HPC and Big Data
Assignment - Machine Learning

Florian Wolf - 12393339(UvA)
flocwolf@gmail.com

February 11, 2021

1 Introduction

The last assignment of the course High Performance Computing and big Data (HPC) focuses on Machine Learning (ML). ML is a part of artificial intelligence (AI) and builds applications which learn to represent and predict data [1]. While there are again various subsections of ML, the challenge of this assignment is a supervised learning classification problem. The applied methods include deep learning, parallel and cluster computing techniques. More specific, the task of this assignment is to profile the performance of training a neural network. Therefore we build a model which learns to successfully classify images from the CIFAR10 dataset [2]. For image classification, convolutional neural nets (CNN) yield great results [3]. Such models have a large number of parameters to train, empirically starting at 10^7, thus require high computational resources. We train the image classifier on the LISA partition of the cluster computer facility SURF [4]. Further we compare the computation time when training a model in parallel fashion on a increasing number of cores.

2 Implementation

To profile the performance of training a neural network on multiple cores/processors/workers, we first need to implement a classifier and a working deep learning pipeline. Deep learning can be accelerated by parallelizing the training by dividing the workload on multiple workers. This does not only speed up deep learning but also makes it attractive for the enhancement of other Big Data applications such as large scale numerical simulations [5]. The implementation is purely in Python language and is executed on LISA through bash scripts. For the implementation of image classifier model, we rely on the ML library Tensorflow [6]. The management and distribution of parallel training on different threats is managed by the Horovod library [7].

Bonus part of the assignment is to use a pre-trained state of the art image classifier and use transfer learning to adopt it to our specific task. Thus, we start by introducing our self-build model and then the implementation of transfer learning for the two models EfficientNet b0 and ResNet50 as well a final fine-tune. We partly follow the implementation example for transfer learning given on the Keras site [8]. Instead of freezing the pre-trained layers, as indicated, we set `effNet.trainable = True` and similar for `resNet`. This allows to retrain the full model and adjust its parameters to the new dataset.

The custom model consists of two convolution layers, followed by max-pooling, dropout and flattening of the image data. The hidden state is then passed through two fully-connected and one dropout layer. The resulting vector is activated by the softmax function and represents the categorical probability of the input image for each of the 10 classes.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Keras is a high-level interface for Tensorflow, which, between many others features, provides the user with pre-trained models [9]. We use this feature to implement EfficientNet B0 [10]. The model comes with pre-trained parameters, yet on a different dataset. Therefore we add an input layer, which adopts the model-input to the image size of the CIFAR10 dataset. We repaet this step for the model-putput with a fully-connected layer, in order to match the length of the final prediction vector to the number of classes `num_classes`= 15.

```
effNet = tf.keras.applications.EfficientNetB0(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(32, 32, 3),
    classes=1000,
    classifier_activation=None,
    )
effNet.trainable = True
inputs = Input(shape=(32, 32, 3))
x = effNet(inputs, training=True)
x = GlobalAveragePooling2D()(x)
outputs = Dense(num_classes, activation='softmax')(x)
model = Model(inputs, outputs)
```

A similar implementation for ResNet50 as base model [11]:

```
resNet = tf.keras.applications.ResNet50(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(32, 32, 3),
    classes=1000,
    classifier_activation=None,
    )
```

In table 1 the parameter count for the different models is displayed. We see, that the pre-trained models with unfrozen parameters have a much higher parameter count. Since the time needed to train a model is linearly correlated with the models parameter count, we choose a normalizing factor α_p which allows us to compare training time between different models.

