing cloud–edge collaboration frameworks based on network pruning encounter two primary issues. (1) These frameworks excessively pursue sparse structure networks during pruning, resulting in poor generalization ability and over-sparse structure of neural networks. (2) These frameworks overlook the communication overhead during cloud–edge collaborative inference or use a symmetric feature coding structure, which fails to account for the difference in computing resources between cloud and edge devices. These issues make it impossible for the cloud–edge collaborative inference frameworks based on network pruning to effectively reduce inference latency and accuracy loss.

In this paper, we propose a cloud–edge collaborative computing method based on network pruning to reduce the demand for computing resources and end-to-end latency in collaborative inference. To effectively utilize the computing resources of edge devices, model splitting was used to make the trade-off between computation and communication. For the model that needs to be deployed on the edge devices after partition, we designed a channel pruning method based on sparsity-aware feature bias minimization to control the FLOPs of the model so that the model can run on more resource-constrained edge devices. Our sparsity-aware feature bias minimization pruning method is based on global preference. We performed pruning on partially unimportant channels to prevent the network structure from being over-sparse. We retained the channel masks during pruning and activate some unimportant channels in the subsequent process to improve the model's generalization ability. Our pruning method addresses the problems of model underfitting and over-sparse in existing frameworks. Furthermore, we built a task-oriented asymmetric feature coding to compress the output feature of the edge devices. We improved the feature coding structure in the existing frameworks by more rationally considering the differences in computing resources between the cloud and edge. A depthwise separable convolution module is introduced as our encoder, allowing us to reduce more computation in the encoding process compared to other frameworks. Our feature coding can balance the computing resources of the edge and cloud to reduce communication costs during the cloud–edge collaborative inference.

In summary, our contributions are as follows:

- We proposed a sparsity-aware feature bias minimization channel pruning method to compress the model deployed at the edge. It effectively solves the model underfitting and over-sparse problems after pruning;
- We designed a task-oriented asymmetric feature coding to decrease the communication overhead required for collaborative inference. It is more rational for the allocation of computing tasks and reduces the computing complexity during the feature coding process;
- We combined our model pruning method and feature coding with model splitting to construct a cloud–edge collaborative inference framework;
- We conducted experiments on the CIFAR-10 dataset, using VGG16, ResNet18, MobileNetV1, and MobileNetV2, the results demonstrated that our framework can reduce inference latency with lower accuracy loss.

The rest of this paper is arranged as follows: Section 2 contains related works on channel pruning, feature coding, and cloud–edge collaborative computing. Section 3 details the proposed sparsity-aware feature bias minimization channel pruning, the designed task-oriented asymmetric feature coding, and the cloud–edge collaborative inference framework. The experiments are conducted in Section 4. Finally, the paper is concluded in Section 5.

2. Related Work

Channel pruning is a structured pruning method. The core idea is to sort channels or features according to specific criteria, and then remove the corresponding channels to generate a non-structured sparsity model [17]. Compared to unstructured pruning, this approach reduces reliance on hardware architecture. With the help of Basic Linear Algebra Subprograms (BLAS) [18], the model compression and acceleration can be convenient to implement on common hardware platforms. Earlier research on channel pruning has compressed DNNs based on filters. Li et al. [19] proposed the Pruning Filter for Efficient ConvNets (PFEC) to calculate the l_1-norm of the filter and remove the filters with a lower ranking. He et al. [20] found that using the l_2-norm as the selection criterion could achieve better compression. As a result, they designed Soft Filter Pruning (SFP), which adopts the l_2-norm of the filter as the criterion. Instead of removing filters, SFP sets filters' values to zero. In addition, researchers have suggested that using the current layer as the selection criterion can also reduce model parameters. Lin et al. [21] proposed HRank, which uses Singular Value Decomposition (SCD) to solve the average rank of the filters, and then prune the model hierarchically to retain the top-k filters. Sui et al. [22] pointed out that the higher the independence of the channel, the greater the impact on the final results of the model; they proposed CHannel Independence (CHIP), which uses the activation maps' nuclear norm change to represent the independence of the channel. For the criteria of channel selection, if the consideration is based on the current level only, it may cause the superposition of global errors and increase the loss of final model accuracy. Yu et al. [23] suggested using global importance as a criterion for selecting removed channels, and they proposed Neuron Importance Score Propagation (NISP) to achieve minimal reconstruction error. The research stated above removes channels according to various criteria, which reduces the number of parameters and computation of DNNs and encourages the deployment of DNNs on edge terminals.

In the study of cloud–edge collaborative computing, splitting DNNs into two parts produces intermediate features, and the latency accrued in transmitting these intermediate features to the cloud through wireless networks is the main communication overhead in the collaborative inference process [24]. Ko et al. [25] suggested encoding intermediate features before transmitting. They combined Huffman encoding and run-length encoding and took DNN as an encoding pipeline to achieve lossless encoding. Mentzer et al. [26] evaluated the rate-distortion of the image compression auto-encoder. They designed a feature coding system to estimate entropy with a context model. Agustsson et al. [27] jointly trained GANs with a feature encoder and proposed a feature compression system

that obtained pleasing image decoding results at low bitrates. Due to the difference in computing resources between edge devices and cloud centers, Yao et al. [28] placed most of the computational burden on the server side and reduced the computation overhead of edge devices with no degradation in inference accuracy through a deep offloading framework. To obtain larger receptive fields, Jiang et al. [29] expanded the size of the kernel based on the feature weights. They used channels to connect the CNN module to achieve deep convolution. Based on this convolution method, they proposed SLIC, which reduces the complexity of the feature encoding process. According to the design of DNNs, the dimension of intermediate features will gradually deepen, which makes us have to carefully consider the choice of partition points when using model splitting, while feature coding allows us to split the model in the early stage and reduce the demand of the model on computing resources of edge devices.

In recent research, attention has been paid to the efficiency-primary cloud–edge collaboration pattern, which employs split deployment [30], splitting the model into two components: one deployed in the cloud and other in the edge, making full use of the computing resources of edge devices and reducing end-to-end latency [31]. Kang et al. [3] adopted model splitting to design a joint inference system that adjusts to the restricted storage space and energy of edge devices while reducing inference delay. Shi et al. [15] offered a two-step pruning cooperative inference framework that combined model pruning and model splitting, reducing wireless communication time and total computation workload in the inference process. Shao et al. [32] concluded that feature compression consumes a significant amount of time during collaborative computing, therefore they introduced an end-to-end architecture BottleNet++ to code features by lightweight convolutional neural networks (CNNs) and achieve a better compression ratio. In subsequent work, they investigated the important trade-off between model computational cost and intermediate features transfer communication overhead [16], using the incremental network pruning and feature encoding designed by DNNs to significantly reduce the inference latency. MATSUBARA et al. [33] applied knowledge distillation to lighten the model deployed at the edge; they introduced a bottleneck structure in the model to reduce inference time and improve accuracy. Banitalebi-Dehkordi et al. [34] proposed a collaborative computing framework for Auto-Split DNNs that maintains model accuracy and reduces end-to-end latency. These works optimize cloud–edge collaborative computing from different perspectives, effectively utilize computation and storage resources during data transmission, reduce the energy consumption of edge devices, and provide solutions for deep learning algorithms to be landed.

3. Methods

In this section, we first describe the shortcomings in common channel pruning algorithms and introduce the feature bias-based pruning method as an alternative to solve these problems. Then, we introduce our proposed task-oriented asymmetric feature coding. Finally, we design a cloud–edge collaborative inference framework based on the above two methods and model splitting to implement DNNs on resource-constrained edge devices.

3.1. Overview of Channel Pruning

Different channel pruning methods use different criteria to determine the importance of channels, and then remove the unimportant channels to reduce the FLOPs of the model. Duan et al. [35] investigated the methods using L1-norm and HRank; they found that these methods were effective on low-dimensional features, but had almost the same performance as random selection methods on high-dimensional features, and the feature distribution of the model changed after pruning. Figure 2 explores the relationship between the compression rate, layer depth, and accuracy of different pruning strategies on VGG16 and ResNet18. These figures show that when the pruning strategy is applied at the shallow layers of the neural network, the accuracy of the HRank and L1-norm strategies is higher than the random pruning strategy, but as layers go deeper, the effect of the random pruning

strategy is similar to the HRank and L1-norm strategies, even the accuracy is higher than the other two strategies on ResNet18. Figure 3 depicts the changes in image features before and after pruning. The figure indicates that image features change as the neural network layers get deeper, and if the network is processed by a pruning algorithm, for the same layers, the distribution of image features changes more. The criteria used by these methods to select channels are usually locally optimal, and as the training process proceeds, the best results cannot be achieved globally, lacking the correlation of global information. To obtain the expected compression rate, most pruning methods adopt sparse regularization for all channels [36], resulting in the weight of crucial channels decaying to zero as the dimension increases, thus leading to the problem of the same criteria in different dimensions, but with different performance.

