Capturing Provenance from Deep Learning Applications Using Keras-Prov and Colab: a Practical Approach

Débora Pina[1], Liliane Kunstmann[1], Felipe Bevilaqua[1], Isabela Siqueira[1], Alan Lyra[1], Daniel de Oliveira[2], Marta Mattoso[1]

[1] COPPE/Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
{dbpina, lneves, felipebfg, isabelags, alanlyra, marta}@cos.ufrj.br
[2] Fluminense Federal University, Niterói, Rio de Janeiro, Brazil
danielcmo@ic.uff.br

Abstract. Due to the exploratory nature of DNNs, DL specialists often need to modify the input dataset, change a filter when preprocessing input data, or fine-tune the models' hyperparameters, while analyzing the evolution of the training. However, the specialist may lose track of what hyperparameter configurations have been used and tuned if these data are not properly registered. Thus, these configurations must be tracked and made available for the user's analysis. One way of doing this is to use provenance data derivation traces to help the hyperparameter's fine-tuning by providing a global data picture with clear dependencies. Current provenance solutions present provenance data disconnected from W3C PROV recommendation, which is difficult to reproduce and compare to other provenance data. To help with these challenges, we present Keras-Prov, an extension to the Keras deep learning library to collect provenance data compliant with PROV. To show the flexibility of Keras-Prov, we extend a previous Keras-Prov demonstration paper with larger experiments using GPUs with the help of Google Colab. Despite the challenges of running a DBMS with virtual environments, DL analysis with provenance has added trust and persistence in databases and PROV serializations. Experiments show Keras-Prov data analysis, during training execution, to support hyperparameter fine-tuning decisions, favoring the comparison, and reproducibility of such DL experiments. Keras-Prov is open source and can be downloaded from `https://github.com/dbpina/keras-prov`.

Categories and Subject Descriptors: H.2 [**Database Management**]: Miscellaneous; H.3 [**Information Storage and Retrieval**]: Miscellaneous; I.7 [**Document and Text Processing**]: Miscellaneous

Keywords: Deep Learning, Metadata, Provenance

1. INTRODUCTION

Over the last decade, Machine Learning (ML) has gained importance both in academia and industry [Iqbal et al. 2021; Beeharry and Fokone 2022]. Among several existing ML techniques [Russell and Norvig 2020], Deep Learning (DL) [Goodfellow et al. 2016a] has been one of the most prominent. The outcomes of Deep Neural Networks (DNN), trained from an input dataset, are quite sensitive to the hyperparameter configuration used for training [Montavon et al. 2012]. It is necessary to fine-tune such hyperparameters, in addition to the automatic tuning. DL dataflow visualization tools such as TensorBoard[1] use logs to represent hyperparameters and metrics. Comparing different runs requires preprocessing these logs, as well as computing derivation paths for deeper analyses. In addition, there is heterogeneity when representing log data in CSV tables, which limits the analytical power of DL visualization tools. To adjust the parameters during model training, it is necessary to have access to the "cause-effect" data, such as which filter was applied to the current hyperparameter configuration

[1]TensorBoard - `https://www.tensorflow.org/tensorboard`

model, when the value of dropout was below a specific threshold. When analyzing the data, the user can decide to accept the trained model or select a new hyperparameter configuration, and from that point re-train the model. A data derivation path from input data to the final DL model can help in this "cause-effect" investigation [Lourenço et al. 2020].

Provenance data [Freire et al. 2008; Moreau and Groth 2013] represent derivation paths, metadata, and relevant parameters from the execution of experiments involving multiple data transformations. Provenance data assist in reproducibility and experiment data analysis. Provenance data have been successfully used to support monitoring, analysis, and reproduction steps in multi-domain experiments, such as Bioinformatics [Almeida et al. 2019], Healthcare [Fairweather et al. 2021], Visualization [Fekete et al. 2020], *etc.*

In ML experiments, provenance data allow for associating configurations and input datasets to the derivation path that leads to an ML model. The life cycle of an ML experiment can be thought of as evaluating several data-centric workflows, each having a different configuration. Generating provenance of ML experiments has been gaining importance [Schelter et al. 2017; Agrawal et al. 2019; Gharibi et al. 2019; Ormenisan et al. 2020]. These approaches propose a framework that encompasses ML solutions to add provenance data capture during ML workflow executions. However, these approaches have proprietary representations for the provenance data, which makes it difficult to interpret and, particularly, to compare ML experiments from different tools. These comparisons can be improved when using a standard data representation such as W3C PROV [Moreau and Groth 2013]. The W3C PROV recommendation defines a data model, serializations, and definitions to support the interchange of provenance information[2]. PROV represents different types of provenance data in a domain-agnostic way.

Recently, the W3C PROV has been specialized to represent the provenance of the ML life cycle in scientific experiments [Souza et al. 2019; Huynh et al. 2019]. This representation allows for a broad view of data derivation from raw data selection to several data transformations even before training the neural network. However, to have access to data from "inside" the training and validation activities, the data capture must be coupled to the DNN solution, and the provenance data needs further specialization. Therefore, domain-agnostic solutions to capture provenance data need ML extensions. Such approaches are usually coupled to a specific programming language *e.g.*, Python in noWorkflow [Pimentel et al. 2017] or notation like in UML2PROV [Sáenz-Adán et al. 2022], which ends up limiting their use in many DL tools. Provenance capture solutions not coupled to a programming language like [McPhillips et al. 2015; Silva et al. 2020; Gehani and Tariq 2012] require an (often) expensive process of instrumentation of the code or a significant time and effort to understand what and how data was modeled to analyze it. Therefore, adding provenance data to DL training to be compared with different configurations is still an open problem in provenance solutions for DL.

