Online Deep Learning Hyperparameter Tuning based on Provenance Analysis

Liliane Kunstmann[1], Débora Pina[1], Filipe Silva[1],
Aline Paes[2], Patrick Valduriez[3], Daniel de Oliveira[2], Marta Mattoso[1]

[1] COPPE/Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
{lneves, dbpina, marta}@cos.ufrj.br, dfilipeaugusto@poli.ufrj.br
[2] Fluminense Federal University, Niterói, Rio de Janeiro, Brazil
{alinepaes, danielcmo}@ic.uff.br
[3] Inria, University of Montpellier, CNRS, LIRMM, France
Patrick.Valduriez@inria.fr

Abstract. Training Deep Learning (DL) models require adjusting a series of hyperparameters. Although there are several tools to automatically choose the best hyperparameter configuration, the user is still the main actor to take the final decision. To decide whether the training should continue or try different configurations, the user needs to analyze online the hyperparameters most adequate to the training dataset, observing metrics such as accuracy and loss values. Provenance naturally represents data derivation relationships (i.e., transformations, parameter values, *etc.*), which provide important support in this data analysis. Most of the existing provenance solutions define their own and proprietary data representations to support DL users in choosing the best hyperparameter configuration, which makes data analysis and interoperability difficult. We present Keras-Prov and its extension, named Keras-Prov++, which provides an analytical dashboard to support online hyperparameter fine-tuning. Different from the current mainstream solutions, Keras-Prov automatically captures the provenance data of DL applications using the W3C PROV recommendation, allowing for hyperparameter online analysis to help the user deciding on changing hyperparameters' values after observing the performance of the models on a validation set. We provide an experimental evaluation of Keras-Prov++ using AlexNet and a real case study, named DenseED, that acts as a surrogate model for solving equations. During the online analysis, the users identify scenarios that suggest reducing the number of epochs to avoid unnecessary executions and fine-tuning the learning rate to improve the model accuracy.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords: Deep Learning, Provenance, Hyperparameter tuning

1. INTRODUCTION

Over the last few years, in the Deep Learning (DL) domain, Convolutional Neural Networks [Matsugu et al. 2003] (henceforth named only as CNNs) have presented an outstanding performance for several applications and scenarios (*e.g.*, visual computing [Liu et al. 2021; Guérin et al. 2021], natural language processing [Ren et al. 2020; Agrawal and Urolagin 2020], finances [Özbayoglu et al. 2020], bioinformatics [de Oliveira et al. 2020], game development [Goulart et al. 2019], engineering [Yang et al. 2021], *etc.*). This performance is due to new architectures, new training procedures of neural networks, and advances in hardware, especially the general-purpose graphics processing units (GPGPUs) [Valero 2016], which demonstrated to be a suitable environment for training CNNs. To develop applications based on CNNs, users commonly gather data, annotate these data, train a model with the data, and evaluate this model in a cycle as represented in Fig. 1. Depending on the results of the evaluation, the model is retrained (revised, by varying hyperparameters), or the data has to be revised. To move from one cycle step to another, the user evaluates intermediate data and log data generated by each step. At a large scale, after many cycles, it becomes difficult to browse and analyze data from all the steps with different file formats and representations, as shown in Fig. 1a. To analyze data, programs must be written to relate these different files. Recent approaches use a provenance

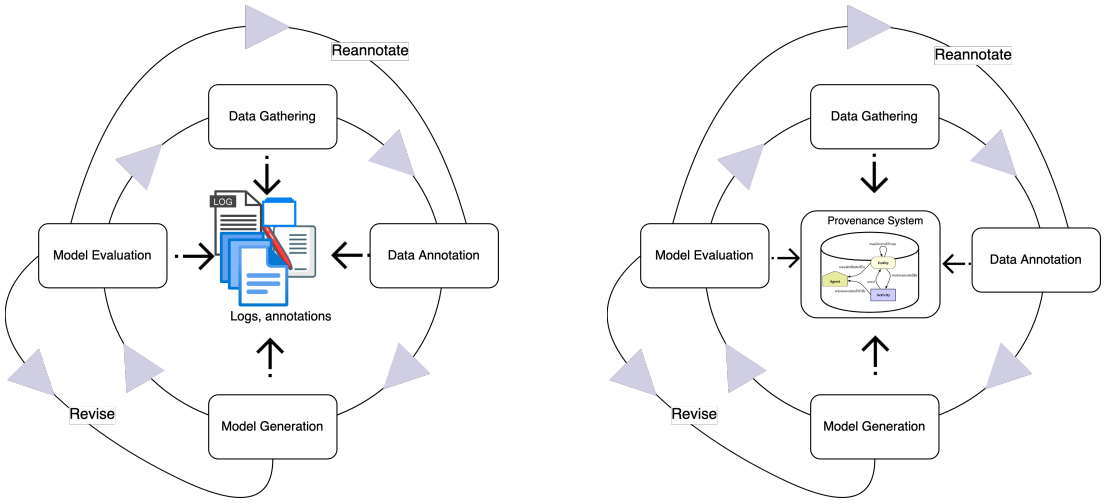


Fig. 1: (a) Life cycle of a CNN application. (b) Life cycle of a CNN application using a provenance database.

database to integrate data from these cycle steps and improve data analyses and decisions (Fig. 1b). In fact, according to [Fekry et al. 2020], this cycle is being adopted as a new development paradigm in many domains.

The performance of a CNN depends on the input dataset and the choice of a set of hyperparameters (a type of parameter that one can use to influence the learning process). However, the user cannot adjust the input dataset in most cases, so the only "control" the user has to improve the quality of the trained model is the proper choice of hyperparameter values. Several hyperparameters need to be set when a user develops a CNN-based application. Examples of commonly effective hyperparameters include the learning rate (α - scales the magnitude of the weights update), batch size (defines the number of subsamples given to the network at each iteration), number of epochs (defines the number of times the training data is provided to the network during the training process), momentum (β - defines the direction of the next step considering previous knowledge), and dropout rate (defines the number of units that are temporarily removed during a training step to avoid overfitting). Choosing the proper values for these hyperparameters is a top priority, yet far from trivial. The reason CNNs are difficult to configure is that several hyperparameters need to be set with a large range of possible values for each one of them and CNNs are particularly sensitive to hyperparameter settings [Hoos and Leyton-Brown 2014]. Besides, several models require a non-negligible amount of time to be trained and a poor hyperparameter configuration can lead to undesired results and time wasted.

In the worst case, when the user does not have any preference over a subset of the hyperparameters, the model should be trained for a large combination of hyperparameter values. Each combination must be registered for future analysis. Experiment tracking tools such as Weights & Biases [Biewald 2020] may help in recording and visualizing the experiment tries. However, exploring and choosing the hyperparameters' values remain a non-trivial exploration process, due to the large search space. There are a plethora of available methods to set hyperparameters for a CNN, *e.g.*, manual search, Grid Search, *Cloud Auto ML*[1], and Bayesian Optimization [Badan and Sekanina 2019]. Such methods are implemented in well-known frameworks such as scikit-learn[2]. Let us take as an example the Grid Search method [Liashchynskyi and Liashchynskyi 2019]. Grid Search generates the model and evaluates it for a specific combination of hyperparameters. It repeats the process for a pre-defined set of hyperparameter combinations. In the end, the combination of hyperparameters values that

[1] `https://cloud.google.com/automl`
[2] `https://scikit-learn.org/`

produced the best score (*e.g.*, accuracy) is chosen.

These methods represent a step forward, but they lack support for user participation [Chevalier-Boisvert et al. 2019; Wang et al. 2019]. Often, based on data analysis, users gain insights that allow reducing the hyperparameter search space. Solutions that present some user support to evaluate which hyperparameter values led to which results adopt a proprietary data representation [Schelter et al. 2017; Tsay et al. 2018]. These proprietary representations of the data derivation path make exploratory data analysis difficult. Thus, the results of the training process on different tools require additional analysis and implementation to be further compared, as there is no uniform way to represent the choice of hyperparameters that leads to the best model. Furthermore, it is not trivial to verify if two models have been trained with the same hyperparameter values.