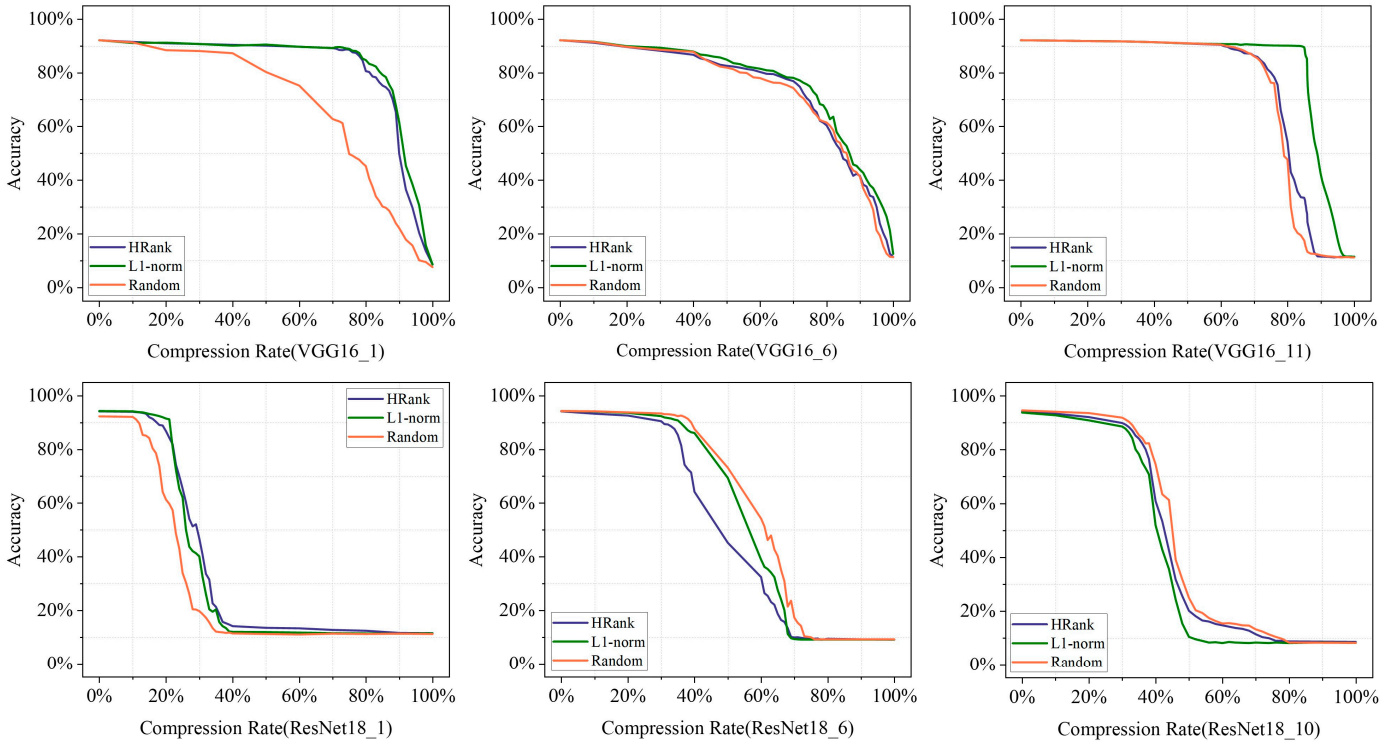


Figure 2. Analyzing the relationship between compression rate, layer depth, and accuracy; "VGG16_1" denotes the 1st layer of the VGG16 network.

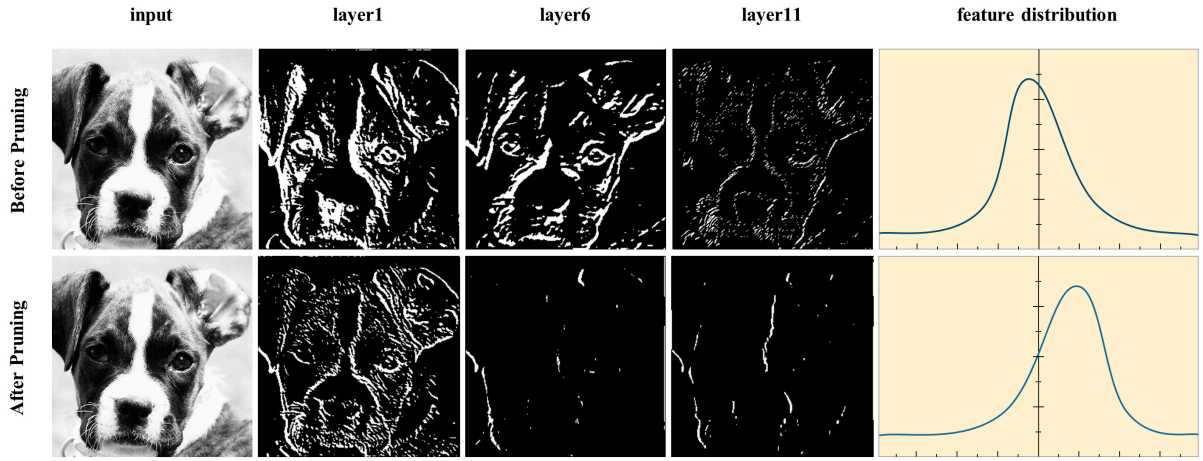


Figure 3. The distribution of image features before and after pruning.

3.2. Sparsity-Aware Feature Bias Minimization Pruning Method

For a convolutional neural network, we use $\mathcal{W}^{(l)} = \{\omega_1^{(l)}, \omega_2^{(l)}, \cdots, \omega_{N_l}^{(l)}\} \in \mathbb{R}^{N_l \times C_l \times k_l \times k_l}$ to represent it, where $\mathcal{W}$ is the weight sets, l is the number of layers, N is the number of weights, C is the channel, and k is the convolution kernel size. The compression rate of each layer is denoted as $R^{(l)} = \{r_1, r_2, \ldots, r_l\}$, so the general channel pruning can be formulated as

$$R^{(l)} = \{r_1, r_2, \ldots, r_l\} = \left(\begin{smallmatrix} arg\ min \\ r_1, r_2, \ldots, r_l \end{smallmatrix}\right) \mathcal{L}\left(\mathcal{N}\left(R^{(l)}; \mathcal{W}^{(l)}\right)\right) = \left(\begin{smallmatrix} arg\ min \\ r_1, r_2, \ldots, r_l \end{smallmatrix}\right) \mathcal{L}\left(\mathcal{N}\left(r_1, r_2, \ldots, r_l; \omega_1^{(l)}, \omega_2^{(l)}, \cdots, \omega_{N_l}^{(l)}\right)\right) \tag{1}$$

where $\mathcal{L}$ is the loss function, and $\mathcal{N}$ is the neural network.

Nowadays, most neural networks have the following basic structure: input layer, convolutional layer, BN layer [37], ReLU layer [38], and output layer. The BN layer can increase the generalization ability of the neural networks as well as convergence speed. ReLU is used as an activation function that enhances the nonlinearity of neural networks. Let $X_l = \{x_1, x_2, \ldots, x_l\}$ be the input of the BN layer and $Y_l = \{y_1, y_2, \ldots, y_l\}$ be the output of the BN layer, for each dimension of the feature map:

$$\hat{x}_l^{(d)} = \frac{x_l^{(d)} - \mu\left[x_l^{(d)}\right]}{\sqrt{\sigma^2\left[x_l^{(d)}\right]}}, \; y_l^{(d)} = \gamma \hat{x}_l^{(d)} + \beta \tag{2}$$

where d is the dimension, $\mu[x_l^{(d)}]$ is the mean of the l-th layer's d-th dimension, $\sigma^2[x_l^{(d)}]$ is the variance of the l-th layer's d-th dimension, and γ and β can modify the BN layer's output to provide the possibility of transforming back to any scales [39]; they can be obtained directly during the training process.

Then, $y_l^{(d)}$ is processed by the ReLU activation function:

$$\hat{y}_l^{(d)} = R\left(y_l^{(d)}\right) = \max\left(0, y_l^{(d)}\right) \tag{3}$$

where $\hat{y}_l^{(d)}$ denotes the output of the ReLU layer, and R stands for the ReLU activation function. It is obvious that $\hat{y}_l^{(d)}$ loses some information compared to $y_l^{(d)}$.

After the channel pruning algorithm compresses the model, the feature distribution usually changes, the difference between the feature distribution of the model before and after pruning is referred to as the feature bias. Conventional network pruning methods remove channels according to specific criteria. We denote the output of these methods as $\hat{y}_{l1}^{(d)}$, and the output of sparsity-aware feature bias minimization is denoted as $\hat{y}_{l2}^{(d)}$. To quantify the feature bias specifically, the mean of each dimension in each layer can be calculated, and then the feature bias of each dimension can be expressed as

$$\left|\Delta\mu_l^{(d)}\right| = \left|\mu\left[y_l^{(d)}\right] - \hat{\mu}\left[\hat{y}_l^{(d)}\right]\right| \tag{4}$$

where $\hat{\mu}$ is the mean after pruning.