In a previous work [Pina et al. 2021] we have shown Keras-Prov as a way to provide automatic provenance data capture at runtime in DL experiments, to support analyses. Keras-Prov identifies the most common data transformations, such as training, testing, and adaptation, and allows the user to extend this automatic provenance capture with new data. In DL experiments with Keras-Prov, provenance data is associated with metadata to support human-guided analysis regarding hyperparameter settings or even training data. Evaluating the various hyperparameters requires the user to be aware of the relationship between many types of metadata, such as the chosen hyperparameter values, performance execution data, epochs monitoring, environment configuration, *etc.* To monitor, track and analyze data during training execution (*i.e.* between epochs of a single model configuration), typical DL provenance data must be captured and made available at runtime. In this article, we extend [Pina et al. 2021] to present Keras-Prov behavior in DL experiments running on GPUs. The evaluation runs part of Keras-Prov components with Google Colab while the components responsible for storing the provenance data run outside Colab. This work contributes to our previous Keras-Prov ex-

[2]`https://www.w3.org/TR/prov-overview/`

periment [Pina et al. 2021] by showing how provenance services can be used in different environments, like Google Colab notebook. We also evaluated the benefits of our provenance services in comparing AlexNet and DenseNet experiments under Google Colab cloud executions. The remainder of this article is structured as follows. Section 2 presents Keras-Prov provenance data representation and describes its architecture components. Section 3 discusses how Keras-Prov was deployed with Colab. Section 4 details the case studies and the experimental evaluation. Finally, Section 5 concludes.

2. KERAS-PROV IN A NUTSHELL

Keras-Prov [Pina et al. 2021] is a library that aims at capturing provenance data from DL experiments automatically. Similar to previous solutions developed in our research group [Silva et al. 2018], the representation of provenance data in Keras-Prov follows the W3C PROV recommendation. By following W3C-PROV, Keras-Prov fosters provenance data analysis, easing the comparison with results generated from other DL libraries (in the case where the DL libraries can export provenance following PROV). In addition to the advantages of provenance data to interpret, reproduce and add quality to the DL data, the target user of Keras-Prov is the user who specifies the DL experiment, being responsible for fine-tuning the hyperparameters and the network architecture. Thus, Keras-Prov stands out for its analytical support during model training and evaluation.

Keras-Prov extends the core functionalities of Keras API[3] (*i.e.*, a deep neural network training library) and adds features of DfAnalyzer (a provenance system for distributed applications) to capture provenance without requiring source code instrumentation. The Keras-Prov architecture (Fig. 1) is composed of three layers: (i) Keras-Prov Core, which is composed by the *Keras Core* and the *Provenance Extractor* component, (ii) Data Layer, which is composed of the *File System* and the *Provenance Database* and (iii) Analysis Layer, which is composed of the *Provenance Exporter* and the *Provenance Viewer* components. As aforementioned, Keras-Prov embeds all native Keras features. When Keras is executing, provenance data is captured and sent to the *Provenance Extractor* to be processed asynchronously to Keras execution. The *Provenance Extractor* is already coupled to the Keras library, so the user does not need to instrument its source code to capture provenance data. The user chooses, among the predefined options, the domain data (datasets) and the set of hyperparameters to be captured. The *Provenance Extractor* automatically extracts the values of the hyperparameters used in each training and corresponding execution time from epochs and time for the training. Once captured, the data is managed asynchronously with the training of the neural network. The *Provenance Extractor* also accesses the *File System* to obtain the paths of JSON files that describe the neural network and its input data.

Ideally, the *Provenance Extractor* should execute in a different processor of Keras. As observed in previous experiments [Silva et al. 2020], this data capturing approach presents negligible overhead in the performance of the DL experiment. Different instances of Keras running the DL experiment in separate GPUs send its local data to the *Provenance Extractor*, which is integrated into the *Provenance Database*. All file paths, hyperparameter values used, metrics, *etc.*, are stored in the same provenance database (in the current version, instantiated in MonetDB[4]). The *Provenance Extractor* structures and stores the captured provenance data in MonetDB, which relates them to other data from previous executions of the experiment. As soon as the data is managed by MonetDB it is ready for queries through the components of the Analysis Layer.

Since W3C PROV is domain-agnostic, Keras-Prov specializes the PROV model, creating a new representation named DNNProv-Df [Pina et al. 2021] shown in Fig. 2. This aims at representing training-specific data from DL experiments. Using the DNNProv-Df representation, it is possible to (i) track epochs, learning rate, accuracy, loss function values, processing time, *etc.* from different

[3]Keras - `https://keras.io/`
[4]MonetDB - `https://www.monetdb.org/`

configurations of the DL experiment, (ii) find out which preprocessing methods were used on the data before training the model, (iii) monitor the training process and fine-tune the hyperparameters, (iv) discover which files were generated in the different execution steps, and (v) interpret the results generated. The *data_ dependency* class allows for tracking the data derivation path and relates input data with the output data of the *task* execution. When mapped to a relational schema, this allows for recursive queries that trace back the execution. The *dataflow_ execution* class allows for grouping information of different trials of the experiments. The *ds_ itraining* class represents input data for training with different configurations and the *ds_ otraining* represents output data captured as performance metrics by epoch. These classes help track and compare the different configurations trials.

