ISBN: 978-953-8246-26-5

Determining the Influence of Hardware on the Execution Times of Trained Machine Learning Models

Sandi Baressi Šegota [1*], Matko Glučina [2], Daniel Štifanić [1], Jelena Musulin [1], Ivan Lorencin [1], Nikola Anđelić [1], Zlatan Car [1]

[1] Faculty of Engineering - University of Rijeka, Vukovarska 58, 51000 Rijeka, Croatia, sbaressisegota@riteh.hr, dstifanic@riteh.hr, jmusulin@riteh.hr, ilorencin@riteh.hr, nandelic@riteh.hr, car@riteh.hr
[2] University of Rijeka, Braće Mažuranića 10, 51000 Rijeka, Croatia, matko.glucina@riteh.hr.

Abstract: While many discussions and observations have been made regarding the execution times of machine learning (ML) model training, not many researchers have shown concern regarding the execution times of trained models when the model are applied for inference. In this paper, the researchers observe the execution times of a realistic hybrid system consisting of a YOLOv3 based detection model and two classification models based on VGG16 and VGG19 convolutional neural networks. The provided hybrid model is tested on five different hardware configurations – single core CPU execution, eight thread CPU execution, 48 thread CPU execution, single GOU execution and five GPU execution. The goal is to compare the execution times and determine the user satisfaction and ease of use on standard consumer hardware, readily available to the end users of the project, in comparison to a high performance computing architectures. The results show that the best performing architecture is a single GPU, but even the slowest solution (single core CPU), does not show an extremely slow time when inference is applied.

Keywords: artificial intelligence, convolutional neural networks, execution timing, high performance computing, hybrid systems, machine learning, model inference

1. introduction

Training times of the machine learning (ML) models are oft discussed [1], but the training processes are often offloaded to high performance computing machines [2, 3]. When it comes to model integration, the trained models will commonly be executed on regular user-accessible machines with comparatively poorer performance. For this reason, this paper investigates the performance of the realistic developed classification models. The models are based on the convolutional neural networks, developed as a hybrid system for detection and classification of the underground infrastructure. The analysis aims to determine the performances of realistic classification models to determine the usability and user satisfaction when using developed data-driven artificial intelligence-based models. For this reason, five different device sets are used, selected in order to compare the regular PC an end user may have access to (1 CPU, 8 CPUs, 1 GPU) to a HPC environment and determining images (48 CPUs, 5 GPUs).

2. Methodology

This section will first describe the used models, followed by the process of timing.

2.1. Used models

The data used for model training are collected as part of the CEKOM SmartCity.4DII project and consists of ground penetrating radar (GPR) imagery with the annotation data provided by the domain partners, who are the infrastructure owners. Three separate AI models are developed – a model for detecting the underground infrastructure location based on the YOLOv3 [4] algorithm, and two models for the material and width classification of the detected infrastructure – based on VGG16 [5] and VGG19 [6] convolutional neural network (CNN) architectures. The models have been trained to achieve a satisfactory model performance. The models are connected within a specific script, in which the output of the detection algorithm serves as the input of both classification models.

2.2. Timing

The developed hybrid algorithm system is trained using the main node of the Z4 HPC cluster [7]. The system was selected for two reasons: the availability of all the devices that wished to be tested, and high enough amounts of memory and input-output (IO) bandwidth where it will not be bottlenecking the performance. Using all systems for tests guarantees that no biases towards execution time are introduced due to the operating system or system libraries. Timing is performed using the Linux time program for time measurements [8]. The time command returns the real, system and user times after the given command has been executed. The user and system times refer to the times the CPU instructions are ran outside and within the kernel space. The real time refers to the so-called wall time – i.e. the time elapsed for the user while the execution is in progress.

Five different device sets are used for the performance measurement. CPU execution is used in three different modes – with a single threaded execution, and a multi-threaded execution with 8 or 48 threads respectively. In addition, the model inference is also run using GPUs – with a single GPU and five GPUs. This was selected to determine the needs and performance that an end user may expect. All the executions are run with niceness set to -20 (highest priority) [9]. The models are executed on 47 different images (a single microlocation images of the underground infrastructure).

