Provenance Supporting Hyperparameter Analysis in Deep Neural Networks*

Débora Pina[1], Liliane Kunstmann[1], Daniel de Oliveira[2], Patrick Valduriez[3], and Marta Mattoso[1]

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

Abstract. The duration of the life cycle in deep neural networks (DNN) depends on the data configuration decisions that lead to success in obtaining models. Analyzing hyperparameters along the evolution of the network's execution allows for adapting the data. Provenance data derivation traces help the parameter fine-tuning by providing a global data picture with clear dependencies. Provenance can also contribute to the interpretation of models resulting from the DNN life cycle. However, there are challenges in collecting hyperparameters and in modeling the relationships between the data involved in the DNN life cycle to build a provenance database. Current approaches adopt different notions of provenance in their representation and require the execution of the DNN under a specific software framework, which limits interoperability and flexibility when choosing the DNN execution environment. This work presents a provenance data-based approach to address these challenges, proposing a collection mechanism with flexibility in the choice and representation of data to be analyzed. Experiments of the approach, using a convolutional neural network focused on image recognition, provide evidence of the flexibility, the efficiency of data collection, the analysis and the validation of network data.

Keywords: Provenance · Deep Learning · Workflow Steering.

1 Introduction

Provenance data [9, 26, 17] constitute a natural solution to assist users in the registration of algorithms, the data derivation path, metadata and parameters relevant to the data transformation steps [18]. Provenance data have already been successfully used in many scenarios and domains over the last decade (*e.g.*, bioinformatics [29, 3], health [7, 5], visualization [8], etc.). Recently, its usage in the Machine Learning (ML) life cycle has gained importance. Among the ML methods, Deep Learning (DL) and Deep Neural Network (DNN) models have gained much attention. Similar to large-scale scientific workflows, ML models are also a result of an iterative process [44]. ML workflows commonly involve several data transformations, users, algorithms, datasets, parameters, and add the challenge of feedback loops [37]. According to Silva *et al.* [37] there is a lack of capabilities for enabling ML workflow steering and dynamic workflow execution. Associating provenance data with the results of an ML workflow can support user steering to improve fine-tuning of parameters (or hyperparameters), at runtime, which is desired by several users [46, 45, 18].

The ML life cycle can be seen as a data centric workflow, as it produces an ML model based on input raw data through a data transformation flow. Fig. 1 presents the data transformation flow in the ML life cycle using DNNs (steps ❶ to ❼). The process starts with the data preparation (step ❶), where outliers can be analyzed, missing values can be imputed and feature engineering may be performed. Once the dataset is prepared, it is split into three subsets (step ❷): training set, test set, and validation set. The first two sets are used to generate an ML model (step ❸), *e.g.*, multi-class classification, regression, etc. These models can be tuned (step ❹), *i.e.*, to set the values of the hyperparameters that produce the most accurate model. ML models are sensitive to hyperparameters [31], adjusting them in DNNs may be costly. Finally, the generated model is evaluated using the validation set (step ❺). There are several evaluation loops where the user may fine-tune parameters. To fine-tune, the user needs to have access to cause and effect data, like what filter was applied to the current model when the dropout value was below a specific threshold. By analyzing the evaluation (step ❻), the user may decide to accept the generated model or to select a new configuration (step ❼) and retrain the model. In this paper, we focus on the training phase where fine-tuning happens based on provenance data analysis at runtime (dotted red rectangle in Fig. 1).

The advantages of capturing provenance data are well-known [9, 26, 21, 17, 18], such as data quality, data interpretation, and reproducibility of the results. According to Cheney *et al.* [4] it is important to consider the use and needs

* This work is funded by CNPq, FAPERJ, and Inria (HPDaSc associated team). D. Pina and L. Kunstmann are supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

