applied sciences

MDPI

Article

Partitioning DNNs for Optimizing Distributed Inference Performance on Cooperative Edge Devices: A Genetic Algorithm Approach

Jun Na [1], Handuo Zhang [2], Jiaxin Lian [2] and Bin Zhang [1,*]

[1] Software College, Northeastern University, Shenyang 110169, China
[2] School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China
* Correspondence: zhangbin@mail.neu.edu.cn

Abstract: To fully unleash the potential of edge devices, it is popular to cut a neural network into multiple pieces and distribute them among available edge devices to perform inference cooperatively. Up to now, the problem of partitioning a deep neural network (DNN), which can result in the optimal distributed inferencing performance, has not been adequately addressed. This paper proposes a novel layer-based DNN partitioning approach to obtain an optimal distributed deployment solution. In order to ensure the applicability of the resulted deployment scheme, this work defines the partitioning problem as a constrained optimization problem and puts forward an improved genetic algorithm (GA). Compared with the basic GA, the proposed algorithm can result in a running time approximately one to three times shorter than the basic GA while achieving a better deployment.

Keywords: DNN partitioning; deployment optimization; genetic algorithm; distributed DNN inferencing

check for updates

Citation: Na, J.; Zhang, H.; Lian, J.; Zhang, B. Partitioning DNNs for Optimizing Distributed Inference Performance on Cooperative Edge Devices: A Genetic Algorithm Approach. *Appl. Sci.* **2022**, *12*, 10619. https://doi.org/10.3390/app122010619

Academic Editors: Neil Yuwen Yen, Chao-Tung Yang, Chen-Kun Tsung and Vinod Kumar Verma

Received: 19 September 2022
Accepted: 18 October 2022
Published: 20 October 2022

1. Introduction

Internet of things (IoT), edge computing (EC), and artificial intelligence (AI) are three technological pillars of the current industrial revolution [1,2]. One hot topic in applying these techniques is edge intelligence [3–5], which involves running deep learning algorithms at the edge of the networks instead of entirely offloading the inferencing tasks to the cloud center. As edge intelligence can alleviate problems such as large bandwidth occupancy, high transmission delay, slow response speed, poor network reliability, and leakage of personal or sensitive information, it is being intensively researched and widely used today. For example, it is common to deploy a trained convolutional neural network (CNN) to the edge for performing real-time video analysis in applications including autonomous driving [6], intelligent monitoring [7], industrial IoT [8], smart cities [9], etc. However, as CNN inference is usually computationally intensive and some CNNs are huge, it is often infeasible to deploy a complete CNN model or perform inference on a single-edge device.

On one hand, the above problem can be solved by pruning, quantization, or knowledge distillation [10–12] to obtain a smaller DNN model before deploying and inferencing. However, these techniques may sacrifice model inference accuracy somewhat. On the other hand, the powerful cloud server is usually adopted to perform part of the inferencing task [13–15]. In such a cloud-assisted approach, a DNN model is often divided into two parts: one remains locally and the other runs at the remote cloud server. As with offloading a complete DNN model to the cloud server, cloud-assisted approaches also need to face the problem of private data leakage, which is natural in cloud computing. Additionally, balancing the calculation accuracy, end-to-end delay, and resource occupancy is also challenging.

To fully unleash the potential of edge devices, it is popular to cut a neural network into multiple pieces and distribute them among available edge devices to perform the inference cooperatively [16,17]. This approach could overcome the problems above by keeping the inferencing process in the edge network. Nevertheless, it is more challenging to partition and distribute a neural network to achieve optimal performance, as it is an NP-hard problem. Although some strategies have been developed in an attempt to split a DNN into several parts effectively [18–20], most of them pay more attention to the methodology of reorganizing the network structure rather than optimizing the process for getting an optimal solution from the perspective of the actual system running. Hence, the problem of partitioning a DNN model to achieve optimal deployment has not been adequately addressed.

This paper proposes a novel layer-based partitioning approach to obtain an optimal DNN deployment solution. In order to ensure the applicability of the resulting deployment scheme, the partitioning problem is defined as a constrained optimization problem and an improved genetic algorithm (GA) is proposed to ensure the generation of feasible candidate solutions after each crossover and mutation operation. Compared to the basic GA, the proposed GA in this paper results a running time that is one to three times shorter than that of the basic GA, while obtaining a better deployment. The main contributions of this paper are as follows:

- Firstly, the DNN model partitioning problem is modeled as a constrained optimization problem and the corresponding problem is introduced.
- Secondly, the paper puts forward a novel genetic algorithm to shorten solving time by ensuring the validity of chromosomes after crossover and mutation operation.
- Finally, experiments are performed on several existing DNN models, including AlexNet, ResNet110, MobelNet, and SqueenzeNet, to present a more comprehensive evaluation.

The remainder of this paper is organized as follows: Section 2 gives an overview of the related work. Section 3 presents the problem definition of the DNN partition problem. Section 4 introduces the details of the proposed algorithm. Section 5 provides the experimental results, and Section 6 concludes the paper.

2. Literature Review

As most modern DNNs are constructed by layers, such as the convolutional layer, the fully connected layer, and the pooling layer, layer-based partitioning is the most intuitive DNN partitioning strategy. For example, Ref. [14] proposed to partition a CNN model at the end of the convolutional layer, allocating the convolutional layers at the edge and the rest of the fully-connected layers at the host. Unlike this fixed partitioning strategy, recent methodologies have focused on adapting their results to the actual inferencing environment. Generally, depending on the construction of the target deployment environment, existing methods are divided into the following two categories.

According to the basic idea of the cloud-assisted approaches, some studies try to divide a given DNN model into two sets and push the latter part to the cloud server. For example, Ref. [13] designed a lightweight scheduler named Neurosurgeon to automatically partition DNN computation between mobile devices and data centers based on neural network layers. Similarly, Refs. [21–23] adopted the same strategy, while they took some further processing. In [21], the authors integrated DNN right-sizing to accelerate the inference by early exiting inference at an intermediate layer. In contrast, Ref. [22] first added early exiting points to the original network and then partitioned the reformed network into two parts. To determine the optimal single cut point, all of [13,21,22] applied exhaustive searching, while [23] solved the problem with mixed-integer linear programming.

