Article

Communication Failure Resilient Distributed Neural Network for Edge Devices

Jonghun Jeong 1, Jong Sung Park 2 and Hoeseok Yang 3,*

1 Department of Artificial Intelligence Convergence Network, Ajou University, 206 Worldcup-ro, Yeongtong-gu, Suwon-si 16499, Korea; kkjh223@ajou.ac.kr
2 Agency for Defense Development, 160, Bugyuseong-daero 488beon-gil, Yuseong-gu, Daejeon 34060, Korea; jsparke61@add.re.kr
3 Department of Electrical and Computer Engineering, Ajou University, 206 Worldcup-ro, Yeongtong-gu, Suwon-si 16499, Korea
* Correspondence: hyang@ajou.ac.kr; Tel.: +82-32-219-2361

Abstract: Recently, the necessity to run high-performance neural networks (NN) is increasing even in resource-constrained embedded systems such as wearable devices. However, due to the high computational and memory requirements of the NN applications, it is typically infeasible to execute them on a single device. Instead, it has been proposed to run a single NN application cooperatively on top of multiple devices, a so-called distributed neural network. In the distributed neural network, workloads of a single big NN application are distributed over multiple tiny devices. While the computation overhead could effectively be alleviated by this approach, the existing distributed NN techniques, such as MoDNN, still suffer from large traffics between the devices and vulnerability to communication failures. In order to get rid of such big communication overheads, a knowledge distillation based distributed NN, called Network of Neural Networks (NoNN), was proposed, which partitions the filters in the final convolutional layer of the original NN into multiple independent subsets and derives smaller NNs out of each subset. However, NoNN also has limitations in that the partitioning result may be unbalanced and it considerably compromises the correlation between filters in the original NN, which may result in an unacceptable accuracy degradation in case of communication failure. In this paper, in order to overcome these issues, we propose to enhance the partitioning strategy of NoNN in two aspects. First, we enhance the redundancy of the filters that are used to derive multiple smaller NNs by means of averaging to increase the immunity of the distributed NN to communication failure. Second, we propose a novel partitioning technique, modified from Eigenvector-based partitioning, to preserve the correlation between filters as much as possible while keeping the consistent number of filters distributed to each device. Throughout extensive experiments with the CIFAR-100 (Canadian Institute For Advanced Research-100) dataset, it has been observed that the proposed approach maintains high inference accuracy (over 70%, 1.53× improvement over the state-of-the-art approach), on average, even when a half of eight devices in a distributed NN fail to deliver their partial inference results.

Keywords: Internet-of-Things device; embedded system; neural network; distributed neural network; neural network optimization

1. Introduction

Recently, resource-constrained computer systems such as Internet-of-Things (IoT) or wearable devices are becoming increasingly popular in the market. Furthermore, with recent advances in AI technology, the demand to run neural network-based (NN-based) applications on resource-constrained systems is increasing [1–3]. However, the resource availability, e.g., memory capacity or computational power, of such devices is usually not rich enough to run recent deep NN-based applications. Typical micro-controller units
(MCUs) that are used to implement such IoT or wearable devices have an operation frequency of 50–500 MHz with 100–500 KB main memory, while the mobile processors have 1–3 GHz with several GB main memory.

While there have been several NN libraries [4] or NN optimization frameworks [5–10] proposed tailored to such MCU-level systems, it is still very challenging to run the state-of-the-art NN applications on top of a single device. Thus, in some approaches, multiple resource-constrained devices are cooperatively used to execute a single NN application, a so-called distributed NN. In this approach, computational and memory overheads of a single big NN application are distributed over multiple tiny devices.

Figure 1 depicts two different distributed NN approaches in comparison. Figure 1a shows the MoDNN approach proposed by Mao et al. [11] where each layer of NN is an unit of distribution. In this case, an intermediate feature map that locates in the boundary of partitioning should be transferred between devices in its entirety, resulting in considerable communication burdens. In order to get rid of such prohibitive communication overheads, Bhardwaj et al. [12] proposed a completely different distribution approach, called Network of Neural Network (NoNN), based on Knowledge Distillation (KD). In their approach, the original NN, referred to as teacher network, is partitioned into multiple independent and smaller NNs, referred to as student networks, by means of KD as illustrated in Figure 1b. In order to determine which parts of the teacher network will be used to extract each student network, the filters in the last convolutional (Conv) layer are grouped into multiple subsets (partitioning). After this, a student network is trained using each subset of the partitioned filters. In inference, each student network is executed independently on a single device and the results are collected only once at the end to infer the final conclusion.

Figure 1. Two different distribution approaches for running a single huge NN applications on multiple devices: (a) MoDNN [11] and (b) Network of Neural Network [12].

While each individual student network can infer its own conclusion separately, these individual inference results before being combined with results from the others would be of lower accuracy. This partial inference can happen any time as long as the distributed NN relies on external communication, thus it is a fundamental limitation of this approach. In fact, many resource-constrained systems such as wearable devices use a low-power short-range communication protocol such as Bluetooth or Zigbee and these are generally considered to be relatively inferior to other conventional communication medium such as Wi-Fi or Ethernet in terms of communication reliability and performance. Moreover, it is expected that wearable devices will communicate with each other though the human body [13] in the near future, making this problem even more serious. i.e., due to its low voltage operation, the human body communication is more vulnerable to the noises that are pervasively injected to the human body [14], thus the communication capability may dictate the performance of the distributed NN.

The partitioning approach used in NoNN may result in unbalanced part sizes and it may also considerably compromise the correlation between filters in the original NN. In case of communication failure, an unacceptable accuracy degradation of NoNN can be observed. In this paper, we improve the partitioning technique of the NoNN approach to make the distributed NN more reliable to communication failures. We enhance the
redundancy of the filters that are used to derive each student network by means of averaging. Then, we also propose a novel partitioning technique, modified from Eigenvector-based partitioning, to preserve the correlation between filters as much as possible while keeping the number of filters for each student network.