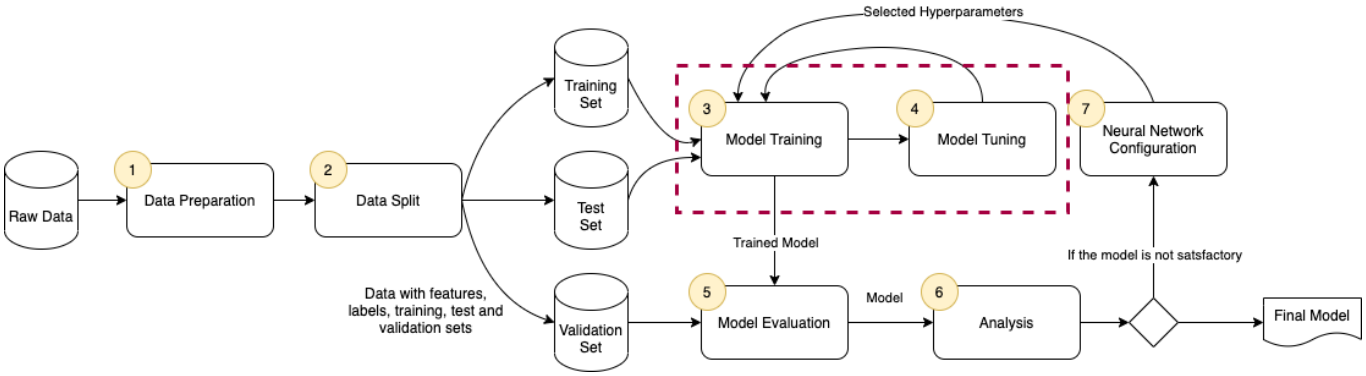

Fig. 1. Data transformation flow in the ML life cycle using DNNs

of provenance at an early stage, before adopting a provenance capture approach. Specifically, in DL-based workflows [15], provenance data have a lot to contribute to data analysis at runtime for user steering. Provenance data along with metadata, when available during execution (*i.e.*, at runtime), have great potential to support the analysis made by humans regarding hyperparameter configurations or even training data. The evaluation of the several hyperparameters requires that the user is aware of the relationship between many types of metadata, *e.g.*, the chosen hyperparameter values, performance data, environment configuration, etc. This data analysis during the training process can support the user in fine-tuning decisions, complementing auto-tuning solutions [43].

Adding provenance data to ML workflows is challenging. Despite the provenance support of several workflow systems, using these systems to invoke popular ML libraries or platforms can result in several execution conflicts, particularly in high-performance computing environments [37]. Basically, there are two approaches to make ML workflows provenance aware. The first is provenance provided for a specific ML platform [46, 35, 24, 40, 2, 20, 36, 30] and the second is the provenance systems that are independent of the domain [32]. In the first approach, each ML platform provides provenance using its proprietary representation, which is difficult to interpret and compare with execution between different platforms. The provenance systems approach is often tightly coupled to a script programming language, limiting the use of well-known ML platforms or it is too generic requiring a lot of data modeling and instrumenting the workflows.

In this paper, we present DNNProv and Keras-Prov, two provenance service management approaches that are designed for supporting hyperparameter analysis in DNNs. DNNProv and Keras-Prov integrate both traditional retrospective provenance data (r-prov) with domain-specific DL data. Both solutions provide an API that allows for users to develop their ML-based workflows using different DL frameworks (*e.g.*, Tensorflow, Theano) while being able to share and analyze captured provenance data using W3C PROV. The remainder of this paper is structured as follows. Section 2 discusses related work, Section 3 details the proposed approach. Section 4 presents the experimental evaluation and discussion of results. Finally, Section 5 concludes the paper.

2 Related Work

With the recent interest in DL methods, several works propose provenance management approaches for data analysis during DNN training [11]. There are several challenges in making ML workflows provenance aware like taking into account the execution framework that may involve CPUs, GPUs, TPUs, and distributed environments such as clusters and clouds as discussed in [36, 42, 14]. In this section, we discuss related work for provenance data management, considering the intention of using provenance for runtime data analysis. We group these works in sections, with approaches that are either focused on the ML domain or that provide domain-agnostic provenance systems. Our provenance services also provide data capture modeled for DL characteristics and analyses, but unlike the approaches that are focused on the DL domain, adopt best practices of provenance systems that follow W3C PROV recommendations. Having preset classes that already represent typical entities and activities from the DL domain improves provenance data preparation for runtime analyses. The result is flexibility in defining new domain data to the DL workflow and being able to execute with different ML platforms with distributed environments having CPUs and GPUs.

2.1 Machine- and Deep Learning-specific Approaches

The approaches in this category manage provenance for several purposes in ML platforms [46, 35, 24, 40, 2, 36, 30, 41, 12]. They are all based on a proprietary representation of provenance data, *i.e.*, that does not follow recommendations

like W3C PROV. These proprietary representations of provenance data can make interoperability and analysis difficult. If one user needs to compare the results of multiple training processes performed in different frameworks, additional implementations will be required. Next, we discuss the approaches that are focused on the use of provenance to interpret and tune hyperparameters and that are closer to our solution. ModelDB [41] is a system that aims at addressing model management. Its goal is to automatically track ML models in their native environment, storing trained models and their results to allow for visual exploration (or using SQL). Currently, ModelDB is customized for models generated using scikit-learn and SparkML, and uses visualization only as a way to perform *post-mortem* analysis of the ML pipeline, *i.e.*, it does not support runtime provenance analysis. Another solution focused on ML experiments is Runway [40]. Runway manages ML and DL artifacts, such as models, data, or experiments, as well as their provenance. In this sense, Runway allows for tracking the model and data, easing reproducibility. However, in addition to being a proprietary solution, which means that the solution does not follow W3C PROV standard to represent provenance data, Runway is restricted to the Python 3 programming language.