For making full use of the available resources in the edge environment, more DNN partitioning strategies have been emerging to divide a DNN model into more than two pieces for distributing the inference task among several edge devices. Generally, based on the object to be partitioned, there are four kinds of main strategies, i.e., partitioning the inputs [24,25], weights [26], and layers [18,19], as well as hybrid strategies [17,20,27–30]. Partitioning the inputs or weights focuses on the large storage requirements for storing large inputs or weights. Partitioning the DNN layers can solve the depth problem of DNN inferencing. Furthermore, the hybrid strategies aim to solve both problems mentioned above. For example, Ref. [27] employed input partitioning after layer-based partitioning to obtain a small enough group of inferencing tasks to be executed. The authors of [20] proposed fused tile partitioning (FTP) to fuse layers and partition them vertically in a grid fashion. The authors of [29] modeled a neural network as a data-flow graph where vertices are input data, operations, or output data and edges are data transfers between vertices. Then, the problem was transformed into a graph partitioning problem.

Nearly all of the above works take inference delay or energy consumption as the optimization objectives. Recently, more studies have begun to focus on the joint optimization of DNN partitioning and resource allocation [31–33]. However, it is still an open and critical challenge to achieve an optimal DNN distributed deployment. Unlike existing approaches, this work models the DNN partitioning problem as a constrained optimization problem, aiming to achieve the optimal inference performance with available resources in the edge environment. Moreover, it proposes a novel genetic algorithm to optimize the solving process of the formulated optimization problem.

3. System Model and Problem Formulation

This section provides an overview of the motivation and fundamental process of the proposed DNN partitioning approach and presents a formal problem description. Suppose that there are N edge devices and an edge server forming an edge network. The edge server acts as a master to receive user requests, partition DNN models, and assign the DNN inferencing tasks for each edge device. Take video-based fall detection in health monitoring as an example. A video-based fall detection application takes a video stream as the input and recognizes if there is a human falling based on a given neural network. Due to the latency and privacy requirements, such applications are best deployed in an edge environment. In order to avoid the edge server becoming the inference bottleneck of all the edge intelligent applications, it is better to distribute the background inferencing task to other edge devices.

As illustrated in Figure 1, after a user deploys a fall detection application in the edge environment through a user interface, the edge server will extract the neural network inferencing task and the corresponding neural network. It partitions and dispatches the neural network according to the current status of each edge device, such as the smart camera, the smart speaker, and the sweeping robot in Figure 1. Then, the neural network is divided into three parts in this example, i.e., $p1$, $p2$, and $p3$, and deployed to the smart camera, the smart speaker, and the sweeping robot, respectively. All these selected devices will cooperate to complete a further distributed inferencing process without the edge server. Specifically, the smart camera will run the partition $p1$ and send its output to the smart speaker as the input of partition $p2$. The sweeping robot, in turn, performs the partition $p3$ after receiving the smart speaker's output, then outputs the recognized result.

As a group of edge devices will cooperate in executing a single DNN, an edge device must receive an input from a preceding device, perform the inferencing task, and deliver the output to the next device. Suppose the DNN model is divided into n pieces and deployed to n different edge devices. Using p_i $(i = 1, 2, \ldots, n)$ to represent a sub-model of a given DNN and $d_j(j = 1, 2, \ldots, n)$ to represent a selected device, if a sub-model p_i is deployed to a device d_j, the corresponding execution time $t_{i,j}$ is defined as

$$t_{i,j} = tc_{i,j} + tr_i + ts_i \tag{1}$$

where $tc_{i,j}$ is the time of executing sub-model p_i on device d_j, and tr_i and ts_i are the time for receiving the input of p_i and sending the output of p_i, respectively. If tt_i is used to represent the total transmission time, then $tt_i = tr_i + ts_i$ and $t_{i,j} = tc_{i,j} + tt_i$.

In addition, because not all sub-models can run directly on any edge device, it also needs to consider whether an edge device can complete a specific inferencing task according to its current state. For example, it is necessary to determine that the available memory is enough and its remaining battery capacity is sufficient. Suppose m_j is the size of available memory on device d_j and rm_i is the required memory for running sub-model p_i. If p_i can be executed on device d_j, the following inequality must be true.

$$rm_i \leq m_j \tag{2}$$

Similarly, if ep_j is the average running power of device d_j and c_j is the remaining battery capacity on device d_j, then if p_i can be executed on device d_j, the following inequality must be true.

$$ep_j \times t_{i,j} \leq c_j \tag{3}$$

Above all, the DNN partitioning problem is formulated as a constrained optimization problem, trying to minimizing the total execution time of the DNN inferencing under given limitation of edge devices' available memory and energy. The corresponding objective function is formulated as follows:

$$\begin{aligned} &\min \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_{i,j} \times t_{i,j} \\ &s.t. \begin{cases} rm_i \leq m_j & \alpha_{i,j} = 1, \\ ep_j \times t_{i,j} \leq c_j & \alpha_{i,j} = 1, \\ \sum_{i=1}^{n} \alpha_{i,j} = 1 & \forall j \in \{1,\ldots,n\}, \\ \sum_{j=1}^{n} \alpha_{i,j} = 1 & \forall i \in \{1,\ldots,n\}, \\ \alpha_{i,j} \in \{0,1\} & \forall i,j \in \{1,\ldots,n\}. \end{cases} \end{aligned} \tag{4}$$

Here, $\alpha_{i,j}$ is a coefficient to indicate whether model p_i will be assigned to device d_j and the value of $\alpha_{i,j}$ is either 0 or 1. When $\alpha_{i,j} = 1$, model p_i will be assigned to device d_j. Otherwise, model p_i will be assigned to other device rather than d_j. Assume that the model will be divided into n parts, and each part will be uniquely deployed to one specific device, then for any device $d_j(j = 1,2,\ldots,n)$, the equation $\sum_{i=1}^{n} \alpha_{i,j} = 1$ is reasonable; the same is true for any sub-model $p_i(i = 1,2,\ldots,n)$, $\sum_{j=1}^{n} \alpha_{i,j} = 1$. All of these $\alpha_{i,j}(\forall i,j \in \{1,\ldots,n\})$ form an n-by-n matrix. The goal is to achieve a specific matrix with the shortest execution time under the constraints of memory and energy consumption.