Substituting $\hat{y}_{l1}^{(d)}$ and $\hat{y}_{l2}^{(d)}$ into Equation (4), we can reach the following formulas:

$$\left|\Delta\mu_{l1}^{(d)}\right| = \left|\mu\left[y_l^{(d)}\right] - \hat{\mu}\left[\hat{y}_{l1}^{(d)}\right]\right| \tag{5}$$

$$\left|\Delta\mu_{l2}^{(d)}\right| = \left|\mu\left[y_l^{(d)}\right] - \hat{\mu}\left[\hat{y}_{l2}^{(d)}\right]\right| \tag{6}$$

where $\left|\Delta\mu_{l1}^{(d)}\right|$ denotes the feature bias of conventional pruning methods, and $\left|\Delta\mu_{l2}^{(d)}\right|$ denotes the feature bias of our pruning method.

Referring to the related research based on feature shift [35], we further analyze the validity of using feature bias as the criterion for channel selection.

Substituting Equation (4) into Equation (2), we can reach the following formulas:

$$y_l^{(d)} = \gamma \hat{x}_l^{(d)} + \beta = \gamma \cdot \frac{x_l^{(d)} - \mu\left[x_l^{(d)}\right]}{\sqrt{\sigma^2\left[x_l^{(d)}\right]}} + \beta = \gamma \cdot \frac{x_l^{(d)} - \left(\Delta\mu_l^{(d)} + \hat{\mu}\left[\hat{x}_l^{(d)}\right]\right)}{\sqrt{\sigma^2\left[x_l^{(d)}\right]}} + \beta \tag{7}$$

Processing Equation (7):

$$\begin{aligned} y_l^{(d)} &= \gamma \cdot \frac{x_l^{(d)} - \left(\Delta\mu_l^{(d)} + \hat{\mu}\left[\hat{x}_l^{(d)}\right]\right)}{\sqrt{\sigma^2\left[x_l^{(d)}\right]}} + \beta \\ &= \gamma \cdot \frac{x_l^{(d)} - \hat{\mu}\left[\hat{x}_l^{(d)}\right]}{\sqrt{\sigma^2\left[x_l^{(d)}\right]}} + \beta - \gamma \cdot \frac{\Delta\mu_l^{(d)}}{\sqrt{\sigma^2\left[x_l^{(d)}\right]}} \\ &= \gamma \cdot \frac{\sqrt{\sigma^2\left[\hat{x}_l^{(d)}\right]}}{\sqrt{\sigma^2\left[x_l^{(d)}\right]}} \cdot \frac{x_l^{(d)} - \hat{\mu}\left[\hat{x}_l^{(d)}\right]}{\sqrt{\sigma^2\left[\hat{x}_l^{(d)}\right]}} + \beta - \gamma \cdot \frac{\Delta\mu_l^{(d)}}{\sqrt{\sigma^2\left[x_l^{(d)}\right]}} \end{aligned} \tag{8}$$

Comparing Equation (7) with Equation (8), we can find that after pruning, the mean of the feature is $\beta - \gamma \cdot \frac{\Delta\mu_l^{(d)}}{\sqrt{\sigma^2\left[x_l^{(d)}\right]}}$ and the standard deviation of the feature is $\gamma \cdot \frac{\sqrt{\sigma^2\left[\hat{x}_l^{(d)}\right]}}{\sqrt{\sigma^2\left[x_l^{(d)}\right]}} \cdot \frac{x_l^{(d)} - \hat{\mu}\left[\hat{x}_l^{(d)}\right]}{\sqrt{\sigma^2\left[\hat{x}_l^{(d)}\right]}}$. This demonstrates the change in feature distribution before and after pruning.

We use feature bias as the criterion for selecting channels in the network pruning so that the model can be as close as possible to the initial distribution during the pruning process. In other words, we choose to minimize $\left|\Delta\mu_l^{(d)}\right|$ as the optimization objective so we can obtain the following relation:

$$\sum_{l=1}^{L}\sum_{d=1}^{D}\left|\Delta\mu_l^{(d)}\right| \le \sum_{l=1}^{L}\sum_{d=1}^{D}\left|\Delta\mu_{l2}^{(d)}\right| \le \sum_{l=1}^{L}\sum_{d=1}^{D}\left|\Delta\mu_{l1}^{(d)}\right| \tag{9}$$

From the comparison expression (9), we can know that the difference between $\sum_{l=1}^{L}\sum_{d=1}^{D}\left|\Delta\mu_l^{(d)}\right|$ and $\sum_{l=1}^{L}\sum_{d=1}^{D}\left|\Delta\mu_{l2}^{(d)}\right|$ is smaller; therefore, during the pruning process, our sparsity-aware feature bias minimization method has a smaller feature bias, which is closer to the original feature distribution than the conventional pruning methods.

Channel pruning is a regularization method that prevents the model from overfitting; however, if all channels are pruned, an excessively sparse model will be generated, causing the model to be underfitting. The sparse pruning approach based on L1 regularization is denoted as follows:

$$\mathcal{L} = \mathcal{L}\left(f\left(I, \mathcal{W}^{(l)}\right), y\right) + \lambda\sum_{l}^{L}\left|\omega^{(l)}\right| \tag{10}$$

where $\mathcal{L}$ is the loss function and λ is a hyperparameter that controls the strength of the regularization.

Existing pruning methods in the cloud–edge collaborative inference frameworks regularize all channels to obtain a sparse network, which has a large impact on the final accuracy. To solve this problem, we only prune some unimportant channels according to the given sparsity rate. We retain the channel masks of each layer and activate a portion of the unimportant channels in the subsequent computation phase to prevent the model from underfitting. We make a penalty on the scaling factors γ of BN. We describe sparsity-aware regularization as

$$\mathcal{L} = \mathcal{L}\left(f\left(I, \mathcal{W}^{(l)}\right), y\right) + \alpha\sum_{l}^{L}\sum_{i=1}^{N_l}\mathcal{M}_i^{(l)}\left|\gamma_i^{(l)}\right| \tag{11}$$

where $\mathcal{L}\left(f\left(I, \mathcal{W}^{(l)}\right), y\right)$ is the loss function of the normal training, α is used for sparsity controlling, $\mathcal{M}$ is the pruning mask consisting of unimportant channels in each layer.

Therefore, our proposed sparsity-aware feature bias minimization pruning method is formulated as

$$\mathcal{L}\left(I, y, \mathcal{W}^{(l)}\right) = \mathcal{L}\left(f\left(I, \mathcal{W}^{(l)}\right), y\right) + \alpha \sum_{l}^{L} \sum_{i=1}^{N_l} \mathcal{M}_i^{(l)} \left|\gamma_i^{(l)}\right| + arg\ min \sum_{l=1}^{L} \sum_{d=1}^{\hat{d}_l} \left|\mu\left[y_l^{(d)}\right] - \hat{\mu}\left[\hat{y}_l^{(d)}\right]\right| \quad (12)$$

where $\sum_{l=1}^{L} \sum_{d=1}^{\hat{d}_l} \left|\mu\left[y_l^{(d)}\right] - \hat{\mu}\left[\hat{y}_l^{(d)}\right]\right|$ denotes the sum of the l-th layer's feature bias, $\hat{d}_l$ is the d-th dimension after pruning.

In the cloud–edge collaborative framework, we adopt the sparsity-aware feature bias minimization method to compress the model deployed at the edge. Figure 4 is a diagram of the sparsity-aware feature bias minimization pruning method. Based on our work, we use the feature bias as the selection criterion for channel pruning and calculate the total feature bias values for each channel. The smaller the channel feature bias values, the smaller the impact of the corresponding channel on the subsequent results. Then, we remove the channels according to the given sparsity rate, which makes our method sparsity-aware. For the output layer of edge devices, a higher sparsity rate can be set to reduce the intermediate data to save communication costs. During each iteration, the pruning steps are as follows: (1) calculating the output expectation of each channel, and then calculating the total feature bias; (2) ranking the channels according to the total feature bias, with larger values corresponding to more important channels; (3) pruning less important channels according to sparsity rate and applying sparse regularization on the scaling factor γ; (4) saving channel masks and activating some channels in subsequent processes; (5) fine-tuning the model for two epochs, and then pruning the next layer.