ModelKB (Model Knowledge Base) [12] aims at automating the life cycle management process for DL models with minimal user intervention. The contributions of ModelKB are to automatically extract and store model metadata and artifacts, in addition to viewing, consulting, comparing experiments, and reproducing models. ModelKB itself is not a modeling tool, but a complementary system that can automatically manage experiments in their native frameworks, such as TensorFlow [1]. However, ModelKB does not make the captured data available for analysis at runtime. Schelter *et al.* [35] provide an automated tool to extract metadata from the model with an interactive view to query and compare experiments. A declarative language is proposed for users to specify their queries. Thus, this solution focuses on tracking metadata and the provenance of ML experiment data. However, this approach does not use the W3C PROV standard, being an obstacle to foster interoperability.

2.2 Domain-agnostic Approaches

There are several approaches for capturing provenance data [32] that can be applied (with specializations) to the ML domain. Approaches for automatic capturing provide very fine granularity, generating a significant execution overhead in the process. Systems like noWorkflow [27], capture and store provenance data from Python scripts in an automatic way. noWorkflow allows provenance data capture without requiring any modifications in the original Python script. However, it is coupled to the Python language and does not execute in distributed and parallel environments, which limits the use of parallelism of popular ML libraries. Similar to noWorkflow, SPADE [10] automatically collects provenance from a workflow script including distributed and parallel environments, but this script has to be compiled using an LLVM compiler. In the automatic capture, the user does not spend time defining what provenance data to capture. However, when it comes to analyzing this provenance, significant time and effort are required to understand what and how data was modeled. In addition, due to the fine granularity, the user has to filter and aggregate data before starting the analysis. Having to do this during the training cycle may not be an option.

Different from the approaches based on automatic provenance capturing, the approaches based on the participation of the user allow to pre-select relevant data for analysis, having less impact on the execution and analysis time. When the user identifies attributes and parameters in W3C PROV entities, activities with their relationships, these chosen names become familiar for runtime analysis. One of these tools is DfAnalyzer [39], which is a tool that allows users to set the relevant data to be captured for runtime analysis in high-performance execution environments. One advantage of DfAnalyzer is that it captures provenance throughout the training of a DNN, and does not interfere in the performance of the training. Although DfAnalyzer is W3C PROV compliant and has been used in different domains, it is not designed for supporting provenance capturing during the ML life cycle. Therefore, repetitive work on designing and instrumenting has to be done. Sumatra [6] captures provenance from the script execution based on a series of annotations in the script. However, Sumatra only supports *post-mortem* analysis, *i.e.*, only after the ML model is generated. YesWorkflow [22] is another example of a tool capable of analyzing provenance data. YesWorkflow does not depend on a programming language, adopting a strategy of adding annotations to the scripts to identify provenance data. However, its queries are based on URIs and hyperparameter runtime analyses are not URI-based. Similar to YesWorkflow, in [13] there is no runtime provenance capture. Instead, they take advantage of applications that provide log files to extract provenance from them. This could be adopted by systems like TensorFlow and Keras, which provide provenance logs. However, the queries are limited to the logged data and it is a *post-mortem* analysis approach.

UML2PROV [34] aims at making UML (Unified Modeling Language) based applications provenance aware automatically. It is an approach that provides a mapping strategy from UML class, State Machine, and Sequence diagrams to define an automatic code generation technique that deploys artifacts for provenance generation in an application. UML2PROV assumes the existence of these diagrams or requires that they be designed or generated through reverse engineering, which limits its use in most ML environments.

Therefore, capturing provenance for runtime analysis in DL domains using domain-agnostic approaches may be complicated due to several reasons: (i) programming language and compiler dependencies, (ii) lack of support for

provenance capturing in HPC and distributed environments, and (iii) lack of support for runtime provenance analysis. The next section describes how DNNProv and Keras-Prov address these limitations.