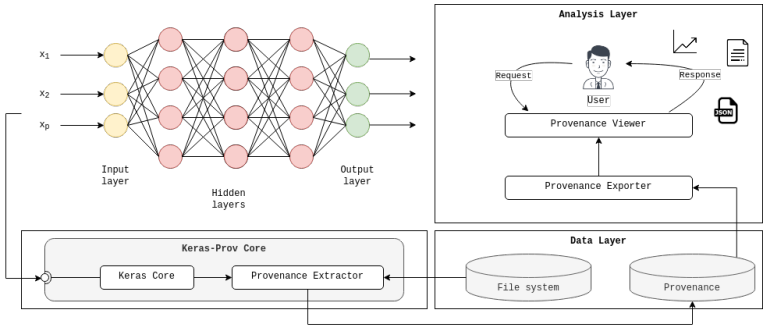

Fig. 1. Keras-Prov architecture adapted from [Pina et al. 2021]

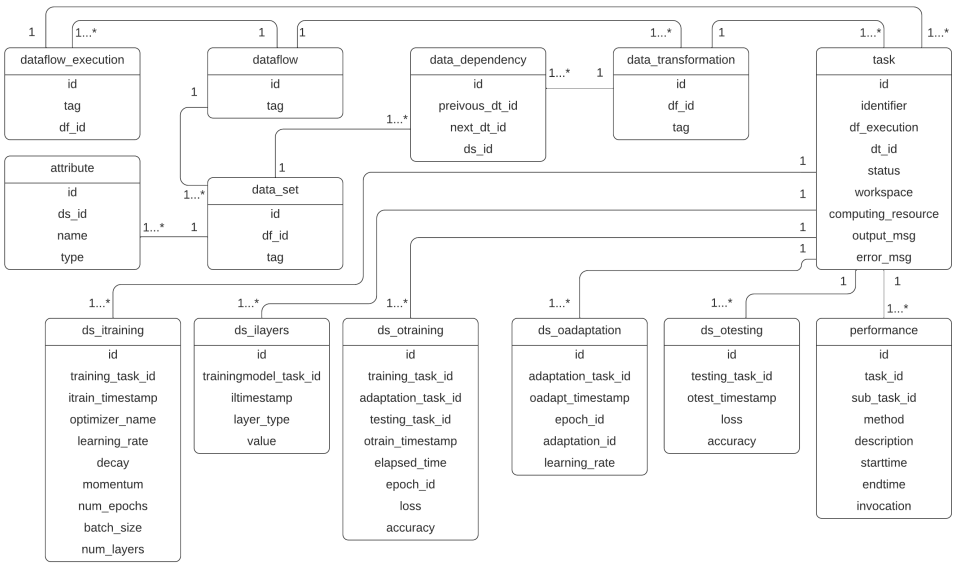

Fig. 2. Keras-Prov Provenance Schema

When the DL experiment is executed, all provenance data is stored in the *Provenance Database*, following a relational schema mapping from DNNProv-Df. Then the *Provenance Exporter* can query the database and sends the query results to the *Provenance Viewer*, which generates a visual representation of the provenance graph for hyperparameter analysis, and it allows for visual analysis (using visualization tools, such as Kibana) for performance analysis, such as loss function values per epoch, *etc.* In addition, PROV-compliant traces can be generated with help of the Prov Python library.

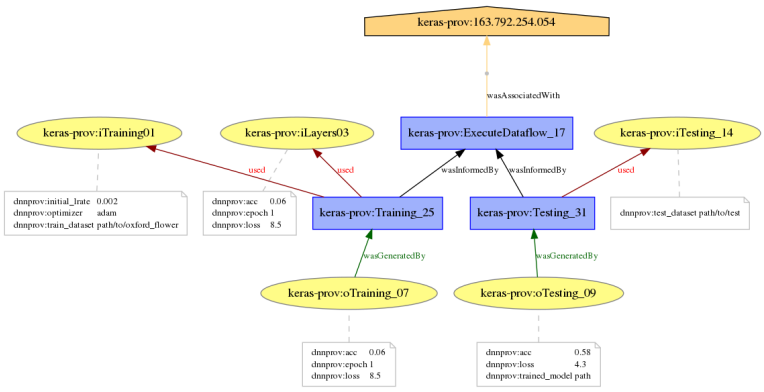

Fig. 3. W3C PROV graph of a DNN training with Keras-Prov

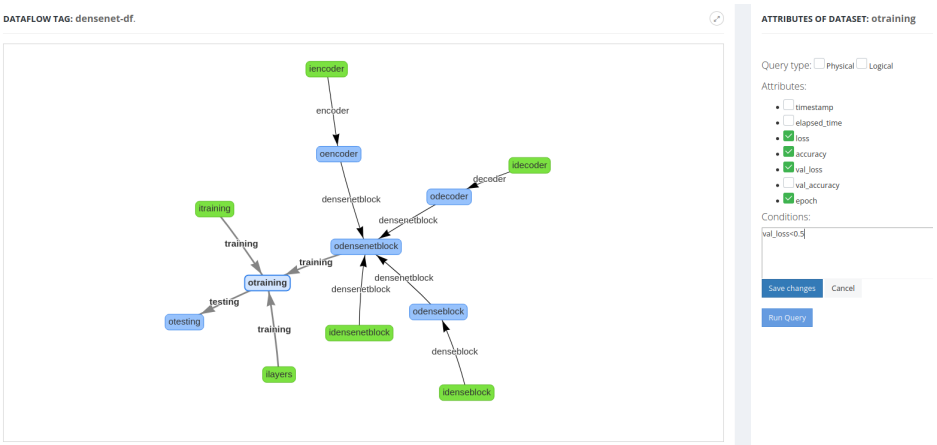

Fig. 4. Keras-Prov query interface

Then, using Graphviz, the user can generate the diagram presented in Fig. 3, which represents activities such as Training and Testing of the DNN training process with Keras-Prov, following the W3C notation proposed by Missier *et al.* [2013]. The orange pentagon represents the Agent concept, yellow ovals represent Entity and blue rectangles represent Activity. When the user has several provenance traces under this same representation, this helps to identify patterns and execution behavior.