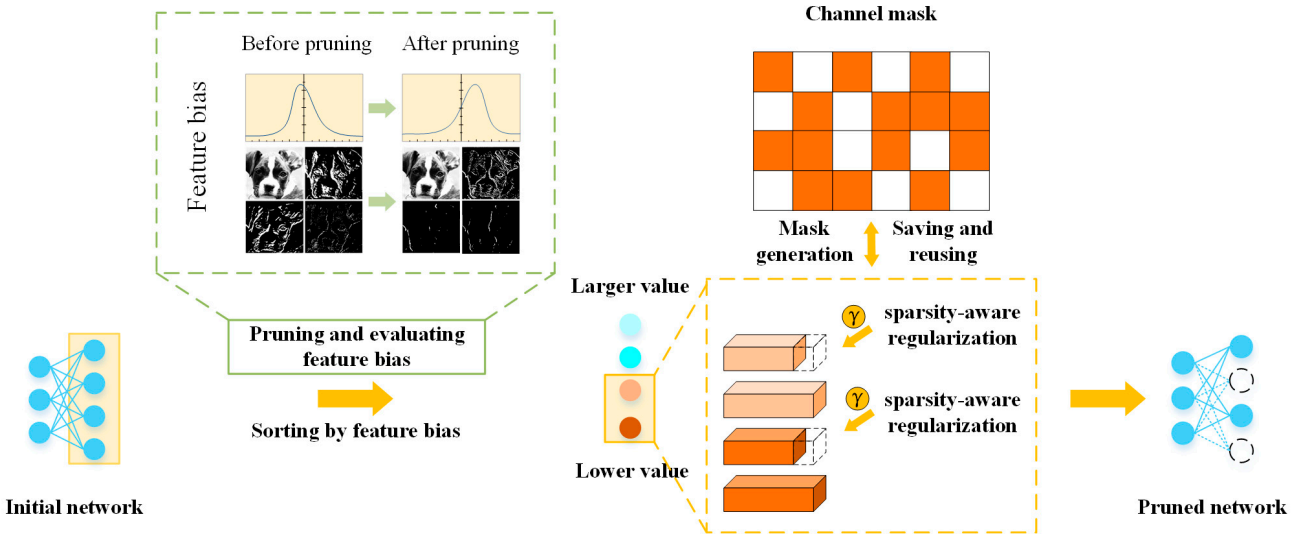
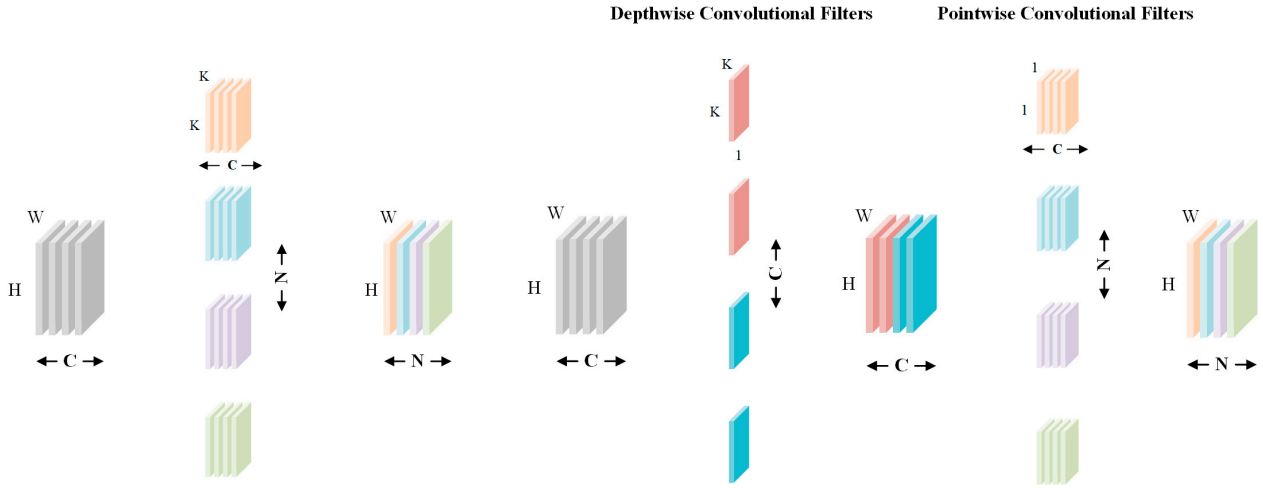


Figure 4. Diagram of the sparsity-aware feature bias minimization pruning method.

3.3. Task-Oriented Asymmetric Feature Coding

Channel pruning reduces model FLOPs deployed in the edge node, but the impact on the overall inference latency of the cloud–edge collaboration framework cannot be neglected due to the communication overhead generated by the partitioned deployment strategy. For this reason, feature coding was introduced by us to compress the intermediate features and save communication costs. Traditional feature coding methods, such as source coding, manually encode features. With the advancement of deep learning, most feature coding is now built on DNNs. These methods use the encoder to compress features at

the edge nodes and then build a symmetric decoder in the cloud server to reconstruct the data. However, in cloud–edge collaborative applications, the gap between computation resources at the edge and the cloud is large, and the symmetric coding distributes the load equally, which is a suboptimal allocation choice. DNNs are fault-tolerant; thus, even if data are over-compressed or partially missing during transmission, their performance will not be greatly affected. Task-oriented training of encoder and decoder allows us to discard unimportant and redundant information. The encoder of our feature coding is deployed at resource-constrained edge nodes, which consist of lightweight neural networks. The encoder consists of a depthwise separable convolution module. Depthwise separable convolution includes depthwise convolution and pointwise convolution [40], a convolution kernel of depthwise convolution only performs convolution operation with one channel of the input feature map to obtain the mapping relationship of the same dimension. After the depthwise convolution, a pointwise convolution is added to realize spatial information extraction for different dimensions. Figure 5 shows the structure of standard convolution and depthwise separable convolution.



(**a**) standard convolution (**b**) depthwise separable convolution

Figure 5. The structure of standard convolution and depthwise separable convolution: (**a**) standard convolution with the kernel size of $K \times K$; (**b**) the convolution kernel of each depthwise convolution has only one channel and the kernel size is $K \times K$. Pointwise convolution is a simple 1×1 convolution.

For an input feature $I \in \mathbb{R}^{H \times W \times C}$, convolution operation applies to produce an output feature $O \in \mathbb{R}^{H \times W \times N}$, where H is the height, W is the width, C is the number of channels of the input feature map, and N is the number of channels of the output feature map. When the kernel size is $K \times K$ and the stride is one, the FLOPs of the standard convolution are expressed as follows:

$$S_{FLOPs} = H \times W \times K \times K \times C \times N \tag{13}$$

The FLOPs of depthwise convolution are expressed as:

$$D_{FLOPs} = H \times W \times K \times K \times C \times 1 \tag{14}$$

The FLOPs of pointwise convolution are expressed as:

$$P_{FLOPs} = H \times W \times 1 \times 1 \times C \times N \tag{15}$$

The FLOPs of depthwise separable convolution:

$$\mathrm{DP}_{FLOPs} = D_{FLOPs} + P_{FLOPs} = H \times W \times K \times K \times C + H \times W \times C \times N \tag{16}$$

Comparing the FLOPs of depthwise separable convolution to standard convolution:

$$\frac{DP_{FLOPs}}{S_{FLOPs}} = \frac{H \times W \times K \times K \times C + H \times W \times C \times N}{H \times W \times K \times K \times C \times N} = \frac{1}{N} + \frac{1}{K^2} \tag{17}$$

In our encoder structure, the kernel size is 3×3, and the number of N is greater than 1. Thus, we can obtain the following formula:

$$DP_{FLOPs} = \left(\frac{1}{N} + \frac{1}{K^2}\right) \cdot S_{FLOPs} < S_{FLOPs} \tag{18}$$

Compared to encoders based on standard convolution, our encoder can significantly reduce the computation introduced by the encoding process. The output feature's dimension on the edge nodes is reduced by point convolution. For image classification tasks, the "manifold of interest" [41], which includes important feature information, is usually formed in high-dimension features, after the dimension reduction by pointwise convolution, the "manifold of interest" can be embedded in a low dimension, and the redundant information is compressed. The computing resources in the cloud are sufficient, and so the decoder in the cloud is complex. It contains a self-attention layer and a bottleneck structure to reconstruct the data from the edge nodes. The self-attention layer learns the relationship between pixel points at different locations [42], giving the compressed data richer detail in the reconstruction process. The bottleneck structure includes standard convolution and pointwise convolution, which allows us to obtain diverse information in high dimensions. Figure 6 shows the structure of our feature coding.

