A Lightweight Execution Manager for Training TensorFlow Models under the Slurm Queuing System

Marcos Lupión[1], Nicolás C. Cruz[2], Felipe Romero[1], Juan F. Sanjuan[1], Pilar M. Ortigosa[1]

[1]Department of Informatics, ceiA3, University of Almería, Sacramento Road, 04120 Almería, Spain, marcoslupion@ual.es, lfromero@ual.es, jsanjuan@ual.es, ortigosa@ual.es

[2]Department of Computer Engineering, Automatic and Robotics, University of Granada, Periodista Daniel Saucedo Aranda Street, 18071 Granada, Spain, ncalvocruz@ugr.es

Abstract: Artificial neural networks currently represent the flagship of Machine Learning and have reached multiple fields alongside Computer Science. This kind of computational model generally needs massive amounts of data and high-performance computing resources. The availability of graphical processing units is especially relevant. Thus, only institutional computing platforms and clusters satisfy such a high demand for computational power and storage resources. These systems rely on resource managers capable of handling multiple users and computing resources. However, the users interested in working with artificial neural networks, especially those without a background in Computer Engineering, might not master system administration. For them, planning their executions within the framework of a resource manager focused on high-performance computing is problematic. This work presents S-TFManager, an easy-to-use open-source web manager for launching and controlling the execution of TensorFlow models consisting of artificial neural networks in a heterogeneous cluster with a Slurm queuing system. Both TensorFlow and Slurm are arguably the most extended tools in their respective fields, so the proposed tool is of public interest. The tool, written in Python, includes built-in batching and visualization capabilities, and its simplicity makes it easy to extend.

Keywords: Machine Learning; TensorFlow; Slurm; Resource Management

1 Introduction

Artificial Neural Networks (ANNs) were conceived to mimic the human brain in learning and solving difficult tasks [1, 22]. In contrast to plain symbolic programming, ANNs can identify, learn, and extrapolate patterns from data, as

human beings do, without a predefined set of rules. Since the first model in the late 1950s, called Perceptron, and its simple classification and regression capabilities, numerous models and architectures have been proposed for many different tasks [7]. Consequently, one can find successful applications of this bio-inspired computing approach in a vast scope. It covers from cybersecurity [10] to energy management [23], including image processing, speech recognition, and automatic translation, among many others [1, 15, 22].

The evolution of ANNs has relied on designing new mathematical models for artificial neurons and operators, their combination, as well as methods for tuning their parameters (training) and hyperparameters. This broad space of design has even resulted in the field of Neural Architecture Search (NAS), which deals with automating the design of the most appropriate ANN for a certain task through optimization [17, 25]. Along with the evolution of theoretical proposals, numerous software frameworks have arisen to support their deployment. Some of them are PyTorch [14] and TensorFlow [8, 17, 24], both open source, which provide users with the fundamental building blocks for designing complex ANNs with state-of-the-art components.

Regardless, advances in High-Performance Computing (HPC) have been critical for implementing such computationally demanding models, especially when concatenating sets of (layers) neurons and operations such as convolutions. In this regard, using Graphical Processing Units (GPUs) as accelerators due to their direct compatibility with the computations involved, i.e., matrix multiplications, is possibly the most relevant milestone [17]. Without the support of modern HPC hardware, building and deploying the sophisticated models currently used becomes infeasible [3]. This need for cutting-edge hardware resources makes Cloud services, such as Google Colab [8], a cost-effective and flexible solution [11]. However, many professionals and researchers opt for private clusters from universities and companies, especially when they deal with confidential data. These clusters serve multiple users concurrently and provide access too many CPU and GPU cores. Thus, they generally feature resource managers and queuing systems, such as the widespread Slurm [12, 20].

At the same time, as a consequence of the broad space of design mentioned, the standard workflow for designing ANNs involves trying multiple architectures and configurations [17, 20, 25]. Accordingly, the experts in charge of building models based on ANNs must plan numerous executions in shared HPC environments. This situation involves extensive use of scripting and general-purpose queuing systems, such as the aforementioned Slurm, which is potentially tedious. Besides, since the popularity of ANNs has brought different professional profiles to this field [6], some may find it difficult to collaborate. For example, the recent work in [9] deals with the relevance of building descriptive and easy-to-use dashboards for using machine learning models on the side of medical users. Similarly, not all developers have a background in Computer Engineering [6]. Hence, scripting and system administration tools can burden their work. Therefore, different tools have

been proposed to simplify the management of software execution relying on the industry standards for ANN building and deployment, such as the referred TensorFlow, in HPC environments [3, 20].