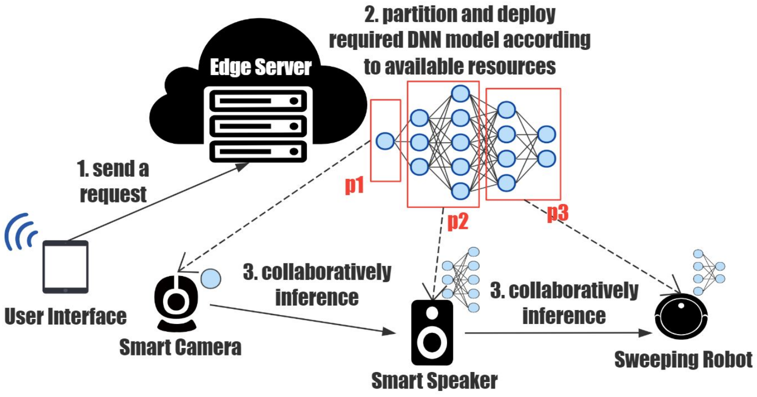


Figure 1. Illustration of a distributed DNN inference by collaboration between multiple edge devices.

4. The Proposed Genetic Algorithm

Genetic algorithms (GA) are a method of searching for an optimal solution by simulating the natural evolution process. When solving complex combinatorial optimization problems, GA can usually obtain better optimization results faster than some conventional optimization algorithms. This section will first analyze the problems that face the basic genetic algorithm when solving the optimization problem formulated in the previous section. On this basis, it puts forward the ideas for the improvements in this work and then describes the corresponding algorithms in detail.

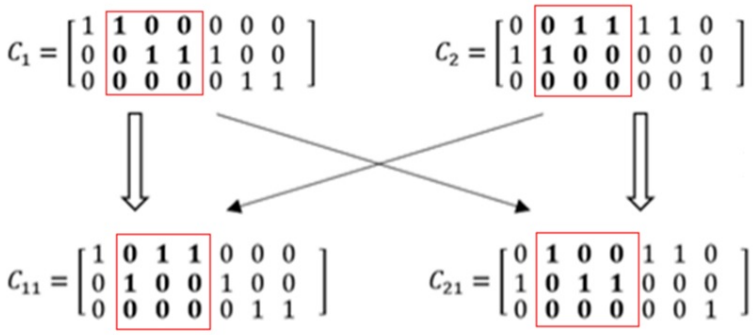
4.1. Problems of Applying Basic GA for DNN Partitioning

The chromosome coding scheme is the fundamental element of GA. In solving the above DNN partitioning problem, assume that each chromosome represents an actual distributed deployment solution. Suppose the required DNN model has l layers, which will be divided into $n(n \leq l)$ pieces and deployed to n different edge devices. To ensure that each sub-model contains only continuous layers to avoid extra data transmission costs, this study constructs a matrix of n rows and l columns to represent a specific deployment scheme, i.e., a chromosome in the GA. For example, the following matrix denotes the deployment scheme that partitions a DNN model with seven layers into three parts and distributes it to three different devices.

$$c = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{5}$$

If L_i is the ith $(i = 1, 2, \ldots, l)$ layer in a given DNN model, this chromosome represents that L_1 and L_2 will be divided into a group and deployed on device d_1, L_3, L_4 and L_5 will be divided into a group and deployed on device d_2, and L_6 and L_7 will be divided into a group and deployed on device d_3.

The basic process of GA starts with generating an initial population, i.e., a set of chromosomes following the above coding scheme. Then, it will run through the loop, including individual evaluation, selection, crossover, and variation, until satisfying the given termination condition. The cross operation plays a core role in GA, which acts on a group of chromosomes and generates new individuals by replacing part of the chromosomes of two father-generation individuals. Figure 2 shows a simple example of computation in a partially mapped crossover operator.



$C_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$
$C_2 = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
$C_{11} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$
$C_{21} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

Figure 2. Illustration of the computing process in the partially mapped crossover operator.

In Figure 2, C_1 and C_2 are the two father individuals, while C_{11} and C_{21} are two new individuals generated by swapping the subsections in each father individual included in the rectangles. It is not difficult to find that the assumption that each sub-model only contains continuous layers is broken during the above crossover operation. For example, layers L_2 and L_5 are grouped together and deployed to device d_2 in the left new individual C_{11}, while layers L_2, L_5, and L_6 are grouped together and L_1, L_3 and $L4$ are grouped together in the right new individual C_{21}. Such deployments will lead to extra network bandwidth and equipment energy consumption caused by repeated transmission between devices. For example, if deploying the DNN according to C_{11}, the output of L_1 will be sent from d_1 to d_2, and then the output of L_2 will be sent back from d_2 to d_1. In turn, the output of L_4 will be sent from d_1 to d_2 again. As a result, the intermediate results need to be transferred four times among the three devices, twice as many as deployed according to C_1.

4.2. The Proposed Improvement

To ensure reasonable individuals that only group continuous layers together after crossover and mutation, this work proposes to distinguish partitioning and deployment by constructing two-layer chromosomes, i.e., partitioning chromosomes and deployment chromosomes. A partitioning chromosome represents a certain partitioning scheme and a deployment chromosome represents a specific deployment scheme. Figure 3 shows an example of the relationship between a DNN structure, a partitioning chromosome, and a deployment chromosome.