The limitations are set using two commands. For setting CPUs, the taskset command is used [10]. This command allows us to set the number of threads the command executed through it (in this instance, the inference model) will use. For setting GPUs, the environmental variable CUDA_VISIBLE_DEVICES [11] is set to either "0" (for a single GPU) or "0,1,2,3,4" (to use all five GPUs). When

the CPU execution is performed, this variable is set to "-1". This environmental variable controls which CUDA capable devices are visible to the code that is executed in the shell.

The hardware used for the timing experiment is:

- CPU – Intel Xeon Gold 6240R with 24 Cores/48 Threads @2.4 GHz (boost up to 4.00 GHz), 35.75 MB of Cache, and
- GPU – NVIDIA Quadro RTX 6000 with 4608 CUDA cores and 24 GB of RAM.

In addition to the above, the server has 768 GB of RAM and all the models are run from Intel D3-S4510 240GB SSD, in RAID 1 [12]. The operating system that is installed on the machine is Linux Ubuntu 18.04 with the kernel version 4.15.0-176.

The execution times of the code can be split into library and model loading time and the poor execution (model prediction) times. Both times are presented, by measuring the execution time of the total program, and the execution time of just the model and module loading – enabling us to determine the execution times as a difference. These results are presented in the following section.

3. Results and discussion

The first table shows the obtained timings for each of the tested devices – focusing on the total execution times. As mentioned, the focus should be given on the real time, as this is the time the user experiences when running the model.

Table 1. *The timings for each of the tested devices*

Device	Real time [s]	User time [s]	System time [s]
1 CPU	117.309	113.514	3.645
8 CPU	103.032	270.567	120.693
48 CPU	50.613	872.58	237.702
1 GPU	33.351	37.632	21.206
5 GPU	41.058	40.235	27.511

The data in question shows the significant improvement in the total execution times, where the single GPU has the lowest execution time. The time for 5 GPUs is somewhat higher – which is probably caused by longer loading times. Observing execution times, we can see that the system times significantly increase when multiple CPUs are used due to the data transfer and execution handling. User times also increase due each of the individual threads being counted separately. Higher amount of CPUs shows the lowering of the times, but it is not significant in regards to the timings.

Table 2 and Figure 2 show that the loading times are equal for the first two CPU cases, but lower when multiple CPUs are used, probably due to multiple cache chips being used for loading the data. In case of GPUs, the real times show a significant increase due to the fact that data needs to be separated and transferred to multiple GPUs.

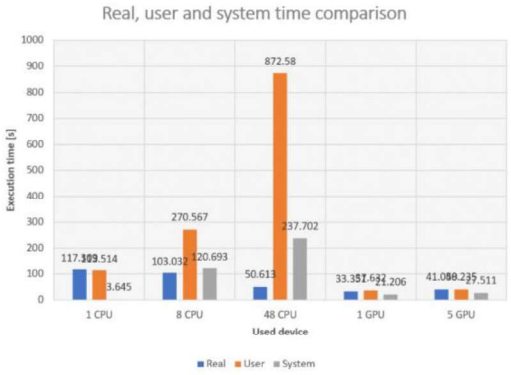

Figure 1. *Visualization of all the total measured times.*

Table 2. *The loading timings for each of the tested devices*

Device	Real time [s]	User time [s]	System time [s]
1 CPU	25.471	9.634	3.189
8 CPU	25.72	9.749	3.194
48 CPU	14.48	16.333	16.323
1 GPU	21.752	17.98	13.598
5 GPU	33.696	27.828	17.84

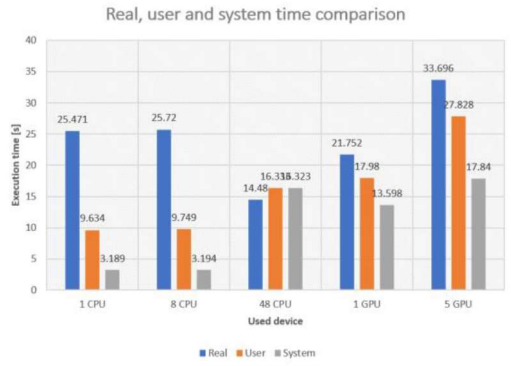

Figure 2. *Visualization of all the measured loading times.*

Table 3 and Figure 3 demonstrate the pure execution times, obtained as a difference between the total timings and the loading times. This is meant to show the time the models actually spend executing on the accelerators. This shows that the 5 GPUs are significantly the fastest in comparison to the other devices.

