Variant Parallelism: Lightweight Deep Convolutional Models for Distributed Inference on IoT Devices

Navidreza Asadi and Maziar Goudarzi

Abstract—Two major techniques are commonly used to meet real-time inference limitations when distributing models across resource-constrained IoT devices: 1) model parallelism (MP) and 2) class parallelism (CP). In MP, transmitting bulky intermediate data (orders of magnitude larger than input) between devices imposes huge communication overhead. Although CP solves this problem, it has limitations on the number of submodels. In addition, both solutions are fault intolerant, an issue when deployed on edge devices. We propose variant parallelism (VP), an ensemble-based deep learning distribution method where different variants of a main model are generated and can be deployed on separate machines. We design a family of lighter models around the original model, and train them simultaneously to improve accuracy over single models. Our experimental results on six common mid-sized object recognition data sets demonstrate that our models can have $5.8 \times - 7.1 \times$ fewer parameters, $4.3 \times - 31 \times$ fewer multiply accumulations (MACs), and $2.5 \times - 13.2 \times$ less response time on atomic inputs compared to MobileNetV2 while achieving comparable or higher accuracy. Our technique easily generates several variants of the base architecture. Each variant returns only $2k$ outputs $1 \leq k \leq \#\text{classes}/2$, representing Top-$k$ classes, instead of tons of floating point values required in MP. Since each variant provides a full-class prediction, our approach maintains higher availability compared with MP and CP in presence of failure.

Index Terms—Distributed machine learning, fault tolerance.

I. INTRODUCTION

CONVOLUTIONAL neural networks (CNNs) are being used in several visual analysis tasks, such as image recognition, object detection and segmentation [1], and tasks such as video analytics [2]. In CNNs, computation grows proportional to the input size, and it eventually leads to higher latency. Although reducing the number of parameters and precision of variables help in improving the latency, they often come at the cost of lower accuracy [3], [4]. Moreover, the complexity of deep learning models and their applications are only to grow over foreseeable future, and hence, overall they require even more resources. Model distribution across multiple machines is one viable solution that has consequently attracted growing attention especially when considering the fact that there are typically quite a number of local compute nodes around us that can help get the jobs done.

For atomic data that cannot be split into further pieces [5], two major solutions are often used to decompose and distribute a model across compute resources; namely, model parallelism (MP) and class parallelism (CP). MP decomposes a model into multiple sequential or parallel slices, each sent over to a separate compute node for processing, then lots of intermediate data values are returned to the master to combine and complete the model. Slicing in MP can be inter- or intra-layer [Fig. 1(a)]. CP decomposes the model into multiple disjoint models, each aiming to predict only one or a few nonoverlapping classes [Fig. 1(b)]; thus, contrary to MP, CP runs a monolithic model on each computing node. CP improves efficiency of each model mostly by applying class-aware pruning policies [5].

Both solutions have their own limitations. In MP, intermediate tensors have to be communicated between different nodes. In current CNNs, the time needed to transmit intermediate tensors can be orders of magnitude higher than...
inference time itself. Hence, MP would drastically increase end-to-end latency [6]. CP does not suffer from this issue since there is no sequential dependency and each computing node only returns a few numbers; these numbers represent the prediction of classes allocated to the submodel executed on that node. Verbatim CP has a firm restriction on scalability that is limited to the number of classes in the application [5], which can debatably be extended by choosing overlapping subsets of classes instead. However, more importantly, both paradigms intrinsically suffer from multiple instances of single point of failure (SPOF) issue because every compute node has to complete its job and send the result back to the requesting node before the top-level classification task can be fully finished. Otherwise, the master node cannot generate the final network prediction. Therefore, failure in a single node compromises availability of the whole system. The scenario of interest in this article is where we use the free times of IoT devices to do a distributed inference job. Failure is a common problem in distributed systems, especially in this scenario of cycle stealing from nearby IoT devices, where worker nodes only contribute in idle times and at any times may preempt the inference task to process their main jobs [7], and some nodes may seriously slow down due to various unexpected reasons. Additionally, communication issues such as network congestion, lead to missed deadlines in real-time applications. These concerns motivate the need for a fault-tolerant mechanism. Obviously, in both MP and CP, higher availability may be achieved by introducing redundant nodes, but of course this comes at the expense of large impact on communication and computation cost.