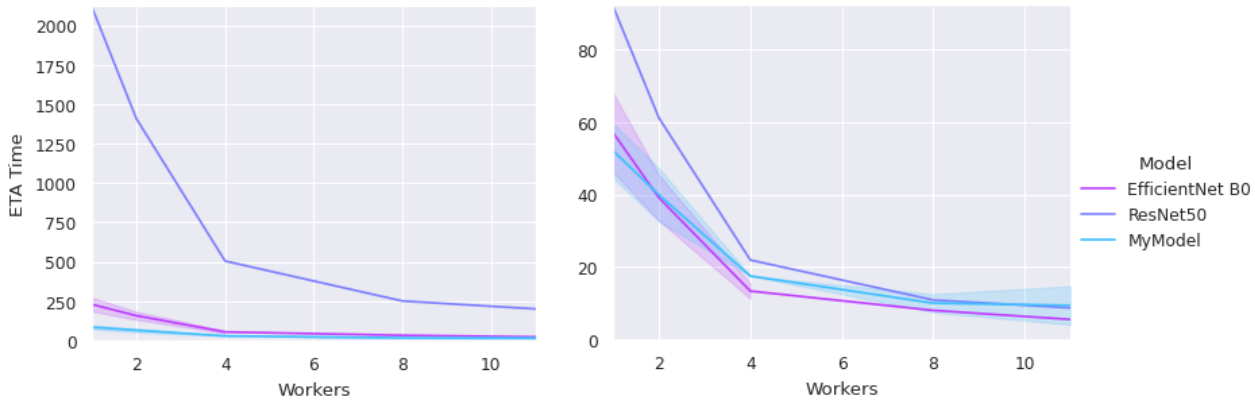
	Custom model	EfficientNet B0	ResNet50
total	1,626,442	4,062,381	23,628,902
trainable	1,626,442	4,020,358	23,555,082
non-trainable	0	42,023	53,120
α_p	1.6	4.0	23.5

Table 1: Parameter counts and α_p for the image classifiers.

3 Parallel Machine Learning

In this section the results of our experiments are presented and compared. We train the three different models under similar conditions. The implementation allows parallel computing on up to 11 cores. Training of deep neural networks is conventionally done on graphics processing units (GPU), which allow numerous parallel computations. While GPU nodes are available on LISA, for this assignment access has only be granted for CPU nodes. Each node on LISA has between $16-24$ CPUs. When reserving a node to process a job, the number of tasks n_t can be specified. The cluster then allocates enough nodes, to provide n_t CPUs. Using Message Passing Interface (MPI) we can run the distribute the training on the available CPUs. The Python library Horovod recognizes the

number of available CPUs and automatically creates the matching amount of workers. Furthermore it adjusts the learning rate and epochs to the number of workers. Since the workload is divided per a number of workers, we expected a speedup factor of $s = n_t$ per additional node. The three models are trained using $n_t \in \{1, 2, 4, 8, 11\}$ within a maximum timeframe of $t_{max} = 150min$ rounded to full minutes. Figure 1 shows the computation time over the number of workers per model. The choice of $tmax$ is by far insufficient for training of ResNet50 on a low number of CPUs, the presented times are therefore the Estimated Time of Arrival (ETA), which is predicted by Tensorflow at the beginning of each epoch. The epoch ETA is multiplied by the total number of epochs and averaged over all estimates, with exception of the first, which has shown to be imprecise. To normalize the impact of the parameter count, the ETA is divided by its corresponding α_p. Figure 1 shows both the ETA and the normalized ETA for the 3 classifiers over the number of workers.



(a) ETA.

Figure 1: Estimated training time in minutes over number of workers.

Within the job script the processing time is logged using the Linux command `time`, which prints the returns the three different time metrics `real`, `user`, `system` [12]. The presented time is the wall clock time for passed while training the model. The same script is used to run all experiments, passing the variable n_t to request the right number of CPUs. In theory, the use of more CPUs allows the work to be distribution to more workers, implying higher parallelism and thus speedup of the training. Yet, in reality more processes are executed around the the execution of the Python script, e.g. the loading of modules moving of data. These processes are not accelerated. For the sake of completeness, figure 2 presents the wall clock time for the training of the different models. Experiments with lower number of workers and higher parameter count exceed the maximum run-time t_{max}, for which reason we found the ETA to be more informative.