Fig. 4 shows the query interface of the *Analysis Layer* that allows for accessing provenance data captured by Keras-Prov. This interface allows for querying the database without using SQL. For example, Fig. 4 shows the composition of the query "What are the accuracy, loss, and validation loss for each epoch when the validation loss is less than 0.5?". On the left side, it shows the provenance graph representation of the dataflow to be queried. The graph contains the names of entities (data) represented as vertices and the names of transformations, represented by directed edges that associate the entities used as input of the transformation to the output data. This representation helps the user to define queries with filters so that the corresponding SQL is generated automatically. The right side of Fig. 4 shows the panel that the user can use to submit queries without using SQL. One can select an entity from the graph and the panel shows its attributes, and add filters such as *val_loss<0.5* as typed on the Conditions box, in this interface is also possible to make joins between datasets.

3. INTEGRATING KERAS-PROV AND GOOGLE COLAB

In this section, we present how Keras-Prov is integrated with Google Colab to foster Keras-Prov usage in GPUs and evaluate its experiment's provenance data analysis. Google Colab is a tool that allows the user to integrate Python source code with text (usually in markdown). This type of environment is commonly called a "Notebook". The advantage of notebooks is that they provide a collaborative environment with zero configuration effort and access to several types of resources. Although there are some disadvantages of using Colab for some ML experiments (*e.g.*, a series of automatic settings are not suitable for ML experiments), the integration of Keras-Prov and Colab seems to be promising, especially because Colab provides access to several generations of GPUs. This type of hardware is not easily available in commodity machines. By the time this article was written, the GPU configuration provided at Google Colab was based on a Tesla T4, which implements the Turing architecture and is targeted at Deep Learning model inference acceleration[5]. By using these GPUs, the time required for each training epoch reduces from 3 minutes (using a CPU) to 2 seconds. This decrease allows for the exploration of more models and hyperparameter values by the user.

To integrate Keras-Prov with Google Colab, an external environment had to be set. Although it is possible (and easier) to deploy both - the provenance system and the ML application - on the same machine, this is not recommended by Google Colab. The reason is that when using Colab, the execution depends on the user's active session (which has a timeout), and data persistence may last for more time than the set timeout. In addition, it is not recommended to install third-party components that depend on a Database Management System (DBMS), *e.g.*, MonetDB, and to execute Java Web Application (*e.g.*, query interface). A cloud solution was chosen to host both DfAnalyzer (*i.e.*, the provenance system) and MonetDB. DigitalOcean[6] was elected because it offers resources with low financial cost and simplicity, providing easy installation of containers. MonetDB and DfAnalyzer, which are deployed as a single container, were installed in a private virtual machine and their services were configured to be accessible from Google Colab through a public IP address.

Fig. 5 presents the architecture that integrates Keras-Prov with Google Colab and DigitalOcean (*i.e.*, external environment). Although there is a communication overhead between both virtual machines (in Colab and DigitalOcean), this configuration has the benefit of separating ML experimentation and provenance infrastructure. Inside an enterprise, for example, there are different expertise and responsibilities: the production team would support the provenance system hosted in an appropriate infrastructure - on-premise or cloud-based - while the data science team would be end-users, training new applications and submitting provenance queries using Python notebooks transparently. For training and evaluating the models trained using Keras-Prov, it is necessary to import libraries `Keras-Prov` and `dfa-lib-python` within the Colab notebook, before training the models. A `model.provenance(parameters)` directive has to be added to the source code while configuring the ML application.

4. EXPERIMENTAL EVALUATION

In this section, we evaluate the integration of Keras-Prov and Google Colab with two real DL experiments. The first experiment uses AlexNet to evaluate aspects of data monitoring and queries when using Google Colab and to analyze the limitations of deploying the MonetDB DBMS, as the Keras-Prov provenance persistent system, to the Colab notebook. The second experiment used DenseNet to explore data analytical issues.

[5]GPU Architecture in Google Colab - `https://colab.research.google.com/github/d2l-ai/d2l-tvm-colab/blob/master/chapter_gpu_schedules/arch.ipynb`
[6]DigitalOcean - `https://www.digitalocean.com/`

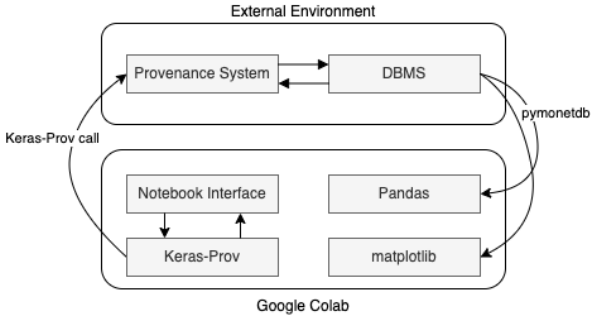

Fig. 5. Architecture that integrates Google Colab with Keras-Prov libraries and services

4.1 AlexNet and DenseNet Case Studies

AlexNet is a convolutional neural network (CNN) architecture developed by [Krizhevsky et al. 2012] for computer vision tasks. AlexNet has eight layers, where the first five are convolutional layers, and the last three are fully connected layers. AlexNet uses a non-saturating ReLU activation function, which presents better performance when compared to activation functions such as tanh and sigmoid. In our experiments, we use AlexNet with the Oxford Flower [Nilsback and Zisserman 2006] dataset, which is composed of 17 species of flowers with 80 images for each class. The flower categories represented in Oxford Flower are chosen to present ambiguities, *e.g.*, some classes cannot be distinguished only by analyzing their colors, such as dandelions and buttercups, others cannot be distinguished only by analyzing shapes, such as daffodils and wild windflowers. The Alexnet dataflow is composed of the following activities: (i) *Training*, (ii) *Adaptation*, and (iii) *Testing*. The dataflow consumes (*used*) the following hyperparameters: the optimizer, the learning rate, the number of epochs, and the number of layers.