In this article, we propose a distribution technique: variant parallelism (VP). Both MP and CP can be viewed as applying a “top-down” model slicing approach: they try to break a baseline model into pieces, although differently in MP versus CP, and distribute each piece across multiple nodes. In contrast, we in VP employ a “bottom-up” approach: we generate multiple variants (described below) from a chosen basic architecture and then combine their results in an aggregator component [Fig. 1(c)]. In fact, CP and VP are both special types of ensemble learning models, but with different intents and approaches.

In VP, each variant is an independent model that can be deployed on a different machine or accelerator. Unlike CP, our variants have full-class prediction heads, i.e., each worker node predicts probabilities for all classes, and this is the key in achieving a fault-tolerant architecture. Consequently, in a system with n worker nodes, VP can still get a full response even if n − 1 nodes fail, while having the variants helps to improve accuracy over naïve replication when more than one worker node is working. VP’s variants can differ in input resolution and input offset. To achieve faster inference time, we apply three network optimizations: 1) narrowing network width; 2) reducing complexity of the compute intensive layers; and 3) replacing with faster operations when possible.

Since each machine generates full-class prediction, the data transmission amount would be higher than CP, resulting in risking a higher response time. To avoid this, we put a compression/decompression module. The compression layer attaches to each worker’s classification head. In the compression part we select k classes with the highest confidence score and send them to the master together with their corresponding indices. In VP, each machine executes a lightweight classification model, and sends its compressed result to the master. The master machine then applies a decompression step followed by a score scaling to weight each prediction based on its corresponding model capacity. It eventually combines results via an aggregation module. There is no trainable parameter in the combination modules. Otherwise, it reduces system’s flexibility and can compromise availability. For evaluation, we consider a smart home scenario and demonstrate VP’s robustness on seven object recognition benchmarks.

Our major contributions can be summarized as follows.

1) We introduce a bottom-up model distribution called VP which contrary to MP and CP is fault tolerant, flexible, and therefore, can be freely distributed across several sporadically available compute nodes, such as the case of IoT devices or across multiple accelerators within a single machine.

2) Our technique generates multiple variants each having different number of parameters, multiply accumulate operations (MACs), latency, and model size that can be deployed on different machines based on their capacity.

3) We propose an aggregation method that is resilient to failures, while matching accuracy of the baseline.

4) We propose a fast and simple technique to reduce bandwidth usage. It can compress the output size of each node by up to 10× while losing less than 0.5% accuracy. The gain would be two orders of magnitude higher on benchmarks with large number of classes, e.g., ImageNet.

5) Our proposed method can have 5.8×–7.1× fewer parameters, 4.3×–31× fewer MACs, and 2.5×–13.2× lower response time on an atomic input compared to the baseline while achieving comparable or higher accuracy.

II. RELATED WORK

MP: In MP [Fig. 1(a)] a deep learning model is sliced into multiple submodels [8], [9]. Each of them can be deployed and concurrently and/or sequentially executed on multiple machines. These submodels have to transfer data from intermediate tensors. The time to transfer these tensors can be orders of magnitude higher than the time to compute operations of the neural network itself. For example, [6] shows that the intermediate outputs of deep learning models are 19×−4500× larger than the compressed input. Recent proposals’ focus has mostly been on reducing communication rounds [10] or its overhead [11]. Though, the SPOF problem is still inevitable and inherent to MP.

CP: It decouples a base model into multiple independent submodels, each can predict one or a few nonoverlapping classes [Fig. 1(b)]. Structured pruning is then applied on each submodel to reduce their latency. The ultimate result can be prepared by combining the prediction of all of these submodels [5], [12]. CP does not have the issue mentioned for MP, and can improve latency compared to a single model. However,
it has a hard limitation on the number of submodels. More importantly, both MP and CP require results from all machines and suffer from multiple SPOFs.

VP works on atomic live data and by design tolerates up to \( n - 1 \) nodes failure. Contrary to other techniques, VP gives us the flexibility to generate enough variants based on different objectives, including latency constraints, model size (proportional to the number of parameters), and compute resources. VP can be combined with other parallelism schemes.

III. VARIANT PARALLELISM

In VP, we consider a basic model, and generate multiple lightweight and fast models based on it. Each of these variants can have different storage sizes, parameters, inference times, and accuracies. We add inference-time augmentation, that leads to feeding slightly different inputs for each model. Each variant can be treated as an independent full-class predictor: every machine can concurrently execute one or more of these variants in-parallel with other machines. Depending on the use case, these machines may be different GPUs on a single node or different edge devices, which is the case of interest in this article. If there is an idle node with limited storage space or computation constraints, VP can generate a lighter variant which provides the opportunity to utilize the available compute capacity. VP aims to reduce response time to enable faster decision making via a distributed deep learning architecture while maintaining availability in presence of failure.