The use of Jupyter Notebooks is arguably the simplest option to provide non-expert users with a kind of development environment to try different approaches, and to record and plot results. However, they lack advanced asynchronous features to plan and manage multiple executions in HPC environments. Therefore, more advanced tools exist for this purpose. For example, the work in [3] focuses on the resources required for using built models, known as inference. The solution built, called ROMA, is an open-source TensorFlow extension equipped with scheduling heuristics and control-theory ideas [2]. It relies on containers and manages the execution of different models in a heterogeneous platform combining multiple CPUs and GPUs while trying to meet user-defined constraints on the execution time.

Nevertheless, most works focus on training rather than inference, i.e., building ANN models by tuning their parameters, as it is significantly more computationally demanding. For example, the proposal in [17] designs an open-source service that manages GPUs and CPUs while building ANNs. Cleverly, instead of redefining a queuing system, it exploits an underlying Slurm installation. Unfortunately, the scope of that work is on automating the design of ANNs, considering different architectures. The requests made are controlled by an optimization algorithm, and it cannot be directly used by human experts. Conversely, our focus is on human users, and several relevant tools exist for this purpose.

One of them is the open-source tool proposed in [20], TensorHive, which targets human users interested in managing machine learning workloads. It allows them to reserve and monitor resources. It also lets them run jobs in a rich web interface that supports interactive work and leading ANN environments, i.e., TensorFlow and PyTorch. Another one is Auto-Keras [13], which simplifies the automation of model selection and training. However, it does not support distributed environments, making it unsuitable for high-performance computing scenarios. There also exists H2O.ai [16], which provides extensive preprocessing and model optimization capabilities but relies heavily on specific distributed implementations. This approach can reduce flexibility and adaptability in diverse computing setups. Another option is Google Cloud AutoML [4]. It is a proprietary solution that offers cloud-based scalability and ease of use. However, its dependency on Google's infrastructure and lack of offline usability limits its application in environments where open-source or on-premises solutions are preferred. Finally, distributed frameworks such as Horovod [21] and Ray [19], while powerful, aim at highly technical users and lack the intuitive interfaces necessary for broader accessibility.

This work also presents an open-source tool, S-TFManager, aimed at human users sharing a cluster for building ANNs. Nevertheless, it differentiates itself from the previous works by providing a mid-term solution between offering a rich set of features at the expense of difficulty of use and modification (TensorHive, Ray, H2O.ai), limited options (Auto-Keras, Horovod), and the lack of accessibility linked to closed-source tools (Google Cloud AutoML). Our proposal follows a "Less is more approach" (LIMA) design philosophy [18] that replaces generality and redundant use cases with a context simpler to deploy, encompass, and extend. More specifically, it offers a self-contained and minimalist web interface that provides users with a convenient environment for launching ANN training jobs using TensorFlow. Resource management is directly outsourced by an existing Slurm installation, and no overlapping with standard monitoring tools, such as Munin [5], occurs either. In return, the proposal includes built-in visualization and systematic hyperparameter exploration capabilities.

The rest of this paper is structured as follows: Section 2 describes the technical design of the proposed execution manager and provides the reader with a link to the source code. Section 3 shows several practical cases where the developed tool exhibits its capabilities and supports the user. Finally, the last section contains the conclusions and states the future work.

2 Developed Solution

This section describes the core components of the proposed tool. First, Section 2.1 contains an overview of its technical requirements. Then, Section 2.2 explains the general architecture of the solution and how its components interact. After that, the main components, i.e., the database (Section 2.3), the interaction between the S-TFManager and Slurm (Section 2.4), and the monitoring script (Section 2.5), are described. Finally, Section 2.6 covers the deployment of the tool, including a basic troubleshooting guideline.

