Title
OptDNN: Automatic Deep Neural Networks Optimizer for Edge Computing

Authors
Luca Giovannesi, Gabriele Proietti Mattia, Roberto Beraldi
{giovannesi, proiettimattia, beraldi}@diag.uniroma1.it

Department of Computer, Control and Management Engineering "Antonio Ruberti"
Sapienza University of Rome, Via Ariosto 25, 00185, Rome, Italy

Abstract
DNNs are widely used for complex tasks like image and signal processing, and they are in increasing demand for implementation on Internet of Things (IoT) devices. For these devices, optimizing DNN models is a necessary task. Generally, standard optimization approaches require specialists to manually fine-tune hyper-parameters to find a good trade-off between efficiency and accuracy. In this paper, we propose OptDNN, a software that employs innovative and automatic approaches to determine optimal hyper-parameters for pruning, clustering, and quantization. The models optimized by OptDNN have a smaller memory footprint, faster inference time, and a similar accuracy to the original models.

Keywords
Deep Neural Networks; DNN Acceleration; DNN Compression; Edge Computing

Code metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1 (Optimizer v1.0.0, Edge Software v1.0.0)
C2	Permanent link to code/repository used for this code version	https://github.com/edgelab-sapienza/optdnn https://github.com/edgelab-sapienza/optdnn-edge
C3	Permanent link to Reproducible Capsule	https://doi.org/10.24433/CO.6645065.v1
C4	Legal Code License	GNU GPLv3
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	python3.9
C7	Compilation requirements, operating environments & dependencies	TensorFlow, Numpy, WebSockets, FastAPI, SQLAlchemy
C8	If available Link to developer documentation/manual	https://github.com/edgelab-sapienza/optdnn/blob/main/README.md
C9	Support email for questions	giovannesi@diag.uniroma1.it

1. Introduction

DNNs are widely used for complex tasks like image and language processing, and they are in increasing demand for implementation on Internet of Things (IoT) devices. These devices work with limited resources such as computing power, memory, and energy requirements. Then, optimizing DNN models becomes critical for reducing memory consumption and calculation time.

Actually, when a DNN is optimized, some hyper-parameters must be set; if these parameters are too aggressive, the optimizations significantly reduce the model's accuracy, whereas if they are too soft, the benefits of the optimization are not visible, so the inference time and model size are similar to the ones of the original model.

Finding the best balance between these two sides is critical. Nowadays, the most common approach is to manually set these hyper-parameters, and after a few trials, an expert can determine the optimal balance point.

The purpose of our software, called OptDNN, is to completely automate this process.

To the best of our knowledge, OptDNN is the first open-source software that, starting from a DNN model, can build an optimized model ready for deployment on an edge device without a prior understanding of DNN optimization by the user. The entire method is delegated to the program.

This software relies on the following DNN optimization strategies:

- **Pruning**. DNNs may contain redundant parts [1]. The main idea of pruning is to remove these parts, in particular, OptDNN employs Global Magnitude Pruning [2], which removes all weights with absolute values less than a specific threshold.

  The crucial part is to find the optimal balance point between removed weights and accuracy.

- **Weight Clustering** [3]. The amount of unique values of the weights are reduced, grouping them in clusters. As a result, several weights are represented by a single value.

  The problem here is to minimize the number of clusters without significantly compromising model accuracy.

- **Quantization** [4]. The weights are transformed from their original floating point data type to an integer data type, which is typically represented by 8 bits. Additionally, also the activation functions can be updated to use the new data type.

Each applied optimization brings benefits in terms of reduced model size and inference time but also introduces an inevitable drop in accuracy. The essential element of DNN optimization is achieving the optimal balance between accuracy loss and performance increases.

2. Software description

OptDNN is an automatic software for neural network optimization, once it has take as inputs: a keras model, a dataset, and some other information, it springs in action its newel agortihms to return a TensorFlow Lite optimized model with an accuracy close to the one of the original model, but with a speed-up of the inference time and a smaller model size.

The optimization procedure is completely automated, and is not required by the user any prior DNN optimization experience.

The user can interact with OptDNN using two interfaces: a CLI or HTTP APIs; it is also available a Docker image to be used on a server using the HTTP APIs.

2.1 Software components

OptDNN is composed by two software, the optimizer and the edge software.

The optimizer is the software which applies the optimizations to a model, it relies on TensorFlow library and require a host provided with GPU to be run smoothly.

The edge device's software waits for a model and a validation dataset from the optimizer, then evaluates the model and provides the measured metrics to the server. Also this software is released as Docker container for a fast and easy deployment.

2.2 How it works

Initially, the program examines the input model and establishes an accuracy baseline for subsequent operations. Then it begins to prune the network with its revolutionary methods. After experimenting with several hyperparameters, it returns the best pruned model.