A. Overview

Our workflow starts with one basic architecture, but is straightforward to extend to more than one model. As shown in Fig. 2, we generate different variants from the base model(s). We then add classification head, compression, decompression, and scaling modules to each variant. Variants are retrained on the desired data set. Each variant can be treated as an independent full-class predictor: every machine can concurrently execute one or more of these variants in-parallel with other machines. Depending on the use case, these machines may be different GPUs on a single node or different edge devices, which is the case of interest in this article. If there is an idle node with limited storage space or computation constraints, VP can generate a lighter variant which provides the opportunity to utilize the available compute capacity. VP aims to reduce response time to enable faster decision making via a distributed deep learning architecture while maintaining availability in presence of failure.

B. Reducing Model Complexity

For variant generation, we reduce complexity of the basic architecture through the following three steps: 1) reducing width factor; 2) reducing complexity of the last few layers; and 3) replacement with faster operations when possible. These steps help us to improve per machine model size, computation requirements, and latency. First, we reduce width factor of depthwise separable convolution layers (first introduced in [13]) by \( \frac{2}{85} \times (2 \times \text{ImageNet variants}) \). It improves network runtime [13]. We recover the lost accuracy by combining multiple parallel variants (Section IV-E). Second, we reduce complexity of the last few layers of each variant based on its input size. The intuition is that in the current efficient CNNs, the last few layers take considerable portion of the computation time. However, it should be proportional to the input resolution and the achievable accuracy. As shown in Table I, each model uses convolutional layers with the number of filters \( f \) proportional to its input size \( (\rho_i) \). For example, input size of \( V_3 \) \((i = 3)\) is 160 and its last convolution layer has 384 output filters while \( V_1 \) \((i = 1)\) with input size of 96 has only 256 output filters. We empirically select the number of these output filters so that they are divisible by 64 for better performance. In that sense, the first and second steps are types of structured pruning that result in fewer parameters and MACs. Finally, we employ faster operations when possible, e.g., replacing \( \text{FullyConnected}, \text{AveragePooling}, \) and \( \text{Average layers by } 1 \times 1 \text{Convolution, MaxPooling, and Add} \)

![Fig. 2. Overview of our proposed workflow, including offline parts (variant generation and model training) and online actions (dispatcher and aggregator).](image1)

![Fig. 3. Flow of data and the online operations on them, for inference.](image2)

### TABLE I

| Input | Operator | \( t \) | \( f \) | \( n \) |
|-------|----------|--------|--------|--------|
| \( 3 \times \rho_1^i \) | conv2d | - | 32 | 1 |
| \( 32 \times \rho_1^i / 4 \) | bottleneck | 16 | 1 |
| \( 16 \times \rho_1^i / 4 \) | bottleneck | 24 | 2 |
| \( 24 \times \rho_1^i / 8 \) | bottleneck | 32 | 3 |
| \( 32 \times \rho_1^i / 16 \) | bottleneck | 64 | 4 |
| \( 64 \times \rho_1^i / 32 \) | bottleneck | 96 | 3 |
| \( 96 \times \rho_1^i / 32 \) | bottleneck | 160 | 3 |
| \( 160 \times \rho_1^i / 64 \) | bottleneck | 320 | 1 |
| \( 320 \times \rho_1^i / 64 \) | conv2d \( 1 \times 1 \) | - | \( 64 \times j \) | 1 |
| \((64 \times j) \times \rho_1^i / 64 \) | globalmaxpool | - | - 1 |
| \((64 \times j) \) | conv2d \( 1 \times 1 \) | - | \( C \) | - |
| \( C \) | compression | - | - | - |