The remainder of this paper is organized as follows. In Section 2, we review the basic notion of knowledge distillation, based on which is the working principle of the state-of-the-art distributed NN technique, NoNN, which is presented in Section 3. Then, in Section 4, we illustrate how we enhance the partitioning strategy of NoNN by the proposed methods in two aspects. In Section 5, we verify the resilience of the proposed distributed NN to communication failures through extensive experiments, followed by concluding remarks in Section 6.

2. Knowledge Distillation

Knowledge distillation (KD) is an NN optimization technique that extracts a relatively small NN (student network) out of a large one (teacher network). It has been proven that student networks derived using KD are more accurate than the ones of the same size that have been trained from scratch [15]. Among several known KD methods [16], we use the following two approaches for extracting student networks from a teacher network in realizing distributed NNs: soft label and attention transfer.

2.1. Soft Label

Ordinary NNs are trained based on a training data set, each of whose elements is a tuple \( \langle x, y \rangle \) where \( x \) and \( y \) denote input data and its label, respectively. In case that there are \( n \) classes in the classification problem, a label is a vector of \( n \) binary values, i.e., \( y \in \{0, 1\}^n \) as illustrated in the bottom-right of Figure 2. On the other hand, the inference output of an NN, denoted as \( P \), is a vector of \( n \) non-negative real values, each of which is less than or equal to 1.0, i.e., \( P \in [0, 1.0]^n \). Ordinary NN training methods try to make the inference results \( P \) as close as possible to labels \( y \), thus minimizing the hard label loss \( L(x) \) that is defined as follows:

\[
L(x) = H(y, P),
\]

where \( H \) denotes cross entropy, a relative difference between \( y \) and \( P \).

Figure 2. An overview of knowledge distillation (KD) that uses soft label and attention transfer.

The goal of KD is to effectively transfer useful information from a teacher network to the derived student network. However, the problem of the label is that it is oblivious to the internal information of teacher network. i.e., all classes except for the correct answer are simply nullified as 0 in the label vector \( y \). The NN inference results actually hold more information including the similarity between different input data. For instance, an input image with a cat might have a lot of features in common with another image with a dog. On the contrary, two different images with a cat and a car would not activate many common neurons at the same time. If the learning is performed only considering the label loss in
Equation (1), this similarity between input classes cannot be taken into consideration in the KD procedure.

In order to overcome this drawback, it has been proposed to utilize an internal information of the teacher network, called soft label (SL). In particular, the input vector of the last softmax layers in the teacher network and student network, denoted as \( P^\tau_t \) and \( P^\tau \), respectively, are used to calculate the loss [15] as illustrated in Figure 2. The revised loss value \( L^{SL}(x) \) that reflect this internal information is defined as follows:

\[
L^{SL}(x) = H(P^\tau_t, P^\tau). \tag{2}
\]

Typically, the actual loss is calculated as the weighted sum of Equations (1) and (2):

\[
L^{KD}(x) = (1 - \alpha) \cdot L(x) + \alpha \cdot L^{SL}(x) \tag{3}
\]

where \( 0 < \alpha < 1.0 \) is a fitting coefficient.

2.2. Attention Transfer

Another KD technique that we rely on in this paper is Attention Transfer (AT) [17], which makes use of the output feature map of the final convolution layer (fconv) of the teacher network. In the AT approach, the loss function is defined as follows

\[
L^{AT}(x) = \left\| \frac{P^F(x)}{\|P^F(x)\|_2} - \frac{P^F_t(x)}{\|P^F_t(x)\|_2} \right\|_2 \tag{4}
\]

where \( P^F \) and \( P^F_t \) are the output feature maps of fconv in the student network and the teacher network, respectively. The rationale is to derive a student network that mimics the teacher network in the output feature map values of fconv. Note that the size of \( F \) and \( F_t \) should be identical to apply AT. Therefore, as shown in Figure 2, resizing should be performed at the end of student networks to make \( F \) the same size as \( F_t \). The AT approach has the advantage over SL in the sense that more internal information of the teacher network is transferred to the student network.

3. Network of Neural Networks (NoNN)

3.1. Overview of NoNN

Bhardwaj et al. [12] proposed a distributed NN technique, called Network of Neural Networks (NoNN), in which multiple independent student networks are derived out of a single teacher network by means of KD. Each of the derived student networks is deployed on top of a resource-constrained embedded system, such as a wearable device, and they perform their own inference independently of each other. The outputs of the student networks are only merged at the end to conclude the final inference decision. In this way, the heavy communication overheads caused by other distributed NN approaches [11] can be mitigated.

Figure 3 illustrates how multiple students are derived from a single teacher network using soft label and attention transfer in NoNN. Note that the size of a student network is limited by the memory/storage capacity of each device, thus the number of filters in a student network is much smaller than that of the teacher network. However, in order to apply the AT technique for deriving student networks, the numbers of filters in fconv should be identical. Therefore, the filters in fconv of the teacher network should be partitioned into a number of subsets, each of which has the same number of filters as the student network’s fconv. This procedure is referred to as partitioning and we call each partitioned subset of filters a part as annotated in Figure 3.
Once the partitioning decision is made, the training of NoNN (the derivation of student networks) should be performed in a way that each student network behaves similarly to its corresponding part. This goal can be achieved by means of AT. In addition, the final inference (merged output of all student networks) should be as similar as possible to the label. Therefore, in NoNN, all (hard) label loss, soft label loss, and AT loss, expressed in Equations (1), (2) and (4), respectively, should be considered in a weighted sum; i.e., the loss of NoNN, \( L_{\text{NoNN}} \), is defined as follows:

\[
L_{\text{NoNN}}(x) = L_{\text{KD}}(x) + \beta \sum_{i=1}^{s} L_{i}^{\text{AT}}(x),
\]

where \( 0 < \beta < \infty \) and \( s \) are a fitting coefficient and the number of student networks (or the number of parts), respectively. \( L_{i}^{\text{AT}}(x) \) denotes the AT loss in the case that \( f_{\text{conv}} \) of the teacher network in Equation (4) is replaced with the \( i \)th part of the teacher network.