One possible alternative to this problem is to represent the hyperparameters' configuration derivations using provenance data. In this case, one can use recommendations such as W3C PROV [Moreau and Groth 2013], which represent provenance data and aim at fostering interoperability. Representing the metadata associated with the training process of a CNN in a provenance repository simplifies associating hyperparameter values and input data with the trained models. Additionally, by leveraging the provenance data, one can improve both hyperparameter tuning and online data analysis. Also, the evaluation of the various hyperparameters requires the relationship between several types of data (*e.g.*, domain-specific data, metrics such as accuracy, metadata from the computational environment used for CNN training [Zaharia et al. 2018]). Correlating hyperparameters and the results is an analysis-intensive task, that requires user expertise and data from several training runs. In addition, queries for evaluating hyperparameter values consumed during the training of a CNN resemble the typical provenance queries already found in different systems [Silva et al. 2018; Silva et al. 2020; Godoy et al. 2020].

In a previous work [Pina et al. 2019], we presented CNNProv, an approach for automatic capture of provenance from CNN applications, based on DfAnalyzer [Silva et al. 2020]. We specialized CNNProv into Keras-Prov [Pina et al. 2020] using the Python DL API Keras[3]. Keras-Prov avoids code annotations from the user, as required in CNNProv, and automatically captures typical hyperparameter values from Keras. Like CNNProv, Keras-Prov keeps compliance to W3C PROV, which eases interoperability. Keras-Prov tracks data transformations and datasets related to CNN training hyperparameters and metrics automatically. Provenance data and user adaptations in configurations are stored in a database that allows for submitting queries during the training. Although Keras-Prov represents a step forward, it lacks relevant features for supporting online provenance analysis.

In this paper, we extend Keras-Prov into Keras-Prov++, to improve online analysis and fine-tuning. Keras-Prov++ provides an analytical dashboard to support users in examining the CNN training online. This visual representation complements database queries by showing the behavior of hyperparameters with the corresponding metrics evolving with epochs. It helps to identify the relationship between a specifically chosen hyperparameter with accuracy and loss, improving the online fine-tuning. This paper goes deeper in presenting Keras-Prov [Pina et al. 2020], including a broader discussion on related work with background details, and introducing Keras-Prov++ with its added visual support for hyperparameter analysis.

In addition to using the well-known CNN AlexNet [Krizhevsky et al. 2017] case study with Keras-Prov++, this paper explores a dense CNN surrogate model in a real scientific application called DenseED [Freitas et al. 2021]. These case studies show how Keras-Prov++ provides new insights into hyperparameter tuning and evidences the relationship of hyperparameter values with metrics such as accuracy. The experiments show how user adaptivity can reduce the number of epochs previously set and consequently the overall training time. This work contributes to a W3C PROV provenance-based architecture that can benefit from several visual data analysis tools. The architecture is designed to

[3]`https://keras.io/`

work with the DL framework chosen by the user, in this case, Keras.

The remainder of this paper is structured as follows. Section 2 presents background on hyperparameters, Section 3 introduces the proposed approach. Section 4 details the case studies, and the experimental evaluation. Section 5 discusses related work. Finally, Section 6 concludes.

2. THE ROLE OF HYPERPARAMETERS IN DEEP LEARNING

The popularity of DL and CNNs has increased at a fast pace over the last ten years [Li et al. 2020]. According to [LeCun et al. 2015], DL methods aim at learning representations from input raw data and using such representations to automatically learn how to solve a task. The representations are obtained through the composition of non-linear functions that transform the representation at each level, starting with the raw input at the lower level to a higher and slightly more abstract level.

Hyperparameters are key components in DL since choosing their values properly may strongly influence the quality of the outcome of an algorithm. According to [Bengio 2012], a hyperparameter is defined as "*a variable to be defined before the actual application of a given learning algorithm to the data, being that variable not directly selected by the learning algorithm itself*". Tuning hyperparameters in DL is particularly challenging since the usual trial-and-error process of experimenting on them is much more expensive in complex and highly dependent models such as deep CNNs [Hoos and Leyton-Brown 2014].

Algorithms for training deep models include a plethora of hyperparameters, such as the number of epochs, optimizers, momentum, learning rate, *etc.* Following, we briefly explain some of the most common hyperparameters. The number of epochs hyperparameter defines how many times an entire dataset is passed forward and backward through the network. When this hyperparameter value increases, the number of times the weights in a network are adjusted also increases and the tendency is that the curve goes from underfitting to optimal. One has to be careful to not pass through the optimal point and reach overfitting when the number of epochs is larger than required. The batch size is the number of examples that are presented to the network during the training. Properly defining the batch size may be a hardware requirement due to the size of the GPU's memory but can also lead to better generalization. The learning rate hyperparameter controls how quickly or slowly the neural network weights are adjusted. The optimizer hyperparameter defines the algorithm or method used to adjust the weights of the network, for example, by computing adaptive learning rates for each parameter [Ioffe and Szegedy 2015]. The output of a loss function works as a guide to optimizers for them to update the parameters of the model. There are several available optimizers. Following, we provide some details about the optimizers available in Keras:

—**SGD** (Stochastic Gradient Descent) is an extension of the gradient descent algorithm that is less expensive computationally. Thus, while gradient descent needs to load an entire dataset of n-points to compute the derivative, SGD computes derivatives for each point;

—**RMSprop** (Root Mean Square Propagation) maintains a moving (discounted) average of the square of gradients and divides the gradient by the root of this average. This optimizer uses plain momentum;

—**Adagrad** adaptively scales the learning rate for each parameter during the training, those updates are relative to the frequency of updates a parameter receives [Duchi et al. 2011];

—**Adadelta** is an extension of Adagrad that allows for a per-dimension learning rate method for SGD. Its main advantage is that it does not require setting a default learning rate method [Zeiler 2012];

—**Adam** [Kingma and Ba 2014] is a stochastic gradient descent method that is based on the adaptive estimation of first-order and second-order moments. Adam requires less memory and is well suited for problems that are large in terms of data and/or parameters;

—**Adamax** is a variant of Adam, based on infinity norm [Kingma and Ba 2014];
—**NAdam** is the optimizer Adam with Nesterov momentum [Dozat 2016]. NAG (Nesterov adaptive gradient) is superior to traditional momentum. So, NAdam is the NAG incorporated to Adam.

A hyperparameter that is often used with some optimizers is Momentum. Momentum is an optimization technique that instead of using only the gradient of the current step to guide the search, like in SGD, it also accumulates the gradient of the past steps to determine the direction to go [Ruder 2016], *i.e.,* it replaces the gradient with a momentum which is the aggregation of gradients. This hyperparameter aims at speeding up learning and avoiding getting stuck in local minima.In addition to the common hyperparameters, the user might set custom hyperparameters according to the network architecture like the number of layers of a specific block, or custom metrics such as loss function with multiple coefficients.

3. AN INTRODUCTION TO KERAS-PROV++

This section presents Keras-Prov++ showing its architecture in section 3.1, and how to use it in section 3.2.

Keras is an API for the well-known and popular TensorFlow[4] library. Keras-Prov++ is a library with a Python interface to provide provenance data to Keras DL applications. The idea behind Keras-Prov++ architecture is to maintain the original Keras core (with its multiple layers) while registering online the values of hyperparameters and their relationships as provenance data. Keras-Prov++ components' architecture acts as provenance plugins to the software that executes the DL, in this case, Keras API. In this approach, the user can continue using Keras without having to run the neural network training under a portal or tool. Keras-Prov++ has a lightweight provenance capture and storage with a negligible computational overhead to the DL execution.

3.1 Keras-Prov++ architecture

The architecture of Keras-Prov++ is shown in Fig. 2 with its three main layers: (i) Training Layer, (ii) Data Layer, and (iii) Analysis Layer.

The *Training* Layer is where Keras core executes and interacts with libraries such as TensorFlow. It is important to note that the original Keras functionalities are not modified. Keras-Prov encapsulates Keras Core so that a component named *Provenance Extractor* can have access to the hyperparameter values to capture them. To provide extensibility, a class called *Provenance* is added to Keras, containing methods that capture data transformations of the DL application. Keras has a class called *Model* and in this class, a method to capture provenance data was added. This method expects a variable named *dataflow_tag* to identify the dataflow (data transformations), and the list of hyperparameters to be captured. If there is an adaptation of the hyperparameters during the training (*e.g.*, update in the learning rate), Keras-Prov uses Keras methods such as *LearningRateScheduler* and *ReduceLROnPlateau* to register the tuning in the provenance database. Then, as Keras executes, Keras-Prov automatically identifies the data transformations in the DL application, the hyperparameters of interest, and the values used in each training to be captured. *Provenance Extractor* interacts with Keras methods to capture these data and to send them asynchronously to the *Data Layer* to manage provenance data. *Data Layer* is prepared to be executed in a different computer node to manage provenance data without competing with the DL execution resources. The *Analysis Layer* of Keras-Prov aims at producing provenance graphs in different representations.