Similar to the original MobileNetV2, we apply a convolution layer on input followed by 19 depth-wise convolutional residual bottleneck blocks. Each row is sequentially applied \( n \) times, and \( t \) is the expansion factor of intermediate layers in bottleneck blocks. \( C \) is the number of classes. \( j = (3 + i) \) when width-factor=0.35 and \( j = \min ([7 + 2i], 20) \) when width-factor=0.5 (for ImageNet variants).
TABLE II
COMPARISON OF DIFFERENT MODELS’ CHARACTERISTICS

| Model   | Input Size | #Params | #MACs |
|---------|------------|---------|-------|
| MobileNetV2 | $3 \times 224^2$ | 2.27M (3.47 on ImageNet) | 300M |
| $V_1$  | $3 \times 320^2$ | 410K | 112M |
| $V_2$  | $3 \times 256^2$ | 390K | 71M |
| $V_3$  | $3 \times 224^2$ | 370K | 54M |
| $V_4$  | $3 \times 192^2$ | 360K | 40M |
| $V_5$  | $3 \times 160^2$ | 340K | 27M |
| $V_6$  | $3 \times 128^2$ | 330K | 17M |
| $V_7$  | $3 \times 96^2$ | 320K | 10M |
| $V_{inv}^1$ | $3 \times 320^2$ | 1.97M | 196M |
| $V_{inv}^2$ | $3 \times 256^2$ | 1.9M | 125M |
| $V_{inv}^3$ | $3 \times 224^2$ | 1.75M | 95M |
| $V_{inv}^4$ | $3 \times 192^2$ | 1.59M | 69M |
| $V_{inv}^5$ | $3 \times 160^2$ | 1.45M | 47M |
| $V_{inv}^6$ | $3 \times 128^2$ | 1.3M | 30M |
| $V_{inv}^7$ | $3 \times 96^2$ | 1.15M | 17M |

Variants $V_{inv}^i$ are exclusively designed for ImageNet benchmark [15].

TABLE III
DATA SETS USED FOR EVALUATION

| Dataset | #C | Input Size | #Trains | #Tests |
|---------|----|------------|---------|-------|
| Food-101 | 101 | $3 \times 256^2$ | 75,750 | 25,250 |
| SVHN | 10 | $3 \times 32^2$ | 73,257 | 26,032 |
| CIFAR-10 | 10 | $3 \times 32^2$ | 50K | 10K |
| CIFAR-100 | 100 | $3 \times 32^2$ | 50K | 10K |
| MNIST | 10 | $28^2$ | 60K | 10K |
| F-MNIST | 10 | $28^2$ | 60K | 10K |
| ImageNet | 1000 | $3 \times 256^2$ | 1.28M | 50K |

D. Scaling and Aggregation

Master is responsible for balancing the prediction vectors of different variants. After decompression step, we apply a Softmax operation to transform the reconstructed prediction vectors into a probability distribution (Fig. 4). We then need to scale each prediction vector based on its network capacity. For example, if variant $V_i$ has more parameters, and thereby returns more accurate predictions than variant $V_j$, then it deserves to have a higher weight in the aggregated result. We use a proxy of the network architecture itself to scale values of every prediction vector via

$$\psi_i = \left(\frac{\rho_i}{\max(P)}\right)\left(\frac{d_i}{\min(D)}\right)^\alpha \psi_i$$

where $\rho_i \in P$, and $d_i \in D$

IV. EXPERIMENTS

A. Experimental Setup

Compute Platform: We use an in-house server with an Intel Xeon E5 CPU, a NVidia GeForce GTX 1080 Ti (11 GB memory) GPU and 32 GB as Main Memory to design, train and evaluate our models. We select common vision data sets that are trainable on our server in a reasonable time.

Benchmarks: We have provided six different mid-sized object recognition data sets that can be categorized into three complexity levels (in terms of the achievable accuracy by the baseline architecture): 1) simple (CIFAR-10 [15], Fashion MNIST (F-MNIST) [16], MNIST [17] and SVHN [18]); 2) mediocre (CIFAR-100 [15]); and 3) difficult (Food-101 [19]). For completeness, we additionally provide accuracy results on ImageNet-1k (i.e., 1k classes) [14] (Table III).
TABLE IV

|                   | Samsung Galaxy Tab A 8.0 with S Pen (2019, SM-P205) |
|-------------------|------------------------------------------------------|
| CPU               | Octa-core big.LITTLE (2x1.8 GHz Cortex-A73 and 6x1.6 GHz Cortex-A53) |
| GPU               | ARM Mali-G71 MP2                                      |
| Memory            | 3GB LPDDR4X                                           |
| Network           | Wi-Fi 802.11, estimation: RTT=2ms bw=1Mbps            |

**Edge Device:** For latency evaluation of different models, we use a mid-range tablet (Table IV) and leverage Google TFLite native benchmark tool [20] to profile the performance.