### 3.2. Partitioning Strategy of NoNN

As stated earlier, the filters in \( f_{\text{conv}} \) of the teacher network should be regrouped into a number of exclusive subsets by the partitioning procedure of NoNN. In doing so, the inter-relationship among the individual filters should be taken into consideration. Therefore, in NoNN, an undirected graph, in which each vertex denotes an individual filter in \( f_{\text{conv}} \), is derived to express how strongly each filter is correlated with others. The correlation between a pair of filters is quantified and annotated as a weight on each edge of the graph. In the NoNN approach, the filters that react simultaneously are encouraged to be evenly distributed over multiple student networks as they intend to keep the knowledge distributed as evenly as possible to all students. Based on this idea, they formulate the Activation Hub (AH) rule that defines the weight between vertices \( i \) and \( j \) as

\[
AH(i,j) = \sum a_i a_j |a_i - a_j|, \tag{6}
\]

where \( a_i \) denotes the mean value of the weights in the \( i \)th filter in \( f_{\text{conv}} \). As can be estimated by Equation (6), if two different filters show big mean values for the same input, the degree of correlation between those two would be small. On the other hand, if two filters have big differences in the mean weight values, they are considered to have a strong correlation (a big \( AH \) value) and are more likely to be partitioned into the same subset.

If there exist \( N \) filters in \( f_{\text{conv}} \), using Equation (6), we can derive a symmetric \( N \times N \) adjacency matrix \( A \) of the graph, i.e., \( A_{i,j} = AH(i,j) = AH(j,i) \). Then, NoNN partitions the vertices into at most \( N \) subsets, i.e., the partitioning of filter \( i \) can be represented by...
$g_i \in \{0, 1, 2, \ldots, N - 1\}$. The partitioning decision is made by choosing $g_i$ values for all $i$ that maximize the following equation:

$$Corr(A) = \sum_{i,j<N} A_{ij} - \frac{1}{\gamma} \frac{k_i k_j}{2m} \delta(g_i, g_j),$$

where $m$, $k_i$, and $\delta$ denote the number of edges in the graph, the number of all edges that are associated with the $i$th filter, and Kronecker delta, respectively. For Kronecker delta, we have $\delta(g_i, g_j) = 1$ if filters $i$ and $j$ are partitioned into the same part and $\delta(g_i, g_j) = 0$ otherwise.

In Equation (7), $0 < \gamma < \infty$ is a user-defined tuneable parameter that determines the size of excluded filters in the partitioning procedure. Before optimizing the partitioning based on Equation (7), any nodes $i$ or $j$ that satisfy the following condition are marked as useless and excluded in partitioning:

$$A_{ij} - \frac{1}{\gamma} \frac{k_i k_j}{2m} < 0.$$

It is worthwhile to mention that the choice of $\gamma$ is tightly associated with the size of this useless filter group that is removed before partitioning. The smaller the $\gamma$ value is, the more filters are marked as useless. In the same regard, the importance of the correlations between filters, i.e., $A_{ij}$, also gets smaller as $\gamma$ gets smaller. Therefore, as exemplified in Figure 4 (a partitioning example of Wide Residual Network (WRN) family with depth = 40 and $k = 4$ (WRN40-4) [18]), a smaller $\gamma$ results in a large useless filter group and useful filters are more likely to be separated in a larger number of groups. On the contrary, when $\gamma$ is relatively large, we have a smaller useless filter group and useful filters tend to be partitioned into a fewer number of groups. It has been reported that the accuracy drop caused by ignoring these useless filters in NoNN was negligible [12].

Figure 4. Three different partitioning results based on the AH rule with different $\gamma$ values for the WRN40-4 example: (a) $\gamma = 0.3$; (b) $\gamma = 0.7$; (c) $\gamma = 1.3$.

One of the drawbacks that NoNN has in partitioning is that the sizes of partitioned groups, i.e., the number of filters assigned in each group, may be asymmetric, as can be observed in Figure 4. Such an unbalanced partitioning may result in an unacceptable accuracy degradation when the result derived from a big filter group is missing due to a communication failure. In order to mitigate the adverse effect of such unbalanced partitioning cases, NoNN merges small groups obtained from the initial partitioning to make the sizes of parts as even as possible as shown in Figure 5. While this post-merge procedure mitigates the unbalance issue to some extent, there could still exist an imbalance between parts in principle, which makes the derived distributed NNs vulnerable to communication failures. Figure 5 illustrates an example of creating four student networks in NoNN when $\gamma$ is chosen to be 0.7. In this example, 197 filters out of 256 in fconv of the teacher network are marked as useless and the remainder are partitioned into five groups initially. Then, the two smallest groups (Groups 3 and 4) are merged into one, resulting in four parts in total at the end.
4. Proposed Distributed Neural Network Partitioning Methods

In this section, we propose to enhance the partitioning strategy of NoNN, in two aspects, to make it more resilient to communication failures.

4.1. Averaging

Although it has been argued that NoNN itself is resilient to communication failures, it is only so to limited information loss. It was observed in [12] that the inference accuracy starts decreasing significantly as the number of unavailable student networks gets more than three devices. This is attributed to be the monolithic one-on-one relationship between the parts and student networks in NoNN. That is, an important part is more likely to be missing if it is reflected to only one student network. As a remedy to this problem, we propose to use parts in a more redundant way when training student networks. Therefore, in the averaging technique, we keep the number of parts ($M$) less than or equal to half of the number of student networks ($S$) to be derived, i.e., $M = \lfloor \frac{S}{2} \rfloor$. Then, the $i$th student network (among $S$) is trained by the $j$th part (among $M$) where $i \mod M = j$.