3 DNNProv and Keras-Prov

Due to the continuous increase in the use of ML and DL methods for developing workflows in different domains, the use of provenance data to support analysis has been gaining importance. The analytical potential of provenance data contributes to the analysis of hyperparameter configurations (and their impact on the accuracy of the generated model) and, consequently, supports the fine-tuning [35, 23, 12]. As the training of DNNs can last for several hours or even days (depending on the computational environment), and a large amount of data is consumed and produced, the user needs to be able to evaluate the training to make adjustments to hyperparameters. Therefore, hyperparameter analysis tools should work during the training of the DNN and quickly provide the required data and entity attributes for analysis without competing for computational resources with the training process.

To capture, store and analyze provenance data from DNNs in an efficient way, we propose two provenance service management approaches, one independent from the DNN framework (named DNNProv) and the other coupled to a DNN framework (named Keras-Prov). DNNProv and Keras-Prov extend DfAnalyzer [39, 38], which allows monitoring, debugging, and analyzing provenance during the execution of scientific workflows. One advantage of DfAnalyzer is that it explores *in-situ* or asynchronous provenance data management, without interfering with ML and DL workflow performance even in HPC environments. Since DfAnalyzer is domain-agnostic, it is not designed for ML and DL domains and targets mainly binary scientific data, this section presents the extensions to DfAnalyzer for ML and DL domains.

DNNProv and Keras-Prov data representation follows the data model based on the recommendations of the PROV-DM of the W3C PROV [26] which is an initiative for the representation of different types of provenance data without being specific to a domain. W3C PROV is based on *Agent*, *Activity* and *Entity* concepts. An agent is is something that bears some form of responsibility for an activity, an entity, or for another agent's activity. An activity is something that occurs over a period of time upon or with entities and an entity is a thing with some fixed aspects. The diagram in Fig. 2, generated by Prov Python[4] and Graphviz[5], represents some activities of the neural network training process with DNNProv and Keras-Prov, following the notation from [25]. The orange pentagon represents the Agent concept, yellow ovals represent Entity and blue rectangles represent Activity. The diagram shows what was used directly from PROV-Df (with the tag dfanalyzer) and what was extended on our approach (with the tag dnnprov).

3.1 Provenance Model

The association of provenance data with domain data can assist DNN users to perform rich analyses at runtime and allows the monitoring of the training process. Thereby, in this paper, we present a solution capable of tracking activities that occur in the DL life cycle, providing efficient provenance data capturing and analysis. The proposed approach considers the steps of the DL life cycle as a dataflow. To represent specific data from the DL training process, a specialization of the PROV-Df [39] model, named DNNProv-Df, is proposed in this subsection. Based on this new model, the user is able to (i) track epochs, learning rate, accuracy, loss function, execution time, etc., (ii) discover which pre-processing methods were used before training the model, (iii) monitor the training process and perform fine-tuning, (iv) discover which files were generated in different execution steps, and (v) interpret the generated results.

Fig. 3 presents the DNNProv-Df model represented as a UML class diagram. The classes inside the dotted red area are classes related to the steps of the DL life cycle considered in this paper, with attributes that represent the data related to training metrics and hyperparameters. In the DL life cycle, several different hyperparameters have to be set, *e.g.*, learning rate, batch size, number of epochs, momentum and dropout. Choosing the best hyperparameter values is far from trivial and many combinations are commonly explored. This process is compute-intensive, usually performed in HPC environments, and, several models are generated. Only one of these models is chosen as the best one and this choice must be registered for *a posteriori* analysis. In addition, as the training process takes a long time, it is necessary to steer it, *e.g.*, by inspecting how the evaluation metrics are evolving during the training so that one can change parameters and start training again if needed.

The classes related to the dataflow specification are inherited from the PROV-Df model, which are *dataflow*, *dataTransformation*, *dataset*, *dataDependency*, *program* and *attribute*. These classes represent prospective provenance (p-prov). The class *dataflow* is responsible for representing the different dataflows whose content has been stored in the provenance database. This class contains a name for each dataflow. The class *dataTransformation* represents the multiple activities associated with a given dataflow. An activity is associated with a *program* or a script that is