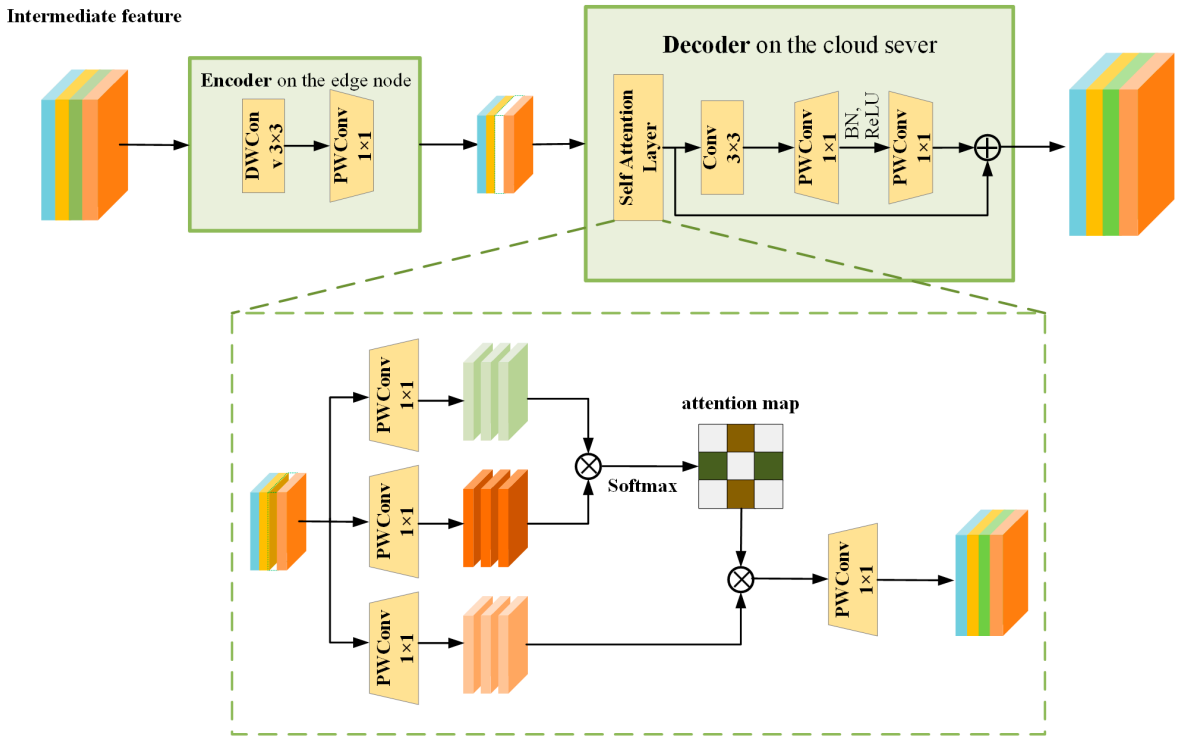


Figure 6. The structure of task-oriented asymmetric feature coding.

3.4. Cloud–Edge Collaborative Inference Framework

According to the above-shown work, we further designed a cloud-edge collaborative inference framework, as shown in Figure 7. The framework is based on model splitting, combined with network pruning and feature coding. It can make full use of computing resources at the edge and the cloud through partition deployment. The framework achieves cloud–edge collaborative computing and reduces end-to-end latency. Our framework

consists of three workflows. First, based on the model structure, the layer with a lower output feature dimension was selected as the partition point. The first part of the splitting model was deployed on the edge devices and the rest of the model was deployed on the cloud. Then, the model deployed on the edge part was compressed by our sparsity-aware feature bias minimization pruning method to remove selected channels layer-by-layer until the partition location, which reduces redundant model parameters deployed at the edge. The rest of the model after model splitting maintains the original structure and parameters on the cloud. Finally, the feature coding designed by us was introduced at the partition location. At the edge, dimension reduction was used to compress the intermediate features. The compressed data transfers to the cloud through the wireless network, where the decoder reconstructs the data, and then the cloud continues to carry out the remaining inference steps.

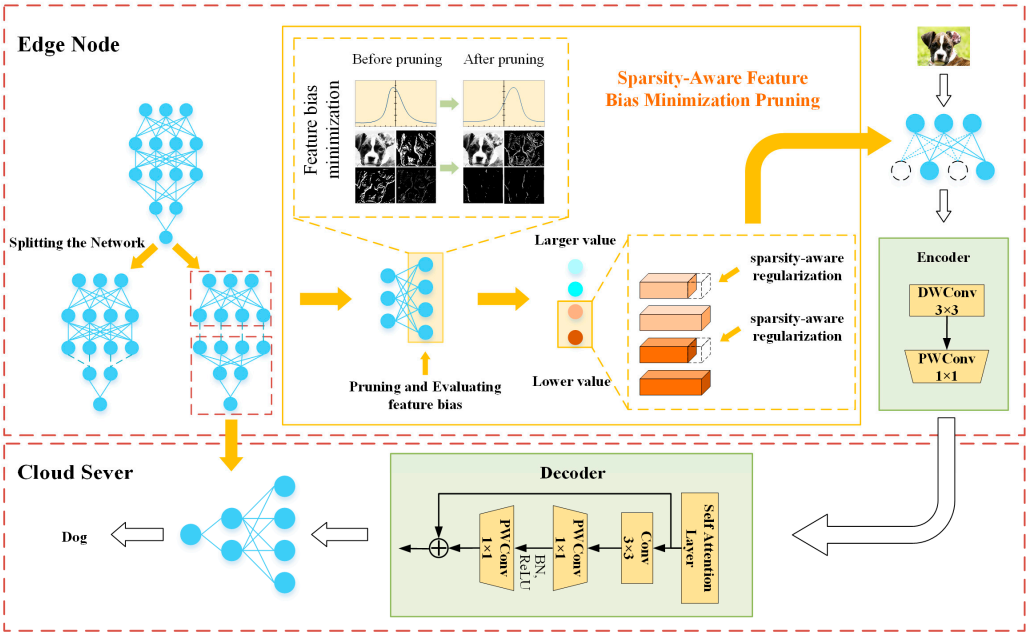


Figure 7. The proposed framework of cloud–edge collaborative inference.

4. Experiments

In this section, we first evaluate the performance of our framework on the image classification task. Then, we compare our framework with similar ones for analysis. Finally, ablation studies are designed to verify the effectiveness of our proposed methods.

4.1. Experimental Setup

4.1.1. Network and Dataset

We selected four representative networks, VGG16 [43], ResNet18 [1], MoblieNetV1 [44], and MoblieNetV2 [41], which have achieved remarkable results in image classification and are widely used in various computer vision tasks. We used the CIFAR-10 [45] dataset as our experimental data, which has 10 classes and contains 60,000 color images with pixels of 32×32.

4.1.2. Baselines

Three methods are used in our framework: model splitting, model compression, and feature coding. To completely verify the effectiveness of our framework, four baselines are considered in our experiments: (1) Only Splitting, a framework for cloud–edge collaborative inference based on model splitting which focuses on the selection of model partition points; (2) two-step Pruning [15], a framework that prunes the model parameters twice, once on the full model and once on the model deployed on the edge devices;

(3) BottleNet++ [32], a cloud–edge collaborative inference framework based on feature coding to reduce communication latency; and (4) CCTO [16], a framework incorporating three techniques: model splitting, model pruning, and feature coding.

4.1.3. Hardware Configuration

In the experiment, we chose Raspberry Pi 3 Model B as the edge node; our cloud server is equipped with an NVIDIA GeForce RTX 3080 Ti GPU. The specific parameters of our hardware platform are shown in Table 1.

Table 1. The specifications of the hardware platform.

Platform	Device	Processor	Freq.	RAM
Edge Node	Raspberry Pi 3 Model B	ARM Cortex-A53 (Advanced RISCMachines, Cambridge, UK)	1.2 GHz	1 GB
Cloud Server	Desktop	Intel i7-11700 CPU (Intel in Santa Clara, CA, USA) + NVIDIA GeForce RTX 3080 Ti GPU (NVIDIA Corporation in Santa Clara, CA, USA)	2.5 GHz	32 GB

4.1.4. Metric

For the cloud–edge collaborative inference framework, end-to-end latency and accuracy loss are the main evaluation metrics. The end-to-end latency consists of three components: edge node inference time T*edge*, cloud–edge data transmission time T*trans*, and cloud inference time T*cloud*. The total end-to-end latency is denoted as T*total* = T*edge* + T*trans* + T*cloud*. Accuracy loss is the change in classification accuracy after using the cloud–edge collaborative inference framework. The factors that affect the model classification accuracy include the model pruning method and feature coding method.

4.2. Results and Analysis

4.2.1. Trade-Off between End-to-End Latency and Accuracy Loss

To verify whether our cloud–edge collaborative inference framework can achieve a better trade-off between end-to-end latency and accuracy loss, we used VGG16 and ResNet18 networks for image classification tasks on the CIFAR-10 dataset. We used the same hardware platform, kept the network bandwidth at 1 Mbps, and measured the end-to-end latency and accuracy loss of the different frameworks. We compared the Only Splitting framework with our framework on MobileNetV1 and MobileNetV2 networks. The experimental results are shown in Table 2.

Based on the experimental results, we can observe that our cloud–edge collaborative inference framework achieved optimal results on both VGG16 and ResNet18. Compared with the Only Splitting framework, for VGG16, our framework reduced end-to-end latency by 82.69% with 0.89% accuracy loss; for ResNet18, its inference speed increased by 83.61% with 0.84% accuracy loss. The overall structure of our framework is most similar to CCTO; compared with it, our approach reduced the end-to-end latency by 14.54% and 15.86% in VGG16 and ResNet18, respectively, and we achieved lower accuracy loss. In addition, MoblieNetV1 and MoblieNetV2 are lightweight neural networks designed for edge computing. The existing frameworks do not provide their inference latency and accuracy loss on both networks. We also use them as our experimental networks. For MoblieNetV1, our framework reduced end-to-end latency by 15.13% with 0.62% accuracy loss. Our framework improved inference speed by 10.02% on MoblieNetV2 with only 0.27% accuracy loss. For MobileNetV1 and MoblieNetV2, which perform well on their own, our framework can further improve the upper bound of their performance, which proves the effectiveness of our framework. The experiments demonstrate that our framework can achieve a better trade-off between end-to-end latency and accuracy loss. It shows that

our framework can adapt to neural networks with more complex structures compared to other frameworks.