The *Data Layer* is the main component of Keras-Prov. It receives provenance data and uses a DBMS to store it. We chose to use MonetDB, a columnar DBMS, to manage provenance data due

[4]www.tensorflow.org

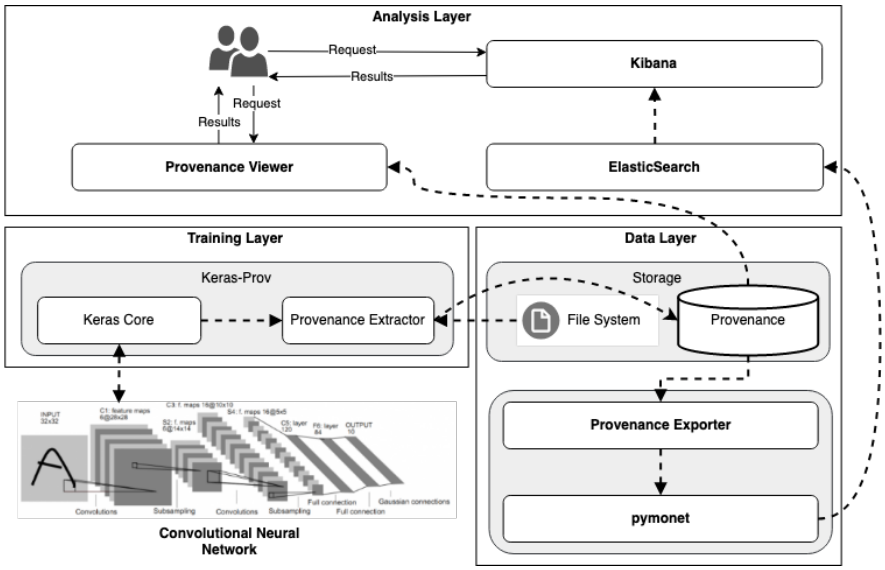


Fig. 2: Architecture of Keras-Prov++

to its efficiency in analytical queries and data ingestion. MonetDB has also scientific data support with plugins for the R library. By choosing a relational DBMS like MonetDB, visualization tools and Python plugins are available to be incorporated by the *Analysis Layer*. The database schema follows DfAnalyzer's PROV-Df representation [Silva et al. 2016], with extensions to model CNN metadata and hyperparameters [Pina et al. 2021]. This relational database schema corresponds to the W3C PROV prospective provenance representation, as shown in Fig. 3. Provenance data received at the *Data Layer* is structured according to this database schema and ingested in the provenance database to provide the corresponding retrospective provenance.

Fig. 3 follows W3C PROV-N notation that is based on three core concepts of Entity, Activity, and Agent. An entity is a physical, digital, or conceptual representation of a thing. In the context of this paper, an example of an entity may be data in a file, entities are represented as yellow ellipses. Activities are actions (processes) that occur in a specific period and act upon entities. The training process of a CNN may be considered an activity, an activity is represented by a blue rectangle. An agent is a person or software agent that is responsible for performing activities, own an entity, etc. Agents are represented as orange pentagons. This representation follows a graphic notation familiar to W3C PROV users, which facilitates comparison and interoperability.

In Fig. 3 *Training*, *Adaptation* and *Testing* are activities. *Training* consumes (arrow *used*) a series of *AttributeValues* such as the name of the optimizer, the hyperparameters learning rate, number of epochs, and number of layers in the network, and produces (arrow *wasGeneratedBy*) a set of metrics, such as the accuracy, the value of the loss function, the elapsed time and the date and time of the end of the execution of each epoch. *Adaptation* consumes (arrow *used*) the data produced by the previous transformation (*Training*), and hyperparameter values such as new learning rate and produces (arrow *wasGeneratedBy*) the value of the epoch and the date and time when the adaptation occurred, in addition to identification for the adaptation. Finally, *Testing* provides data on the evaluation of the model according to the training dataset and outputs the accuracy and loss function values. Entities represent data related to the training, divided into the classes *ds_ itrainingmodel*, *ds_ otrainingmodel*, *ds_ iadaptation*, *ds_ oadaptation*, and *ds_ otestingmodel*, which follow the description of the activities previously presented (*Training*, *Adaptation*, and *Testing*).

There are several advantages in choosing W3C PROV. The provenance data captured online can be exported using a W3C PROV-compliant representation, *e.g.*, JSON, as presented in Fig. 4. Keras-

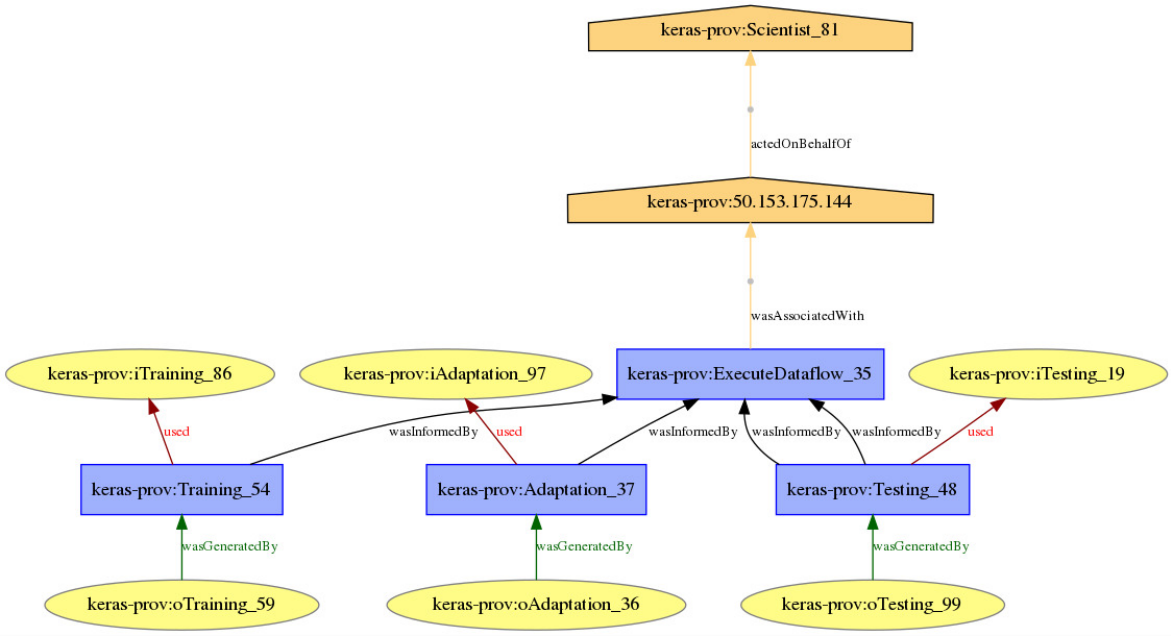


Fig. 3: Keras-Prov provenance representation with W3C PROV notation.

Prov++ generates the representation in Fig. 3 using Prov Python[5] and Graphviz[6], and still, validates the W3C PROV compliance using ProvValidator[7].

To empower the *Analysis Layer*, Keras-Prov++ has a dashboard for analyzing captured provenance data. Four new components were added to Keras-Prov to become the Keras-Prov++ architecture: (i) Provenance Exporter, (ii) pymonet, (iii) ElasticSearch, and (iv) Kibana, as presented in Fig. 2. The Provenance Exporter aims at extracting provenance data from the provenance database and sends it to the pymonet[8]. Pymonet ingests data into ElasticSearch periodically during execution. ElasticSearch is a search engine built on top of the Lucene library, it can search several kinds of documents and provides scalable search. Due to the volume of captured provenance data, ElasticSearch is a natural choice for searching data. Then, ElasticSearch is accessed by Kibana to produce the Keras-Prov++ Dashboard. Kibana aims at providing rich visualization capabilities on top of the content indexed on ElasticSearch (*i.e.*, provenance data). Data analysis is not real-time, since ElasticSearch only updates its information within a certain interval, but is fast enough for runtime tuning. Finally, the *Provenance Viewer* component generates a visual representation of the provenance graph and accesses the provenance database, to simplify the user analysis.