In the example illustrated in Figure 6, part 0 is used to train student networks 0 and 2, while part 1 is used for others. In this way, each part can be used to derive at least two student networks.

Figure 5. A partitioning example in NoNN ($\gamma = 0.7$) for four student networks.

Figure 6. A partitioning example with averaging.
If a student network is trained from a part with $S$ filters, its inference result is represented as a $(1 \times 1 \times S)$ vector. When merging the partial results to conclude the final inference result, we collect the results from the student networks that are trained from the same part and output the average of them as illustrated in Figure 6. In this example, the results from student networks 0 and 2 are combined by averaging into a $(1 \times 1 \times 30)$ vector and another $(1 \times 1 \times 29)$ vector is obtained in the same way from student networks 1 and 3. At last, the final inference result is represented as a concatenation of them, i.e., a $(1 \times 1 \times 59)$ vector.

4.2. Eigenvector-Based Partitioning

Note in Equation (7) that the correlation between filters in $f_{conv}$ is significantly compromised by $\gamma$ in NoNN. A similar partitioning problem exists in the analysis of social network and, in that domain, Newman [19] proposed a partitioning approach that groups the vertices with strong correlation into the same group. They partition a given graph into two exclusive subsets and, in order to quantitatively evaluate the partitioning, a quality factor $Q$ is defined as follows for a graph with $N$ vertices:

$$Q = \frac{1}{4m} \sum_{i,j < N} (A_{ij} - \frac{k_{ij}}{2m})(s_i s_j + 1)$$

where $s \in [-1, 1]^N$ denotes the binary partitioning decision for all $N$ vertices in the graph. Thus, if the $i$th and $j$th vertices are partitioned into the same group, we have $s_i s_j = 1$, while $s_i s_j$ becomes $-1$ if the two vertices are separated. Then, the quality factor $Q$ can be rewritten as follows with $B$ being an $N \times N$ matrix ($B_{ij} = A_{ij} - \frac{k_{ij}}{2m}$), based on Eigenvector decomposition (Eigenvector of a $z \times z$ square matrix $Y$ is defined as a $z \times 1$ vector $x$ that satisfies $Y \cdot x = \lambda x$, where $\lambda$ is a nonzero scalar value, called Eigenvalue). There can exist at most $z$ pairs of Eigenvector $x$ and Eigenvalue $\lambda$ for a $z \times z$ matrix:

$$Q = \frac{1}{4m} x^T \cdot B \cdot s.$$

It has been shown in [19] that maximizing the quality factor $Q$ in Equation (9) is equivalent to finding the largest eigenvalue of $B$ in Equation (10).

In the proposed partitioning approach, we borrow the basic Eigenvector-based partitioning presented in [19]. However, this Eigenvector-based decomposition has a couple of limitations to be applied as it is to the target partitioning problem of distributed NNs. First, it partitions the vertices into only two subsets. In distributed NNs, the partitioning should be performed generically to any arbitrary number of devices. Second, the size of the two partitioned subsets may differ significantly. In this Eigenvector-based decomposition, only the connectivity between the vertices is taken into consideration in maximizing $Q$, which can result in considerably unbalanced sizes between the partitioned subsets. In order to tackle these issues, we propose to apply the basic partitioning algorithm recursively and adaptively, as will be discussed in what follows.

Algorithm 1 delineates how we revise the basic Eigenvector-based partitioning that partitions a given filter group into two. The modified partitioning, procedure $E$-PARTITION, takes three arguments as input: a set of filters to be partitioned ($Group$), an adjacency matrix ($Adj$), and the desired minimum number of filters for a single part ($psize$). As the Eigenvector-based partitioning divides the original group into two subgroups, a partitioning result is characterized by the followings: two divided sets of filters ($subGroup[2]$), corresponding adjacency matrices for them ($subAdj[2]$), and the quality factor of the partitioning (Eigenvalue) $eval$. Therefore, a return value of $E$-PARTITION is a tuple of ($subGroup[2], subAdj[2], eval$). The working flow of the proposed algorithm is as follows.
Algorithm 1 Proposed modified Eigenvector-based partitioning.

1: Group: a set of $N$ filters to be partitioned (input argument)
2: Adj: $N \times N$ adjacency matrix (input argument)
3: psize: desired number of vertices in a part (input argument)
4: subGroup[2]: two partitioned filter subsets (return value)
5: subAdj[2]: two adjacency matrices for subGroup (return value)
6: eval: quality factor of the resultant partitioning, i.e., Eigenvalue (return value)

7: 

8: procedure E-PARTITION(Group, Adj, psize)
9: subGroup[0], subGroup[1] $\leftarrow$ $\phi$; $\triangleright$ Initialization of return value
10: subAdj[0], subAdj[1] $\leftarrow$ NULL;
11: eval $\leftarrow -\infty$;
12: 
13: if $N \leq 2 \cdot $ psize then $\triangleright$ If too small to be partitioned into two subgroups
14: return $\langle$subGroup, subAdj, eval$\rangle$; $\triangleright$ Not valid
15: end if
16: 
17: $\text{EigenList} \leftarrow \text{EigenSearch}(\text{Adj});$ $\triangleright$ Ordered list of Eigenvectors/values
18: for $i$ in 0 to $|\text{EigenList}| - 1$ do $\triangleright$ For all possible Eigenvectors
19: partition[0], partition[1] $\leftarrow$ $\phi$;
20: currEvec $\leftarrow$ EigenList.$i$.Evec;
21: currEval $\leftarrow$ EigenList.$i$.Eval;
22: $M$ $\leftarrow$ number of rows in currEvec; $\triangleright$ currEvec is an $M \times 1$ matrix
23: for $j$ in 0 to $M - 1$ do
24: if currEvec.$j$ > 0 then
25: partition[0] $\leftarrow$ partition[0] $\cup \{j\}$;
26: else
27: partition[1] $\leftarrow$ partition[1] $\cup \{j\}$;
28: end if
29: end for
30: if min($|\text{partition[0]}|, |\text{partition[1]}|$) $\geq$ psize then $\triangleright$ Check if valid
31: eval $\leftarrow$ currEval;
32: break;
33: end if
34: end for
35: 
36: for $i$ in 0 to 1 do $\triangleright$ For two partitioned subsets,
37: for $j$ in 0 to $|\text{partition[i]}| - 1$ do $\triangleright$ For two partitioned subsets,
38: $jj$ $\leftarrow$ the $j$th element of partition[$i$];
39: subGroup[$i$] $\leftarrow$ the $jj$th element of Group; $\triangleright$ Calculate subGroup[$i$]
40: for $k$ in 0 to $|\text{partition[i]}| - 1$ do
41: $kk$ $\leftarrow$ the $k$th element of partition[$i$];
42: subAdj[$i$].$kk$ $\leftarrow$ Adj.$jj$.$kk$; $\triangleright$ Calculate subAdj[$i$]
43: end for
44: end for
45: end for
46: 
47: return $\langle$subGroup, subAdj, eval$\rangle$; $\triangleright$ Return the best possible partitioning result
48: end procedure