[4] https://prov.readthedocs.io/en/latest/prov.html
[5] http://www.graphviz.org/

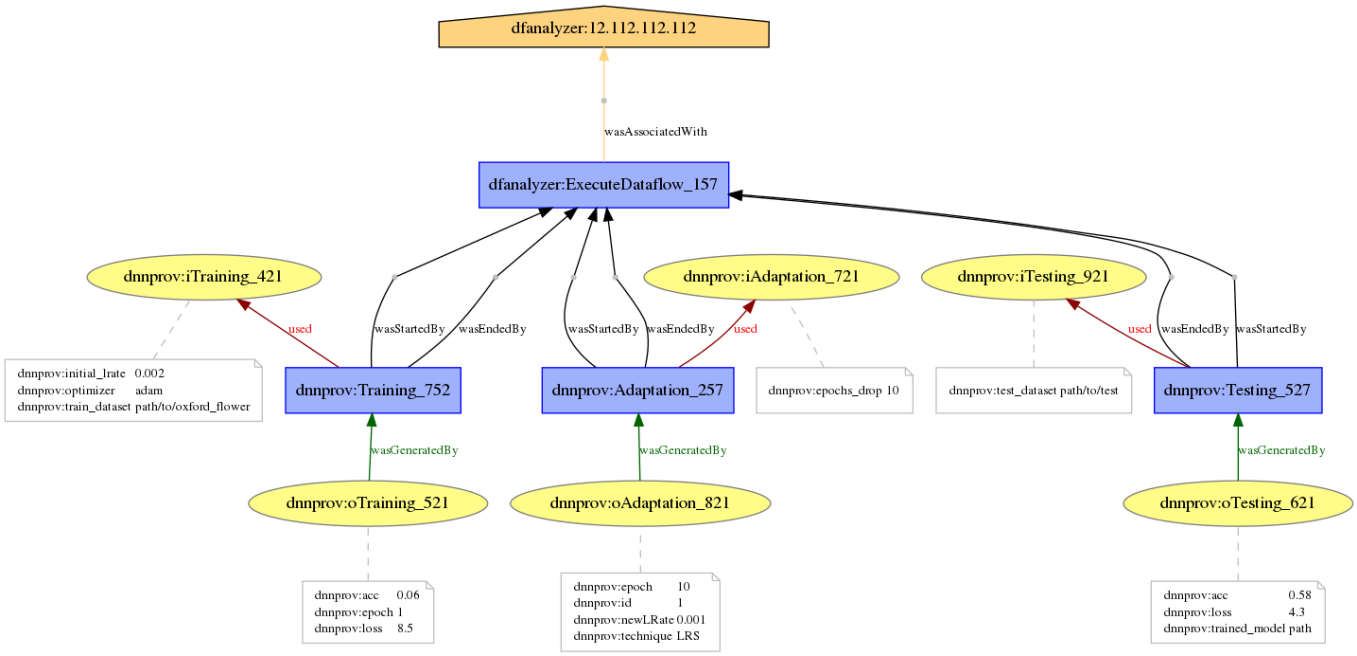

Fig. 2. W3C PROV graph fragment of DNNProv and Keras-Prov

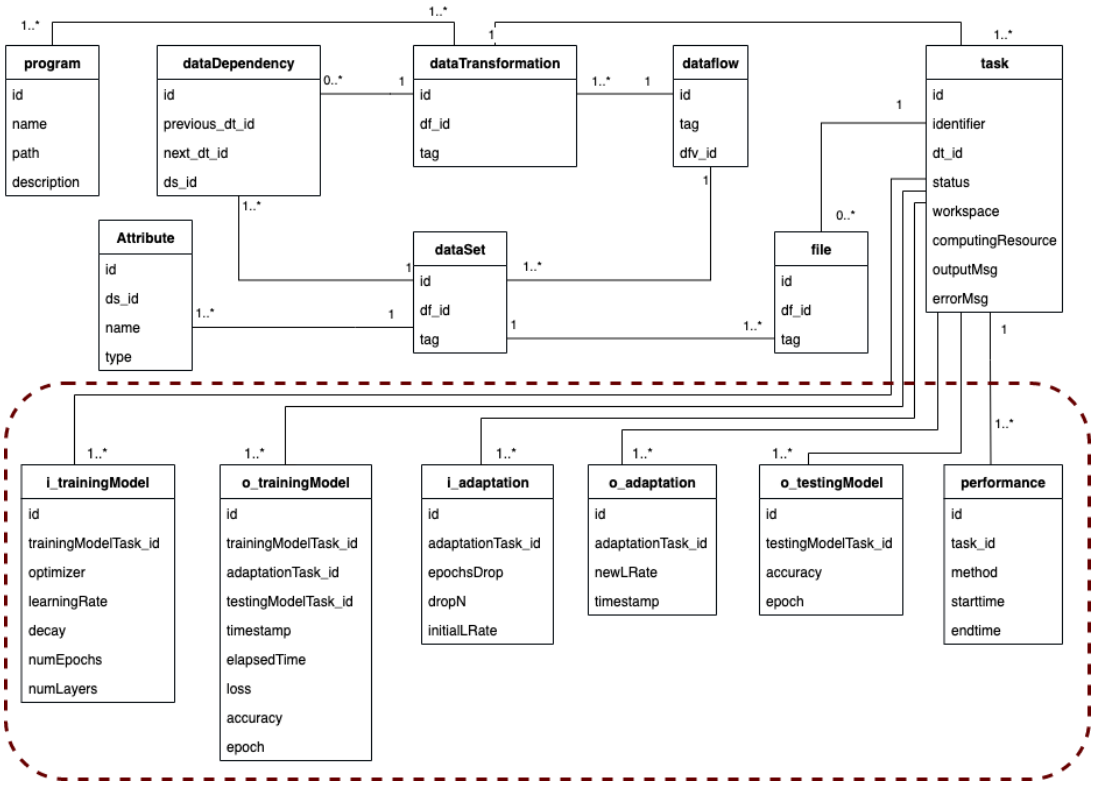

Fig. 3. DNNProv-Df Model

executed. To represent retrospective provenance (r-prov), DNNProv-Df contains the classes *task* and *file*. The class *task* represents the instances of activities that are executed, and the class *file* represents the entity attributes that were eventually used and generated at runtime.

As described in [23], the hyperparameters are adjusted after or during each iteration. To make decisions regarding the fine-tuning of these hyperparameter values, the user needs to evaluate several data associated with the behavior of the DNN with the hyperparameter configuration used in that iteration, *e.g.*, the training time for each epoch, and the loss associated with each epoch. Thus DNNProv-Df contains the following classes to represent domain-specific data regarding the DNN training process: *i_trainingModel*, *o_trainingModel*, *i_adaptation*, *o_adaptation* and *o_testingModel*.

The class *i_trainingModel* contains the hyperparameters that have to be set in the DNN before the training process (the prefix *i* refers to the input parameters of a task and the prefix *o* in the classes refers to the output parameter values). Some of these hyperparameters are learning rate, number of epochs, optimizer, momentum, and decay. Meanwhile, the class *o_trainingModel* contains performance metrics for the generated model by epoch. This class contains attributes such as elapsed time and the date and time of the end of the epoch's execution. The elapsed time attribute defined in this class allows users to check whether the execution of an epoch is taking longer than expected. Adaptations are made during the training of a DNN. For example, an adaptation can generate a new learning rate at the end of an epoch, using a function that significantly decreases the value of the learning rate every n times by a m factor. Thus, the activity *Adaptation* receives as input data the