Although similar data is stored in Keras logs, this comparative and visual analysis would become much more labor-intensive requiring preprocessing of logs for ElasticSearch ingestion. Adopting W3C PROV representation, documents like PROV-N can be generated and post-processed by libraries such as Prov Python libraries.

3.2 Keras-Prov++ in action

To use Keras-Prov++ the user needs to download, install and start the library from `https://github.com/hpcdb/keras-prov`, DfAnalyzer, MonetDB, ElasticSearch, and Kibana. Once all those compo-

[5]https://prov.readthedocs.io/en/latest/prov.html
[6]http://www.graphviz.org/
[7]https://openprovenance.org/services/view/validator
[8]https://pypi.org/project/pymonet/

```
{
  "prefix": {
    "keras-prov": "http://github.com/dbpina/keras-prov"
  },
  "entity": {
    "keras-prov:Dataflow": {"prov:type": "Plan",
      "ex:creator": "dbpina"
    },
    "keras-prov:File": {"prov:type": "File",
      "ex:path": "/usr/dbpina/IMG410.png",
      "ex:creator": "dbpina"
    },
  },
  "agent": {
    "keras-prov:Machine": {"prov:type": "prov:SoftwareAgent",
      "ex:name": "127.0.0.1"},
    "keras-prov:Scientist": {"prov:type": "prov:Person",
      "ex:name": "dbpina"}
  },
  "activity": {
    "keras-prov:ExecuteDataflow": {"prov:startTime": "2021-03-07T21:34:33.555472"}
  },
  "actedOnBehalfOf": {
    "_:id17": {
      "prov:delegate": "keras-prov:Machine",
      "prov:responsible": "keras-prov:Scientist"

Continues....
```


Fig. 4: A fragment of the PROV-compliant JSON file.

nents are installed, the user can execute the DL application with Keras. One only has to change the code by adding the *provenance* method, as shown in Fig. 5, to set the hyperparameters to be captured. When this code is not added, the execution proceeds with Keras and without provenance capture. In the instantiation of the method *provenance*, the attributes *dataflow_tag* and *hyps* are mandatory. If all the attributes inside the variable *hyps* are set as False, Keras-Prov++ will register the CNN architecture and the adaptation values if this variable is set as True. To access the captured data the user interacts with the Provenance Viewer or, the Keras-Prov++ dashboard, both through a browser.

Behind this hyperparameters setting, Keras-Prov uses the predefined provenance representation, so that provenance data is captured and stored accordingly. Relationships between hyperparameters (*e.g.*, optimizer and learning rate) can be queried as they are ingested in the database. In case the user needs to analyze additional metadata or domain data, one can instrument the code for this purpose. For more information about the instrumentation in Keras-Prov please visit `https://github.com/hpcdb/keras-prov`.

The user can access the Keras-Prov++ database and make analyses through the Keras-Prov++ Dashboard and the Provenance Viewer, during and after the training execution. The Keras-Prov++ Dashboard is customizable and the user can set which available data to use to generate the charts and also a time interval for ElasticSearch to refresh its information. In addition to monitoring with Kibana, the Provenance Viewer allows the user to query the specifications of dataflows that have already been registered in the database, without having to write SQL. For example, the user can obtain directly the execution time of each epoch, learning rate, with the current accuracy for each epoch, the adaptations applied to the training configuration, and at which point they happened. The Provenance Viewer component uses a graphical interface that presents tables and views (predefined joined tables) as datasets with attributes for the users to define filters and aggregates. It is necessary

```
hyps = {
    "OPTIMIZER_NAME": True,
    "LEARNING_RATE": True,
    "DECAY": False,
    "MOMENTUM": False,
    "NUM_EPOCHS": True,
    "BATCH_SIZE": False,
    "NUM_LAYERS": True}

model.provenance(
    dataflow_tag="keras-alexnet-df",
    adaptation=True, hyps = hyps)
...
```

Fig. 5: A Fragment of Keras-Prov Code

to specify the datasets that will be used in the analysis, the attributes for the projection clause, and the attributes for the selection clause. Then, this component uses these arguments to generate the corresponding SQL query, which is sent to the Data Layer to be executed by MonetDB on the provenance database. In sections 4.2 and 4.3 we show examples of Keras-Prov++ in action.

4. EVALUATING HYPERPARAMETERS USING KERAS-PROV++

This section presents KerasProv++ in action with experiments that show the potential of provenance data in hyperparameter analyses in two DL applications. It is worth mentioning that these analyses were performed online with the user submitting queries to the Keras-Prov++ Dashboard and the Provenance Viewer. During the execution, the provenance database is ingested with provenance data from the training, test, and validation phases, as they evolve.

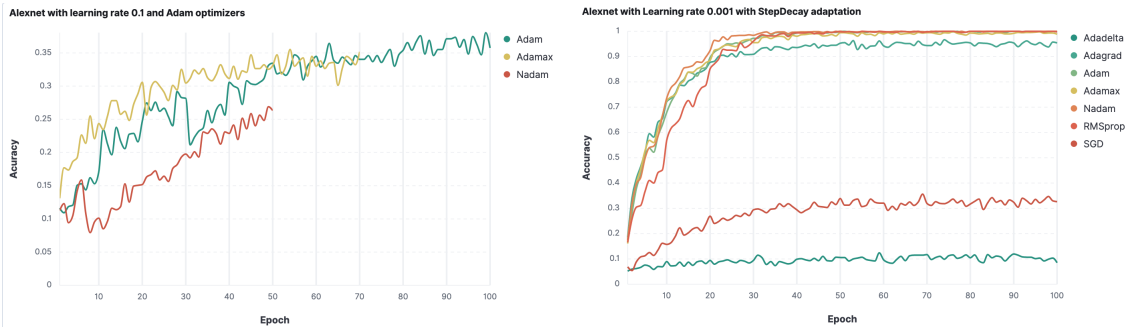
4.1 Experiment Setup

Our experiments were performed on the Lobo Carneiro computer cluster, also known as LoboC, from the High-Performance Computing Center (NACAD) at COPPE/UFRJ. LoboC is an SGI ICE-X Linux cluster with 504 Intel Xeon E5-2670v3 (Haswell) CPUs, totaling 6,048 processors. The processors feature Hyper-Threading (HT) technology, offering 48 processing threads per node, with 64GB of RAM. Compute nodes are interconnected with InfiniBand FDR–56 Gbs (Hypercube) technology. The cluster runs under a shared disk architecture, with an Intel Luster parallel file system with 500TB storage capacity.

The experiments are done with MonetDB, instantiated in a dedicated computational node to manage the provenance database. Keras-Prov++ receives HTTP requests with data to be stored and then establishes a connection with MonetDB through the JDBC driver to ingest data. The first case study, Alexnet, is executed with two nodes and the second case study, DenseED, used four nodes at LoboC.

4.2 Case Study: Alexnet

AlexNet [Krizhevsky et al. 2012] is a CNN for image classification, with a focus on high-resolution images. AlexNet classifies images into more than 1,000 categories. The network has an image input size of 227 X 227. The architecture of AlexNet consists of eight layers: five convolutional layers and three fully connected layers. In addition, as an activation function, AlexNet uses Rectified Linear Units (ReLU) instead of the Hyperbolic tangent (tanh), which was standard at the time, and this reduced the training time.



(a) (b)

Fig. 6: (a) Accuracy of Alexnet with initial learning rate of 0.1 and optimizers Adam, Adamax and NAdam and (b) Accuracy of Alexnet with initial learning rate of 0.001 and optimizers available in Keras, and adaptation with StepDecay function.

In its seminal paper is discussed that in the lowest layers of the network, the model learns feature extractors that resembled traditional image processing filters. The layers in the middle can represent larger structures such as eyes or noses and the higher layers can represent entire objects like flowers, people, airplanes, *etc.* The final hidden layers learn a compact summarized representation of the image content in a way to differ diverse categories. Due to this, AlexNet won the 2012 annual challenge of ImageNet, which is a challenge focused on creating methods and learning architectures for the classification of the ImageNet database [Deng et al. 2009]. Thus, AlexNet is widely recognized as the architecture that popularized CNNs for computer vision.