2.1 Technical Overview

2.1.1 License and Distribution

The software package is publicly available under the Creative Commons Attribution-Noncommercial 4.0. Accordingly, it can be freely used, distributed, and modified by anyone without commercial purposes and referring to this source.

Specifically, the tool can be found at the following code repository: https://gitlab.hpca.ual.es/marcoslupion/s-tfmanager. In addition to the source code, the repository contains a detailed explanation of the main features of the tool with

graphical examples, installation and execution instructions, and contact information. However, the most relevant aspects are summarized later in this section.

It is noteworthy that, as usual with open-source projects, even though the tool is well intentioned and has been successfully tested, it has no warranty of stability or correctness. This statement makes the authors not liable for any problem caused by unexpected malfunctioning.

2.1.2 Requirements

S-TFManager consists of Python and shell scripts with HTML parts and JSON files. Thus, the main requirement for running it is having a Python installation in a Linux-like environment. Among the necessary libraries, Flask[1] has the highest requirements by expecting Python 3.8 or higher, so it defines the minimum version supported. Also, the tool uses Slurm to submit jobs. Since its *--gres* option is necessary, the Slurm version must be either 2.6 or higher. Other requirements result from needing a TensorFlow installation, including the appropriate cuDNN, cudatoolkit, keras, and jinja. The interested reader can check the *requirements.txt* file in the code repository for further information.

2.1.3 Supported Platforms

The platform where the tool has been tested uses OpenHPC 2, based on CentOS 8. It should be equivalent to Ubuntu 20.04 or similar. However, as detailed above, any Linux-like system with the appropriate versions of Slurm, TensorFlow, and Python should be compatible. The same occurs with the referred software packages and libraries. Although we have listed in the repository the specific versions for which S-TFManager has been developed and tested, similar versions, especially newer ones, should be valid as long as they keep backward compatibility. Regardless, the preferred installation process relies on creating a virtual environment, which allows staying with the checked versions. More information about this process is provided in Section 2.6. The interested reader can also check the repository for further details.

2.2 General Architecture

S-TFManager consists of several open-source software components that make it easy to integrate into Slurm-managed clusters. Figure 1 depicts how users and the different elements of the tool interact. The core element of the tool is a REST API developed in Flask. It is in charge of providing the web dashboard and launching the Slurm jobs in the cluster. Users can interact with the API through the web

[1] https://flask.palletsprojects.com/en/3.0.x

dashboard, which allows uploading the training scripts, defining the hyperparameters to consider, and monitoring their performance in real-time. Nonetheless, the API can be directly accessed, which makes it accessible programmatically.

The API manages the system database, which consists of raw JSON files, and interacts with the Slurm queuing system. Depending on the availability of GPU nodes, the training requests may be executed or enqueued. For the same user and configuration of tasks, the user perceives them to run in FIFO order. However, the queuing strategy depends on the system's configuration. By default, Slurm follows its backfill[2] scheduling paradigm. According to it, Slurm tries to maximize the system's throughput by prioritizing jobs and considering variables such as the possibility of job preemption and the availability of resources. The interested reader can check the documentation of Slurm for further information.

Concurrently, an auxiliary thread tracks the execution of scripts and provides information about their status and performance metrics. Thus, this element represents the keystone of the claimed integrated monitoring support. Along with the API, it runs in a CPU-only node so as not to interfere with the computational resources aimed at ANN training.

