To Compress, or Not to Compress: Understanding Deep Learning Model Compression for Embedded Inference

Qin Qing‡, Jie Ren†, Ling Gao‡, Jianbin Fang§, Yansong Feng¶, Zheng Wang*
‡ Northwest University, China, † Shaanxi Normal University, China, § National University of Defense Technology, China
¶ Peking University, China, * Lancaster University, United Kingdom

***Abstract*—Abstract comes here...**

***Keywords*-Deep learning, embedded systems, parallelism, energy efficiency**

I. Introduction

FIX:ZW: I will come back to this later.

II. Background and Motivation

A. Background

In this work, we consider xx commonly used model compression techniques, described as follows.

Pruning.

Data quantization.

xx.

B. Motivation

Choosing the right compression technique is non-trivial. As a motivation example, consider applying two widely used model compression techniques, *pruning* [1] and *data quantization* [], to two influential `CNN` models, `VGG_16` and `Resnet_50`. Our evaluation platform is a NVIDIA Jetson TX2 embedded deep learning platform (see Section **??**).

Setup. We apply the compression technique to the pre-trained model (which has been trained on the ImageNet ILSVRC 2012 training dataset []). We then test the original and the compressed models on ILVRSC 2012 validation set which contains 50k images. FIX:Describe the setup of your compression techniques. E.g. what are the pruning and quantization thresholds. We use the GPU for inferencing.

Motivation Results. Figure 1 compares the model size, inference time and accuracy after applying model compression. By removing some of the pathways of the network, *pruning* is able to reduces the inference time by 28%. However, it offers little saving in storage size because network weights still deaminate the model size. By contrast, by using a few number of bits to represent the weights, *quantization* significantly reduces the model storage size by 75%. However, the reduction in the model size does not translate to faster inference time; on the contrary, the inference time increases by 1.45x. This is because the sparsity in network weights brought by *quantization* leads to irregular computation which causes poor GPU performance []. Applying both compression techniques has modest impact on the prediction accuracy, on average, less than 5%. This suggests that both techniques can be profitable.

Lessons Learned. This example shows that the compression technique to use depends on what to be optimized for. If storage space (e.g., RAM and disk) is a limiting factor, *quantization* could be used, but a more powerful processor unit is required to achieve quick on-device inference. If faster on-device turnaround time is a priority, *pruning* can be employed but it would require sufficient memory resources to store the model parameters. Since offloading the computation into the cloud is often infeasible due to privacy concerns, high latency, or the lack of connectivity, there is a need to understand the pros and cons of model compression techniques for embedded deep learning. This work provides an extensive study to characterize the benefit and cost of commonly used deep learning compression techniques on embedded systems.

III. Experimental Setup

A. Platform and Models

Hardware. Our experimental platform is the NVIDIA Jetson TX2 embedded deep learning platform. The system has a 64 bit dual-core Denver2 and a 64 bit quad-core ARM Cortex-A57 running at 2.0 Ghz, and a 256-core NVIDIA Pascal GPU running at 1.3 Ghz. The board has 8 GB of LPDDR4 RAM and 96 GB of storage (32 GB eMMC plus 64 GB SD card).

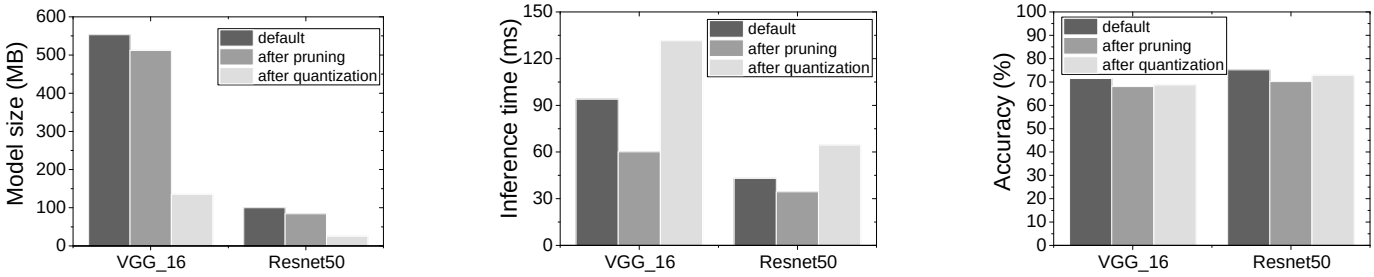
System Software. Our evaluation platform runs Ubuntu 16.04 with Linux kernel v4.4.15. We use Tensorflow v.1.0.1, cuDNN (v6.0) and CUDA (v8.0.64).