In the experiments presented in this section, we trained Alexnet using the Oxford flowers [Nilsback and Zisserman 2006] dataset. This dataset contains images of flowers belonging to 17 different categories. The images are acquired by searching the Web. There are 80 images available for each category. All executions were performed with 100 epochs, the optimizers used were SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, and NAdam, and to evaluate and score each experiment we use accuracy and loss as reference metrics. All other hyperparameters were set with default values.

Several experiments have been conducted with the Oxford dataset to analyze the behavior of different optimizers and learning rates. To analyze each combination, we split the training dataset into 80%/20% to train and validate the dataset. In all experiments, we used the default settings of each optimizer and the size of all images was constant. All data presented in this subsection was captured by Keras-Prov++ and shown with its dashboard.

Figures 6a and 6b show the training accuracy of different executions with Alexnet. In the first execution (Fig. 6a) the user chooses Adam, Adamax, and NAdam optimizers with a learning rate of 0.1. In Fig. 6a, around epoch 30, the accuracy values of the network when using ADAM optimizer are much lower than expected, which suggests further investigation. The user queries the current learning rate value and decides to drop it from 0.1 to 0.05. After this action, the user keeps following the execution through the dashboard. After epoch 40, a comparison between the running optimizers suggests that NAdam's execution is not worth continuing. At epoch 50, Fig. 6a shows that accuracy is still low for Adam and Adamax, so a new learning rate update is applied by the user, who decides to drop it again from 0.05 to 0.025, without any improvement. Since, by epoch 70, Fig. 6a does not suggest much difference between the training accuracy of Adamax and Adam, the training of the Adamax optimizer is also interrupted by the user. Those actions have an impact on energy and resource consumption. When the execution of Adam ends, the user decides to try again with other optimizers with a learning rate of 0.001, using StepDecay as a learning rate adaptation function, this case is shown in Fig. 6b. StepDecay is a function that drops the learning rate value by a factor every n epochs, in this case, the factor is defined as 0.5 and $n = 10$. The execution of Alexnet with a smaller learning rate and different optimizers are shown in Fig. 6b.

Table I: Q1: What is the loss value of epoch 30 of each optimizer in Alexnet with learning rate 0.001?

Optimizer	Loss	Validation loss
Adadelta	3.3	2.75
Adagrad	0.29	2.64
Adam	0.09	2.07
Adamax	0.14	2.16
NAdam	0.07	2.08
RMSprop	0.11	1.76
SGD	2.3	2.27

Table II: Q2: List the layers of the model.

Name	Attribute type	Value
activation_1	activation	relu
activation_2	activation	relu
activation_3	activation	relu
activation_4	activation	relu
activation_5	activation	relu
activation_6	activation	relu
dropout_1	dropout	0.5
activation_7	activation	relu
dropout_2	dropout	0.5
activation_8	activation	relu
dropout_3	dropout	0.5
activation_9	activation	softmax

Table III: Q3: What are the average time and average loss for the NN's training with each optimizer?

Average time	Average Loss	Optimizer
23.46	2.68	Adadelta
22.48	1.41	Adagrad
23.27	2.24	Adam
23.11	2.84	Adamax
23.55	2.07	NAdam
22.83	2.15	RMSprop
22.41	1.95	SGD

Through the charts shown in the dashboard, the user might have a clear perspective of each run. Even though a visual tool is useful when there is a need to analyze a significant number of values, query analysis can be more effective to investigate specific values. Using the Provenance Viewer, the user can submit queries such as Q1: "What is the loss value of epoch 30 of each optimizer in Alexnet with learning rate 0.001?", Q2: "List the layers of the model.", Q3: "What are the average time and loss for training each epoch?". The result of these queries is presented in Tables I, II, III and can be retrieved through the Provenance Viewer.

Q1 (Table I) is a query that helps decision-making, such as changing a learning rate or dropout values. Q2 (Table II) shows the activation and dropout layers of AlexNet. This is important because, in some trials, the user may change the dropout value or the activation. To be able to present these differences between trials, Keras-Prov++ captures and stores the name that identifies the layer (e.g., activation_1, dropout_1), the type of the layer (activation or dropout), and the value of this layer (e.g., the value for activation is relu, the value for dropout is 0.4). Q3 (Table III) can be used to detect execution anomalies, such as a NN training that takes longer than expected with a specific optimizer

4.3 Case Study: DenseED

Our second case study uses the neural network DenseED and its datasets, as proposed by [Freitas et al. 2021]. DenseED uses a Physics-guided CNN as a surrogate model to enable the quantification of uncertainties [Zhu and Zabaras 2018]. The network architecture is a dense CNN as shown in Fig. 7. DenseED aims to replace the calculation of the Reverse Time Migration (RTM) equations with a trained model. In this way, the RTM calculation that would have to be performed for each probability distribution can be reduced. DenseED's architecture adopts a fully convolutional Bayesian encoder-decoder network. The number of dense layers in DenseED is variable and its evaluation metric is the Mean Squared Error (MSE). This case study requires more data to be analyzed than the Keras-Prov++ default hyperparameter values. Therefore, it required additional modeling and instrumentation to track the model's architecture. The provenance graph representation of the dense blocks and their transformations can be seen in Fig. 8.

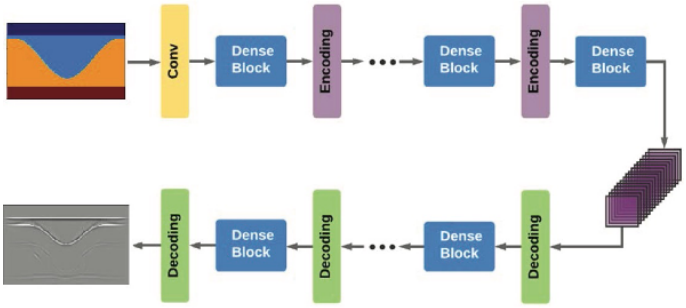


Fig. 7: DenseED: Deep Convolutional Encoder-Decoder network architecture from [Freitas et al. 2020]

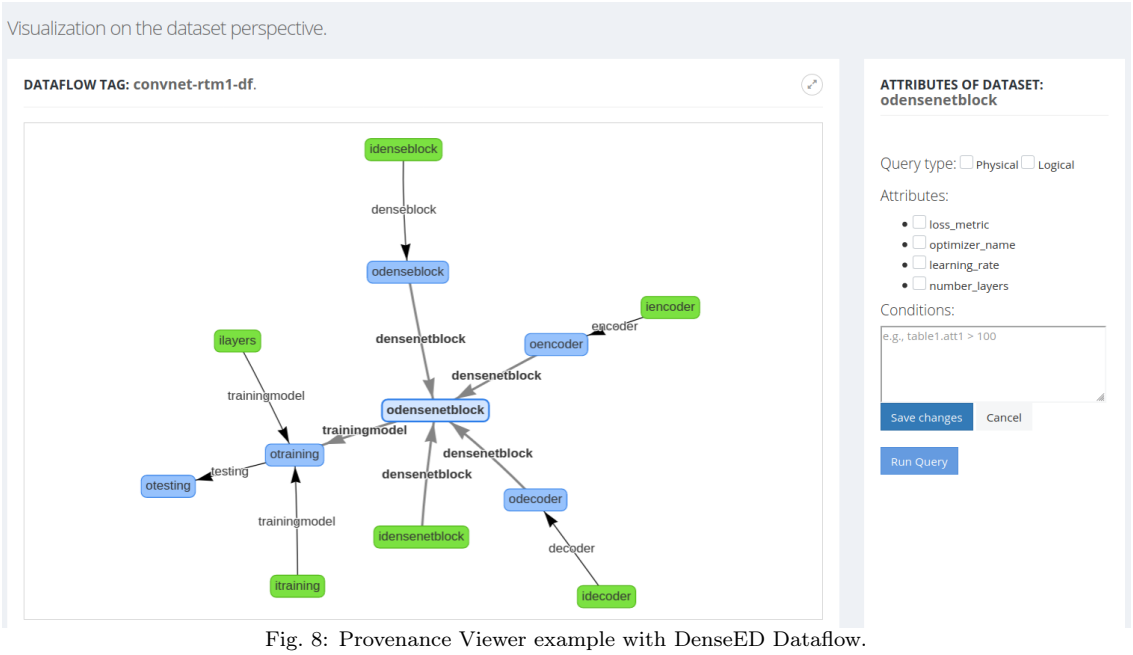
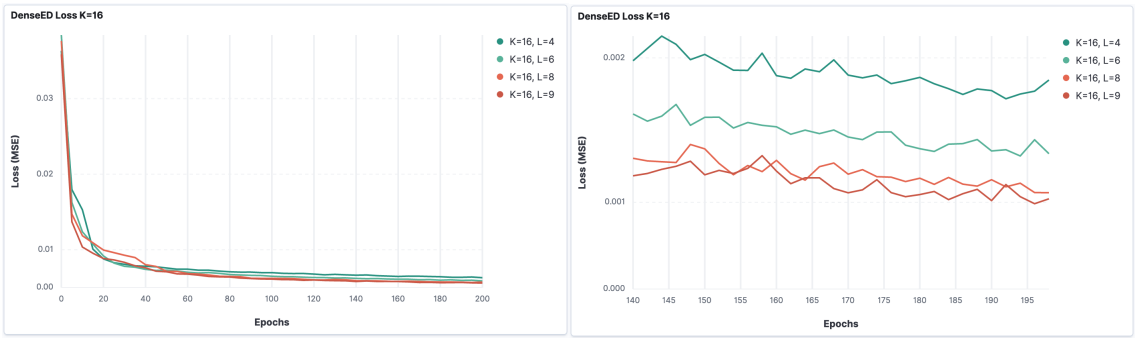