**Baseline Model:** We consider MobileNetV2 [3] as our basic architecture (Table II). It is regarded as an efficient model designed for constrained edge devices. Thus, if we show gains in comparison to MobileNetV2, as we did, then one can expect similar results on models with more redundancy.

**B. Training**

We trained $V_1$ through $V_7$ for 135 epochs. Our training procedure has two steps. For the first 30 epochs, all layers except the last two convolutions were frozen. For the ImageNet variants ($V_{im1[i]}$), we iterate 75 epochs (the first 40 for fine-tuning). We port pretrained weights from ImageNet-1k [14] whenever available. We used batch size of 128 in this step, and 64 (32 in Food-101) for the rest of 105 epochs. We used Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Learning rate decays by a factor of 0.98 once the accuracy stagnates. We do not apply hyperparameter optimization. Thus, one may improve the results by doing so. Since our tasks are classification, and each model can independently predict all of the classes, we apply a separate categorical cross entropy for each model. Contrary to CP, our variants can observe the same data during training. Therefore, we train multiple models along with the combination components all together. The number of models that can be trained simultaneously is a parameter that can be set based on the computation capacity. To avoid overfitting, we apply typically used augmentation methods, namely, random horizontal flip, random rotation, and MixUp.

**C. Accuracy of Single Models**

First, we evaluate accuracy of each variant and compare it to the accuracy of our baseline. To have a fair comparison, we trained the baseline with the same training policy, and made sure it has the same or higher accuracy compared with original reports, if applicable.\(^1\) For each data set, we report the accuracy of five (seven for Food-101 and ImageNet) different models. We also report top-2 and top-5 in aggregator for data sets with $C < 100$ and $C \geq 100$, respectively. Here, by top-$m$ we mean the ground-truth label is among top $m$ class predictions with the highest confidence scores. As depicted in Table V, as we go from the variant $V_1$ to $V_7$ ($V_{im1}$ to $V_{im7}$ on ImageNet), the accuracy improves. For the data sets with few number of classes, the accuracy of $V_3$ is close to the baseline. Nevertheless, our $V_3$ has $\sim 6\times$ fewer parameters and MACs. This shows that for simple tasks with few classes, we can already reach to the baseline accuracy level without the need for parallelism. In fact, our bottom-up ensemble design gives us the flexibility to pick only one or a few variants for a custom use-case scenario while in model and CP, all models are mandatory. Yet, by using VP, as we explain next, we can achieve even higher accuracy compared with the baseline.

**D. Impact of Compression Method**

In this section, we analyze the effect of our compression algorithm on accuracy and the required bandwidth. We feed the same input data to all variants, and evaluate the impact of different values for $k$ in our compression module.

**Bandwidth Saving:** Output size is a linear function of $k$

$$h(k) = \left\{ \begin{array}{ll} \min \{k \times [\text{size}(fp) + 1], \text{size}(fp) \times C\} & , k > 1 \\ 1 & , k = 1 \end{array} \right. \quad (3)$$

where $h(k)$ is the output size in byte, size$(fp)$ is the number of bytes required for storing a floating point value and $C$ is the number of classes. Among different $k$ values, $k = 1$ is a special case. Because the compressor selects only one element, we can further reduce communication cost, sending only an integer representing its index. It can be found that choosing $k > (C/2)$ is not logical as transmitting the whole vector actually costs less than compressing it.

**Ensemble Accuracy:** In this experiment, we evaluate how different values of $k$ can impact the final accuracy. Fig. 5 illustrates a significant jump in top-1 accuracy from $k = 1$ to $k = 2$. For $k > 2$, the accuracy does not significantly increase. Another interesting observation is on top-$m$ accuracy: sending the prediction with the highest confidence score per variant improves top-$m$ accuracy. The reason is that when aggregating multiple prediction vectors, classes with lower scores behave like noise. By omitting these elements from our prediction vectors, we indeed achieve higher top-$m$ accuracy. It appears to be a tradeoff between higher top-1 and top-$m$ accuracies. Apparently, $k = 2$ is a good choice and we use it for the rest of our experiments.

**E. Scaling Accuracy**

We generated different variants each having unique characteristics. Here, we analyze the impact of combining different variants on the ensemble accuracy (Fig. 6). We show results on CIFAR-10, CIFAR-100, Food-101, and ImageNet. The other three data sets behave similarly to CIFAR-10. We show the final combined accuracy with respect to the average per-device MACs. Note that the horizontal axis is the average MACs of our experiments.