Table 2. Comparison of different collaborative inference frameworks.

Test Network	Comparison Framework	Params (M)	FLOPs (M)	End-to-End Latency (ms)	Acc.(TOP-1) (%)	Acc. Loss (%)
VGG16	Only Splitting	15.8	436.8	65.52	92.63	-
	Two-step Pruning [15]	15.8	436.8	19.83	90.17	2.46
	BottleNet++ [32]	15.8	436.8	16.48	91.06	1.57
	CCTO [16]	15.8	436.8	13.27	90.94	1.69
	SAFBMP (Ours)	15.8	436.8	11.34	91.74	0.89
ResNet18	Only Splitting	11.69	37.52	43.68	94.18	-
	Two-step Pruning [15]	11.69	37.52	12.09	91.36	2.82
	BottleNet++ [32]	11.69	37.52	9.34	93.02	1.16
	CCTO [16]	11.69	37.52	8.51	92.53	1.65
	SAFBMP (Ours)	11.69	37.52	7.16	93.34	0.84
MobileNetV1	Only Splitting	3.2	46.35	10.64	88.57	-
	SAFBMP (Ours)	3.2	46.35	9.03	87.95	0.62
MobileNetV2	Only Splitting	3.47	7.39	9.58	89.54	-
	SAFBMP (Ours)	3.47	7.39	8.62	89.27	0.27

4.2.2. Trade-Off between Computation and Communication

Our cloud–edge collaborative inference framework aims to enable DNNs to be better deployed on resource-constrained edge devices, so the evaluation of computation and communication trade-offs can validate the effectiveness of our methods on edge devices. We deployed tasks with different computational complexity on edge nodes to test the communication overhead. Then, we analyzed the performance of our framework on resource-constrained edge devices and made a comparison with other frameworks. The dataset for the experiment was CIFAR-10 and the test network was VGG16. The experimental results are shown in Figure 8. According to the communication-computation trade-off curves, our framework enabled a superior trade-off between computation and communication overhead when performing tasks with the same FLOPs on edge devices. In particular, our framework significantly reduced the communication overhead compared with the cloud–edge collaborative inference framework that only uses model splitting.

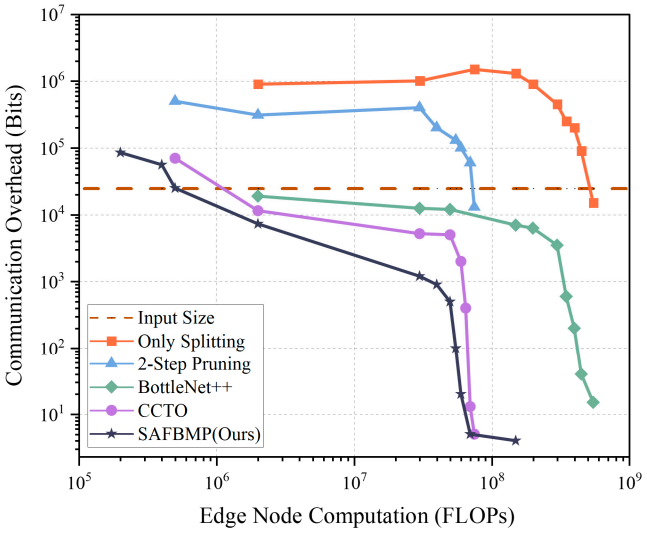


Figure 8. The computation-communication trade-off among different cloud–edge collaborative inference frameworks.

4.2.3. Inference Latency under Low Network Bandwidth

We kept the computation capability and storage capacity fixed and tested the inference latency under low network bandwidth. Figure 9 shows the comparison of the relationship between inference latency and network bandwidth for different frameworks. Among all the frameworks, our framework had the lowest latency while achieving the most stable performance for the same network bandwidth. Our framework includes a sparsity-aware module that enables us to flexibly adapt the model's FLOPs on edge devices when the network bandwidth changes. Meanwhile, because our framework has a task-oriented asymmetric feature coding structure, allowing our framework to have more choices for computation offloading.

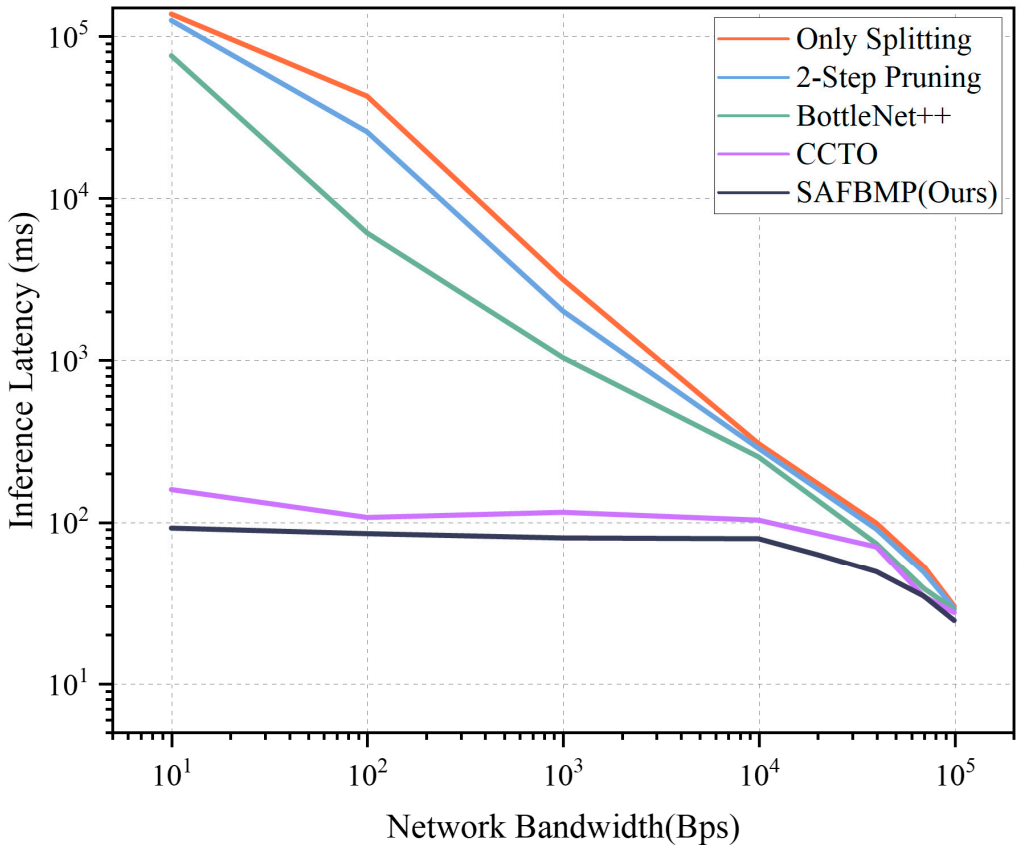


Figure 9. The comparison of the relationship between inference latency and network bandwidth for different frameworks.

4.2.4. Impact of Communication Environment

This section explores the impact of the communication environment on the inference latency of our framework. We used the WiFi network as the external traffic load and changed the WiFi network bandwidth from 1 Mbps to 5 Mbps to simulate the external communication environment. Figure 10 shows the impact of communication environment changes on our framework. The results showed that the inference latency gradually decreased with the increase in network bandwidth, but the decreasing trend was not obvious. Even if the network communication bandwidth was sufficient, our framework still maximized the edge devices' computing resources rather than simply transferring all the computational tasks to the cloud. It indicates that our cloud–edge collaborative inference framework can maintain stability when facing changing network conditions.