Fig. 8: Provenance Viewer example with DenseED Dataflow.

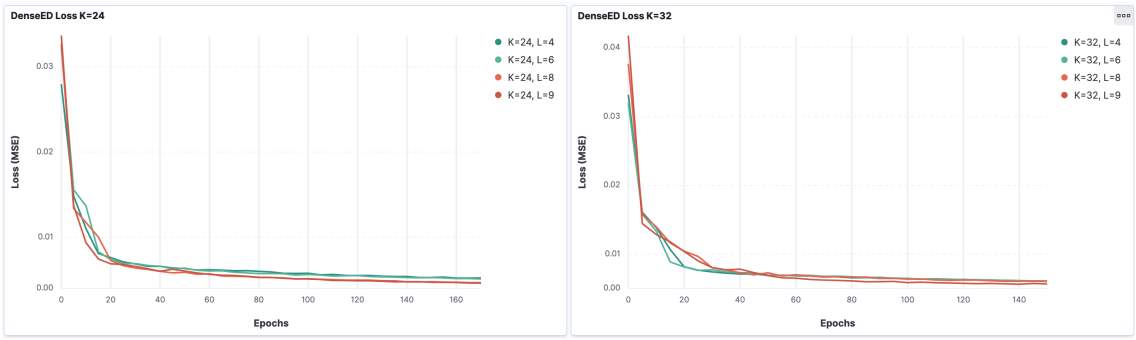
As reported in [Freitas et al. 2021], the user conducted several hyperparameter fine tunings. Without provenance, it is time consuming to gather the different training executions to compare and register adjustments. In this case study we reexecuted some of these trainings with the provenance support from Keras-Prov++ to show its analytical support. The input dataset of DenseED contains 100,000 velocity fields. From the input dataset 12000 samples were randomly selected for the training and 1000 for the testing set. In the validation of this NN, 50% of the training set was used. This neural network generates a surrogate model for RTM, thus, the expected output is a seismic image. The loss in this model is the Mean Squared Error (MSE) and it is monitored during the training and the testing phases, where the Coefficient of Determination (R^2) is the main metric in the test phase. The training explores alternatives for the Growth Rate ($k = (16, 24, 32)$) and the number of layers in the dense block ($l = (4, 6, 8, 9)$). All these data are stored in the provenance database.

In Figures 9a, 10a and 10b, we show the DenseED training results with different k and l values. In the cases explored, after finishing the training of combinations where $k = 16$, the user observes in Fig. 9a that at epoch 140 and beyond, there is a tendency that MSE values stay within a range of 0 and 0.001, zooming the chart (Fig. 9b) and querying the database, this tendency is confirmed. Assuming the same behavior for the combinations of $k = 24$, he decides to decrease the number of epochs from 200 to 170 epochs. As expected, in Fig. 10a, the MSE from epoch 140 stabilizes within



(a) (b)

Fig. 9: (a) Graphical view of the training results of DenseED with $k = 16$ and $l = (4, 6, 8, 9)$ (b) Zoomed view of the training results for DenseED with $k = 16$ and $l = (4, 6, 8, 9)$ from epoch 140 to epoch 200.



(a) (b)

Fig. 10: (a) Graphical view of the training results of DenseED with $k = 24$ and $l = (4, 6, 8, 9)$ (b) Graphical view of the training results of DenseED with $k = 32$ and $l = (4, 6, 8, 9)$.

values 0 and 0.001, and then the user trains the $k = 32$ combinations with 150 epochs. These changes in the number of epochs to be trained decrease the resource consumption and the execution time of combinations of $k = 24$ and $k = 32$, with no significant change in MSE. Looking at these results through queries is also possible, but the amount of information presented in a table would take longer to generate insight and decision.

The user can submit analytical queries to fine-tune DenseED parameters like Q4: "What are the initial learning rate, optimizer and the number of layers when the training for DenseED achieved the highest R^2?", Q5: "Retrieve the combinations of k and l that achieved the top three MSE values and their R^2.", Q6: "What are the top five training MSE values and their elapsed time for each epoch when $k = 32$ and $l = 9$?", Q7: "What are the combinations of k and l that consume less time during the training?". Tables IV,V, VI, and VII show query results. Q4 (Table IV) shows the characteristics of the model that had the best value for R^2 and Q5 (Table V) might suggest that greater values of l lead to smaller MSE values. The results of Q6 (Table VI) might be a case where, if more values were required, a graphical representation would be more helpful. Q7 (Table VII) shows that the training time increases with the number of layers (l), this is important for the user to evaluate the tradeoff between time cost and the value of R^2. For DenseED, the user considers that $R^2 \geq 0.95$ as a satisfactory R^2. The combination $k = 16$ and $l = 4$ was chosen since it already reached the user's criteria and costs less time. Queries that filter and order results to show highest and lowest values, such as Q4-Q7, assist in evaluating extreme values and outliers as the training evolves. Without this data, keeping track of this progress associated with execution data and the hyperparameter set would be costly and error-prone.

Table IV: Q4: What are the initial learning rate, optimizer, and the number of layers when the training for the combination of k and l that achieved the highest R^2?

Initial Learning Rate	Optimizer Name	Number of Layers	k	l	R^2
0.01	Adam	118	32	9	0.9979

Table V: Q5: Retrieve the combinations of k and l that achieved the top three MSE values and their R^2

MSE	R^2	k	l
0.00093	0.9979	32	9
0.00103	0.9976	16	9
0.00111	0.9975	24	8

Table VI: Q6: What are the top five training MSE values and their elapsed time for each epoch when $k = 32$ and $l = 9$?

Epoch	Elapsed time	MSE
144	41.7	0.000589
138	41.93	0.000609
142	41.71	0.000612
141	41.74	0.000617
145	41.8	0.000619

Table VII: Q7: What are the combinations of k and l that consume less time during the training?

Average elapsed time	k	l
7.3	**16**	**4**
9.2	**24**	**4**
11	**32**	**4**
13.2	16	6
17.1	24	6
20.3	16	8
21.5	32	6
25	16	9
28.1	24	8
34.1	24	9
35.6	32	8
43.8	32	9

4.4 Threats to Validity

The case study of Alexnet aims at using a well-known neural network to show how outcomes may change varying one hyperparameter, the amount o data generated by each attempt, and how an online analysis (through charts and queries) may benefit the life cycle. DenseED case study is a real case and the variations explored were used in the process of designing this model.

We are yet to explore the heterogeneity of the execution environment, Alexnet and DenseED case studies were executed at Lobo Carneiro Supercomputer only. Studies to analyze the time and storage overhead of the functionalities associated with Keras-Prov++ Dashboard are yet to be performed. The current version of Keras-Prov++ is limited to Tensorflow 2.2. Also, information is limited by what can be provided by Keras, such as how to identify the NN architecture.