Differently from AlexNet, Dense Convolutional Networks (*i.e.*, DenseNet) [Huang et al. 2017] connect each layer of the neural network to every other layer in a feed-forward way. This way, DenseNet presents L(L+1)/2 connections, which is different from the L connections found in most neural networks. The use of DenseNets in many problems presents several advantages, *e.g.*, they foster feature propagation and (possibly the most important advantage) reduce the number of parameters. In our experiments, a pre-trained DenseNet model with the ImageNet dataset is used as a base layer with fixed parameters. We add, on top of this pre-trained model, a trainable dense layer with dropout, batch normalization, and max-pooling, and a last dense layer with softmax activation.

4.2 Environment Setup

Although the integration of Keras-Prov and Colab is promising and opens room for many refinements, it presents a drawback regarding resource reservation. Since Colab provides free computational resources, it automatically adjusts the hardware availability at runtime (*e.g.*, during the training process of a neural network). This is a behavior already found in many cloud providers, *e.g.*, the AWS Spot market [Portella et al. 2019]. Although Colab offers ways to provide on-demand virtual machines that guarantee performance, this type of resource was not used in the experiments presented in this article. Colab also offers many types of GPUs to use, however, the types of GPUs that are available vary over time. By the time the experiments were executed, the GPU configuration provided by Colab is based on a Tesla T4 (which implements the Turing architecture). In addition, it is not guaranteed that the experiments are executed on a dedicated card. The amount of memory in Colab virtual machines also varies from 12GB to 25GB over time, but the amount of memory does not vary during the life cycle of a specific virtual machine.

4.3 Experiment Setup

The AlexNet network is trained using Keras-Prov in Google Colab, for classifying flowers using the Oxford Flower dataset. Similar to most DL experiments, we need to vary a set of hyperparameters to get the best results possible. For AlexNet, we have chosen to evaluate the impact of the activation function choice. Seven different activation functions are explored, *i.e.*, Rectified Linear Activation (ReLU) [Nair and Hinton 2010], Logistic (Sigmoid), Softmax [Goodfellow et al. 2016b], Softplus [Glorot et al. 2011], Softsign [Goodfellow et al. 2016b], Hyperbolic Tangent (Tanh) and Scaled exponential linear unit (SELU) [Klambauer et al. 2017]. The hyperparameter settings for AlexNet are presented in Table I. The type of optimizer, the learning rate, the number of epochs, and the dropout value are fixed, and the activation function of the internal layers of the AlexNet network is varied. Thus, experiments are performed for each of the aforementioned activation functions. All these hyperparameters are automatically captured by Keras-Prov.

Table I. AlexNet Hyperparameter Configuration.

Fixed Parameters				**Varied Parameters**
Optimizer	**Learning Rate**	**Epochs**	**Dropout Rate**	**Activation Function**
Adam	0.0001	100	0.4	ReLU, Sigmoid, Softmax, Softplus, Softsign, Tanh, SELU

With DenseNet, two experiments are performed. In the first, different activation functions are evaluated in the dense layer of the network, as presented in Table II. While in AlexNet the activation functions are related to multiple layers, in this experiment the activation function that is varied is the one at the dense layer (except for the final dense layer) used on top of the pre-trained layers. In the second experiment, different learning rates and dropout rates are evaluated, while the number of epochs, the optimizer, and the activation function are fixed, as presented in Table III.

Table II. DenseNet Hyperparameter Configuration - Experiment 1.

Fixed Parameters				**Varied Parameters**
Optimizer	**Learning Rate**	**Epochs**	**Dropout Rate**	**Activation Function**
Adam	0.0001	100	0.4	ReLU, Sigmoid, Softmax, Softplus, Softsign, Tanh, SELU

Table III. DenseNet Hyperparameter Configuration - Experiment 2.

Fixed Parameters			**Varied Parameters**	
Optimizer	**Activation Function**	**Epochs**	**Dropout Rate**	**Learning Rate**
Adam	Softplus	10	0.2, 0.4, 0.6, 0.8	0.001, 0.01, 0.1

It is worth noticing that the volume of data used in the experiments is relatively small, and no cross-validation techniques were used. Thus, many of the analyses on the accuracy of the models may be the result of random oscillations and, therefore, should not be seen as effective conclusions about the overall performance of the models. The experiments presented are data collected including all executions in which the environment had to be set up. We used the free version of Colab, the computational environment was shared among many users altering computational performance and measurements. Also, using two environments generated the additional cost of sending external calls to the Provenance System. It is noteworthy that, despite this, there is no harm to the goals of this article, which are to explore the use of the Keras-Prov integrated with Google Colab and not to evaluate the effectiveness of the models.

4.4 Results Discussion

AlexNet. As aforementioned, all collected provenance data is stored in the Provenance Database in MonetDB. In the Colab notebook, `pymonetdb` is used to retrieve the provenance data stored by Keras-Prov. A series of queries can be submitted using `pymonetdb`. The combination of such features, especially Keras-Prov, made the results analysis of hyperparameter variations easier and faster. In this experiment, the SQL-like query `"SELECT * FROM all_100epoc.ds_otrainingmodel"` is submitted to retrieve the accuracy and the loss for each evaluated activation function. Based on this provenance

query result, Fig. 6 presents accuracy and the loss charts generated with `Matplotlib`. By analyzing the results in Fig. 6, one can state that the activation function that obtained the best accuracy and lowest loss is the Softplus activation function. This network configuration presents a maximum accuracy of 76% and a minimum loss of 1.19, and both values occurred at epoch 50. The performance of the accuracy and loss of other functions over the epochs can be seen in Fig. 6.