\(^1\) For ImageNet data set, we could not reach the reported accuracy. This is likely due to the choice of hyperparameters and fewer iterations as it was not feasible for us to provide the same infrastructure. Yet, since the same policy was applied to all models including the baseline, the results should hold fair.
### Table V

| Dataset      | Metric | MobileNetV2 | $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ | $V_7$ |
|--------------|--------|-------------|-------|-------|-------|-------|-------|-------|-------|
| CIFAR-10     | Top-1  | 95.16       | 91.76 | 93.07 | 93.81 | 94.43 | 94.62 | -     | -     |
|              | Top-5  | 98.57       | 97.24 | 97.87 | 98.20 | 98.76 | 98.29 | -     | -     |
| MNIST        | Top-1  | 99.37       | 99.38 | 99.41 | 99.41 | 99.42 | 99.46 | -     | -     |
|              | Top-2  | 99.87       | 99.92 | 99.95 | 99.89 | 99.92 | 99.95 | -     | -     |
| CIFAR-100    | Top-1  | 80.51       | 69.62 | 73.57 | 74.46 | 74.67 | 75.20 | -     | -     |
|              | Top-5  | 94.9        | 91.17 | 93.36 | 93.47 | 93.41 | 93.67 | -     | -     |
| SVHN         | Top-1  | 95.52       | 93.81 | 95.16 | 95.34 | 95.89 | 95.91 | -     | -     |
|              | Top-2  | 98.37       | 97.47 | 98.21 | 98.25 | 98.46 | 98.32 | -     | -     |
| Fashion-MNIST| Top-1  | 94.41       | 93.38 | 94.16 | 94.60 | 94.67 | 94.55 | -     | -     |
|              | Top-2  | 98.82       | 98.30 | 98.51 | 98.76 | 98.84 | 98.82 | -     | -     |
| Food-101     | Top-1  | 79.68       | 58.78 | 65.35 | 70.13 | 71.43 | 74.23 | 74.64 | 75.79 |
|              | Top-5  | 93.5        | 83.50 | 87.70 | 90.19 | 90.78 | 92.43 | 92.52 | 93.15 |
| ImageNet     | Top-1  | 67.5        | 48.8  | 54.7  | 57.9  | 61    | 62.25 | 63.2  | 64.9  |
|              | Top-5  | 87.9        | 73.65 | 78.7  | 80.9  | 83.3  | 84.2  | 85    | 86    |

Fig. 5. Impact of different $k$ values on accuracy of different benchmarks. $k$: the number of top predictions returned by each node. (a) CIFAR-10 aggregator Top-1. (b) CIFAR-100 aggregator Top-1. (c) Food-101 aggregator Top-1. (d) ImageNet aggregator Top-1. (e) CIFAR-10 aggregator Top-2. (f) CIFAR-100 aggregator Top-2. (g) Food-101 aggregator Top-5. (h) ImageNet aggregator Top-5.

In all the benchmarks, increasing $k$ in the compression step or transmitting the predictions without any compression gives better results but the differences are not significant. Interestingly, by increasing the number of variants to $>=6$ in the Food-101, this gap disappears. This can be due to the increase in the number of variants, particularly one or two more accurate ones.

### F. Effect of Every Variant on Accuracy

To find out whether each variant actually impacts the final accuracy, we evaluate the predictions returned by the combination of our variants while excluding only one of them. As shown in Fig. 7, in all data sets, omitting each variant has a negative impact on the final accuracy. This is especially interesting in less accurate variants. For example, on CIFAR-10, $V_1$ has $\approx 70\%$ accuracy itself which is more than $10\%$ lower than the baseline, yet when included in the aggregation, it can improve the final accuracy. Same is true in the other benchmarks. On ImageNet, however, we do not observe this
Fig. 6. Impact of different combinations of variants on the final accuracy (Please zoom in). (a) CIFAR-10. (b) CIFAR-100. (c) Food-101. (d) ImageNet.

Fig. 7. How omitting a variant affects the final accuracy (Please zoom in). (a) CIFAR-10. (b) CIFAR-100. (c) Food-101. (d) ImageNet.

pattern. Interestingly, although addition of variants increases accuracy [Fig. 6(d)], excluding one of them does not seem to negatively impact the final accuracy.

