A Hardware Prototype Targeting Distributed Deep Learning for On-device Inference

Allen-Jasmin Farcas[1], Guihong Li[1], Kartikeya Bhardwaj[2], Radu Marculescu[1]

[1]Department of Electrical and Computer Engineering
The University of Texas at Austin, Austin TX 78712
{allen.farcas, lgh, radum}@utexas.edu

[2]Arm Inc
San Jose, CA 95134
kartikeya.bhardwaj@arm.com

Abstract

This paper presents a hardware prototype and a framework for a new communication-aware model compression for distributed on-device inference. Our approach relies on Knowledge Distillation (KD) and achieves orders of magnitude compression ratios on a large pre-trained teacher model. The distributed hardware prototype consists of multiple student models deployed on Raspberry-Pi 3 nodes that run Wide ResNet and VGG models on the CIFAR10 dataset for real-time image classification. We observe significant reductions in memory footprint (50×), energy consumption (14×), latency (33×) and an increase in performance (12×) without any significant accuracy loss compared to the initial teacher model. This is an important step towards deploying deep learning models for IoT applications.

1. Introduction

Running computer vision tasks on edge devices requires efficient on-device inference due to the real-time demands; this local processing has also become necessary to avoid sending private data to the cloud. Given the memory, power, and computation constraints of a single edge device, even models compressed with well-known techniques such as pruning, quantization [1] or KD [2] need to be distributed in a smart manner among multiple edge devices in order to minimize the inter-device communication. Indeed, if done naively, these model compression techniques can result in high accuracy loss and/or significant communication increase among the distributed devices.

To meet the tight memory and power constraints of edge devices, we rely on a new communication-aware model compression called Network-of-Neural Networks (NoNN) [3]. This approach draws from KD and consists of two major pillars: first, the activation patterns of a neural network, called teacher, are used to build a network of filter activations. Second, community detection techniques are applied to this network to partition the teacher's knowledge into simpler functions. These functions are used to train the individual students to mimic each partition and hence perform the same task as the teacher without compromising the accuracy.

Experimental results show that partitioning based on community detection achieves orders of magnitude reduction in the number of model parameters, memory footprint, energy, and latency, with only a negligible accuracy loss compared to the original teacher model.

Finally, by creating a software framework, we enable users to select the teacher and student models for deploying on a network of edge devices and evaluate various power/performance trade-offs.

2. Approach

Our framework (Fig.1.) starts with an initial teacher model that is pre-trained. All filters are extracted from the teacher's last convolutional layer (LCONV), and used to create a network of filter activations. This network is then used to distribute teacher's knowledge into disjoint partitions via community detection [4]. The partitions are used to train multiple disjoint student models and distribute teacher's knowledge. The obtained partitions may not have the same number of filters, hence our framework balances the number of partitions learned by each student so that every student learns approximately the same number of filters in total. The student models are carefully selected so they can be deployed on edge devices while satisfying tight memory (e.g., 500KB), power (e.g., 2W), and computational constraints (e.g., ARM Cortex-A53)and maintaining teacher's accuracy at image classification. The outputs of all students are merged by a Fully Connected (FC) layer, similar to the one used by the teacher, to make the final prediction. After training, each student model is deployed on a single edge device.

2.1. Teacher partitioning

We suppose that the teacher contains several convolutional layers and one or more FC layers for prediction. When passing an image (φ) from the validation set (*val*) through the teacher network, in the LCONV, each f_i filter has a certain feature map. By computing the average value of the feature map which corresponds to a filter, we obtain the average activity (a_i) for filter f_i. The higher the a_i value of filter f_i, the more important the filter f_i is for classifying image φ. Using all *val* images from a given class, we can detect which filters matter the most for that class using a_i [3].

The framework builds a network with f_i filters as nodes and each two nodes (f_i, f_j) are connected by a weighted edge with $F_{ij} = \sum_{val} a_i\, a_j |a_i - a_j|$ as the weight. This *filter activation network* is partitioned using community detection via Activation Hubs (AHs). The AHs are created in such a manner that they are sparsely interconnected, resulting in N disjoint partitions [3].