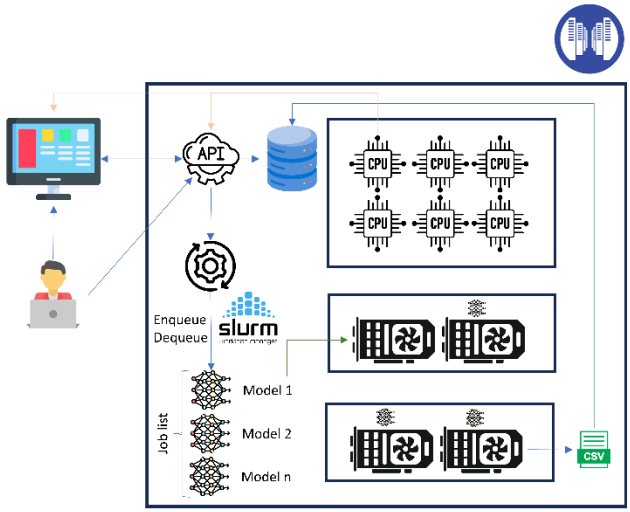

Figure 1
Architecture of the tool

[2] https://slurm.schedmd.com/sched_config.html#backfill

2.3 Structure of the Database

As mentioned, the API relies on a database of JSON files to track pending, running, and failed executions. The standard approach might have been using SQLite instead. However, the direct use of JSON files was considered easier to integrate, as it avoids a relational database management system and its related queries.

The database contains two kinds of data types: jobs and tasks. A job stores the information of the training name, description, and the associated training script. The task is the particular training that is sent as a Slurm task. It references the job and also stores the information of the hyperparameters (batch size, learning rate, number of epochs, early stopping) of the execution. Furthermore, the status of the task is stored, as well as the submission, start, and finalization date and time.

2.4 Slurm Management

The proposed solution mainly focuses on simplifying the interaction of users with the Slurm queuing system for training ANNs. The latter procedure may involve trying multiple parameters, such as batch sizes and learning rates. Hence, when the API receives a list of potential values for variables, it automatically creates their combinations with the rest. Each permutation will result in a new training task. The API also generates the appropriate launch command and submits it to Slurm using the Process Python module.

Slurm commands include the parameters shown in Code 1. The *--partition* parameter specifies the nodes where the training can run, as they feature GPUs.

The *--gres*[3] parameter states that the task requires one GPU to be executed. The *--ntasks-per-node* parameter indicates that the script will launch a single task in the assigned node. For example, in the cluster used for experimentation, the gpu_volta partition has two nodes, each containing two GPUs. The script in Code 1 will result in up to four processes running concurrently, two at each node, and one for each GPU.

Code 1
Slurm launching script

```
1. # SBATCH --partition=gpu_volta
2. # SBATCH --gres=gpu:1
3. # SBATCH --ntasks-per-node=1
```

[3] https://slurm.schedmd.com/gres.htm

2.5 Monitoring Script

Since the API ultimately delegates task managing to Slurm asynchronously, the tool needs an extra component to track the requests made. For this purpose, as mentioned, it is equipped with a monitoring script as well. This script contains an infinite loop checking the different tasks every minute. When a task is initialized, it creates a file called START, which indicates that the task started. When it finishes, it creates another file called END. The monitoring script registers the initialization and finalization times, as well as the current status in the database. Besides, while every training occurs, the monitoring script creates and maintains a figure that shows its evolution. It also registers the lowest loss value and the epoch when it takes place, which is valuable information for the users.

2.6 Tool Installation, Initialization, and Troubleshooting

The installation process for S-TFManager has been designed to be as straightforward as possible while ensuring compatibility with the necessary dependencies and configurations.

The installation procedure should start by checking the requirements previously summarized and detailed in the code repository. It will also be necessary to clone the repository or download it as a zip file to a preferred directory.

Then, the next step is to create an isolated virtual environment[4]. This approach is widespread as users are not generally allowed to install software packages globally in a shared cluster. Accordingly, it will allow the user to install the dependencies listed in the above-mentioned *requirements.txt* file. These include Flask, TensorFlow, and other auxiliary libraries needed for execution. Code 2 contains the commands for creating a virtual environment (first line) and launching the installation of the software packages listed in the *requirements.txt* file (second line).

Code 2

Commands for virtual environment creation and installation of packages in *requirements.txt*

```
1. python -m venv {path}
2. pip -r install requirements.txt
```

After that, the user must ensure that the configuration files, such as the JSON settings, are correctly updated to reflect the system's specifics (e.g., paths to Slurm and TensorFlow installations, job submission parameters, etc.). Once installed and configured, the tool can be executed directly by running the appropriate scripts.

[4] https://docs.python.org/3/library/venv.html

The main script is *script.sh*, which initializes the API in a CPU node. The number of threads can be defined with the SBATCH *--cpus-pertask* parameter. The more threads, the more concurrent queries the system can handle flawlessly. Another script, *process_results_tasks.sh*, sets up the monitoring component in a similar way. The other relevant script, *models/model_script.sh*, is used to launch the model training to the SLURM queuing system.

