ACCELERATING HYPERPARAMETER TUNING OF A DEEP LEARNING MODEL FOR REMOTE SENSING IMAGE CLASSIFICATION

Marcel Aach[1,2]*, Rocco Sedona*[1,2]*, Andreas Lintermann*[2]*, Gabriele Cavallaro*[2]*, Helmut Neukirchen*[1]*, Morris Riedel*[1,2]

[1] School of Engineering and Natural Sciences, University of Iceland, Iceland
[2] Jülich Supercomputing Centre, Forschungszentrum Jülich, Germany

ABSTRACT

Deep Learning models have proven necessary in dealing with the challenges posed by the continuous growth of data volume acquired from satellites and the increasing complexity of new Remote Sensing applications. To obtain the best performance from such models, it is necessary to fine-tune their hyperparameters. Since the models might have massive amounts of parameters that need to be tuned, this process requires many computational resources. In this work, a method to accelerate hyperparameter optimization on a High-Performance Computing system is proposed. The data batch size is increased during the training, leading to a more efficient execution on Graphics Processing Units (GPUs). The experimental results confirm that this method reduces the runtime of the hyperparameter optimization step by a factor of 3 while achieving the same validation accuracy as a standard training procedure with a fixed batch size.

***Index Terms*—** Hyperparameter tuning, Deep Learning, Batch Size, High-Performance Computing, Remote Sensing.

1. INTRODUCTION

The enormous investments that made freely available data acquired by modern Earth Observation (EO) programs have democratized access to timely satellite imagery of the entire planet. Missions such as the Copernicus Sentinel-2 can re-observe the same area every 5 days (under cloud-free conditions) by exploiting two polar orbiting satellites. Its free data represent an invaluable asset for tackling challenges as wide-ranging and important as quantifying the effects of climate change through land cover classification, vegetation mapping, environmental monitoring, etc. [1]. Nevertheless, the extraction of valuable information from raw satellite data is complex and requires large amounts of labelled training samples when using supervised learning with Deep Learning (DL) models. Furthermore, to achieve the best performance from a DL model, it is fundamental to optimize the values of its parameters. This optimization step requires several processing steps that may consume a lot of computing power, i.e., leading to long processing times.

The present manuscript contributes to the Remote Sensing (RS) community by exploring a way to reduce these computational costs. While the group's previous work [2] focused on using evolutionary methods, this work aims at reducing hyperparameter tuning costs by training with a large batch size BS without sacrificing validation accuracy. By running the hyperparameter tuning more efficiently, it becomes faster and cheaper for the community to find the best performing models. The experiments make use of the BigEarthNet-19 dataset [3]. It consists of 590,326 patches extracted from 125 Sentinel-2 tiles, each associated to one or more of the 19 labels of the simplified legend of the CORINE Land Cover [4], a thematic map from 10 European countries updated in 2018.

2. PROBLEM FORMULATION

The performance of deep neural networks depends on the hyperparameters set by the user before training. This usually involves a lot of manual tuning but may yield huge gains in performance. The main problem in finding the right set of hyperparameters is the expensive evaluation of different configurations. Each of them requires a full model training run. In principle, two main strategies for reducing the overall computational costs exist: (1) improving the choice of hyperparameters with optimization algorithms and (2) reducing the runtime of the training runs. This work focuses on the latter.

Large batch size parameter values BS are necessary when a large dataset is used for training on multiple Graphics Processing Units (GPUs) on an High-Performance Computing (HPC) system. In this case, a large BS value (that still fits into the GPU memory) leads to a higher GPU utilization, increasing the efficiency. BS values of up to 80,000 samples have been reported for a Convolutional Neural Network (CNN) [5].

This work was performed in the Center of Excellence (CoE) Research on AI- and Simulation-Based Engineering at Exascale (RAISE) receiving funding from EU's Horizon 2020 Research and Innovation Framework Programme H2020-INFRAEDI-2019-1 under grant agreement no. 951733. The authors gratefully acknowledge the computing time granted by the JARA Vergabegremium and provided on the JARA Partition part of the supercomputer JURECA at Forschungszentrum Jülich.

However, training with large BS values usually results in a lower validation performance, which is a general problem in distributed DL. Several techniques have been proposed to circumvent this problem, i.e., scheduling the learning rate LR to slowly increase at the beginning and then decaying it over time [6], and using optimizers such as LARS [7] or LAMB [8] that introduce layerwise adaptive scaling mechanisms.