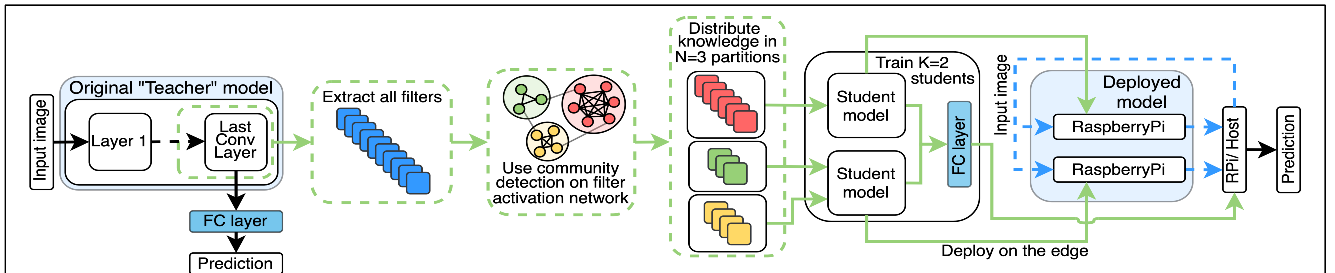

Fig. 1. Proposed distributed model compression framework for disjoint edge devices

2.2. Training student models

Our framework currently enables users to choose the number of desired students (K), and the neural network model for each student. Each student model needs to be chosen (or designed by the users) to fit within a single edge device under given constraints. To train all students at once, we create a single neural network using the chosen student model which is replicated K times. All students have the same model and are connected via a FC layer. We train this network with a KD-based loss function such that each student can mimic a certain partition from the original teacher [3].

2.3. Deploying student models on edge devices

The obtained student models are extracted from the unified network and compiled independently for deployment on the destination devices using the Tensor Virtual Machine (TVM) compiler. After compilation and deployment, a host device controls the inference process by sending the images required to be classified, collecting the outputs of each deployed student and making the final prediction based on those outputs. The host can be an edge device or a general-purpose computer.

3. Experimental setup and results

For the teacher model, we have conducted two experiments: one with the WRN network with a depth of 40 and a widening factor of 4 (WRN40-4), and another with VGG-19. In both cases we consider FP32 precision and we use the framework to partition the teacher model. As student models, we use WRN with a depth 16 and a widening factor of 1 (WRN16-1) in the first experiment, and VGG-11 in the second experiment. The results (see Table I and Table II) are obtained from inferencing 10000 images from the CIFAR10 test dataset on a NoNN with $K = 2$ students.

Table I. WRN experimental results

Metrics	Teacher	Student	Improvement
Accuracy	96.78%	95.92%	-0.86%
Parameters	8.9M	0.18M	49.89×
Latency [ms]	1878	150	12.44×
Energy [mJ]	3430.67	238.98	14.36×

We compare the inference process for a teacher deployed on a RaspberryPi 3B+ (RPi) and a NoNN of two students deployed on the same device. The number of parameters, latency, and energy corresponding to the Student column (in both tables) is measured on one RPi device to emphasize the impact of the teacher and a student model deployed on a single RPi.

Table II. VGG experimental results

Metrics	Teacher	Student	Improvement
Accuracy	93.00%	91.47%	-1.53%
Parameters	20M	9.2M	2.16×
Latency [ms]	847	414	2.04×
Energy [mJ]	436.27	366.39	1.2×

4. Conclusion

In this paper, we have presented a hardware prototype and a software framework to compress large teacher models for distributed power- and latency-constrained on-device inference at the edge. The student models perform on-device inference efficiently and deliver real-time performance while deployed on RPi edge devices.

Our approach demonstrates significant reductions in memory footprint (50×), energy consumption (14×), latency (33×) and an increase in performance (12×), while maintaining the accuracy of the initial teacher model.

Our demonstration consists of a live deployment of the resulted compressed models on RaspberryPi 3B+ boards with real-time on-device inference and actual performance and power measurements.

References

[1] S. Han, H. Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv:1510.00149.

[2] G. Hinton, O. Vinyals, and J. Dean. 2015. Distilling the knowledge in a neural network. arXiv:1503.02531

[3] K. Bhardwaj, C. Y. Lin, A. Sartor, and R. Marculescu, "Memory- And communication-aware model compression for distributed deep learning inference on IoT," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 5s, 2019, doi: 10.1145/3358205.

[4] M. E. J. Newman. Modularity and community structure in networks. PNAS 103, 23 (2006), 8577–8582.