After initializing the return values (lines 9–11 of Algorithm 1), it first checks the input filter group is big enough to be partitioned into two with respect to the desired size, i.e., the number of filters in a partitioned group (lines 13–15 of Algorithm 1). If the size of input set, Group, is less than $2 \cdot $ psize, the procedure immediately returns and no partitioning is performed. Otherwise, all possible Eigenvector and Eigenvalue pairs are calculated and stored in EigenList in a descending order of their Eigenvalues by
invoking EigenSearch (line 17 of Algorithm 1). For example, after line 17 of Algorithm 1, EigenList[0].Eval and EigenList[0].Evec are the the biggest possible Eigenvalue and its corresponding Eigenvector for the $N \times N$ matrix Adj. Then, in lines 18–34 of Algorithm 1, it examines all possible Eigenvector candidates in a descending order of their Eigenvalues to see if the corresponding partitioning from the current Eigenvector satisfies the size requirement enforced by $psize$. To be more specific, given a candidate Eigenvector, it tries to partition Group into two subsets (lines 23–29 of Algorithm 1). If both of the partitioned subsets have at least $psize$ filters, the partitioning is deemed as valid and it stops searching the remaining candidates (lines 30–33 of Algorithm 1). Otherwise, it continues to search for a valid partitioning from the next available Eigenvector/Eigenvalue pair in EigenList. After finding a valid partitioning, it is necessary to build new partitioned subgroups and their corresponding adjacency matrices; these are illustrated in lines 36–45 of Algorithm 1. At last, the procedure returns the found partitioning, characterized by subGroup and subAdj, as well as the quality factor of the resultant partitioning, eval.

Note that E-PARTITION shown in Algorithm 1 only partitions the given group into two subsets. In order to partition a given filter group into an arbitrary number of subgroups, we apply E-PARTITION in an adaptive and iterative way to obtain $pnumber$ parts, each of which consists of $psize$ filters, as described in Algorithm 2. At the beginning, a sorted list (SortedList), that maintains partitioning candidates in a descending order of eval, is initialized with $f_{conv}$ of the teacher network (lines 1–3 of Algorithm 2). Until the number of partitions reaches the desired number, $pnumber$, it performs the followings iteratively: (1) it takes a candidate partitioning with the biggest eval from SortedList (line 5 of Algorithm 2), (2) and performs the partitioning if it is a valid partitioning (lines 6–8 of Algorithm 2); (3) two newly created groups are inserted back to SortedList to be considered for further partitioning in the successive iterations (lines 9–10 of Algorithm 2); (4) then, it increases the number of parts by one as one part is removed and two new ones are created (lines 11 of Algorithm 2). After line 12 of Algorithm 2, we have $pnumber$ subsets of filters in SortedList, each of which includes at least $psize$ filters. Thus, in lines 14–22 of Algorithm 2, we remove less important filters in each part to make the sizes of all parts consistent. In doing so, the filters assigned to each group is sorted in a descending order of average activation value (line 16 of Algorithm 2) and only first $psize$ filters remain in the part (lines 18–20 of Algorithm 2). Finally, we obtain $pnumber$ parts stored in partList and each part has exactly $psize$ filters in it.

Figure 7 shows how the proposed distributed NN partitioning technique works for the same partitioning example with four student networks illustrated in Figure 5. Unlike NoNN, we are able to adjust the desired part’s size in the proposed distributed NN partitioning method; in this example each part is expected to have 30 filters, i.e., $psize = 30$. Firstly, the adjacency matrix Adj of 256 filters in $f_{conv}$ is derived by the AH rule, Equation (6). By invoking E-PARTITION with $f_{conv}$, Adj, and $psize = 30$, $f_{conv}$ is firstly partitioned into two subgroups GroupA and GroupB as shown in the figure. In the two following iterations of the while loop in Algorithm 2, they are partitioned into four groups: Group0 with 50 filters, Group1 with 60 filters, Group2 with 79 filters, and Group3 with 67 filters. As each part is expected to have only $psize = 30$ filters, total 136 filters are eliminated by the last for loop of Algorithm 2.
Algorithm 2 Proposed NN Partitioning using E-PARTITION.