After deploying the API and the monitoring scripts, the tool can be accessed at the IP of the installation node (port 54322 by default). Due to standard safety policies, the API is not expected to be available outside the cluster network. To cope with this situation, users must open an SSH tunnel. It provides a secure channel between local and remote machines.

Code 3 shows an example of building an appropriate SSH tunnel. The -L parameter defines the port forwarding. Accordingly, any accessing *http://localhost:* is redirected to its instance of the application running in the cluster.

Code 3
SSH tunneling command

```
1. ssh -N -L < local port >:<node name>:<API
port(default:54322)><username>@<cluster ip>
```

Although unlikely, the user may encounter problems with the installation and use of S-TFManager. If troubles arise, the first aspect to check is that dependencies are provided. Then, it is necessary to ensure the correct versions are installed as specified in the requirements and check the repository's documentation for further details. If the tool fails to interact with Slurm, the user should verify that the `--gres` option is enabled. It is also advisable to revise the configuration files to ensure the paths and parameters are correctly set. If TensorFlow cannot access the GPU or reports CUDA/cuDNN mismatches, ensure the correct versions of these libraries are installed and properly configured. In this regard, it is critical to remember to execute the tool under the virtual environment created. Finally, if the problems persist or S-TFManager presents unexpected problems, the users can ask for further support through the contact details provided in the repository.

3 Features and Use Cases

This section exhibits the functionality of the tool with the aid of examples and figures. It starts with the form designed for submitting models to train. It ends by showing the visualization dashboard of models being trained.

3.1 Submission of Training Tasks

Users can submit new training tasks to the underlying Slurm system by completing the web form shown in Figure 2. As can be seen, it has four main parts: i) the training information, i.e., name and description; ii) the Python script defining the model and its training; iii) the script parameters and callback; and iv) the hyperparameters to try. The default training options are the ones shown, i.e., batch size, early stopping, number of epochs, and learning rate. Since users must parameterize the submitted script accordingly, the form warns about this requirement below the file input. They must also include an appropriate CSV callback in their model so that the tool can offer its visualization capabilities. Notice that the fields "batch size" and "learning rate" can contain several values separated by commas. This way, users indicate that there will be one configuration for each value in the list. Regardless, the simplicity of our tool allows for including new parameters and options to consider.

3.2 Training Task Monitoring

As introduced, when a user submits a job with comma-separated hyperparameters, the API creates as many tasks as combinations of them exist. After submitting the form correctly, the dashboard page appears. There, the user can see all the launched tasks and their status (see Figure 3). At first glance, the name and description of each job are shown alongside its state. The possible states are the following:

- **Submitted**: Task sent to the Slurm queue yet pending of execution.
- **Running**: Task already running in a node.
- **Error**: The task finished with an error and did not complete.
- **Stopped**: The user decided to stop the training process.
- **Finished**: The training process completed successfully.

The user can click on a job to see further details (see Figure 4), i.e., the model's name, the number of associated training tasks, and submission time. Besides, if at least one of the training procedures has started, the validation loss will be automatically displayed in a common per-model graph. This feature aims to facilitate users to compare different configurations of their model at a glance. The user can also stop and remove the selected job. Any linked task is automatically removed in this case.