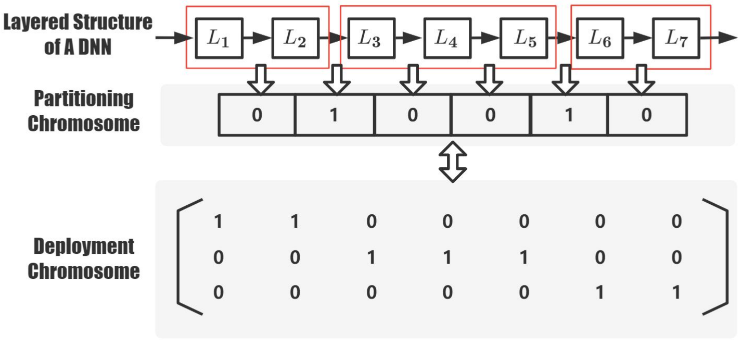


Figure 3. An example the relationship between a DNN structure, a partitioning chromosome and a deployment chromosome.

In Figure 3, there is a DNN with seven layers. A partitioning chromosome is represented by a one-dimensional vector whose length is $l - 1$ (l is the number of layers in a given DNN model), and each gene is a possible cut point. The given partitioning chromosome represents that the DNN is divided into three parts by splitting at the end of L_2 and L_5. According to this partitioning scheme, a group of corresponding deployments can be generated. The meaning of the example deployment chromosome in Figure 3 is the same as introduced in the above section.

Based on the description above, there is a one-to-many relationship between the partitioning chromosome and the deployment chromosome. Especially if there are n devices to participate in the collaborative inferencing, there would be $n!$ different deployment chromosomes generated from one partitioning chromosome. Conversely, only one partitioning chromosome can be abstracted from a given deployment chromosome. The detail of the conversion algorithms between partitioning chromosomes and deployment chromosomes are presented in Algorithms 1 and 2 as below.

Algorithm 1 begins with initializing two empty sets DC_lines and DC to store possible lines in a deployment chromosome and the required number of deployment chromosomes, respectively. Then, all possible lines for a deployment chromosome are constructed through lines 4 to 15. Then, the loop from lines 16 to 19 composed these possible lines in a random order to construct n specific deployment chromosomes which make up the deployment chromosome set DC.

Extracting a partitioning chromosome from a given deployment chromosome is more straightforward than generating deployment chromosomes based on a given partitioning chromosome. As shown in Algorithm 2, it only needs to read through the input deployment chromosome and compare whether every two adjacent elements are the same or not (see the for loop in Algorithm 2). If the two adjacent elements are the same, append a $'0'$ to the vector pc. Otherwise, append a $'1'$ to pc (see the if-else statement in Algorithm 2). At last, after checking the last pair of elements, the corresponding partitioning chromosome is achieved.

Algorithm 1: Deployment Chromosome Generation Algorithm

```
Input: a partitioning chromosome $pc$, the number of deployment chromosomes to
       be generated $n$
Output: $n$ deployment chromosomes
begin
    $DC\_lines = \{\}$
    $DC = \{\}$
    Extract all positions with 1 in $pc$ and put these positions in a vector $P$
    $bp = 0$
    for $p_i \in P$ do
        Create a vector $v$ with $len(pc) + 1$ zeros
        if $p_i$ is not the last element in $P$ then
            Set the value from position $bp + 1$ to $p_i$ in $v$ to 1.
        else
            Set the value from position $bp + 1$ to $len(pc) + 1$ in $v$ to 1.
        end
        Add $v$ into $DC\_lines$.
        $bp = i$
    end
    for $i = 0$ to $n$ do
        Build a matrix $dc$ by composing all vectors in $DC\_lines$ in a random order
        Add $dc$ into $DC$
    end
    return $DC$
end
```

Algorithm 2: Partitioning Chromosome Extraction Algorithm

```
Input: a deployment chromosome $dc$
Output: a partitioning chromosome $pc$
begin
    $pc = []$
    for Each line $v_i$ in $dc$ do
        $e = v_i[0]$
        for $j = 1$ to $len(v_i) - 1$ do
            if $e == v_i[j]$ then
                Append 0 into $pc$
            else
                Append 1 into $pc$
            end
            $e = v_i[j]$
        end
    end
    return $pc$
end
```

On this basis, the basic GA needs to be improved in the following two aspects.

- On the one hand, the initial population generation needs to be modified according to the above chromosome classification. The initialization process should be divided into two steps: first, the random generation of a partitioning population. Then, the derivation of the corresponding deployment population based on Algorithm 1.
- On the other hand, after selecting excellent individuals out of the deployment population, the corresponding partitioning population should be extracted based on

Algorithm 2. Then, crossover and mutation should be performed on these partitioning chromosomes and corresponding deployment individuals should be selected to produce a new deployment population.

To summarize, Algorithm 3 shows the complete framework of the improved genetic algorithm in this paper. As mentioned above, if there are n candidate devices, each partitioning chromosome can directly derive $n!$ different deployment chromosomes. To control the population size, the algorithm adopts a proportion p_{dc} $(0 < p_{dc} <= 1)$. Then, it only needs to generate $n! \times p_{dc}$ deployment chromosomes for each partitioning chromosome.

Algorithm 3: The Framework of the Proposed Genetic Algorithm

```
   Input: a DNN model description $model$, a performance description of a group of
          candidate devices $D$, initial partitioning chromosome population size $PCN$,
          the proportion of the initial deployment chromosome population $p_{dc}$,
          crossover probability $p_c$, mutation probability $p_m$, maximum number of
          iterations $MAXGEN$, the number of consecutive occurrences of the same
          optimal value $SVG$
   Output: a deployment scheme $dc$
 1 begin
 2     $n \leftarrow$ number of candidate devices in $D$
 3     $l \leftarrow$ number of layers in $model$
 4     $latency[n][l] \leftarrow$ each DNN layer's execution times on each device in $D$
 5     $PP \leftarrow$ randomly generate $PCN$ partitioning chromosomes
 6     $population \leftarrow$ generate $n! \times p_{dc}$ deployment chromosomes for each
        partitioning chromosome in $PP$
 7     $currentGen = 0$
 8     $maxf = 0$
 9     while $currentGen < MAXGEN$ do
10         $currentGen++$
11         $OP$ = selection($population$)
12         $PP \leftarrow$ extract partitioning chromosomes from $OP$
13         update $PP$ through crossover and mutation
14         $population \leftarrow$ deployment chromosomes generated based on $OP$
            conforming to $PP$
15         $maxf = maxfitness(population)$
16         if $maxf$ has appeared $SVG$ times continuously then
17             break
18         end
19     end
20     return the individual that has the maximal fitness in current population
21 end
```