While these approaches focus on adjusting the parameter LR, this work adapts the parameter BS itself to accelerate the process. Empirically, this has a similar effect as decaying the LR over time [9] while using less parameter updates. Smith et al. [9] train a CNN on ImageNet starting with $BS = 8{,}000$ images, which is increased to $BS = 16{,}000$ after $E = 30$ epochs. Compared to the baseline of keeping the BS constant at 8,000 throughout the whole training process, a $\approx 33\%$ faster convergence with no drop in validation accuracy (76.1%) is reached. McCandlish et al. [10] introduce a metric called the Gradient Noise Scale (GNS) to predict the largest useful BS value to be employed during each part of training. When using Stochastic Gradient Descent (SGD) with a small BS value, the gradient update is a noisy approximation of the true gradient. A big batch resembles the true gradient much better. Following this intuition, the GNS measures the ratio of noise (variance) to signal (size) of the gradient. A large noise to signal ratio indicates that a bigger BS value should be used and vice versa. Libraries such as Pollux [11] or KungFu [12] use the GNS and similar metrics to systematically optimize the throughput of DL models on HPC systems.

3. METHODOLOGY

3.1. Distributed Deep Learning

Training a neural network on large datasets can be time consuming as the the model needs to iterate though the whole input data once per epoch. One method to accelerate this process is to use data parallel training: the input data is split and distributed to different GPUs that all train separately on their own batches but perform a gradient synchronization at the end of each epoch. This way, the model on each GPU is the same but the data is different. Horovod [13] is an easy-to-use Python library that implements efficient data parallel training and was already used by us in the past to train on an RS dataset with up to 128 GPUs [14].

3.2. EfficientNet

The benefits of employing CNNs come at the cost of an increased computational budget. EfficientNet [15] is an architecture that, maintaining a fixed ratio between the width and the depth of the network, aims at decreasing the amount of parameters (i.e., weights and biases of the network) while maximizing the extraction of fine-grained and high-level features. It consequently curbs the usage of resources as compared to other benchmark models such as ResNet [16], reaching higher test accuracies while being of smaller size. Here, the EfficientNet-B0 [15] is used, a model achieving better test accuracies than ResNet-50 while having much less parameters ($5.3\,\mathrm{M}$ for EfficientNet-B0 vs. $26\,\mathrm{M}$ for ResNet-50).

3.3. Hyperparameter Optimization with Ray

Tuning the hyperparameters of a neural network involves training a lot of different sets of hyperparameters (*configurations*). A complete training run of said configuration is called a *trial*. Allocating resources and launching each trial manually is inefficient. Therefore, Ray[1] is used, which is an open-source library for distributed computing. Its subpackage Ray Tune can run distributed hyperparameter tuning at scale. It provides options to specify: the number of resources to use per trial, the hyperparameters, which range they are sampled from, and a scheduling or optimization algorithm. With this approach, a single Ray Tune job is started and Ray deals with all scheduling and communication tasks.

3.4. Changing the Batch Size

DL libraries like Tensorflow usually require the BS to be set in the beginning of a training run and remain fixed throughout. To change the BS using such libraries, it is necessary to train up to a certain epoch value E with a fixed BS value, check-point, and then continue the training with a different BS value. This would introduce expensive memory access operations. Here, a different approach is followed. With the *GradientTape* mode of TensorFlow, an iterative way of calling the optimization steps has to be implemented, exposing the current batch in each iteration. This way, a big batch can be subdivided and the optimization step can be called on each of the smaller batches individually. Except for the epoch where the switch between BS values occurs, tests have shown that no additional computational overhead is required when using this method, see Alg. 1 for a Python implementation.

4. EXPERIMENTAL RESULTS

4.1. Experimental Setup

The experiments are executed on the Jülich Research on Exascale Cluster Architectures (JURECA) system [17]. Its DC (data-centric) module features 192 accelerated compute nodes, each equipped with four NVIDIA A100 GPUs (with 40 GB high bandwidth memory each). The experiments use 24 nodes (96 GPUs in total) concurrently. The following Python libraries are employed: Horovod/0.23.0, TensorFlow/2.5.0, and Ray/1.8.0. The overall hyperparameter tuning run is launched with Ray Tune. Ray then allocates 4 nodes (16 GPUs) to each trial, within each trial data-parallel

[1] `https://www.ray.io/`

Algorithm 1 Implementation of varying the batch size by batch subdivision.

```
# training iteration loop
for batch, (images, labels) in enumerate(dataset):
    split = 32 # factor of bigger to smaller batch
    # small batch case
    if (epoch < 20):
        # split up the original big batch into
            smaller batches
        images_split = np.array_split(images,
            split)
        labels_split = np.array_split(labels,
            split)
        # call the training step on each of the
            small batches
        for i in range(split):
            loss_value = training_step(
                images_split[i], labels_split[i])
    # big batch case
    else:
        loss_value = training_step(images, labels)
```

training is executed via Horovod. While on NVIDIA GPUs the preferred way of communication is through the NCCL[2] backend, Horovod in combination with Ray only supports the slower Gloo[3] backend, though.