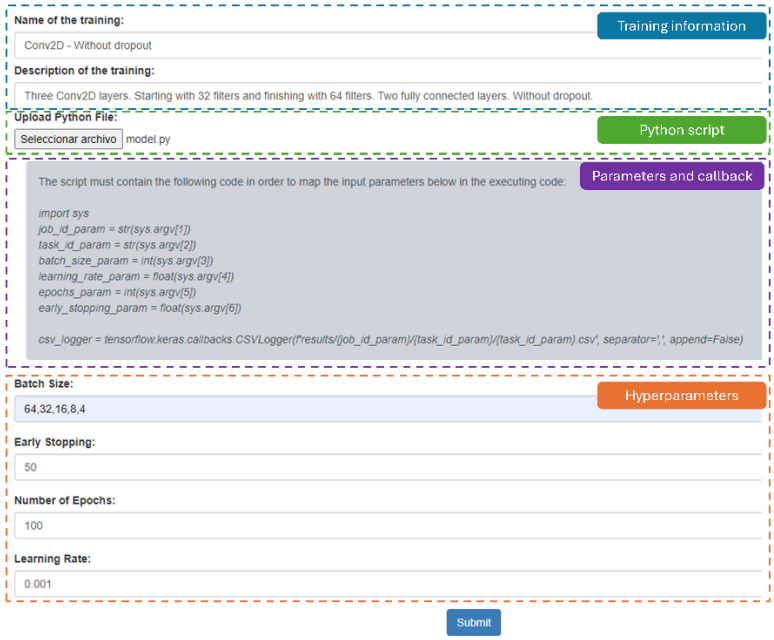

Figure 2
Form for submitting a new training job

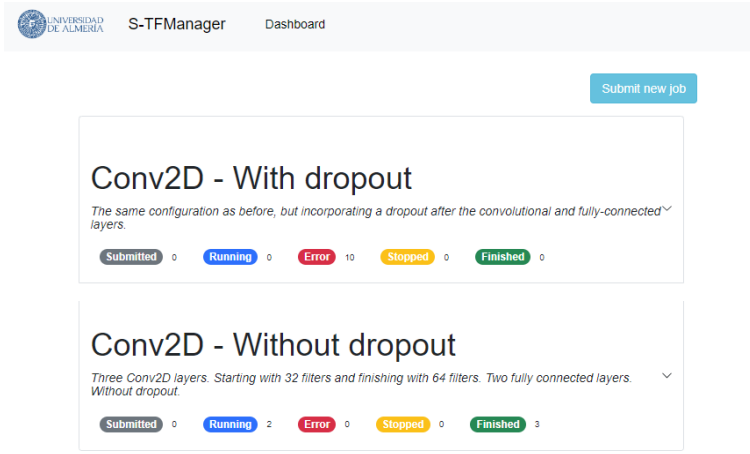

Figure 3
General view of the dashboard. List of submitted jobs

Figure 4

Job details. Execution evolution of the different configurations defined in training script

After the graph, the application shows the different training configurations, like jobs, indicating their state. The user can see further details by clicking on them (see Figure 5). The hyperparameters appear first. After that, the training results are summarized, including the current epoch, the best validation loss achieved, and when it was achieved. The training and validation loss evolution are plotted as well. Running tasks display a progress bar linked to the remaining epochs. Finally, the submission, initialization, and finalization times are indicated. If there is an error in the execution, the log file of the error is included in the graphical interface as well (see Figure 6). It is also relevant to highlight that users can stop pending and running tasks. After stopping, the user can restart them from scratch. Jobs that are either completed or failed can also be restarted.

Additionally, notice that the task visualization capabilities of the proposal are compatible with other monitoring tools from the command line for expert users and system administrators.