set produced by the previous activity (activity *Training*) and a dataset with information such as the factor m, the value of n and the initial learning rate. This dataset is represented by the class *i_adaptation*. The dataset produced by this activity, represented by the class *o_adaptation*, contains the new learning rate, epoch and the date and time when the adaptation occurred, in addition to identification for the adaptation. Finally, the class *o_testingModel* is related to the activity *Testing* and provides data about the evaluation of the model. Thus, similarly to the *o_trainingModel* class, *o_testingModel* contains performance metrics, such as accuracy and loss function values. It is worth mentioning that this model can be extended as new hyperparameters and metrics are needed.

3.2 Architecture of DNNProv and Keras-Prov

We present two provenance service management solutions for hyperparameters analysis. The architectures of DNNProv and Keras-Prov are presented in Fig. 4. The main difference between DNNProv and Keras-Prov is where the *Provenance Extractor* component is deployed. The architecture of both services is composed of three layers, according to Fig. 4, which are: (i) Training Layer, (ii) Data Layer, and (iii) Analysis Layer. The *Training Layer* is where the training library executes and interacts with the *Provenance Extractor*, which accesses the hyperparameter values at runtime and gets their values.

For DNNProv, the *Provenance Extractor* is implemented and deployed outside the DNN library. This way, the user can choose any DNN library or framework, and then instrument the code (adding calls to the provenance extractor component) to define which data to capture. Using DNNProv we assume that the programs in the DL life cycle are gray boxes, *i.e.*, part of their source code can be adapted, while the other part can invoke a private source code (black box). In DNNProv, the user defines which domain data and hyperparameters values will be captured and where they should be captured. With this instrumentation, the *Provenance Extractor* can access the data during training.