The following hyperparameter ranges are evaluated: $LR \in [10^{-3}, 1.0]$, momentum $M \in [0.0, 0.9]$, nesterov momentum $NM \in \{false, true\}$, and weight decay $WD \in [5 \cdot 10^{-5}, 10^{-1}]$. The selection of hyperparameters for a trial is performed with a random search. All models are trained for $E = 100$ epochs.

The training starts with a small batch size of $BS_{local} = 32$ per GPU ($BS_{global} = 16 \cdot 32 = 512$) and switches to $BS_{local} = 1{,}024$ per GPU ($BS_{global} = 16{,}384$) after $E = 20$. The choice of the BS is motivated by our earlier results [14], where training on the BigEarthNet dataset was stable for $BS_{global} = 512$ but diverged for $BS_{global} = 16{,}384$. Switching BS at $E = 20$ (at 20% of the total training time) gives the optimizer sufficient time to equalize instabilities in early epochs and leads to a yet computationally efficient training with a larger BS for 80% of the time. The application of metrics like the GNS to achieve a more accurate guess of which BS to use were unsuccessful. To evaluate the influence of this BS value switching mechanism on the validation metrics and the computational resource consumption, the same 24 randomly sampled hyperparameter configurations are run once with and without varying BS. We provide the corresponding code in a GitLab repository[4].

[2] https://developer.nvidia.com/nccl

[3] https://github.com/facebookincubator/gloo

[4] https://gitlab.jsc.fz-juelich.de/CoE-RAISE/FZJ/switching_bs

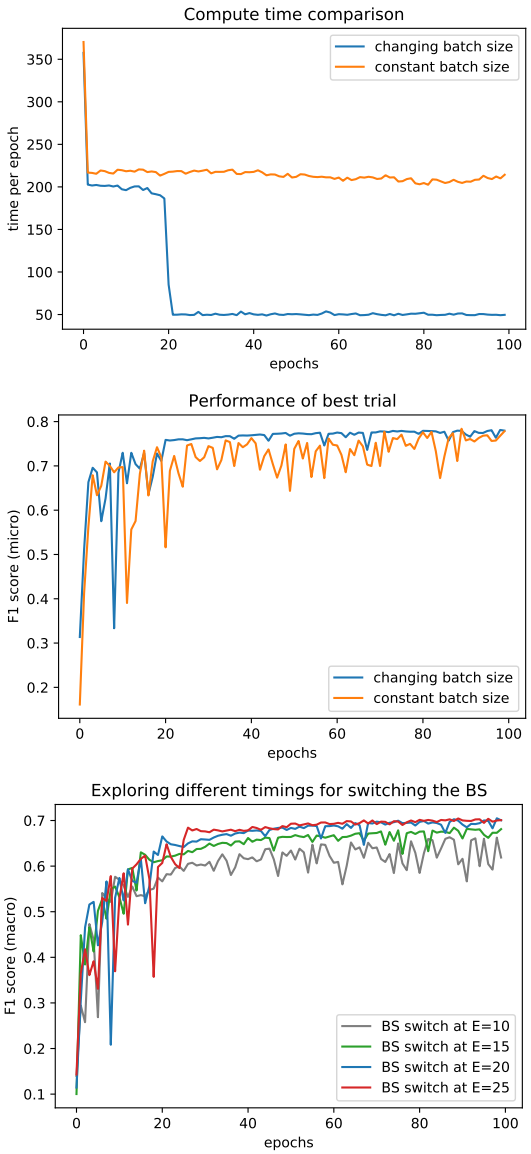

Fig. 1. **Top**: Mean time per epoch. **Center**: Best performing trial. **Bottom**: Comparison of timing for switch.