5. RELATED WORK

To decide whether the DL training should continue or try different configurations, the user needs to analyze online the hyperparameters most adequate to the training dataset, observing metrics such as accuracy and loss values. Three approaches provide provenance data to support DL hyperparameter analysis. The first is having provenance data as part of the DL system, like Keras logs, which is difficult to analyze. The second is to add a provenance system [McPhillips et al. 2015; Pimentel et al. 2016; Pimentel et al. 2017; Silva et al. 2020; Silva et al. 2018] to the DL application. However, these generic provenance approaches require that the user defines data representations and functions to explicit and analyze the relationships between hyperparameters and provenance data, for each DL application. The work of Souza et al. [Souza et al. 2021] supports the whole ML life cycle, including provenance support from several steps before the training. Souza et al. [Souza et al. 2021] present a rich provenance data representation, W3C PROV compliant, already associated with ML data, but the approach needs the user participation to instrument the DL application with the corresponding provenance capture library calls.

The third approach encompasses provenance-based hyperparameter analysis solutions that are coupled to DL systems. However, related work in this approach, like [Miao et al. 2015; Schelter et al. 2017; Tsay et al. 2018], adopt a specific provenance data representation, disregarding open standards and recommendations. Some of these tools require running the DL application under a portal or framework, limiting the choice of DL environment by the user. The approach of Keras-Prov is to provide a provenance library to be plugged into DL libraries to access hyperparameters and support online analysis. In addition, Keras-Prov is W3C PROV compliant. Next, we discuss the closest approaches to Keras-Prov and present a comparative table between them.

The HParams panel of TensorFlow[9] provides several tools to assist in defining the hyperparameters. The HParams panel allows for the user to list all executions and associate hyperparameters values and their metrics and generate charts (*e.g.*, parallel coordinates that show each execution as a line passing through an axis for each hyperparameter and metric) to evaluate the results. Although HParams panel represents a step forward, it does not allow for online analysis neither can import data from other approaches to perform comparisons (which is possible by using open standards such as PROV).

Neptune[10] is a lightweight analysis tool for DL experiments. It is similar to the HParams tool. It logs hyperparameters and metrics values from multiple executions and generates charts to compare results. One advantage of Neptune is that it can be integrated with notebooks, which eases the analysis process. It does not allow for online analysis (the data is also provided at the end of the training process, for example) neither can import data from other approaches to perform comparisons.

[Chatzimparmpas et al. 2020] propose VisEvol, a visual analytics tool that aims at supporting interactive analysis of hyperparameters in DL experiments. One of the advantages of VisEvol is that it allows for interventions during the training. Through exploration of the parameter value space, the user can create new models within the tool. VisEvol allows for online analysis, but only if the model is generated within it. If the users choose to use an external DL framework, it is not possible to capture the metadata for analysis. ModelHub [Miao et al. 2017] is a life cycle management system for DL which aims at storing models in different versions. ModelHub has a proprietary model for representing the training metadata of the neural network, which makes interoperability difficult. Runway [Tsay et al. 2018] aims at managing DL artifacts, such as models, data, and experiments. Runway allows for the user to track the model and data used during the training process. However, in addition to being a proprietary solution, it is restricted to the Python programming language. [Schelter et al. 2017] propose an automated tool to extract the metadata from the DL model and present it with interactive visualization to assist the comparison of experiments. However, the approach proposed by [Schelter et al. 2017] does not use standards, such as W3C PROV, affecting interoperability.

Table VIII presents a comparison between provenance-based hyperparameter analysis solutions and Keras-Prov. The column Hyperparameter selection refers to whether the tool assists in defining which hyperparameters are to be analyzed. The column Analysis interface refers to how the user queries and monitors DL data, through a graphical interface or a specific language. The column Data storage is related to how the captured data is being stored and represented, which in the case of a DBMS, can ease online querying. The column Provenance refers to whether the tool captures prospective provenance (p-prov) in addition to retrospective provenance (r-prov), and if it follows any known recommendation for provenance data representation. The column Online analysis refers to the possibility of online DL data analysis during the training execution.

Despite the importance and approaches for provenance support in DL applications, to the best of our knowledge, no solution provides online W3C PROV provenance data analysis using an independent provenance capture component with the DL system. The provenance capture component of Keras-Prov can be compiled to other DL systems, rather than having to execute under a specific platform.

[9] `https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams`
[10] `https://neptune.ai/`

Table VIII: Comparison of provenance support in DL training data analysis.

	Hyperparameter selection	Analysis interface	Data storage	Provenance	Online analysis
Hparams	Predefined	Graphical	Directory files	r-prov	No
Neptune	Predefined	Graphical	Metadata database	r-prov	No
VisEvol	Predefined	Graphical	Logs	r-prov	Limited
ModelHub	Predefined	Graphical, DLV	Directory files	r-prov	Yes
Runway	Predefined	Graphical	Cloudant DB (Document Store)	r-prov	No
[Schelter et al. 2017]	Customizable	Graphical	Document database	r-prov	No
KerasProv++	Customizable	Graphical, SQL	MonetDB (RDBMS)	p-prov, r-prov (W3C PROV)	Yes

Keras-Prov, through Keras-Prov++, presents visual graphic support to monitoring training data in addition to a query interface that generates SQL automatically for online analysis based on a W3C PROV compliant database.

6. CONCLUSION

In this paper, we presented Keras-Prov and its extension, Keras-Prov++, to improve online hyperparameter fine-tuning based on provenance analysis. Keras-Prov aims at reducing the effort to adapt and instrument the DL application code to capture provenance data during the training of CNNs. Keras-Prov++ builds on Keras-Prov using powerful online analytical features. By automatically capturing provenance data, it is possible to analyze the chosen hyperparameter values related to the training stages and adjust them to achieve results with more quality. Case studies have been conducted with the AlexNet CNN and the real CNN application DenseED surrogate model. We showed experiments where the analysis of hyperparameters during the training of CNNs has led to fine-tunings that improved the overall training. Our experiments showed how using the Keras-Prov++ approach for online provenance query analysis and monitoring can support decision-making.

As future work, we intend to use GPUs in the evaluations of Keras-Prov. We also plan to extend the solution to better assist the training of Physics Guided Neural Networks, where loss function specification and analysis are much different from typical CNNs. Also, we look forward to making this solution portable (even for HPC environments) using a hierarchy of containers to improve provenance management deployment.

ACKNOWLEDGEMENTS

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

REFERENCES

Agrawal, T. and Urolagin, S. 2-way arabic sign language translator using CNNLSTM architecture and NLP. In *BDET 2020: 2nd International Conference on Big Data Engineering and Technology, Singapore, January 3-5, 2020*. ACM, pp. 96–101, 2020.

Badan, F. and Sekanina, L. Optimizing convolutional neural networks for embedded systems by means of neuroevolution. In *TPNC 2019*. Vol. 11934. pp. 109–121, 2019.

Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*. Springer, pp. 437–478, 2012.

Biewald, L. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

Chatzimparmpas, A., Martins, R. M., Kucher, K., and Kerren, A. Visevol: Visual analytics to support hyperparameter search through evolutionary optimization. *CoRR* vol. abs/2012.01205, 2020.

Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., and Bengio, Y. Babyai: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019.

de Oliveira, G. B., Padilha, R., Dorte, A., Cereda, L., Miyazaki, L., Lopes, M., and Dias, Z. COVID-19 x-ray image diagnostic with deep neural networks. In *Advances in Bioinformatics and Computational Biology - 13th Brazilian Symposium on Bioinformatics, BSB 2020, São Paulo, Brazil, November 23-27, 2020, Proceedings*, J. C. Setubal and W. M. C. Silva (Eds.). Lecture Notes in Computer Science, vol. 12558. Springer, pp. 57–68, 2020.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255, 2009.

Dozat, T. Incorporating nesterov momentum into adam, 2016.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12 (7), 2011.

Fekry, A., Carata, L., Pasquier, T., Rice, A., and Hopper, A. To tune or not to tune? in search of optimal configurations for data analytics. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. New York, NY, USA, pp. 2494–2504, 2020.

Freitas, R. S., Barbosa, C. H., Guerra, G. M., Coutinho, A. L., and Rochinha, F. A. An encoder-decoder deep surrogate for reverse time migration in seismic imaging under uncertainty. *arXiv preprint arXiv:2006.09550*, 2020.