In the case of Keras-Prov, the *Provenance Extractor* is already implemented within the Keras library, so the user does not need to instrument the code. The user chooses, among the predefined provenance choices, domain data and hyperparameters to be captured. The *Provenance Extractor* automatically extracts the values of the hyperparameters used in each training. In both Keras-Prov and DNN-Prov, once captured, the provenance data is managed asynchronously with the DL execution. After, the *Provenance Extractor* interacts with the *Data Layer* to get the JSON file paths describing the DNN. These file paths, with hyperparameters values, metrics and, etc., are sent to the provenance database. Then, the *Provenance Exporter* queries the provenance database and sends query results to the *Provenance Viewer*, which generates a visual representation of the provenance graph as an alternative to the user runtime analysis.

3.3 Using DNNProv and Keras-Prov

If the user chooses to use DNNProv, one important step is the code instrumentation. In this step, the user defines which data to capture. The first step is to define the dataflow structure in the source code of the DNN workflow (p-prov). Let us assume that a user needs to capture the data defined in Fig. 3, thus one has to include in the source code the fragment shown in Fig. 5. In this fragment, the user specifies an identifier for the dataflow (dataflow_tag = "alexnet") and sets the automatic definition of activities to *True*. By setting the automatic definition of p-prov, the DNNProv queries the provenance database and identifies the registered activities. If the user chooses to set the automatic definition of p-prov to *False*, one has to define which activities are part of this dataflow manually.

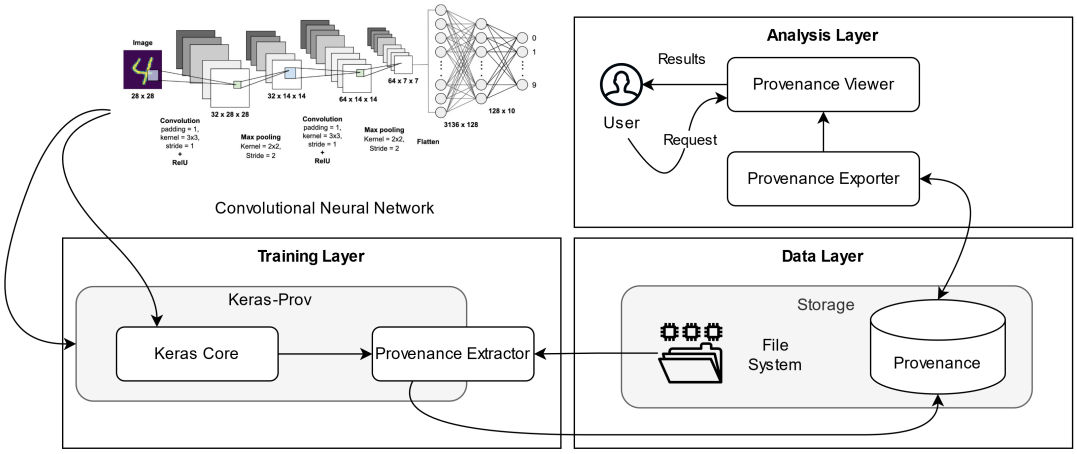

Fig. 4. Architecture of DNNProv and Keras-Prov

Once the dataflow and its transformations are defined, the user needs to set in the source code where a task starts (and finishes) and where DNNProv can extract (at runtime) the values of the hyperparameters. Note that the dataflow and data transformations specifications are p-prov, while the definition of tasks is r-prov. When the user is defining a task in the source code, one can also define the data dependencies among tasks, *i.e.*, the user specifies in the source code which tasks are responsible for the production and consumption of data in a given dataflow. This step is also shown in Fig. 5 (*dependency=t1* - this means that task *t2* only starts after task *t1* produces data). Multiple dependencies are modeled with binary relationships and queried with join predicates. Once set, this provenance schema can be reused for further DNN analyses.

```
#Prospective provenance
df = Dataflow("alexnet", True)
df.save()

#Retrospective provenance
...
t2 = Task(2, "alexnet", "Adaptation", dependency=t1)
initial_lrate = 0.002
drop = 0.5
epochs_drop = 10.0
t2_input = DataSet("iAdaptation", [Element([epochs_drop,
drop, initial_lrate])])
t2.add_dataset(t2_input)
t2.begin()
...
def on_epoch_begin(self, epoch, logs=None):
    lr = K.get_value(self.model.optimizer.lr)
    temp_lr= schedule(self.epoch)
    K.set_value(self.model.optimizer.lr, lr)
    t2_output = DataSet("oAdaptation", [Element([lr,
    datetime.now(), epoch, adaptation_id])])
    t2.add_dataset(t2_output)
    t2.save()
```