4.2. Evaluation

A comparison of the time per epoch with and without changing the BS is shown in Fig. 1 (top). For the first $E = 20$ epochs, both methods take about the same time. Once the threshold is reached, the effect of the BS value switch is visible as it is more than 4 times faster. The whole hyperparameter tuning run is about 3 times faster when varying the BS as shown in Tab. 1. In terms of validation F1 scores (a weighted average of precision and recall), the best performing configuration for both approaches is $LR = 0.20735$, $M = 0.26415$, $WD = 5 \cdot 10^{-5}$, and $NM = false$. The scores achieved are in line with our earlier work [14, 2]. As the scores are almost similar for $BS = const.$ and the changing BS method,

Table 1. Runtime of the hyperparameter tuning and accuracy of the best performing run for constant and changing batch size BS, showing accumulated and average trial runtime and validation F1 micro (macro) score.

BS_{global}	total runtime	trial runtime	F1 scores
512	27 hrs	355 mins	0.78 (0.72)
512 $\rightarrow$16,384	10 hrs	136 mins	0.78 (0.70)

the latter does not seem to suffer from the problem of a lower validation accuracy that large batch sizes usually come with.

The graphs in Fig. 1 (center) show the detailed training progress of the best performing trial. Overall, the training with a larger BS value (changing BS method) seems to be smoother than training with a smaller BS (constant BS method) for the whole duration. Furthermore, much of the training progress is already made in the first 20 epochs. Still, the last 80 epochs are necessary to achieve the final F1 scores. For the F1 micro score, the changing BS approach even seems to perform slightly better, but this might be due to the smoothness of its training curve. In the end, both methods converge to a similar F1 score.

Figure 1 (bottom) evaluates the impact of the epoch switching on the training progress. Switching at $E = 10$ results in a lower final F1 macro score while with a switch at $E = 15$, the training performs similar to the original. Switching later in time leads to a better accuracy but also increases the runtime. However, as the graph for a switch at $E = 25$ shows, the gain is only marginal, so the original choice of $E = 20$ seems to be a good trade-off between accuracy and runtime. In all cases, the training never diverges, which indicates that the found hyperparameters seem to stabilize the optimizer even when training with larger batches earlier.

5. CONCLUSIONS

In this manuscript, a method to successfully accelerate the hyperparameter tuning process of a CNN trained with a RS dataset has been presented. Increasing the batch size during training from a smaller one in the beginning (to let the optimizer stabilize) to a larger one (to run data parallel training more efficient) seems to be a promising approach and does not reduce the validation accuracy. Compared to running the hyperparameter tuning with a fixed batch size, a speedup from 27 hours to 10 hours runtime on 96 GPUs has been achieved. While in this study, the focus has been on the BigEarthNet dataset, it would be interesting to see if the approach can also be transferred to other datasets and models.

6. REFERENCES

[1] J. Aschbacher, "ESA's earth observation strategy and Copernicus," in *Satellite Earth Observations and Their Impact on Society and Policy*. Springer, 2017.

[2] D. Coquelin et al., "Evolutionary optimization of neural architectures in remote sensing classification problems," in *IGARSS*. 2021, IEEE.

[3] G. Sumbul et al., "BigEarthNet dataset with a new class-nomenclature for remote sensing image understanding," 2021, arXiv: 2001.06372.

[4] M. Bossard, J. Feranec, and J. Otahel, "CORINE land cover technical guide – Addendum 2000," Tech. Rep. 40, European Environment Agency, Copenhagen, 2000.

[5] S. Kumar et al., "Exploring the limits of concurrency in ML training on Google TPUs," 2021, arXiv: 2011.03641.

[6] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," 2014, arXiv:1404.5997.

[7] Y. You et al., "Large batch training of convolutional networks," 2017, arXiv:1708.03888.

[8] Y. You et al., "Large batch optimization for deep learning: Training bert in 76 minutes," 2020, arXiv: 1904.00962.

[9] S. L Smith et al., "Don't decay the learning rate, increase the batch size," 2017, arXiv: 1711.00489.

[10] S. McCandlish et al., "An empirical model of large-batch training," 2018, arXiv: 1812.06162.

[11] A. Qiao et al., "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," 2021, arXiv: 2008.12260.

[12] L. Mai et al., "KungFu: Making training in distributed machine learning adaptive," in *OSDI 20*. Nov. 2020, pp. 937–954, USENIX Association.

[13] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," 2018, arXiv:1802.05799.

[14] R. Sedona et al., "Scaling up a Multispectral RESNET-50 to 128 GPUs," in *IGARSS*. 2020, IEEE.

[15] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," 2020, arXiv: 1905.11946.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, arXiv: 1512.03385.

[17] P. Thörnig, "JURECA: Data centric and booster modules implementing the Modular Supercomputing Architecture at Jülich Supercomputing Centre," *Journal of large-scale research facilities*, vol. 7, 10 2021.