1: SortedList ← φ;                        ▷ Initial partitioning
2: SortedList.push(E-PARTITION(f conv, Adj, psize));
3: numParts ← 1;
4: while numParts < pnumber do          ▷ Until the number of parts becomes pnumber
5:   curr ← SortedList.pop();           ▷ take a candidate partition with the biggest eval
6:   if curr.eval == −∞ then            ▷ No available partitioning
7:     break;
8: end if
9: SortedList.push(E-PARTITION(curr.subGroup[0], curr.subAdj[0], psize));
10: SortedList.push(E-PARTITION(curr.subGroup[1], curr.subAdj[1], psize));
11: numParts ← numParts + 1;            ▷ One part eliminated; two new ones created
12: end while
13: ▷ Now, SortedList has pnumber elements
14: partList ← φ;
15: for ∀ e ∈ SortedList do              ▷ Prune out useless filters
16:   FilterList ← (e.subGroup[0] ∪ e.subGroup[1]) sorted by average activation value;
17:   newPart ← φ;
18:   for i in 0 to psize − 1 do         ▷ newPart = newPart ∪ {FilterList.pop()};
19:     newPart ← newPart ∪ {FilterList.pop()};
20: end for
21: partList.push(newPart);
22: end for

Figure 7. A partitioning example with the proposed Eigenvector-based partitioning technique.

5. Experiments and Discussion

In this section, we verify the resilience of the proposed distributed NN to communication failure through experiments. As a benchmark, we used an image classification application which was trained with the CIFAR-10 (Canadian Institute For Advanced Research-10) and CIFAR-100 (Canadian Institute For Advanced Research-100) data sets [20]. As we use NoNN as a comparison target, we adopted the same teacher and student network structures that were used in [12] as summarized in Tables 1 and 2. While the NoNN technique is not limited to a certain number of parts in principle, they used only two parts, throughout all their evaluations, even for the cases with more than two student networks. Then, each of these two parts is used to train multiple student networks. To make comparisons, we implemented the NoNN technique in the same way, i.e., the number of parts in NoNN is
always two. To emulate the communication failures, we zeroed out the inference results of the devices that were considered to experience communication failures.

Table 1. Basic teacher network models used for experiments.

| Dataset   | Network   | # of Parameters | Accuracy   |
|-----------|-----------|-----------------|------------|
| CIFAR-10  | WRN40-4   | 8.9 M           | 95.82%     |
| CIFAR-100 | WRN28-10  | 36.5 M          | 80.88%     |

Table 2. Basic student network models used for experiments.

| Dataset   | Network    | Teacher       | # of Parameters (Each Student) |
|-----------|------------|---------------|-------------------------------|
| CIFAR-10  | WRN-based  | WRN40-4       | 0.43 M                        |
| CIFAR-100 | WRN-based  | WRN28-10      | 0.82 M                        |

5.1. Effects of Averaging and Eigenvector-Based Partitioning

In this set of experiments, we observe how the averaging technique in partitioning affects the distributed NNs in terms of resiliency to communication failures. As a baseline comparison target, we generated 8 students using NoNN for both CIFAR-10 and CIFAR-100. We performed inference evaluations with the test images of CIFAR-10 and CIFAR-100 for all possible communication failure cases and, for each number of failed devices, we recorded the maximum, minimum, and average accuracy.

As shown in Figure 8a, the basic NoNN technique was already resilient to communication failures by nature, thanks to the duplicated parts for multiple devices (while the NoNN technique in principle is not restricted to partitioning into two parts all the evaluations made in [12] used just two parts even if there are more than two devices, as stated at the beginning of this section. To make fair comparisons, we followed this when we implemented NoNN and this is the reason why NoNN in this experiment has some duplicated parts per device). However, the minimum inference accuracy started decreasing sharply as the number of failed devices became more than half. On the other hand, as can be seen in Figure 8c, this accuracy drop could be improved by the proposed averaging technique; the average inference accuracy could be kept over 70% even if only one device is alive. Considering all possible cases, the accuracy improvement from the averaging technique for CIFAR-10 was 15.072% on average. This improvement was also very apparent with CIFAR-100; the accuracy degradation over the number of lost devices was very steep as can be observed in Figure 8b. When three devices are failed for CIFAR-100, the inference drop was from 77.22% to 54.966% while the accuracy was kept over 70% with the averaging technique as shown in Figure 8d. The average accuracy improvement from the averaging technique for CIFAR-100 was 15.072%.

In order to verify the effectiveness of the proposed Eigenvector-based partitioning, we performed the same set of experiments using the proposed Eigenvector-based partitioning without and with averaging and the results are reported in Figure 8e,f, respectively. The partitioning parameters were set to ps\text{size} = 30 and p\text{number} = 2. (While the p\text{number} and ps\text{size} can be customized as desired in the proposed method, we used the same parameters as NoNN for fair comparison. Note that, for ps\text{size} = 30, NoNN may result in parts with more than 30 filters while the number of filters is always 30 in the proposed method) For both CIFAR-10 and CIFAR-100, it has been observed that the proposed partitioning approach (Eigenvector-based partitioning with averaging) outperformed the basic NoNN and NoNN with averaging in all cases. As can be observed in Figure 8f, the proposed approach showed that the distributed inference results for the hard problem (CIFAR-100) were maintained over 70%, on average, even when a half of the devices were failed. It is a significant improvement, compared with the basic NoNN approach in Figure 8b, 46.908% on average.
5.2. Effects of Different Part Numbers and Sizes

One of the merits that the proposed technique has over the basic NoNN partitioning approach is that the size and number of parts can be customized as desired. The size of each part is an important design parameter as it is directly related to the communication overhead of the distributed NN. For instance, within a low-bandwidth communication environment, a large part size may result in a prohibitively big inference latency. Similarly, the number of parts may also affect the behavior of inferences. The selected part size and number should be carefully optimized with respect to the given communication environment.

In order to see how the number of parts and their sizes affect the inference accuracy, we performed another set of experiments with the same configuration, but varying $p_{size}$ and $p_{number}$ in Algorithm 2: $p_{number} = 1, 2, \text{ or } 4$ and $p_{size} = 5, 10, 15, \text{ or } 30$. Based on the proposed partitioning technique, we could successfully derive multiple distributed NNs with different $p_{number}$ and $p_{size}$ values and the results are reported in Figure 9.