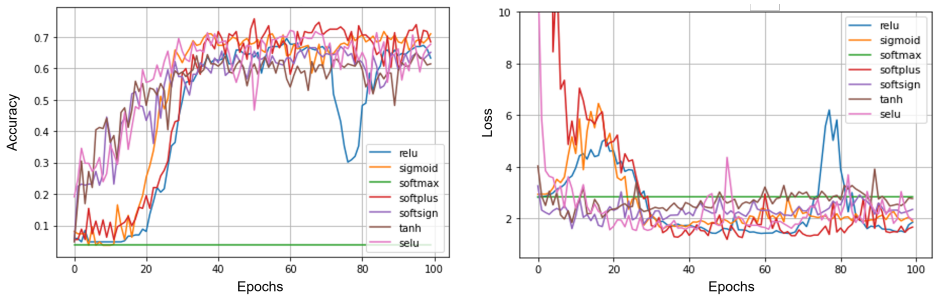

Fig. 6. Accuracy and Loss over the epochs for the different activation functions used in AlexNet.

DenseNet. To further explore the capabilities and limitations of Keras-Prov, two experiments were carried out with DenseNet. The DenseNet model was pre-trained with the ImageNet dataset, and we use this pre-trained model as a base layer with fixed parameters, while on top of this model we add a trainable dense layer. In the first experiment with DenseNet, we analyze which activation function presents the best accuracy and loss. While in the AlexNet experiment the activation functions are related to multiple layers of AlexNet, in this experiment the activation function that is changed is the one at the only trainable dense layer used on top of the pre-trained DenseNet. Table IV shows the two epochs that present the best accuracy and loss, in both of them, Softplus is the activation function. Thus, the Softplus activation function is the choice for the second experiment with DenseNet.

Table IV. The best accuracy among all explored activation functions for DenseNet

Model	Activation Function	Learning Rate	Dropout Rate	Epoch	Validation Accuracy	Validation Loss	Processing Time
DenseNet	Softplus	0.01	0.4	17	0.863	0.529	13.77
DenseNet	Softplus	0.01	0.4	18	0.863	0.527	13.79

Having the same database schema helped us to compare results from AlexNet and DenseNet. Also, the fact that Keras-Prov follows a public recommendation helps coupling applications to analyze and manage the data in the way that is best for the user, and with no pre-processing. Keras-Prov associates performance execution data to the provenance data, like the execution time of each (or set of) epoch and the total training time. Table V presents the average processing time (*i.e.*, $\overline{x}$) in seconds with AlexNet and DenseNet for each network configuration. DenseNet presents a more complex architecture when compared to AlexNet, so it requires higher processing times. In the experiments, both networks consider learning rate 0.0001, dropout 0.4, 100 epochs, and the following activation functions: ReLU, Sigmoid, Softmax, Softplus, Softsign, Tanh, SELU.

Finally, in the second experiment with DenseNet, the activation function (*i.e.*, Softplus), the Optimizer (*i.e.*, Adam) and the number of epochs (*i.e.*, 10) are fixed. We varied the dropout rate (0.2, 0.4, 0.6, and 0.8) and the learning rate (0.001, 0.001, and 0.1) to evaluate the accuracy and the loss for each combination. Fig. 7 and Fig. 8 present the accuracy and loss for each combination, respectively. This type of analysis is fundamental for the user to set the proper number of epochs for a specific experiment. In many cases, adding more epochs does not help much in improving the accuracy of the model.

Table V. Comparison of the average processing time of epochs of each optimizer for different networks

Activation	Alexnet+Keras-Prov	DenseNet+Keras-Prov
ReLU	3.107	13.833
Sigmoid	3.121	13.839
Softmax	3.772	13.848
Softplus	3.217	13.834
Softsign	3.207	13.850
Tanh	3.204	13.863
SELU	3.839	13.351
$\overline{x}$	**3.352**	**13.77**

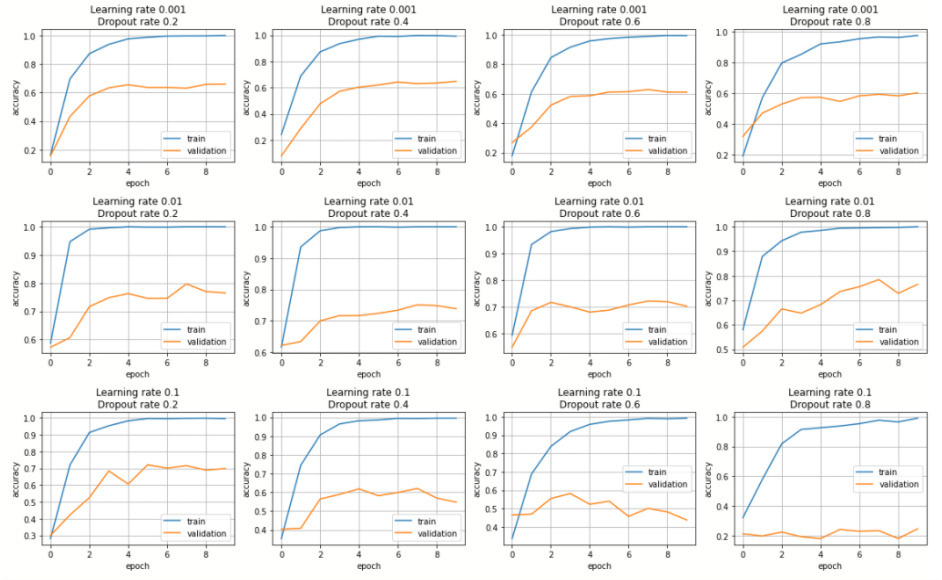

Fig. 7. Accuracy rates obtained for training and validation of each combination of parameters with DenseNet.