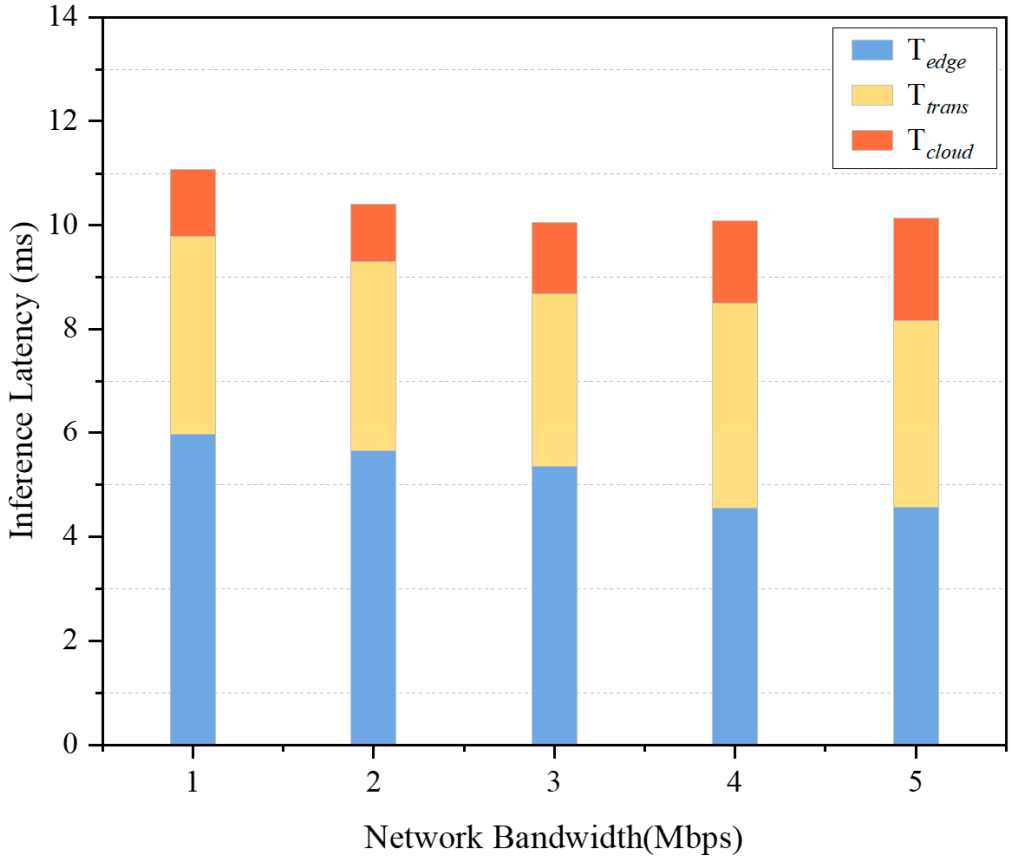


Figure 10. The relationship between inference latency and network bandwidth.

4.3. Ablation Study

Our cloud–edge collaborative inference framework adopts channel pruning to lighten the models deployed on edge devices and introduces a feature coding approach to compress the intermediate features generated by model splitting to reduce the communication overhead between cloud and edge. To further explore the role played by our designed network pruning approach and feature coding structure in reducing the model inference latency, we designed the following comparison frameworks: (1) a cloud–edge collaborative inference framework based on model splitting; (2) a cloud–edge collaborative inference framework based on model splitting and network pruning (without feature coding); (3) a cloud–edge collaborative inference framework based on model splitting and feature coding (without network pruning); and (4) a cloud–edge collaborative inference framework based on model splitting, network pruning, and feature coding. We used VGG16 as the experimental network, CIFAR-10 as the experimental dataset, and the average upload transmission rate was 1 Mbps.

In the first part of the neural network, the intermediate features are larger than the input features due to the increase in the number of feature mappings, which increases the inference latency. Figure 11 shows that our designed channel pruning and feature coding can effectively compress the redundant data. When using our designed pruning method on the neural network alone, it is better than the framework with model splitting at different network layers, and in the best of circumstances, it can reduce the inference latency by 2.84×. The latency at different network layers when only adding feature coding to the network is lower than the way without feature coding, which demonstrates that although our feature coding introduces extra computation, the time cost of extra computation is much lower than the communication overhead of directly transmitting intermediate features.

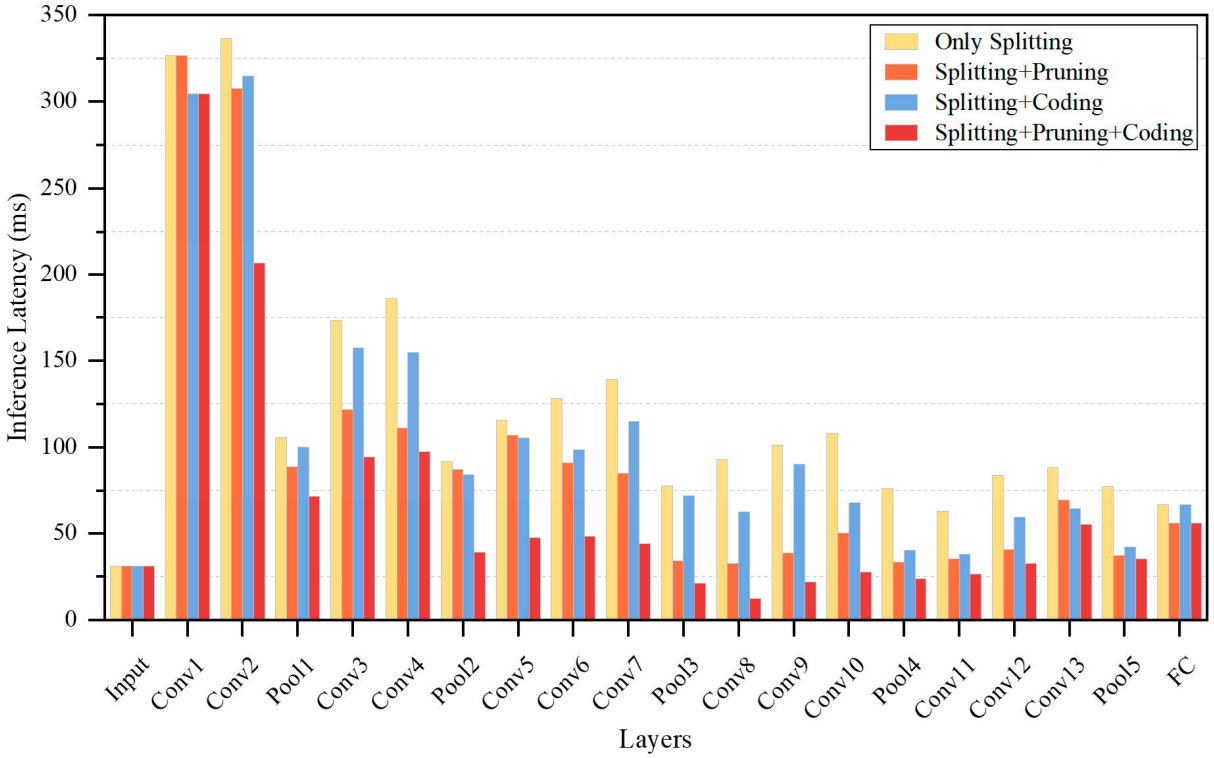


Figure 11. The relationship between inference latency and partition layers on different frameworks and the comparison of different collaborative inference frameworks.

5. Conclusions

In this paper, we have designed a cloud–edge collaborative inference framework for the deployment of DNNs on resource-constrained edge devices. Our framework improved the inference speed by 82% to 84% compared to cloud–edge collaborative inference frameworks with model splitting because of the sparsity-aware feature bias minimization method in our framework, enabling us to reduce the model FLOPs deployed on the edge devices; additionally, our task-oriented asymmetric feature coding method compresses the intermediate features to further reduce the communication overhead between cloud and edge. Our sparsity-aware feature bias minimization method uses feature bias as the criterion for removing channels, which solves the problem of pruning methods in similar frameworks that tend to be locally optimized and over-sparse. Furthermore, our task-oriented asymmetric feature coding accounts for the load distribution between cloud and edge more rationally. As a result, our framework outperforms similar frameworks on the same computing tasks with lower inference latency, accuracy loss, and communication overhead.

We explore the possibility of using multiple compression methods for cloud–edge collaborative computing, providing a new approach to DNNs for resource-constrained devices (such as embedded devices, and smart sensors). We validated the performance of our framework on typical network structures, but edge devices need to use different networks to perform better. More work is needed to refine our framework's methods to enable it to adapt to complex and diverse network structures. In addition, energy consumption is a factor that should be considered when deploying DNNs on edge devices; we plan to combine methods such as reinforcement learning to save energy consumption and enhance energy efficiency. For future work, we will conduct more experiments to explore the application of our framework in the Internet of Vehicles, smart manufacturing, and smart cities.

Author Contributions: Conceptualization, M.L. and X.Z.; methodology, M.L.; software, M.L.; validation, J.G.; formal analysis, M.L.; investigation, M.L.; resources, X.Z.; writing—original draft preparation, M.L.; writing—review and editing, M.L., F.L. and X.Z.; visualization, M.L.; supervision, X.Z.; project administration, M.L. and X.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Science and Technology Key Projects of Guangxi Province, grant number 2020AA21077007; the Innovation Project of Guangxi Graduate Education, grant number YCSW2022042; the Guangxi New Engineering Research and Practice Project, grant number XGK2022003; and the Central Guidance on Local Science and Technology Development Fund of Guangxi Province, grant number 202201002. The APC was funded by Science and Technology Key Projects of Guangxi Province, grant number 2020AA21077007.