Figure 8. Inference accuracy of distributed NNs with and without the averaging technique (X-axis: number of failed devices, Y-axis: accuracy): (a) NoNN w/o averaging (CIFAR-10), (b) NoNN w/o averaging (CIFAR-100), (c) NoNN w/ averaging (CIFAR-10), (d) NoNN w/ averaging (CIFAR-100), (e) Eigenvector-based partitioning w/ averaging (proposed, CIFAR-10), and (f) Eigenvector-based partitioning w/ averaging (proposed, CIFAR-100).

Figure 9. Inference accuracy for different configurations with $p_{number}$ and $p_{size}$; (a) CIFAR-10; (b) CIFAR-100.
Given the fixed structure and size of student networks, the accuracy differences between \( p_{\text{number}} = 1 \) and \( p_{\text{number}} = 4 \) tend to be larger when the part size \( (p_{\text{size}}) \) is small. This implies that, sometimes, small parts may not capture the behavior of the teacher network and this can be complemented by having multiple parts. Thus, if the distributed NNs are to be deployed in a communication environment with low bandwidth, i.e., enforcing a small part size, it is desirable to have a bigger number of parts during partitioning.

On the other hand, the accuracy differences between different \( p_{\text{number}} \) values tend to diminish as the part size grows. That is, if the communication performance is good enough allowing each student network having a big part size, it would be desirable to have a small number of parts in favor of resiliency to communication failures (redundant usage of a part for multiple devices).

### 5.3. Effects of Teacher Network Size

Whereas the size of student networks is restricted by the memory or storage budget of the wearable or embedded devices that the student networks are deployed on, teacher network does not have such a constraint. Though it can be generally said that a larger teacher network would have higher probability to have a higher accuracy, it is not certain if this tendency still holds true for the distributed NN. Therefore, in order to study the effects of the size of teacher networks, we compared three different teacher networks, summarized in Table 3, for deriving distributed NN with the same student network configuration. We kept all the partitionin parameters the same as previous experiments.

#### Table 3. Teacher network models used for comparison.

| Dataset  | Network  | # of Parameters | Accuracy  |
|----------|----------|-----------------|-----------|
| CIFAR-10 | WRN40-2  | 2.2 M           | 95.17%    |
|          | WRN40-4  | 8.9 M           | 95.82%    |
|          | WRN22-8  | 17.2 M          | 96.28%    |
| CIFAR-100| WRN40-4  | 8.9 M           | 78.65%    |
|          | WRN22-8  | 17.2 M          | 80.52%    |
|          | WRN28-10 | 36.5 M          | 80.88%    |

Tables 4 and 5 show the inference results of the distributed NNs derived from different teacher network models for CIFAR-10 and CIFAR-100, respectively. In contrast to the inference results of teacher networks (Table 3), a larger teacher networks did not always result in better inference accuracy in distributed NNs as shown in Tables 4 and 5. In both, the highest accuracy was obtained in the second largest teacher network. This suggests that the quality of distributed NN is not only dependent upon the accuracy of the original teacher network, but also, and possibly more, on how the knowledge distillation is performed from the teacher network to students. Given a limited and fixed size of student networks, a large teacher networks may make the knowledge distillation even more challenging. Thus, given the complexity of the problem and the size of student networks, care must be taken for the choice of teacher network.

#### Table 4. Distributed inference results from different teacher network models on CIFAR-10.

| Teacher  | # of Parameters (Each Student) | Accuracy  |
|----------|--------------------------------|-----------|
| WRN40-2  | 0.43 M                         | 95.50%    |
| WRN40-4  | 0.43 M                         | 95.52%    |
| WRN22-8  | 0.43 M                         | 95.01%    |
Table 5. Distributed inference results from different teacher network models on CIFAR-100.

| Teacher    | # of Parameters (Each Student) | Accuracy  |
|------------|-------------------------------|-----------|
| WRN40-4    | 0.82 M                        | 78.70%    |
| WRN22-8    | 0.82 M                        | 78.81%    |
| WRN28-10   | 0.82 M                        | 78.55%    |

5.4. Effects of the Number of Student Networks

The last set of experiments was performed to confirm if having more distributed devices actually enhances the inference accuracy. For that, we trained a general standalone convolutional NN with the same size and structure as a student network from scratch as a baseline (first rows of Tables 6 and 7). Then, we also applied the proposed distributed NN technique for 2, 4, and 8 student networks (the number of student networks is equal to the number of distributed devices). For the partitioning parameters, we kept the old values used for the previous experiments.

Table 6. An experiment comparing the results of different number of students on CIFAR-10.

| Teacher    | # of Students | # of Parameters (Each Student) | Accuracy  |
|------------|---------------|-------------------------------|-----------|
| -          | 1             | 0.43 M                        | 94.28%    |
| WRN40-4    | 2             | 0.43 M                        | 94.78%    |
| WRN40-4    | 4             | 0.43 M                        | 95.23%    |
| WRN40-4    | 8             | 0.43 M                        | 95.52%    |

Table 7. An experiment comparing the results of different number of students on CIFAR-100.

| Teacher    | # of Students | # of Parameters (Each Student) | Accuracy  |
|------------|---------------|-------------------------------|-----------|
| -          | 1             | 0.82 M                        | 74.09%    |
| WRN28-10   | 2             | 0.82 M                        | 75.52%    |
| WRN28-10   | 4             | 0.82 M                        | 77.45%    |
| WRN28-10   | 8             | 0.82 M                        | 78.55%    |

As summarized in Tables 6 and 7, the inference results from distributed NNs always outperformed those from standalone NNs for the both problems. Furthermore, it has been consistently observed that having more student networks improves the inference accuracy. The accuracy improvement was more significant in CIFAR-100 as the complexity of inferences is bigger, i.e., more room for improvement by distributed NNs.