G. Availability in Presence of Failure

One serious drawback of utilizing idle nodes is preemption in presence of higher priority jobs (e.g., their main tasks). Missing real-time deadlines due to unexpected communication or computation issues are other reasons that indicate the demand for a fault-tolerant parallelism approach. Nevertheless, MP and CP need the results of all machines to provide the final outcome. By design, our architecture is fault tolerant even if \( n - 1 \) worker nodes are not able to return their predictions. Our aggregation module also supports this behavior. As shown in Fig. 6, we can still have results by contribution of a fraction of variants being executed on different machines. As the number of faulty nodes increases (i.e., fewer contributing variants), the final accuracy decreases. The achievable accuracy depends on the variants remained for the aggregation. In the worst case, all nodes fail except the one that executes \( V_1 \) as it has the lowest accuracy among all, which is still better than MP/CP that fail to provide any prediction. Our VP can be viewed as providing a graceful degradation in presence of increasing levels of faults/preemptions in the environment.

H. Other Metrics: Inference Time, MACs, Model Size

We report the performance of our variants on the other common metrics. We convert each variant designed for and trained on Food-101 to TFLite [21], and execute them for 100 iterations. The results are illustrated in Table VI. Since each variant has different characteristics, the slowest one determines the eventual response time. Therefore, depending on the available combination of nodes, one can get \( 2.5 \times -13.2 \times \) speedup compared to the baseline. Considering a smart home scenario with a star topology and the round-trip time \( RTT = 2 \text{ ms} \), the estimated speedup would be \( 2.4 \times -10.7 \times \)

I. Variant Parallelism versus Class Parallelism

We concisely compare our method with the current state-of-the-art deep learning parallelism algorithm, sensAI [5] which is a CP technique.

Fault Tolerance: We demonstrated that in case of failure on even \( n - 1 \) worker nodes, we can still get a response. In CP, however, since each model predicts different class(es), all contributing machines have to successfully process and send their results to the master. One can improve it by executing replications on additional machines, but it increases both complexity and cost. Another idea is to apply overlapping class-aware pruning [12]. In this type of replication, in presence of failure in one machine, the results can be aggregated from other machines with accuracy drop. Both ideas increase computation and communication costs.

Response Time, MACs, Model Size, and Training Time: As depicted in Table VI, by leveraging CP, sensAI can reach up to \( \sim 2 \times \) speedup using ten parallel machines. VP gets \( \sim 2.5 - 3.2 \times \) speedup compared to the baseline while requiring roughly half of in-parallel compute nodes compared to sensAI. Further note that, we trained our variants for 135 epochs which is 32.5% less than CP, yet we achieve comparable or higher accuracy. However, since our infrastructures and scheduling policies are different, direct comparison might be unfair.

Flexibility: Since each variant is being executed independently, it gives us more flexibility than CP to achieve different goals. We can generate additional variants based on more than
one basic architecture, or design a highly customized variant for contributing machines. We can have several variants with different characteristics to be deployed on heterogeneous compute machines. VP can also be combined with other distribution schemes, including CP.

V. Conclusion and Future Directions

In this article, we presented VP, a more flexible and fault-tolerant distribution scheme based on ensemble learning. Our evaluation on six different object recognition data sets demonstrates that our method can improve performance in number of parameters, MACs, and response time compared to the baseline and the current state-of-the-art. We note potential direction to be considered in future works. We intend to extend it to more complicated tasks, such as object detection and semantic segmentation. We expect more performance gain is achievable when tasks become more complicated. Both VP and CP are based on the well-known ensemble learning techniques. Therefore, they can have similar weaknesses and strengths. For example, in VP combining variants with identical characteristics, may not help achieving significant accuracy boost unless we retrain each of them with different random seeds [22]. Contributing machines in CP and VP observe the same data. In some scenarios, e.g., a smart city with long latency or bandwidth bottlenecks, transmitting input data can impact the end-to-end latency. Combining split computing paradigm [23], [24] with CP or VP can help mitigate the problem. On that end, a hierarchy of resources with different computation and communication characteristics (e.g., end-edge–cloud) is another direction which has recently gained attention and has its own challenges [25], [26], [27]. For generating variants we apply untargeted structured pruning. Investigating targeted pruning methods (e.g., [28]) to further improve retraining and inference times, therefore, can be considered as part of the future work to this article.