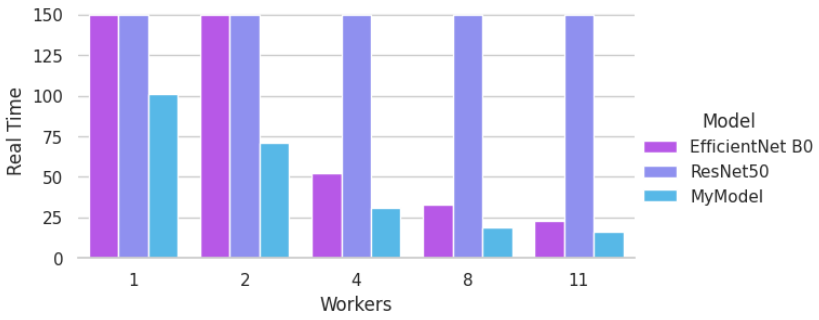
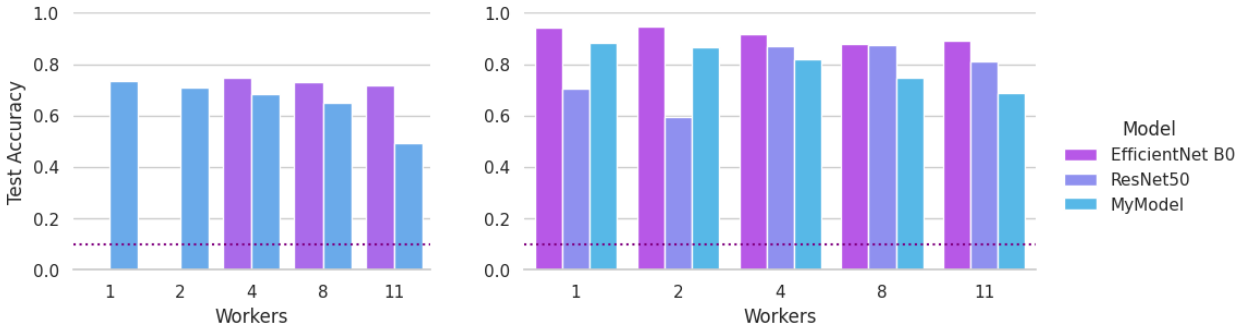


Figure 2: Wall clock run-time of the training over number of workers. Results are capped at $150min$.

In both figures 1 and 2 we observe a decline in computation time and therefore a clear speedup with increasing number of workers. Figure 1 indicates an inverse logarithmic drop in estimated

processing time, whereas in figure 2 the decline seems almost linear for EfficientNet and the custom model with and estimated speedup of $s \approx 0.7n_t$. When normalizing the estimated processing time over the parameter count, a similar behaviour for all three models is observed.

To get a deeper insight in the impact of parallel on deep learning, we further compare the prediction accuracy of the different models over number of workers. For the same reason as in the previous experiment, some results are limited by t_{max}, meaning the model did not complete the training and misses the final evaluation on the unseen test set. Thus, we include the last recorded accuracy score on the train set. For this experiment the baseline accuracy is a random guess. Considering that we predict on 10 classes, the baseline accuracy of guessing the right class without prior knowledge is 0.1. The accuracy scores on both test and train set are presented in figure 3.



(a) Test set.

(b) Train set.

Figure 3: Train and test accuracy of the different models over number of workers. Dotted line indicates the random baseline.

Eventough the pre-trained models do not finish training with a low number of workers, they outperform the custom model on the train set, which gives reason to assume that they also perform better on the test set. With 4 or more workers, EfficientNet consistently scores higher than the custom model. ResNet seems to gain momentum with more workers, yet, in no experiment it completes training.