Data Availability Statement: The dataset presented in this study is openly available at http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz, and accessed on 21 May 2023.

Acknowledgments: The authors would like to express appreciation to the editors and reviewers for their valuable comments and suggestions. Relevant persons from the School of Computer, Electronics and Information, Guangxi University are greatly appreciated for their valuable comments and suggestions. This work is supported by Guangxi Key Laboratory of Multimedia Communications and Network Technology, Guangxi, China, School of Computer, Electronics and Information, Guangxi University, Nanning, Guangxi, China.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition(CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
2. Teerapittayanon, S.; McDanel, B.; Kung, H.T. Distributed Deep Neural Networks Over the Cloud, the Edge, and End Devices. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems(ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 328–339.
3. Kang, Y.; Hauswald, J.; Gao, C.; Rovinski, A.; Mudge, T.; Mars, J.; Tang, L. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Comput. Archit. News ACM* **2017**, *45*, 615–629.
4. Li, H.; Hu, C.; Jiang, J.; Wang, Z.; Wen, Y.; Zhu, W. Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution. In Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11–13 December 2018; pp. 671–678.
5. Lane, N.D.; Bhattacharya, S.; Georgiev, P.; Forlivesi, C.; Jiao, L.; Qendro, L.; Kawsar, F. DeepX: A software accelerator for low-power deep learning inference on mobile devices. In Proceedings of the 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Vienna, Austria, 11–14 April 2016; p. 23.
6. Jeong, H.J.; Jeong, I.; Lee, H.J.; Moon, S.M. Computation offloading for machine learning Web apps in the edge server environment. In Proceedings of the IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2–6 July 2018; pp. 1492–1499.
7. Pacheco, R.G.; Couto, R.S.; Simeone, O. Calibration-aided edge inference offloading via adaptive model partitioning of deep neural networks. In Proceedings of the ICC 2021-IEEE International Conference on Communications, Montreal, QC, Canada, 14–23 June 2021; pp. 1–6.
8. Pacheco, R.G.; Oliveira, F.D.; Couto, R.S. Early-exit deep neural networks for distorted images: Providing an efficient edge offloading. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021; pp. 1–6.
9. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both weights and connections for efficient neural network. In Proceedings of the Advances in Neural Information Processing Systems 28 (NIPS 2015), Montreal, QC, Canada, 7–12 December 2015; pp. 1135–1143.
10. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. Xnornet: Imagenet classification using binary convolutional neural networks. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 525–542.
11. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. In Proceedings of the NeurIPS Learning and Representation Learning Workshop, Montreal, QC, Canada, 7–12 December 2015.
12. Denton, E.L.; Zaremba, W.; Bruna, J.; LeCun, Y.; Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In Proceedings of the Advances in Neural Information Processing Systems 27 (NIPS 2014), Montreal, QC, Canada, 8–13 December 2014; pp. 1269–1277.
13. Zhu, M.; Gupta, S. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.

14. Li, L.; Ota, K.; Dong, M. Deep Learning for Smart Industry: Efficient Manufacture Inspection System with Fog Computing. *IEEE Trans. Ind. Inf.* **2018**, *14*, 4665–4673.
15. Shi, W.; Hou, Y.; Zhou, S.; Niu, Z.; Zhang, Y.; Geng, L. Improving Device-Edge Cooperative Inference of Deep Learning via 2-Step Pruning. In Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Paris, France, 29 April–2 May 2019; pp. 1–6.
16. Shao, J.; Zhang, J. Communication-Computation Trade-off in Resource-Constrained Edge Inference. *IEEE Commun. Mag.* **2020**, *58*, 20–26.
17. He, Y.; Zhang, X.; Sun, J. Channel pruning for accelerating very deep neural networks. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 1389–1397.
18. He, Y.; Xiao, L. Structured Pruning for Deep Convolutional Neural Networks: A survey. *arXiv* **2023**, arXiv:2303.00566.
19. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning filters for efficient convnets. In Proceedings of the International Conference on Learning Representations, ICLR 2017, Toulon, France, 24–26 April 2017.
20. He, Y.; Kang, G.; Dong, X.; Fu, Y.; Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. In Proceedings of the International Joint Conference on Artificial Intelligence, Stockholm, Sweden, 13–19 July 2018; pp. 2234–2240.
21. Lin, M.; Ji, R.; Wang, Y.; Zhang, Y.; Zhang, B.; Tian, Y.; Shao, L. Hrank: Filter pruning using high-rank feature map. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 1529–1538.
22. Sui, Y.; Yin, M.; Xie, Y.; Phan, H.; Aliari Zonouz, S.; Yuan, B. Chip: Channel independence-based pruning for compact neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Virtual, 6–14 December 2021; Volume 34, pp. 24604–24616.
23. Yu, R.; Li, A.; Chen, C.F.; Lai, J.H.; Morariu, V.I.; Han, X.; Gao, M.; Lin, C.Y.; Davis, L.S. Nisp: Pruning networks using neuron importance score propagation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 9194–9203.
24. Eshratifar, A.E.; Abrishami, M.S.; Pedram, M. JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Trans. Mobile Comput.* **2019**, *20*, 565–576. [CrossRef]
25. Ko, J.H.; Na, T.; Amir, M.F.; Mukhopadhyay, S. Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. In Proceedings of the 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), Auckland, New Zealand, 27–30 November 2018; pp. 1–6.
26. Mentzer, F.; Agustsson, E.; Tschannen, M.; Timofte, R.; Van Gool, L. Conditional probability models for deep image compression. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4394–4402.
27. Agustsson, E.; Tschannen, M.; Mentzer, F.; Timofte, R.; Gool, L.V. Generative adversarial networks for extreme learned image compression. In Proceedings of the IEEE International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 221–231.
28. Yao, S.; Li, J.; Liu, D.; Wang, T.; Liu, S.; Shao, H.; Abdelzaher, T. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In Proceedings of the 18th Conference on Embedded Networked Sensor Systems, Virtual, 16–19 November 2020.
29. Jiang, W.; Ning, P.; Wang, R. Slic: Self-conditioned adaptive transform with large-scale receptive fields for learned image compression. *arXiv* **2023**, arXiv:2304.09571.
30. Yao, J.; Zhang, S.; Yao, Y.; Wang, F.; Ma, J.; Zhang, J.; Chu, Y.; Ji, L.; Jia, K.; Shen, T.; et al. Edge-Cloud Polarization and Collaboration: A Comprehensive Survey for AI. *IEEE Trans. Knowl. Data Eng.* **2021**, *35*, 6866–6886. [CrossRef]
31. Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; Zhang, J. Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing. *IEEE* **2019**, *107*, 1738–1762. [CrossRef]
32. Shao, J.; Zhang, J. Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems. In Proceedings of the 2020 IEEE International Conference on Communications Workshops, Dublin, Ireland, 7–11 June 2020; pp. 1–6.
33. Matsubara, Y.; Callegaro, D.; Baidya, S.; Levorato, M.; Singh, S. Head Network Distillation: Splitting Distilled Deep Neural Networks for Resource-Constrained Edge Computing Systems. *IEEE Access* **2020**, *8*, 212177–212193. [CrossRef]
34. Banitalebi-Dehkordi, A.; Vedula, N.; Pei, J.; Xia, F.; Wang, L.; Zhang, Y. Auto-Split: A General Framework of Collaborative Edge-Cloud AI. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD'21). Association for Computing Machinery, New York, NY, USA, 14–18 August 2021; pp. 2543–2553.
35. Duan, Y.; Zhou, Y.; He, P.; Liu, Q.; Duan, S.; Hu, X. Network Pruning via Feature Shift Minimization. In Proceedings of the Asian Conference on Computer Vision, Macau SAR, China, 4–8 December 2022; pp. 4044–4060.
36. Jiang, N.F.; Zhao, X.; Zhao, C.Y.; An, Y.Q.; Tang, M.; Wang, J.Q. Pruning-aware sparse regularization for network pruning. *Mach. Intell. Res.* **2023**, *20*, 109–120. [CrossRef]
37. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning, ACM, Lille, France, 7–9 July 2015; pp. 448–456.
38. Glorot, X.; Bordes, A.; Bengio, Y. Deep Sparse Rectifier Neural Networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR, Fort Lauderdale, FL, USA, 11–13 April 2011; pp. 315–323.
39. Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; Zhang, C. Learning efficient convolutional networks through network slimming. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2755–2763.

40. Chen, J.; Kao, S.H.; He, H.; Zhuo, W.; Wen, S.; Lee, C.H.; Chan, S.H. Run, Don't Walk: Chasing Higher FLOPS for Faster Neural Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 18–22 June 2023; pp. 12021–12031.
41. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
42. Zhang, H.; Goodfellow, I.; Metaxas, D.; Odena, A. Self-Attention Generative Adversarial Networks. In Proceedings of the 36th International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; Volume 97, pp. 7354–7363.
43. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015; pp. 1–14.
44. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
45. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; Technical Report; University of Toronto: Toronto, ON, Canada, 2009.