6. Conclusions

In this paper, we identified limitations of the state-of-the-art distributed NN, called NoNN: their partitioning technique may result in unbalanced filters for the distributed devices and the correlation between filters cannot be considered enough during partitioning. Such limitations, in turn, often result in an unacceptable accuracy degradation when results from some devices are missing due to communication failures. In order to overcome these drawbacks, we propose to enhance the partitioning strategy of NoNN, in two aspects. First, we proposed the averaging technique to improve the redundancy of the filters used to train different student networks. Second, we also proposed a novel partitioning technique, modified from Eigenvector-based partitioning, to preserve the correlation between filters as much as possible while keeping the number of filters distributed to each device consistent. We verified the effectiveness of the proposed distributed NN on communication failures with extensive experiments with an image classification application, i.e., it has been shown that the proposed partitioning approach improved the distributed inference results significantly in case of communication failures; for CIFAR-100; the accuracy was maintained over 70% (1.53× improvement over NoNN), on average, even when half of eight devices
failed. In the future, we plan to work on network topology, packet routing, and traffic control for distributed NN, i.e., at the transport or network layer of the Open Systems Interconnection (OSI) model as we believe that the communication between multiple edge or wearable devices will become the bottleneck of the inference. Furthermore, it remains as a future work to improve the partitioning considering class-specific accuracy measures.

**Author Contributions:** Conceptualization, J.J. and H.Y.; methodology, J.J. and H.Y.; software, J.J.; validation, J.J.; resources, J.S.P.; writing—original draft preparation, J.J. and H.Y.; writing—review and editing, J.J. and H.Y.; visualization, J.J.; supervision, J.S.P.; project administration, J.S.P.; funding acquisition, J.S.P. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work has been supported by the Future Combat System Network Technology Research Center program of Defense Acquisition Program Administration and Agency for Defense Development (UD190033ED).

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

**Abbreviations**
The following abbreviations are used in this manuscript:

- IoT: Internet of Things
- AI: Artificial Intelligence
- NN: Neural Network
- MCU: MicroController Unit
- NoNN: Network of Neural Network
- KD: Knowledge Distillation
- Conv: Convolution
- fconv: Final Convolution layer
- SL: Soft Label
- AT: Attention Transfer
- AH: Activation Hub
- WRN: Wide Residual Network
- CIFAR: Canadian Institute For Advanced Research
- OSI: Open Systems Interconnection

**References**
1. Yuan, J.; Yu, S. Privacy preserving back-propagation neural network learning made practical with cloud computing. *IEEE Trans. Parallel Distrib. Syst.* 2013, 25, 212–221. [CrossRef]
2. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* 2016, 3, 637–646. [CrossRef]
3. Strubell, E.; Ganesh, A.; McCallum, A. Energy and policy considerations for deep learning in NLP. *arXiv* 2019, arXiv:1906.02243.
4. Lai, L.; Suda, N.; Chandra, V. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv* 2018, arXiv:1801.06601.
5. Lin, J.; Chen, W.M.; Lin, Y.; Cohn, J.; Gan, C.; Han, S. McuNet: Tiny deep learning on iot devices. *arXiv* 2020, arXiv:2007.10319.
6. Banbury, C.; Zhou, C.; Fedorov, I.; Navarro, R.M.; Thakkar, U.; Gope, D.; Reddi, V.J.; Mattina, M.; Whatmough, P.N. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. *arXiv* 2020, arXiv:2010.11267.
7. Liberis, E.; Dudziak, L.; Lane, N.D. µNAS: Constrained Neural Architecture Search for Microcontrollers. In Proceedings of the 1st Workshop on Machine Learning and Systems, Scotland, UK, 26 April 2021; pp. 70–79. [CrossRef]
8. Zhao, Z.; Barijough, K.M.; Gerstlauer, A. Deepphings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 2018, 37, 2348–2359. [CrossRef]
9. Hegde, G.; Ramasamy, N.; Kapre, N. CaffePresso: An optimized library for deep learning on embedded accelerator-based platforms. In Proceedings of the 2016 International Conference on Compilers, Architectures, and Synthesis of Embedded Systems (CASES), Pittsburgh, PA, USA, 2–7 October 2016; pp. 1–10.
10. Zhang, Y.; Suda, N.; Lai, L.; Chandra, V. Hello edge: Keyword spotting on microcontrollers. *arXiv* 2017, arXiv:1711.07128.
11. 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; pp. 1396–1401.
12. Bhardwaj, K.; Lin, C.Y.; Sartor, A.; Marculescu, R. Memory-and communication-aware model compression for distributed deep learning inference on iot. *ACM Trans. Embed. Comput. Syst. (TECS)* **2019**, *18*, 1–22. [CrossRef]

13. Kim, S.; Ko, J. IB-MAC: Transmission latency-aware MAC for electro-magnetic intra-body communications. *Sensors* **2019**, *19*, 341. [CrossRef]

14. Sakuma, J.; Anzai, D.; Wang, J. Performance of human body communication-based wearable ECG with capacitive coupling electrodes. *Healthc. Technol. Lett.* **2016**, *3*, 222–225. [CrossRef]

15. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* 2015, arXiv:1503.02531.

16. Gou, J.; Yu, B.; Maybank, S.J.; Tao, D. Knowledge distillation: A survey. *arXiv* 2020, arXiv:2006.05525.

17. Komodakis, N.; Zagoruyko, S. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *Proceedings of the ICLR 2017: 5th International Conference on Learning Representations*, Paris, France, 24–26 April 2017.

18. Zagoruyko, S.; Komodakis, N. Wide residual networks. *arXiv* 2016, arXiv:1605.07146.

19. Newman, M.E. Modularity and community structure in networks. *Proc. Natl. Acad. Sci. USA* **2006**, *103*, 8577–8582. [CrossRef] [PubMed]

20. Krizhevsky, A.; Hinton, G. Learning Multiple Layers of Features from Tiny Images. Available online: [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf) (accessed on 6 July 2021).