Deep Learning Models. We consider FIX:14 pre-trained `CNN` models for image recognition from the TensorFlow-Slim library [**?**]. The models are built using TensorFlow and trained on the ImageNet ILSVRC 2012 training set.

B. Evaluation Methodology

Evaluation Metrics We consider the following metrics:

- ***Inference time*** *(lower is better)*. Wall clock time between a model taking in an input and producing an output, excluding the model load time.
- ***Energy consumption*** *(lower is better)*. The energy used by a model for inference. We deduct the static power used by the hardware when the system is idle.
- ***Accuracy*** *(higher is better)*. The ratio of correctly labeled images to the total number of testing images.
- ***Precision*** *(higher is better)*. The ratio of a correctly predicted images to the total number of images that are predicted to have a specific object. This metric answers



(a) Model size (b) Inference time (c) Accuracy

Figure 1: The achieved model size (a) inference time (b) and accuracy (c) before and after the compression by *quantization* and *pruning*. The compression technique to use depends on the optimization target.

e.g., "*Of all the images that are labeled to have a cat, how many actually have a cat?*".

- ***Recall*** *(higher is better)*. The ratio of correctly predicted images to the total number of test images that belong to an object class. This metric answers e.g., "*Of all the test images that have a cat, how many are actually labeled to have a cat?*".
- ***F1 score*** *(higher is better)*. The weighted average of Precision and Recall, calculated as $2 \times \frac{Recall \times Precision}{Recall + Precision}$. It is useful when the test datasets have an uneven distribution of object classes.

Performance Report. To collect inference time and energy consumption, we run each model on each input repeatedly until the 95% confidence bound per model per input is smaller than 5%. In the experiments, we exclude the loading time of the CNN models as they only need to be loaded once in practice. To measure energy consumption, we developed a lightweight runtime to take readings from the on-board energy sensors at a frequency of 1,000 samples per second. We then matched the energy readings against the time stamps of model execution to calculate the energy consumption.

IV. Experimental Results

A. Overall Results

B. Compare to Oracle

C. Analysis

V. Related Work

DNNs have shown astounding successes in various tasks that previously seemed difficult []. Despite the fact that many embedded devices require precise sensing capabilities, adoption of DNN models on such systems has notably slow progress. This mainly due to DNN-based inference being typically a computation intensive task, which inherently runs slowly on embedded devices due to limited resources.

FIX:Talk about different compression techniques.

There is an extensive body of work on how to accelerate DNN training using xx, xx, and xx. Our work aims to understand how to accelerate deep learning inference by choosing the right model compression technique.

As an alternative to on-device inferencing, off-loading computation to the cloud can accelerate DNN model inference [**?**]. Neurosurgeon [**?**] identifies when it is beneficial (*e.g.* in terms of energy consumption and end-to-end latency) to offload a DNN layer to be computed on the cloud. The Pervasive CNN [**?**] generates multiple computation kernels for each layer of a CNN, which are then dynamically selected according to the inputs and user constraints. A similar approach presented in [**?**] trains a model twice, once on shared data and again on personal data, in an attempt to prevent personal data being sent outside the personal domain. Computation off-loading is not always applicable due to privacy, latency or connectivity issues. The work presented by Ossia ifnextchar.et alet al.partially addresses the issue of privacy-preserving when offloading DNN inference to the cloud [**?**]. Our work is complementary to prior work on computation off-loading by offering insights to choose the optimal compression technique to best optimize local inference.

VI. Conclusions

This paper has presented

References

[1] Franco Manessi et al. Automated pruning for deep neural network compression. *arXiv*, 2017.