Algorithm 3 first predicts and stores the execution time of each DNN layer according to Equation (1) for calculating individual fitness (from line 2 to line 4). Line 5 and line 6 initialize a deployment population. The while statement from line 9 to line 19 is the main loop in the algorithm. First, line 10 updates the current number of iterations and line 11 selects outstanding individuals from the current population to OP. Then, line 11 and line 12 extract the corresponding partitioning chromosomes from OP and perform crossover and mutation to generate a new partitioning population PP. Line 14 constructs a new deployment population according to OP and PP, each of which has a corresponding partitioning chromosome in PP. In the end, the individual with the maximum fitness in the current population is resulted through a specified number of times consecutively as the stop condition. If so, the loop is exited. Otherwise, search is continued until the maximum number of iterations MAXGEN is reached. Finally, the algorithm returns the deployment chromosome corresponding to the current maximum fitness as the final optimal deployment scheme.

According to the optimization objectives described in Section 3, the fitness function is defined as follows.

$$fitness(dc) = \begin{cases} \frac{10^5}{\sum_{i=1}^{n}\sum_{j=1}^{n} dc_{i,j} \times t_{i,j}} & \text{if } dc \text{ satisfies all constraints} \\ 10^{-6} & \text{if } dc \text{ does not satisfy all constraints} \end{cases} \quad (6)$$

In the above fitness function, dc is the α matrix in the formulated problem definition (shown in Equation (4)) for calculating the fitness of a specific deployment chromosome.

5. Performance Evaluation

This section evaluates the performance of the proposed DNN partitioning method on four real-world CNNs. It presents experimental results and compares them to other existing methodologies to demonstrate that the proposed algorithm can execute given CNN inference on a group of distributed collaborative edge devices in a shorter time.

5.1. Experiment Setting

To provide a comprehensive comparison, four common CNNs designed for running on edge devices are adopted, namely AlexNet [34], ResNet110 [35], MobileNet [36] and SqueenzeNet [37]. All of these CNNs have a diverse number of layers, memory requirements, and performance. The CNN training and inferencing is based on the Cifar-10 data set [38], which consists of 32 × 32 color images divided into ten classes.

In addition, a simulated distributed system with seven devices with different configurations is set up as shown in Table 1.

Table 1. Performance parameters of edge devices.

Device No.	GFLOPS	Battery Capacity (J)	I/O Bandwidth (MBPS)
1	0.218	250	140.85
2	9.92	20	1525.63
3	0.213	500	135.89
4	13.5	10	1698.25
5	0.247	300	140.91
6	3.62	200	159.45

To achieve the performance description of the candidate devices (D in Algorithm 3), PALEO [39] is adopted, which is an analytical performance model that can efficiently and accurately model the expected scalability and performance under a given deployment assumption. Based on PALEO, the memory requirements and execution time of different DNN layers on each given device are evaluated. In PALEO, the execution time of a single DNN layer consists of the time it takes to receive input from the upper layer, the time it takes for the current layer's computation, and the time it takes to write the output to local memory. Based on this, the energy consumption required to perform a given layer on a specific device is calculated by multiplying device power with execution time.

The other parameter values in Algorithm 3 are set as follows. The initial size of the partitioning chromosome population is five. In order to ensure fairness in comparing with the basic GA, the population size of deployment chromosomes is set to 50 fixedly, which is the same as population size set in basic GA. In both basic GA and the improved GA, crossover probability is 0.5, mutation probability is 0.01, the maximum iteration number is 200, and the algorithm will be terminated when the optimal fitness value remained unchanged for 50 consecutive generations. In basic GA, the chromosomes are generated according to the structure shown in Section 4.1.

The experiments are executed on a laptop with an AMD Ryzen7 5700U CPU and 16 GB memory in a Pycharm environment. The following results are collected by running a same algorithm ten times as a group.

5.2. Comparison of Inference Performance

In order to achieve the optimally distributed deployment scheme, the partitioning optimization and deployment optimization are considered either separately or simultaneously. The following experiments first compare the inferencing delay under a given partition scheme and then compare both inferencing delay and device average energy cost in considering partitioning and deployment simultaneously.

5.2.1. Comparison in Considering Partitioning Optimization and Deployment Optimization Separately

To obtain an optimal partition scheme, the experiment adopts the DNN partitioning algorithm proposed in [13,40] to optimally divide a given DNN into two parts and calculates the optimal deployment by exhaustive searching. The average value, maximum value, minimum value, mode, and standard deviation are achieved by running each algorithm ten times. In Table 2, method-1 and method-2 present partitioning DNN based on [13,40], respectively.

Table 2. Comparison on inferencing delay (ms).