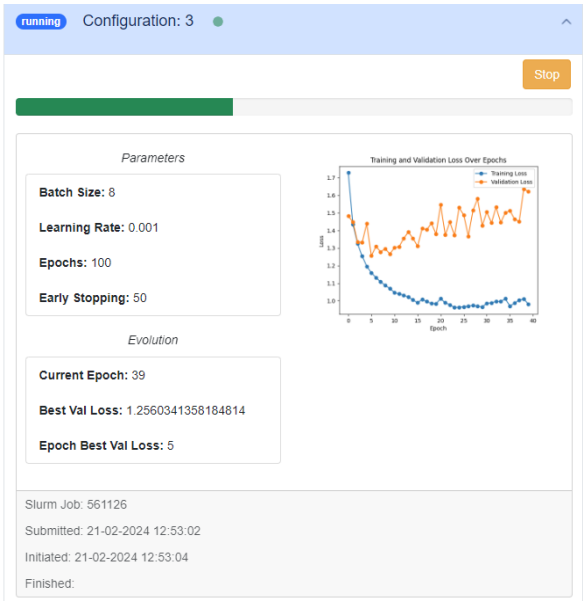

Figure 5
Execution details of a concrete ANN configuration

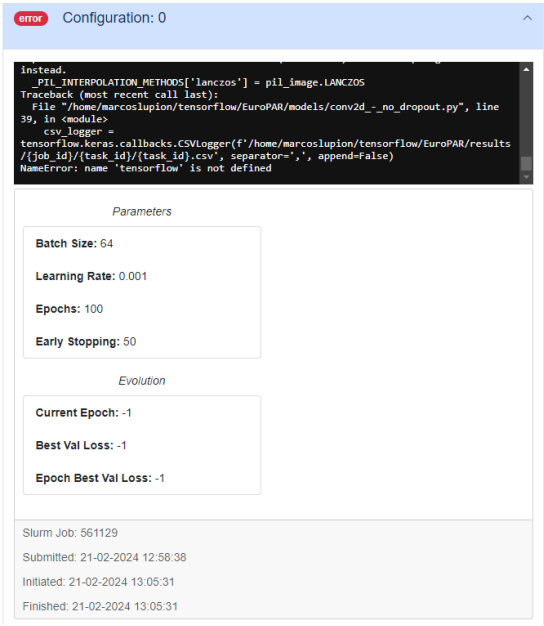

Figure 6
Error handling on a concrete ANN configuration

Conclusions

ANNs represent a very active research branch of Machine Learning. Alongside the conceptual advances in this field, HPC platforms have been critical for supporting the implementation of models that have progressively become more sophisticated and well-performing.

In this context, experts working with ANNs have two main options, i.e., cloud platforms and institutional clusters. While the former is a cost-effective and flexible option, the latter is also widespread, especially when dealing with confidential data. However, working in a shared computing cluster involves managing multiple users and resources. Besides, given the current activity and ubiquity of the field, not every researcher or engineer working with ANNs masters traditional command-line and scripting tasks.

This work has presented and described a lightweight yet powerful Python web app (S-TFManager) for working with TensorFlow models in a shared cluster. Although other alternatives exist, there are few options, and the proposal offers a perfect trade-off between design simplicity, which facilitates its adoption and adaptation, and features. It allows users to execute, queue, and track jobs.

Instead of duplicating components already expected in a standard cluster, the tool relies on an existing Slurm system for managing the available resources. Hence, users do not need to type the commands required by Slurm for launching tasks. The proposed system takes care of this interaction. It also integrates visualization capabilities for tracking and comparing the performance of different models. Furthermore, as considering multiple model configurations is part of the standard workflow in machine learning, the tool includes an option to automate the execution of models with different parameter sets.

For future work, supporting other machine learning frameworks, such as PyTorch and Mxnet, will be studied to increase the tool's compatibility. Moreover, more advanced optimization strategies will be incorporated, allowing for a more precise and efficient hyperparameter search. Finally, the inclusion of some NAS algorithms will also be considered. They should allow users to discover more performant architectures than those hand-crafted by them.

Acknowledgement

This work has been funded by the projects R+D+i PID2021-123278OB-I00 and PDC2022-133370-I00 from MCI-N/AEI/10.13039/501100011033/ and ERDF funds; and the Department of Informatics of the University of Almería. M. Lupión is a fellowship of the FPU program from the Spanish Ministry of Education (FPU19/02756). N.C. Cruz is supported by the Ministry of Economic Transformation, Industry, Knowledge and Universities from the Andalusian government (PAIDI 2021: POSTDOC_21_00124).

References

[1] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., Arshad, H.: State-of-the-art in artificial neural network applications: A survey. *Heliyon* 4(11), e00938 (2018)

[2] Baresi, L., Leva, A., Quattrocchi, G.: Fine-grained dynamic resource allocation for big-data applications. *IEEE Transactions on Software Engineering* 47(8), 1668-1682 (2019)

[3] Baresi, L., Quattrocchi, G., Rasi, N.: Resource management for Tensorflow inference. In: *International Conference on Service-Oriented Computing*. pp. 238-253, Springer (2021)

[4] Bisong, E. (2019) Google Cloud AutoML. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform* (pp. 297-308)

[5] Brinke, B. T.: *Instant Munin Plugin Starter*. Packt Publishing (2013)