Freitas, R. S., Barbosa, C. H., Guerra, G. M., Coutinho, A. L., and Rochinha, F. A. An encoder-decoder deep surrogate for reverse time migration in seismic imaging under uncertainty. *Computational Geosciences* 25 (3): 1229–1250, 2021.

Godoy, W. F., Podhorszki, N., Wang, R., Atkins, C., Eisenhauer, G., Gu, J., Davis, P. E., Choi, J., Germaschewski, K., Huck, K. A., Huebl, A., Kim, M., Kress, J., Kurç, T. M., Liu, Q., Logan, J., Mehta, K., Ostrouchov, G., Parashar, M., Poeschel, F., Pugmire, D., Suchyta, E., Takahashi, K., Thompson, N., Tsutsumi, S., Wan, L., Wolf, M., Wu, K., and Klasky, S. ADIOS 2: The adaptable input output system. A framework for high-performance data management. *SoftwareX* vol. 12, pp. 100561, 2020.

Goulart, Í., Paes, A., and Clua, E. Learning how to play bomberman with deep reinforcement and imitation learning. In *Entertainment Computing and Serious Games - First IFIP TC 14 Joint International Conference, ICEC-JCSG 2019, Arequipa, Peru, November 11-15, 2019, Proceedings*, E. D. V. der Spek, S. Göbel, E. Y. Do, E. Clua, and J. B. Hauge (Eds.). Lecture Notes in Computer Science, vol. 11863. Springer, pp. 121–133, 2019.

Guérin, J., Thiery, S., Nyiri, E., Gibaru, O., and Boots, B. Combining pretrained CNN feature extractors to enhance clustering of complex natural images. *Neurocomputing* vol. 423, pp. 551–571, 2021.

Hoos, H. and Leyton-Brown, K. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*. pp. 754–762, 2014.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. PMLR, pp. 448–456, 2015.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* vol. 25, pp. 1097–1105, 2012.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60 (6): 84–90, May, 2017.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature* 521 (7553): 436, 2015.

Li, G., Lee, C. H., Jung, J. J., Youn, Y. C., and Camacho, D. Deep learning for EEG data analytics: A survey. *Concurr. Comput. Pract. Exp.* 32 (18), 2020.

Liashchynskyi, P. and Liashchynskyi, P. Grid search, random search, genetic algorithm: A big comparison for NAS. *CoRR* vol. abs/1912.06059, 2019.

Liu, Z., Yang, C., Huang, J., Liu, S., Zhuo, Y., and Lu, X. Deep learning framework based on integration of s-mask R-CNN and inception-v3 for ultrasound image-aided diagnosis of prostate cancer. *Future Gener. Comput. Syst.* vol. 114, pp. 358–367, 2021.

Matsugu, M., Mori, K., Mitari, Y., and Kaneda, Y. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks* 16 (5): 555–559, 2003. Advances in Neural Networks Research: IJCNN '03.

McPhillips, T. M., Song, T., Kolisnik, T., Aulenbach, S., Belhajjame, K., Bocinsky, K., Cao, Y., Chirigati, F., Dey, S. C., Freire, J., Huntzinger, D. N., Jones, C., Koop, D., Missier, P., Schildhauer, M., Schwalm, C. R., Wei, Y., Cheney, J., Bieda, M., and Ludäscher, B. Yesworkflow: A user-oriented, language-independent tool for recovering workflow information from scripts. *CoRR* vol. abs/1502.02403, 2015.

Miao, H., Li, A., Davis, L. S., and Deshpande, A. Modelhub: Lifecycle management for deep learning. *Univ. of Maryland*, 2015.

Miao, H., Li, A., Davis, L. S., and Deshpande, A. Towards unified data and lifecycle management for deep learning. In *2017 IEEE 33rd ICDE*. IEEE, pp. 571–582, 2017.

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

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.

Özbayoglu, A. M., Gudelek, M. U., and Sezer, O. B. Deep learning for financial applications : A survey. *Appl. Soft Comput.* vol. 93, pp. 106384, 2020.

Pimentel, J. F., Dey, S. C., McPhillips, T. M., Belhajjame, K., Koop, D., Murta, L., Braganholo, V., and Ludäscher, B. Yin & yang: Demonstrating complementary provenance from noworkflow & yesworkflow. In *Provenance and Annotation of Data and Processes - 6th International Provenance and Annotation Workshop, IPAW 2016, McLean, VA, USA, June 7-8, 2016, Proceedings*, M. Mattoso and B. Glavic (Eds.). Lecture Notes in Computer Science, vol. 9672. Springer, pp. 161–165, 2016.

Pimentel, J. F., Murta, L., Braganholo, V., and Freire, J. noworkflow: a tool for collecting, analyzing, and managing provenance from python scripts. *Proc. VLDB Endow.* 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 *International Provenance and Annotation Workshop, IPAW*, 2021.

Pina, D., Kunstmann, L., Oliveira, D., Valduriez, P., and Mattoso, M. Uma abordagem para coleta e análise de dados de configurações em redes neurais profundas. In *Anais do XXXV Simpósio Brasileiro de Bancos de Dados*. SBC, Porto Alegre, RS, Brasil, pp. 187–192, 2020.

Pina, D., Neves, L., Paes, A., de Oliveira, D., and Mattoso, M. Análise de hiperparâmetros em aplicações de aprendizado profundo por meio de dados de proveniência. In *Anais do XXXIV Simpósio Brasileiro de Banco de Dados*. SBC, Porto Alegre, RS, Brasil, pp. 223–228, 2019.

Ren, X., Fu, X., Zhou, X., Liu, C., Gao, S., and Peng, L. Bilingual word embedding with sentence combination CNN for 1-to-n sentence alignment. In *NLPIR 2020: 4th International Conference on Natural Language Processing and Information Retrieval, Seoul, Republic of Korea, December 18-20, 2020*. ACM, pp. 119–124, 2020.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Schelter, S., Böse, J.-H., Kirschnick, J., Klein, T., and Seufert, S. Automatically tracking metadata and provenance of machine learning experiments. In *ML Systems workshop*, 2017.

Silva, V., Campos, V., Guedes, T., Camata, J. J., de Oliveira, D., Coutinho, A. L. G. A., 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., Mattoso, M., and Valduriez, P. Dfanalyzer: Runtime dataflow analysis of scientific applications using provenance. *Proc. VLDB Endow.* 11 (12): 2082–2085, 2018.

Silva, V., De Oliveira, D., Valduriez, P., and Mattoso, M. Analyzing related raw data files through dataflows. *Concurrency and Computation: Practice and Experience* 28 (8): 2528–2545, 2016.

Souza, R., Azevedo, L. G., Lourenço, V., Soares, E., Thiago, R., Brandão, R., Civitarese, D., Vital Brazil, E., Moreno, M., Valduriez, P., Mattoso, M., Cerqueira, R., and Netto, M. A. S. Workflow provenance in the lifecycle of scientific machine learning. *Concurrency and Computation: Practice and Experience* n/a (n/a): e6544, 2021.

Tsay, J., Mummert, T., Bobroff, N., Braz, A., Westerink, P., and Hirzel, M. Runway: machine learning model experiment management tool. In *SysML*, 2018.

Valero, M. Runtime aware architectures. In *Proceedings of the 9th Annual Workshop on General Purpose Processing using Graphics Processing Unit, GPGPU@PPoPP 2016, Barcelona, Spain, March 12 - 16, 2016*, D. R. Kaeli and J. Cavazos (Eds.). ACM, pp. 1, 2016.

Wang, D., Weisz, J. D., Muller, M., Ram, P., Geyer, W., Dugan, C., Tausczik, Y., Samulowitz, H., and Gray, A. Human-ai collaboration in data science: Exploring data scientists' perceptions of automated ai. *Proceedings of the ACM on Human-Computer Interaction* 3 (CSCW): 1–24, 2019.

Yang, L., Meng, X., and Karniadakis, G. E. B-pinns: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *J. Comput. Phys.* vol. 425, pp. 109913, 2021.

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., and Zumar, C. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.* vol. 41, pp. 39–45, 2018.

Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zhu, Y. and Zabaras, N. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics* vol. 366, pp. 415–447, 2018.