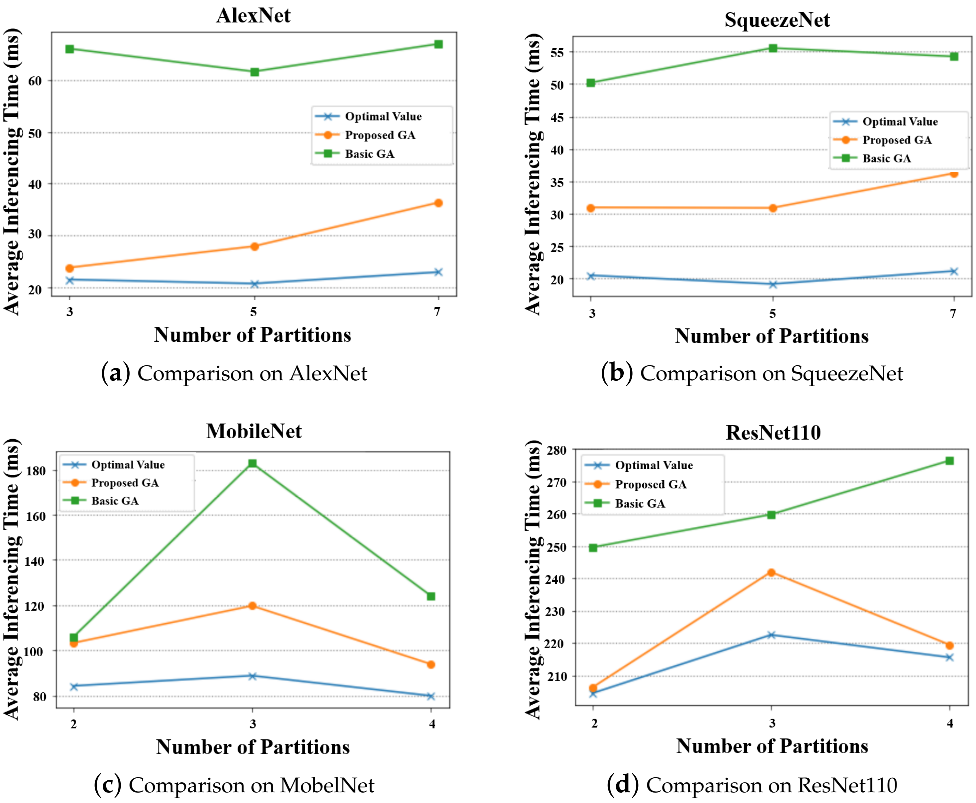
DNN Model	Method	Avg	Max	Min	Mode	SD
AlexNet	Method-1	49.73	92.88	20.97	20.97	35.23
	Method-2	42.54	92.88	20.97	20.97	32.95
	Proposed method	**28.90**	92.88	20.97	20.97	**21.44**
SqueezeNet	Method-1	36.13	51.81	20.45	51.81	15.68
	Method-2	32.99	51.81	20.45	20.45	15.36
	Proposed method	**30.54**	52.04	20.45	20.45	**14.05**
MobileNet	Method-1	96.20	114.10	84.27	84.27	14.61
	Method-2	93.22	114.10	84.27	84.27	13.67
	Proposed method	**95.86**	117.41	84.27	114.10	**15.11**
ResNet110	Method-1	232.05	259.54	204.56	259.54	27.49
	Method-2	226.55	259.54	204.56	204.56	26.93
	Proposed method	**220.03**	254.99	204.56	233.89	**16.77**

In Table 2, the average inferencing time by distributing inferencing according to the proposed algorithm is superior to other methods. In addition, the proposed algorithm results in smaller standard deviation values for partitioning most of these CNNs, which means the proposed algorithm is relatively more stable.

5.2.2. Comparison in Considering Partitioning Optimization and Deployment Optimization Simultaneously

As a genetic algorithm is an approximate algorithm, it cannot ensure obtaining the absolute optimal solution. This experiment adopts an exhaustive method to obtain the optimal inferencing delay and energy cost in any given setting as the baseline, and then compares the corresponding results by running basic GA and the improved GA in considering partitioning optimization and deployment optimization simultaneously. The following experimental results are all average values from running each algorithm ten times in every test case.

Firstly, Figure 4 shows the comparison of inferencing delay in each test case. In this figure, each sub-picture refers to a different CNN where the horizontal axis represents the number of partitions and the vertical axis represents the average inferencing time in milliseconds. It demonstrates that the inferencing delay resulting from the proposed GA is closer to the optimal value than basic GA. In addition, the trends of the proposed GA and the optimal value are also more similar.



(**a**) Comparison on AlexNet

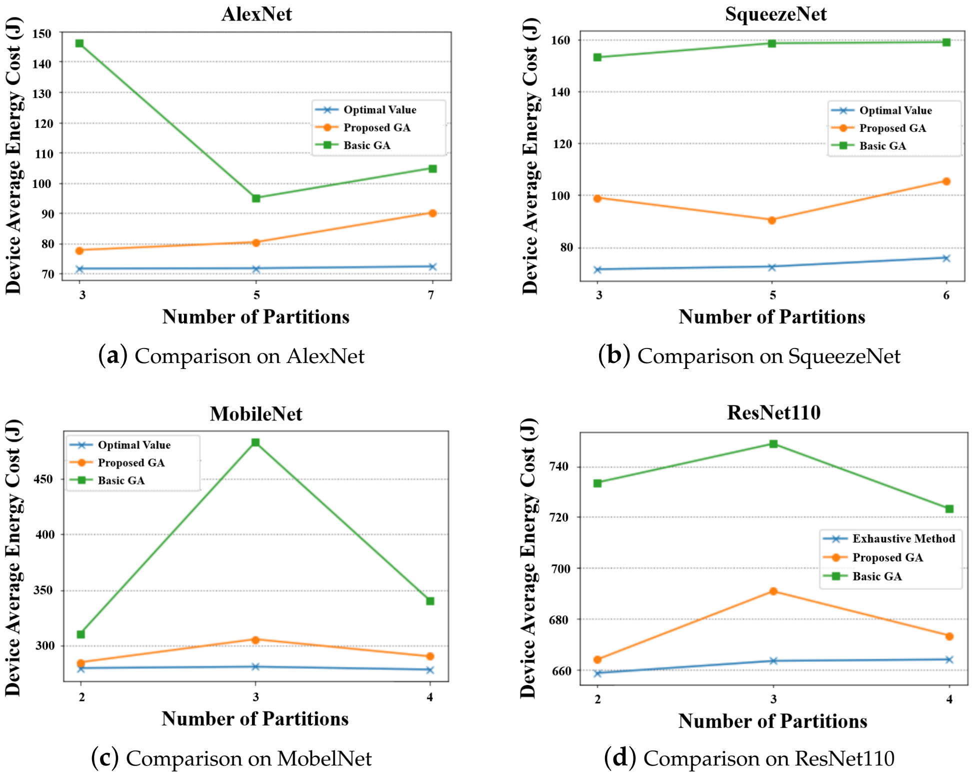
(**b**) Comparison on SqueezeNet

(**c**) Comparison on MobelNet

(**d**) Comparison on ResNet110

Figure 4. Average Inferencing Time Comparison.