Next, weight clustering is applied, starting from the pruned model, the program invokes the algorithm particularly intended to determine the optimal number of clusters, returning a model pruned from the previous step and clustered with the current one.

Finally, it tests multiple quantization approaches on the generated model, ordered from the more aggressive to the gentler, and returns the first that matches the desired target accuracy.

At this stage, the software has generated the optimized model based on all of the previously determined optimal parameters.

Then, the program tests the model on one or more edge devices to provide to the user a real-world feedback of the applied optimization.

In the edge devices, there is a WebSocket server that listens for incoming commands from the optimizer. With these commands, the edge software receives instructions to download the model and the dataset via HTTP GET requests, which are solved by an HTTP server expressly created by the optimizer for this purpose.

When the edge device has all the necessary data, it evaluates the models and returns the results over WebSocket.

Since multiple edge devices can be used, this evaluation part may also be utilized to assess how an optimized model performs on various edge devices, allowing a comparison between the architectures.

Indeed, if any edge devices are not provided to the optimizer, the software use the local machine for the evaluations.

An overall overview of the entire workflow is shown in figure 1.

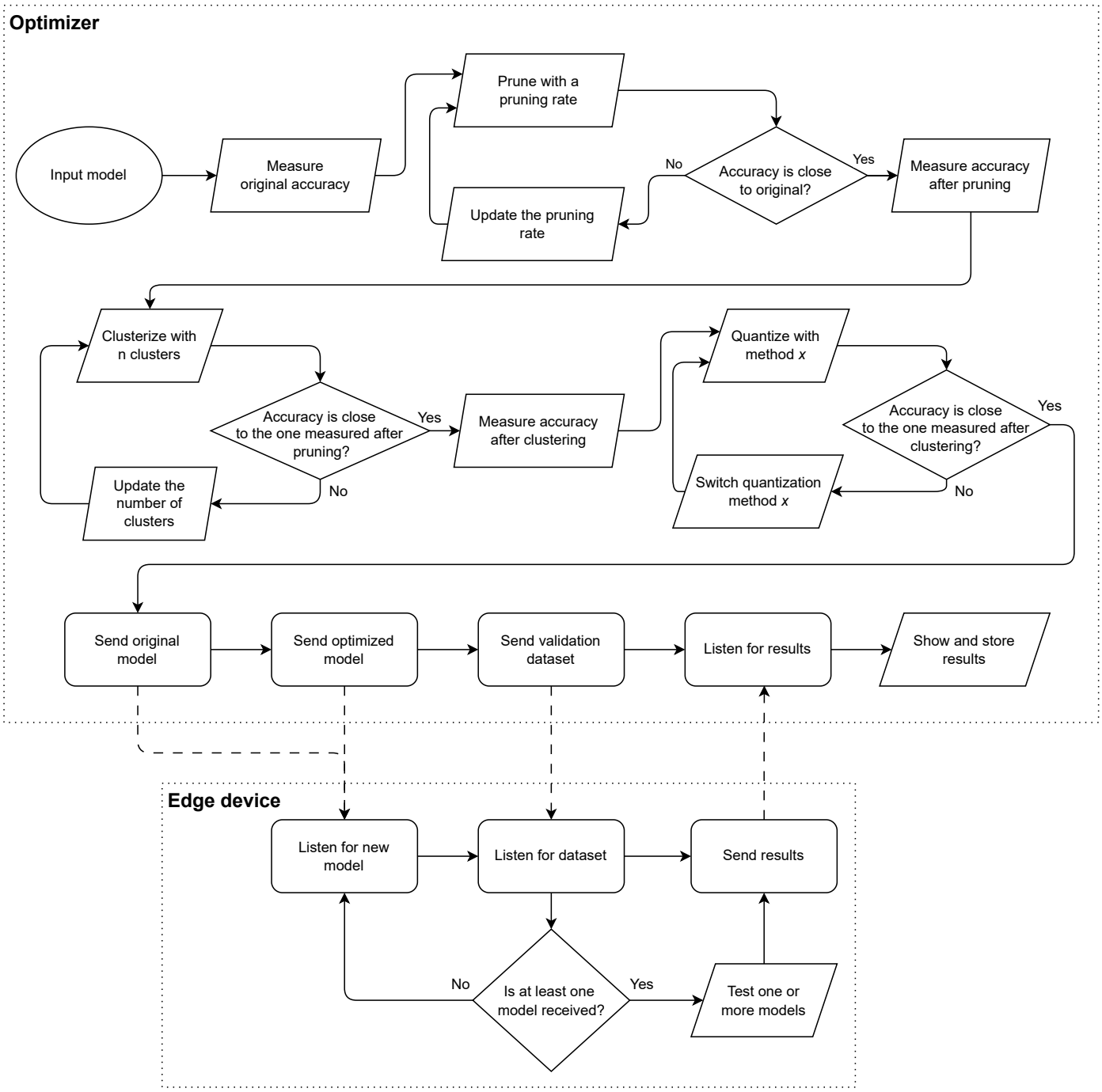

Figure 1: Overall optimization procedure workflow

The optimization procedure takes time, especially with large model, furthermore only one model at time can be processed. When the software is used with the HTTP APIs, the requests are handled asynchronously. The requests for new optimization tasks are queued on a list stored on a SQLite server. Then the stored tasks will be processed when there isn't any model being processed.