Table 3. *The execution timings for each of the tested devices*

Device	Real time [s]	User time [s]	System time [s]
1 CPU	91.838	103.88	0.456
8 CPU	77.312	260.818	117.499
48 CPU	36.133	856.247	221.379
1 GPU	11.599	19.652	7.608
5 GPU	7.362	12.407	9.671

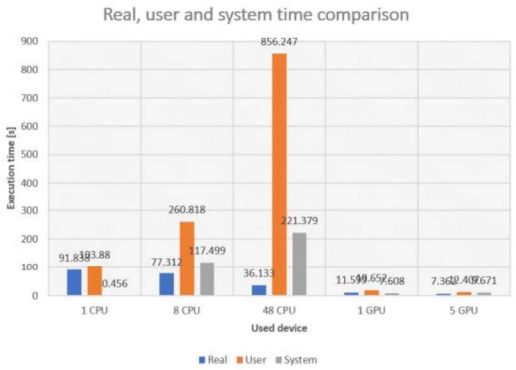

Figure 3. *Visualization of all the determined execution times.*

4. Conclusion

The data in question shows that the fastest model execution times, when observing the total real times, which is what a user would experience, is achieved when using the GPU. Still, the cost to the end user needs to be considered. Taking the cost into the account, it van be seen that all of the used devices provide satisfactory times, with a maximum total real time below 2 minutes.

Acknowledgments

This research has been (partly) supported by the CEEPUS network CIII-HR-0108, European Regional Development Fund under the grant KK.01.1.1.01.0009 (DATACROSS), project CEKOM under the grant KK.01.2.2.03.0004, Erasmus+ project WICT under the grant 2021-1-HR01-KA220-HED-000031177, University of Rijeka scientific grant uniri-tehnic-18-275-1447.

References

[1] Seiffert, Chris, et al. "RUSBoost: Improving classification performance when training data is skewed." *2008 19th international conference on pattern recognition*. IEEE, 2008.
[2] Car, Zlatan, et al. "Modeling the spread of COVID-19 infection using a multilayer perceptron." *Computational and mathematical methods in medicine* 2020 (2020)..
[3] Kahle, James A., Jaime Moreno, and Dan Dreps. "2.1 Summit and Sierra: designing AI/HPC supercomputers." *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019.
[4] Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." *arXiv preprint arXiv:1804.02767* (2018).
[5] Lorencin, Ivan, et al. "On urinary bladder cancer diagnosis: Utilization of deep convolutional generative adversarial networks for data augmentation." *Biology* 10.3 (2021): 175.
[6] Lorencin, Ivan, et al. "Automatic evaluation of the lung condition of COVID-19 patients using X-ray images and convolutional neural networks." Journal of Personalized Medicine 11.1 (2021): 28.
[7] Musulin, Jelena, et al. "Multiclass Classification of Oral Squamous Cell Carcinoma." Ri-STEM-2021 (2021): 7.
[8] Linux manual page, "time(1)", https://man7.org/linux/man-pages/man1/time.1.html
[9] Jonk, Ruben, et al. "Timing prediction for service-based applications mapped on linux-based multi-core platforms." 2018 21st Euromicro Conference on Digital System Design (DSD). IEEE, 2018.
[10] Linux manual page, "timeset(1)", https://man7.org/linux/man-pages/man1/taskset.1.html
[11] Šego Sanders, Jason, and Edward Kandrot. CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional, 2010.ta,
[12] Sandi Baressi, et al. "A Brief Note on the Influence of Storage Choices on Machine Learning Algorithm Training Times." Ri-STEM-2021 (2021): 19.