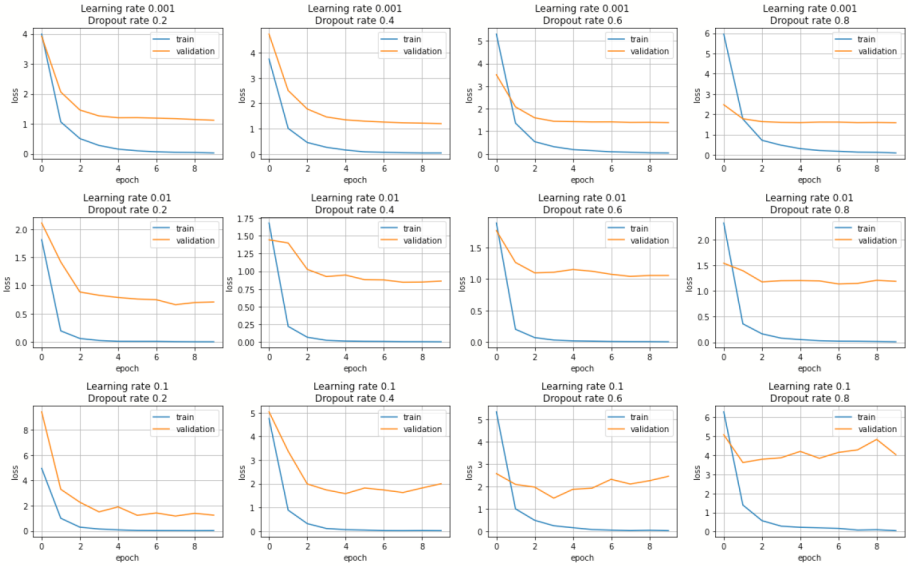

Fig. 8. Loss rates obtained for training and validation of each combination of parameters in Experiment 2 with DenseNet

4.5 Limitations

The original version of Keras-Prov has its container to help with deployment, but it was not compatible with Google Colab. Most Keras-Prov services are not dependent on the Keras version, however, the provenance capturing accesses the Keras internal methods, which limits its usage with newer Keras versions. It is worth mentioning that this limitation is also found in many ML services provided by Tensorflow and Colab, *e.g.*, new versions of Tensorflow may not be entirely compatible. Despite the containerized version of Keras-Prov, the deployment at Colab required "manual" (and non-trivial) configurations. Another issue is related to the provenance schema of Keras-Prov. Having a provenance schema allows for richer queries than CSV files or the ML environment support. Although Keras-Prov data representation (Fig. 2) specializes PROV with ML classes, there are other PROV specializations for ML that could be used. There is no consensus on that issue in the provenance community, and our evaluation suggests that the provenance representation in Keras-Prov must be continuously improved, incorporating features and characteristics of other proposed models.

5. CONCLUSION

In this article, we have shown Keras-Prov being used in Google Colab to generate provenance traces of DL experiments in different computational environments. Keras-Prov captures provenance from Keras executions automatically, and allows querying through a graphical interface and integration with various analysis tools (e.g., Pandas, ElasticSearch, Kibana). Hence, this provenance database approach does not replace, but rather complements typical DL data visualization and allows for more specific visualizations, taking advantage of sophisticated query results, which can, for example, make use of aggregations. Due to following the W3C PROV recommendation, its database adopts a standard data structure, which provides uniform DL training representation and comparisons with other PROV-compliant data. Also, using Keras-Prov with Colab has shown the possibility of executing Keras-Prov with a popular notebook solution, and GPUs in different environments to collect and analyze provenance. We have shown two case studies, DenseNet and AlexNet, to solve the same learning task and compared them at different training stages using data collected through Keras-Prov. Some provenance database queries presented in our experiments show the power of comparing these experiments, like presenting the average epoch's processing time for the same optimizer in the two networks. This approach avoids having to run the DL training under a specific framework to make some similar comparisons or the need to organize CSV files in directories and write scripts to compare them. Despite defining configurations on small scale, having so many variations to compare and evaluate has evidenced the value of having a provenance database that organizes and persists these data. Using W3C PROV allowed to produce provenance traces in a standard notation. These provenance data analyses evidenced the benefits of having database community techniques coupled with ML solutions.

As future work, we plan to make Keras-Prov services more platform-independent and easier to be deployed to different DL systems. Our main goal is not to provide a provenance solution on its own, but rather to offer provenance services to the main DL system developers so that they can incorporate our provenance services to offer as part of their DL releases.

ACKNOWLEDGEMENTS

This work was partially funded by CNPq, FAPERJ and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

REFERENCES

Agrawal, P., Arya, R., Bindal, A., Bhatia, S., Gagneja, A., Godlewski, J., Low, Y., Muss, T., Paliwal, M. M., Raman, S., Shah, V., Shen, B., Sugden, L., Zhao, K., and Wu, M.-C. Data platform for machine

learning. In *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD '19. Association for Computing Machinery, New York, NY, USA, pp. 1803–1816, 2019.

Almeida, R. F., da Silva, W. M. C., Castro, K., de Araújo, A. P. F., Walter, M. E. T., Lifschitz, S., and Holanda, M. Managing data provenance for bioinformatics workflows using aprovbio. *Int. J. Comput. Biol. Drug Des.* 12 (2): 153–170, 2019.

Beeharry, Y. and Fokone, R. T. Hybrid approach using machine learning algorithms for customers' churn prediction in the telecommunications industry. *Concurr. Comput. Pract. Exp.* 34 (4): e6627, 2022.