3. Impact

This software's target users include: researchers of IoT or edge computing projects, companies, amateurs, and anyone who wants to deploy a DNN on an edge device. OptDNN allows them to easily run efficient DNN models on their edge devices without effort.

OptDNN allows for large models, which were previously impossible to be executed on low-performance hardware, to be run on these devices. The software allows also to optimize a model specifically for 8-bit devices with a slight accuracy loss, expanding the pool of compatible devices.

In general, since the inference time has been reduced, the compute unit spends more time in idle state. As a result, energy consumption is reduced [5, 6], bringing also an extension of the battery life for battery-powered devices.

Given that the optimization process is totally automated, the presented software can be incorporated into a CI/CD pipeline via the HTTP APIs, ensuring that new features are automatically distributed on target devices.

This software expands the capabilities in research projects, for example, all projects that require image processing on edge devices, such as for animal or nature phenomenal monitoring, usually use small boards for computation, commonly powered by batteries recharged by solar panels, making the computational efficiency a priority. OptDNN enables them to run neural networks that meet the efficiency constraints, previously, this was not possible without an expert in optimization techniques on the team.

Moreover, while researchers are focused on other tasks, OptDNN optimizes the provided models in background. This allows the team to dedicate more time to the critical aspects of their projects, delivering high-quality results.

Even companies take advantage of OptDNN. The productivity is increased since human resources are not needed for model optimization, allowing them to reduce labor costs or move the human resources to other vital tasks.

Furthermore, this method has a quick scalability. If the amount of different neural network models increases, the software can be allocated on new GPUs to increase the parallelization, while with the previous manual method, looking for new experts with the desired skills could take time.

Users who will use the final device will also benefit from this software because low-power boards can be adapted by manufacturers, which are usually cheaper, and as previously stated, the adoption of this software reduces labor costs, allowing the final product to be priced lower.

Because the final items will have faster software, the device's quality will increase, providing the user with a better usage experience. Finally, as previously reported, due to lower energy consumption, battery devices have a longer battery life, requiring fewer recharges by the user. This advantage also allows for some devices to be powered by a solar panel, making installation easier and free from power plugs.

This software has been developed within the project "EdgeVision against Varroa (EV2): Edge computing in defence of bees," which is founded by the Italian MUR PRIN2022 (ERC PE6) research program and by European Union - Next Generation EU (project no. 202277 WMAE CUP B53D23012820006). In this study, solar-powered edge devices use computer vision to identify Varroa mites in beehives. OptDNN allows to increase the efficiency of the DNN model used for the detection.

4. Conclusion

This article proposes the first version of an automatic deep neural network model optimizer, which can be easily integrated into deploy pipelines bringing advantages for a large set of users. However, the proposed solution's simplicity allows for future refinement. For example, by employing the same search algorithms, we may utilize superior pruning strategies, such as filter pruning, to significantly boost the inference time.

The present version of the framework can only handle models for classification tasks, thus our next goal is to implement the most often used object detection and segmentation tasks.

Actually, because a large number of fine-tuning phases are necessary, the optimizer can take a long time to converge, especially with big models; therefore, it would be desirable to improve the used algorithms, employing more targeted solutions to reduce convergence times.

The proposed software produces promising results, optimizing ResNet50 [7] in 125 minutes on a nVidia RTX 1080, returning a model with a size reduction of 6.35x and a speedup of 2.91x on a Radxa Rock 5B.

References

[1] M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems* (C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds.), vol. 26, Curran Associates, Inc., 2013.

[2] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015.

[3] J. S. Larsen and L. Clemmensen, "Weight sharing and deep learning for spectral data," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4227–4231, 2020.

[4] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021.

[5] I. Galanis, I. Anagnostopoulos, C. Nguyen, G. Bares, and D. Burkard, "Inference and energy efficient design of deep neural networks for embedded devices," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 36–41, 2020.

[6] L. Papa, G. Proietti Mattia, P. Russo, I. Amerini, and R. Beraldi, "Lightweight and energy-aware monocular depth estimation models for iot embedded devices: Challenges and performances in terrestrial and underwater scenarios," *Sensors*, vol. 23, no. 4, 2023.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.