[6] Cullell-Dalmau, M., Noé, S., Otero-Viñas, M., Meic, I., Manzo, C.: Convolutional neural network for skin lesion classification: understanding the fundamentals through hands-on learning. *Frontiers in Medicine* 8, 644327 (2021)

[7] Dong, S., Wang, P., Abbas, K.: A survey on deep learning and its applications. *Computer Science Review* 40, 100379 (2021)

[8] Estrada, J. F. S., Cruz, N. C., Lupión, M., Garzón, E. M., Ortigosa, P. M.: Teamwork using colab notebooks in the cloud. In: *EDULEARN23 Proceedings*. pp. 2710-2718, IATED (2023)

[9] Fritz, B. A., Pugazenthi, S., Budelier, T. P., Pennington, B. R. T., King, C. R., Avidan, M. S., & Abraham, J.: User-centered design of a machine learning dashboard for prediction of postoperative complications. *Anesthesia & Analgesia*, 138(4), 804-813 (2024)

[10] Issa, A. S. A., & Albayrak, Z.: DDoS attack intrusion detection system based on hybridization of CNN and LSTM. *Acta Polytechnica Hungarica* 20(2), 1-19 (2023)

[11] Jahani, A., Lattuada, M., Ciavotta, M., Ardagna, D., Amaldi, E., Zhang, L.: Optimizing on-demand GPUs in the cloud for deep learning applications training. In: 2019 *4th International Conference on Computing, Communications and Security (ICCCS)* pp. 1-8, IEEE (2019)

[12] Jette, M. A., Wickberg, T.: Architecture of the Slurm Workload Manager. In: *Workshop on Job Scheduling Strategies for Parallel Processing*. pp. 3-23, Springer (2023)

[13] Jin, H., Chollet, F., Song, Q., & Hu, X. (2023) AutoKeras: An AutoML Library for Deep Learning. *Journal of Machine Learning Research*, 24(6), 1-6

[14] Ketkar, N., Moolayil, J.: Introduction to PyTorch, pp. 27-91, *Apress, Berkeley*, CA (2021)

[15] Krogh, A.: What are artificial neural networks? *Nature Biotechnology* 26(2), 195-197 (2008)

[16] LeDell, E., & Poirier, S. (2020) H2O AutoML: Scalable Automatic Machine Learning. In: *Proceedings of the ICML 2020 AutoML Workshop*

[17] Lupión, M., Cruz, N. C., Sanjuan, J. F., Paechter, B., Ortigosa, P. M.: Accelerating neural network architecture search using multi-GPU high-performance computing. *Journal of Supercomputing* 79(7), 7609-7625 (2023)

[18] Mladenovic, N., Pei, J., Pardalos, P. M., Urosevic, D.: Less is more approach in optimization: A road to artificial intelligence. *Optimization Letters* 16(1), 409-420 (2022)

[19] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., & Stoica, I. (2018) Ray: A Distributed Framework for Emerging AI Applications. In: *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation* (pp. 561-577)

[20] Rosciszewski, P., Martyniak, M., Schodowski, F.: TensorHive: management of exclusive GPU access for distributed machine learning workloads. *The Journal of Machine Learning Research* 22(1), 9766-9770 (2021)

[21] Sergeev, A., & Del Balso, M. (2018) Horovod: Fast and Easy Distributed Deep Learning in TensorFlow. *arXiv preprint* arXiv:1802.05799

[22] Sharma, N., Sharma, R., Jindal, N.: Machine learning and deep learning applications–A vision. *Global Transitions Proceedings* 2(1), 24-28 (2021)

[23] Shvets, O., Seebauer, M., Naizabayeva, A., & Toleugazin, A.: Monitoring and Control of Energy Consumption Systems, using Neural Networks. *Acta Polytechnica Hungarica* 20(2), 125-144 (2023)

[24] Singh, P., Manure, A.: Introduction to TensorFlow 2.0, pp. 1-24, *Apress, Berkeley*, CA (2020)

[25] Talbi, E. G.: Automated design of deep neural networks: A survey and unified taxonomy. *ACM Computing Surveys* 54(2), 1-37 (2021)