A similar conclusion can be achieved by observation of Figure 5, which compares device average energy costs in each test case.



(**a**) Comparison on AlexNet

(**b**) Comparison on SqueezeNet

(**c**) Comparison on MobelNet

(**d**) Comparison on ResNet110

Figure 5. Device Average Energy Consumption Comparison.

As a result, the proposed algorithm can produce better deployments under different scenarios compared to the basic genetic algorithm. The performance of some solutions is even close to the optimal deployment generated by the exhaustive method.

5.3. Comparison of Algorithm Efficiency

From the perspective of the actual running process of an intelligent application system, as mentioned in Section 3, the usability of each edge device may keep changing. It is necessary to dynamically divide and deploy a DNN according to the latest status of edge devices when a request arrives. In this scenario, the algorithm's execution time will be accumulated to the actual system response time. Therefore, this section compares the algorithm running time of exhaustive method, basic genetic algorithm, and the proposed method in different deployment scenarios. Table 3 shows the detailed comparison results.

Table 3. Comparison on algorithm running-time (ms) under different scenarios.

DNN Model	Number of Partitions	Exhaustive Method	Improved GA	Basic GA
AlexNet	3	490.43	**267.20**	275.46
	5	7870.74	**6563.56**	7214.8
	7	3,675,603.00	**6563.56**	24,991.9
SqueezeNet	3	490.97	**372.15**	381.81
	5	41,385.8	**6568.28**	6713.38
	6	1,476,833	**11,197.10**	12,572.80
MobileNet	2	485.28	**231.70**	268.69
	3	751.76	**335.14**	341.12
	4	29,311.00	**1238.22**	1357.69
ResNet110	2	530.85	**269.12**	381.98
	3	12,665.30	**1620.24**	4086.40
	4	5,217,793.75	**7322.81**	19,345.10

The above table shows that the running time of all three algorithms increases significantly with the growing number of devices or DNN layers. However, the improved GA needs the least time to obtain a better solution. For example, in partitioning AlexNet, the improved GA needs about 1.84×, 1.26×, and 166.74× shorter time than the exhaustive method in each scenario. In partitioning ResNet110 into three parts, the improved GA can save 712.13× running time compared to applying the exhaustive method and nearly 3× running time compared to applying the basic GA. It can be seen that when the problem size gets larger, the propose GA has better execution efficiency.

6. Conclusions

This paper establishes a dynamic DNN partitioning and deployment system model to represent the actual application requirements of distributed DNN inferencing in an edge environment. On this basis, the problem of optimal deployment-oriented DNN partitioning is modeled as a constrained optimization problem. Considering that the crossover and mutation operators in a basic genetic algorithm may produce many infeasible solutions, it aims to distinguish two types of chromosomes, i.e., partitioning chromosomes and deployment chromosomes. Then, it performs the crossover and mutation operations on partitioning chromosomes to ensure generating reasonable deployment chromosomes and produce new deployment chromosomes based on the updated partitioning population and the select excellent deployment individuals for the next iteration. The experimental results show that the proposed algorithm can not only result in shorter inferencing time and lower device average energy cost, but also needs less time to achieve an optimal deployment.

To further improve this work, a potential future research direction is to try to reduce working on CPU by constructing proper mathematical models. In addition, 3D image-related applications will be considered in the future.

Author Contributions: Conceptualization, J.N. and B.Z.; methodology, J.N., H.Z. and J.L.; validation, J.L.; investigation, J.N. and H.Z.; data curation, H.Z. and J.L.; writing—original draft preparation, J.N.; writing—review and editing, J.N.; supervision, B.Z.; funding acquisition, B.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Key Project of the National Natural Science Foundation of China: U1908212.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets can be obtained from http://www.cs.toronto.edu/~kriz/cifar.html (accessed on 17 November 2021).