Finally addressing the elephant in the room, we observe a clear decline of the accuracy scores with increasing number of workers. A significant drop in classification performance for all models on both train and test set occurs with increasing n_t. While Horovod might provide deep learning with the perks of parallel computing, it seems to come with a price. Possible explanations for the observed phenomenon are discussed in the following section.

4 Conclusion

Knowledge is a constantly transforming and growing matter, today's science will be without a doubt outdated and overwritten in a few years and as Plato said, all I know, is that I know nothing [13]. Thus, rather than providing the absolute answers, we aim at asking the right questions and proposing new hypotheses based on our gained insights, allowing our curiosity to strive for more knowledge.

What impact has parallel computing on the model's training speedup s?

In this regard the experiments reveal 4 insights. The required time for training a model is reduced when adding workers. The increase of s is not linear to the increase of number of workers n_t. While the training time differs per model, s is similar, indicating no correlation between s and the number of parameters. More than 8 workers does not reduce the training time any further, which can be related to $e = 15$, the number of epochs. If we divide e by the n_t and take the next higher integer, it results in 2 epochs per worker for both experiments with $n_t \in \{8, 11\}$. We assume that the remaining cores stand by unused, bringing up the question: is it better to adjust n_t or e?

Yet, the most prominent question is: *Why does the performance decrease with higher n_t?*

Since during each epoch a core trains a separate model, we state the hypothesis that it is be harder for the combined model to converge to the global optimum. We wonder, what the right

ratio of epochs would be between two different n_t, to reach the same accuracy score? For a fact, we can say that none of the models completely converges within 15 epochs.

Finally we ask the question: *Does Higher complexity yield better performance and did transfer learning even work?*

The results give reason to assume that ResNet50 is unproportionaly complex for this use case. EfficientNet might be a better fit in terms of adequate complexity, as it scores best in all experiments. The custom model can be seen as a Vanilla image classifier with adequate structure and room for improvement. For the second part of the question we should take a glance at the data sets. Both models were pre-trained on Imagenet [14], a dataset with a much higher number of data points and classes compared to CIFAR10 [2], resulting in a higher variance of the data distribution. Combining this insight with the fact that both models started the training with an accuracy score similar to the random-guess baseline. we state the hypothesis, that transfer learning was unsuccessful and pre-training unnecessary, implying that the same untrained models would score similar.

An unsolved mystery to the curiosity of the author stays the vast difference between `real` and `user` time.

List of Figures

List of Tables

References

[1] IBM. Ibm cloud education. `https://www.ibm.com/cloud/learn/education`, 2 2021.

[2] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] Song Feng, ETH Torsten Hoefler, Switzerland Jessy Li, Damian Podareanu, Qifan Pu, Judy Qiu, Vikram Saletore, Mikhail E Smorkalov, and Jordi Torres. Valeriu codreanu, surfsara, netherlands ian foster, uchicago & anl, usa zhao zhang, tacc, usa.

[5] Caspar van Leeuwen, Damian Podareanu, Valeriu Codreanu, Maxwell X Cai, Axel Berg, Simon Portegies Zwart, Robin Stoffer, Menno Veerman, Chiel van Heerwaarden, Sydney Otten, et al. Deep-learning enhancement of large scale numerical simulations. *arXiv preprint arXiv:2004.03454*, 2020.

[6] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[7] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[8] Keras. Transfer learning & fine-tuning. `https://keras.io/guides/transfer_learning/`, February 2021.

[9] Nikhil Ketkar. Introduction to keras. In *Deep learning with Python*, pages 97–111. Springer, 2017.

[10] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[11] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition*, 90:119–133, 2019.

[12] Stack Overflow. What do 'real', 'user' and 'sys' mean in the output of time. `https://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1`, February 2021.

[13] Edith Hamilton, Huntington Cairns, Lane Cooper, et al. *The collected dialogues of Plato*. Princeton University Press, 1961.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.