References

[1] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A ConvNet for the 2020s,” in Proc. IEEE/CVF CVPR, 2022, pp. 11976–11986.
[2] M. Khani et al., “RECL: Responsive resource-efficient continuous learning for video analytics,” in Proc. USENIX NSDI, 2023, pp. 1–16.
[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in Proc. IEEE CVPR, 2018, pp. 4510–4520.
[4] M. Tan and Q. Le, “EfficientNetV2: Smaller models and faster training,” in Proc. ICLR, 2021, pp. 10096–10106.
[5] G. Wang et al., “SensAI: ConvNets decomposition via class parallelism for fast inference on live data,” in Proc. MLSys, vol. 3, 2021, pp. 669–679.
[6] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winston, “Cracking open the DNN black-box: Video analytics with DNNs across the camera-cloud boundary,” in Proc. HotEdgeVideo, 2019, pp. 27–32.
[7] X. Tan and I. Hakala, “StateOS: A memory-efficient hybrid operating system for IoT devices,” IEEE Internet Things J., vol. 10, no. 11, pp. 9523–9533, Jun. 2023.
[8] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, “CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices,” IEEE/ACM Trans. Netw., vol. 29, no. 2, pp. 595–608, Apr. 2021.
[9] P. Yu and M. Chowdhury, “Fine-grained GPU sharing primitives for deep learning applications,” in Proc. MLSys, 2020, pp. 98–111.
[10] J. Du, M. Shen, and Y. Du, “A distributed in-situ CNN inference system for IoT applications,” in Proc. IEEE ICCD, 2020, pp. 279–287.
[11] H. Zhou, M. Li, N. Wang, G. Min, and J. Wu, “Accelerating deep learning inference via model parallelism and partial computation offloading,” IEEE Trans. Parallel Distrib. Syst., vol. 34, no. 2, pp. 475–488, Feb. 2023.
[12] Y. Yang et al., “Robust class parallelism-error resilient parallel inference with low communication cost,” in Proc. IEEE ASILOMAR, 2020, pp. 1064–1065.
[13] A. G. Howard et al., “MobileNets: Efficient convolutional neural networks for mobile vision applications,” 2017, arXiv:1704.04861.
[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in Proc. IEEE CVPR, 2009, pp. 248–255.
[15] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Dept. Comput Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009-09.
[16] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms,” 2017, arXiv:1708.07747.
[17] Y. LeCun, C. Cortes, and C. J. Burges, “The MNIST database of handwritten digits,” ATT Labs. 2010. [Online]. Available: yann.lecun.com/exdb/mnist
[18] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” in Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn., 2011, pp. 1–9.
[19] L. Bossard, M. Guillaumin, and L. Van Gool, “Food-101—Mining discriminative components with random forests,” in Proc. ECCV, 2014, pp. 446–461.
[20] “TFlite model benchmark tool with C++ binary,” 2022. [Online]. Available: github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/tools/benchmarks
[21] “TensorFlow lite,” 2022. [Online]. Available: tensorflow.org/lite
[22] Z. Allen-Zhu and Y. Li, “Towards understanding ensemble, knowledge distillation and self-distillation in deep learning,” 2020, arXiv:2012.09816.
[23] Y. Kang et al., “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge,” ACM SIGARCH Comput. Archit. News, vol. 45, no. 1, pp. 615–629, 2017.
[24] M. Sbai, M. R. U. Saputra, N. Trigoni, and A. Markham, “Cut, distil and encode (CDE): Split cloud-edge deep inference,” in Proc. IEEE SECON, 2021, pp. 1–9.
[25] N. Asadi and M. Goudarzi, “An ensemble mobile-cloud computing method for affordable and accurate glucometer readout,” 2023, arXiv:2301.01758.
[26] H. Zhou, Z. Zhang, D. Li, and Z. Su, “Joint optimization of computing offloading and service caching in edge computing-based smart grid,” IEEE Trans. Cloud Comput., vol. 11, no. 2, pp. 1122–1132, Apr.–Jun. 2023.
[27] H. Zhou, T. Wu, X. Chen, S. He, D. Guo, and J. Wu, “Reverse auction-based computation offloading and resource allocation in mobile cloud-edge computing,” IEEE Trans. Mobile Comput., early access, Jul. 18, 2022, doi: 10.1109/TMC.2022.3189050.
[28] K. Zhao, A. Jain, and M. Zhao, “Automatic attention pruning: Improving and automating model pruning using attentions,” in Proc. AISTATS, 2023, pp. 10470–10486.