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Dec, G.; Stadnicka, D.; Paśko, Ł.; Mądziel, M.; Figliè, R.; Mazzei, D.; Tyrovolas, M.; Stylios, C.; Navarro, J.; Solé-Beteta, X. Role of Academics in Transferring Knowledge and Skills on Artificial Intelligence, Internet of Things and Edge Computing. *Sensors* **2022**, *22*, 2496. [CrossRef] [PubMed]
2. Paśko, Ł.; Mądziel, M.; Stadnicka, D.; Dec, G.; Carreras-Coch, A.; Solé-Beteta, X.; Pappa, L.; Stylios, C.; Mazzei, D.; Atzeni, D. Plan and Develop Advanced Knowledge and Skills for Future Industrial Employees in the Field of Artificial Intelligence, Internet of Things and Edge Computing. *Sustainability* **2022**, *14*, 3312.
3. Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; Zhang, J. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* **2019**, *107*, 1738–1762. [CrossRef]
4. Murshed, M.S.; Murphy, C.; Hou, D.; Khan, N.; Ananthanarayanan, G.; Hussain, F. Machine learning at the network edge: A survey. *Acm Comput. Surv.* **2021**, *54*, 1–37. [CrossRef]
5. Chen, J.; Ran, X. Deep learning with edge computing: A review. *Proc. IEEE* **2019**, *107*, 1655–1674. [CrossRef]
6. Liang, X.; Liu, Y.; Chen, T.; Liu, M.; Yang, Q. Federated transfer reinforcement learning for autonomous driving. *arXiv* **2019**, arXiv:1910.06001.
7. Zhang, Q.; Sun, H.; Wu, X.; Zhong, H. Edge video analytics for public safety: A review. *Proc. IEEE* **2019**, *107*, 1675–1696. [CrossRef]
8. Liang, F.; Yu, W.; Liu, X.; Griffith, D.; Golmie, N. Toward edge-based deep learning in industrial Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 4329–4341. [CrossRef]
9. Qolomany, B.; Al-Fuqaha, A.; Gupta, A.; Benhaddou, D.; Alwajidi, S.; Qadir, J.; Fong, A.C. Leveraging machine learning and big data for smart buildings: A comprehensive survey. *IEEE Access* **2019**, *7*, 90316–90356. [CrossRef]
10. Cheng, Y.; Wang, D.; Zhou, P.; Zhang, T. A survey of model compression and acceleration for deep neural networks. *arXiv* **2017**, arXiv:1710.09282.
11. Deng, L.; Li, G.; Han, S.; Shi, L.; Xie, Y. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* **2020**, *108*, 485–532. [CrossRef]
12. Choudhary, T.; Mishra, V.; Goswami, A.; Sarangapani, J. A comprehensive survey on model compression and acceleration. *Artif. Intell. Rev.* **2020**, *53*, 5113–5155. [CrossRef]
13. 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* **2017**, *45*, 615–629. [CrossRef]
14. 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; IEEE: Piscataway, NJ, USA, 2018; pp. 1–6.
15. Jeong, H.J.; Lee, H.J.; Shin, C.H.; Moon, S.M. IONN: Incremental offloading of neural network computations from mobile devices to edge servers. In Proceedings of the ACM Symposium on Cloud Computing, Carlsbad, CA, USA, 11–13 October 2018; pp. 401–411.
16. Jouhari, M.; Al-Ali, A.; Baccour, E.; Mohamed, A.; Erbad, A.; Guizani, M.; Hamdi, M. Distributed CNN Inference on Resource-Constrained UAVs for Surveillance Systems: Design and Optimization. *IEEE Internet Things J.* **2021**, *9*, 1227–1242. [CrossRef]
17. Tang, E.; Stefanov, T. Low-memory and high-performance CNN inference on distributed systems at the edge. In Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion, Leicester, UK, 6–9 December 2021; pp. 1–8.
18. Zhou, J.; Wang, Y.; Ota, K.; Dong, M. AAIoT: Accelerating artificial intelligence in IoT systems. *IEEE Wirel. Commun. Lett.* **2019**, *8*, 825–828. [CrossRef]

19. Zhou, L.; Wen, H.; Teodorescu, R.; Du, D.H. Distributing deep neural networks with containerized partitions at the edge. In Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19), Renton, WA, USA, 9 July 2019.
20. Zhao, Z.; Barijough, K.M.; Gerstlauer, A. Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 2348–2359. [CrossRef]
21. Li, E.; Zeng, L.; Zhou, Z.; Chen, X. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Trans. Wirel. Commun.* **2019**, *19*, 447–457. [CrossRef]
22. Wang, H.; Cai, G.; Huang, Z.; Dong, F. ADDA: Adaptive distributed DNN inference acceleration in edge computing environment. In Proceedings of the 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), Tianjin, China, 4–6 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 438–445.
23. Gao, M.; Cui, W.; Gao, D.; Shen, R.; Li, J.; Zhou, Y. Deep neural network task partitioning and offloading for mobile edge computing. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–6.
24. Mao, J.; Chen, X.; Nixon, K.W.; Krieger, C.; Chen, Y. Modnn: Local distributed mobile computing system for deep neural network. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1396–1401.
25. Mao, J.; Yang, Z.; Wen, W.; Wu, C.; Song, L.; Nixon, K.W.; Chen, X.; Li, H.; Chen, Y. Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns. In Proceedings of the 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, USA, 13–16 November 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 751–756.
26. Shahhosseini, S.; Albaqsami, A.; Jasemi, M.; Bagherzadeh, N. Partition pruning: Parallelization-aware pruning for deep neural networks. *arXiv* **2019**, arXiv:1901.11391.
27. Kilcioglu, E.; Mirghasemi, H.; Stupia, I.; Vandendorpe, L. An energy-efficient fine-grained deep neural network partitioning scheme for wireless collaborative fog computing. *IEEE Access* **2021**, *9*, 79611–79627. [CrossRef]
28. Hadidi, R.; Cao, J.; Woodward, M.; Ryoo, M.S.; Kim, H. Musical chair: Efficient real-time recognition using collaborative iot devices. *arXiv* **2018**, arXiv:1802.02138.
29. de Oliveira, F.M.C.; Borin, E. Partitioning convolutional neural networks for inference on constrained Internet-of-Things devices. In Proceedings of the 2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Lyon, France, 24–27 September 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 266–273.
30. Mohammed, T.; Joe-Wong, C.; Babbar, R.; Di Francesco, M. Distributed inference acceleration with adaptive DNN partitioning and offloading. In Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; IEEE: Piscataway, NJ, USA,2020; pp. 854–863.
31. He, W.; Guo, S.; Guo, S.; Qiu, X.; Qi, F. Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in IoT. *IEEE Internet Things J.* **2020**, *7*, 9241–9254. [CrossRef]
32. Tang, X.; Chen, X.; Zeng, L.; Yu, S.; Chen, L. Joint multiuser dnn partitioning and computational resource allocation for collaborative edge intelligence. *IEEE Internet Things J.* **2020**, *8*, 9511–9522. [CrossRef]
33. Dong, C.; Hu, S.; Chen, X.; Wen, W. Joint Optimization With DNN Partitioning and Resource Allocation in Mobile Edge Computing. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 3973–3986. [CrossRef]
34. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*. [CrossRef]
35. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
36. 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.
37. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
38. Krizhevsky, A.; Hinton, G.; *Learning Multiple Layers of Features from Tiny Images*; Technical Report, University of Toronto, Toronto, ON, Canada, 2009.
39. Qi, H.; Sparks, E.R.; Talwalkar, A. Paleo: A performance Model for Deep Neural Networks. 2016. Available online: https://openreview.net/pdf?id=SyVVJ85lg (accessed on 12 June 2021).
40. Tian, X.; Zhu, J.; Xu, T.; Li, Y. Mobility-included DNN partition offloading from mobile devices to edge clouds. *Sensors* **2021**, *21*, 229. [CrossRef]