Fairweather, E., Wittner, R., Chapman, M., Holub, P., and Curcin, V. Non-repudiable provenance for clinical decision support systems. In *Provenance and Annotation of Data and Processes*. Springer International Publishing, Virtual Event, pp. 165–182, 2021.

Fekete, J., Freire, J., and Rhyne, T. Exploring reproducibility in visualization. *IEEE Computer Graphics and Applications* 40 (5): 108–119, 2020.

Freire, J., Koop, D., Santos, E., and Silva, C. T. Provenance for computational tasks: A survey. *Computing in Science & Engineering* 10 (3): 11–21, 2008.

Gehani, A. and Tariq, D. Spade: Support for provenance auditing in distributed environments. In *Middleware 2012*, P. Narasimhan and P. Triantafillou (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 101–120, 2012.

Gharibi, G., Walunj, V., Rella, S., and Lee, Y. Modelkb: Towards automated management of the modeling lifecycle in deep learning. In *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*. pp. 28–34, 2019.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík (Eds.). Proceedings of Machine Learning Research, vol. 15. PMLR, Fort Lauderdale, FL, USA, pp. 315–323, 2011.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016a. `http://www.deeplearningbook.org`.

Goodfellow, I. J., Bengio, Y., and Courville, A. C. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016b.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, pp. 2261–2269, 2017.

Huynh, T., Stalla-Bourdillon, S., and Moreau, L. *Provenance-based Explanations for Automated Decisions: Final IAA Project Report*, 2019.

Iqbal, S., Hassan, S., Aljohani, N. R., Alelyani, S., Nawaz, R., and Bornmann, L. A decade of in-text citation analysis based on natural language processing and machine learning techniques: an overview of empirical studies. *Scientometrics* 126 (8): 6551–6599, 2021.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks, 2017.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. pp. 1097–1105, 2012.

Lourenço, R., Freire, J., and Shasha, D. E. Bugdoc: A system for debugging computational pipelines. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo (Eds.). ACM, pp. 2733–2736, 2020.

McPhillips, T., Bowers, S., Belhajjame, K., and Ludäscher, B. Retrospective provenance without a runtime provenance recorder. In *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*. USENIX Association, Edinburgh, Scotland, 2015.

Missier, P., Belhajjame, K., and Cheney, J. The w3c prov family of specifications for modelling provenance metadata. In *Proceedings of the 16th International Conference on Extending Database Technology*. EDBT '13. Association for Computing Machinery, New York, NY, USA, pp. 773–776, 2013.

Montavon, G., Orr, G., and Müller, K.-R. *Neural networks: tricks of the trade*. Vol. 7700. Springer, 2012.

Moreau, L. and Groth, P. Provenance: an introduction to prov. *Synthesis Lectures on the Semantic Web: Theory and Technology* 3 (4): 1–129, 2013.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Omnipress, Madison, WI, USA, pp. 807–814, 2010.

Nilsback, M.-E. and Zisserman, A. A visual vocabulary for flower classification. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE, pp. 1447–1454, 2006.

Ormenisan, A. A., Ismail, M., Haridi, S., and Dowling, J. Implicit provenance for machine learning artifacts. *Proceedings of MLSys* vol. 20, 2020.

Pimentel, J. F., Murta, L., Braganholo, V., and Freire, J. noworkflow: a tool for collecting, analyzing, and managing provenance from python scripts. *VLDB* 10 (12): 1841–1844, 2017.

Pina, D., Kunstmann, L., de Oliveira, D., Valduriez, P., and Mattoso, M. Provenance supporting hyperparameter analysis in deep neural networks. In *Provenance and Annotation of Data and Processes*. Springer International Publishing, Virtual Event, pp. 20–38, 2021.

Pina, D., Neves, L., de Oliveira, D., and Mattoso, M. Captura automática de dados de proveniência de experimentos de aprendizado de máquina com keras-prov. In *Anais Estendidos do XXXVI Simpósio Brasileiro de Bancos de Dados*. SBC, Porto Alegre, RS, Brasil, pp. 69–74, 2021.

Portella, G., Nakano, E. Y., Rodrigues, G. N., and Melo, A. C. M. A. Utility-based strategy for balanced cost and availability at the cloud spot market. In *12th IEEE International Conference on Cloud Computing, CLOUD 2019, Milan, Italy, July 8-13, 2019*, E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, and K. Oyama (Eds.). IEEE, pp. 214–218, 2019.

Russell, S. J. and Norvig, P. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

Sáenz-Adán, C., Pérez, B., Izquierdo, F. J. G., and Moreau, L. Integrating provenance capture and UML with UML2PROV: principles and experience. *IEEE Trans. Software Eng.* 48 (2): 53–68, 2022.

Schelter, S., Boese, J.-H., Kirschnick, J., Klein, T., and Seufert, S. Automatically tracking metadata and provenance of machine learning experiments. In *Machine Learning Systems Workshop at NIPS*. pp. 27–29, 2017.

Silva, V., Campos, V., Guedes, T., Camata, J., de Oliveira, D., Coutinho, A. L., Valduriez, P., and Mattoso, M. Dfanalyzer: Runtime dataflow analysis tool for computational science and engineering applications. *SoftwareX* vol. 12, pp. 100592, 2020.

Silva, V., de Oliveira, D., Valduriez, P., and Mattoso, M. Dfanalyzer: runtime dataflow analysis of scientific applications using provenance. *VLDB* vol. 11, pp. 2082–2085, 2018.

Souza, R., Azevedo, L., Lourenço, V., Soares, E., Thiago, R., Brandão, R., Civitarese, D., Brazil, E. V., Moreno, M., Valduriez, P., Mattoso, M., Cerqueira, R., and Netto, M. A. S. Provenance data in the machine learning lifecycle in computational science and engineering. In *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, pp. 1–10, 2019.