Fig. 5. A fragment of the source code instrumented for DNNProv

This instrumentation of the source code to capture the provenance data can be a barrier to the adoption of provenance tools by many scientists that are not computer science experts, given the effort that may be required

from the user. With this in mind, we proposed Keras-Prov, which is the second provenance service management approach proposed in this paper. Keras-Prov is an extension of the DNN library Keras[6] and its goal is to reduce the effort to adapt the code to capture provenance data during the DNN training process by reducing the need for instrumentation. In the implementation of Keras-Prov, modifications were performed to the source code of Keras[7] to embed the *Provenance Extractor* component, since Keras does not capture provenance data natively. A class *Provenance* was created containing methods to deal with the creation of the activities that follow the DL life cycle, following the representation presented in Fig. 3. In addition to this data, Keras-Prov captures and stores information about the layers of the DNN, that is, the name that identifies the layer (*e.g.*, activation_1, dropout_1), the type of layer (*e.g.*, activation, dropout) and the value of this layer (*e.g.*, for activation the value is *relu*, the value for dropout is 0.4). Although Keras-Prov captures several hyperparameter values automatically, it is worth noticing that the user can define new data to be captured. For more information about defining new data to be captured using Keras-Prov please visit `https://github.com/dbpina/keras-prov`.

In the *Model* class of Keras, a *provenance* method was created to capture provenance data. This method receives a *tag* to identify the dataflow, if there is an adaptation of the hyperparameters during training (e.g., an update of the learning rate), that is, the use of methods such as *LearningRateScheduler* offered by Keras, and the list of hyperparameters to be captured. The data received by the *provenance* method are defined by the user in the source code of the DL workflow following the example presented in Fig. 6. Different from DNNProv, when using Keras-Prov the user needs only to set which hyperparameters to capture, and no additional instrumentation is required. After setting *True* to the hyperparameters of interest, the user adds a call to the method *provenance*.

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

```
model.provenance(
                dataflow_tag= "alexnet",
                adaptation=False,
                hyps = hyps)
```

Fig. 6. Setting the hyperparameters of interest in Keras-Prov

Considering that ML workflows follow the life cycle of Fig. 1, the inclusion of DNNProv in popular ML systems like Keras, requires the identification, in the source code, of the points at which such activities (network configuration, training, and testing) are performed and how the neural network data, hyperparameters, and metrics (from the training and testing steps) are being manipulated. This step allows to define PROV relationships between them and establishes data extraction into the database for automatic provenance design and capture in those systems.

4 Evaluation

In this section, we evaluate DNNProv and Keras-Prov. We discuss the results obtained using both approaches for the analysis of hyperparameter configurations during the training of DNNs using provenance data. In the experiments presented in this section, we trained AlexNet [19] in both DNNProv and Keras-Prov in the cluster Lobo Carneiro (SGI cluster with 504 CPUs Intel Xeon E5-2670v3 (Haswell) - total of 6.048 processors) at COPPE/UFRJ using the Oxford Flower [28] dataset, which consists of 17 species of flowers with 80 images for each class. The flower categories in this dataset are deliberately chosen to have some ambiguity in each aspect. For example, some classes cannot be distinguished only in colors, such as dandelions and buttercups, others cannot be distinguished only in shapes, such as daffodils and wild windflowers. The images of the flowers were retrieved from different websites and some images from the authors' own photographs [28].

The Alexnet dataflow is composed of the following activities: (i) *Training*, (ii) *Adaptation*, and (iii) *Testing*. *Training* consumes (*used*) the following hyperparameters: the name of the optimizer, the learning rate, number of epochs, and number of layers in the network, and produces (*wasGeneratedBy*) a set of metrics that helps in the evaluation of results obtained during training, *e.g.*, accuracy, the value of the loss function, the elapsed time and the date and time

[6] https://keras.io/
[7] https://github.com/keras-team/keras

of the end of the execution of each epoch. *Adaptation* consumes (*used*) the dataset produced by the previous activity (*Training*), a dataset with information for the adaptation that has taken place and the output contains the values for the new learning rate, the value of the epoch and the date and time when the adaptation occurred, in addition to identification for the adaptation. *Testing* provides data on the evaluation of the model according to the training dataset and outputs the accuracy and loss function values.

Several training runs were performed for AlexNet, with variations in hyperparameters values, *e.g.*, learning rate (0.0005, 0.001, 0.002), optimizer (Adam, SGD) and dropout (0.4, 0.7). After a few training executions, the user decided to apply a filter to convert the images (input dataset) to a gray-scale. Because of that, the user needed to add a new activity called *Filters*, also wanting this activity to be registered at the provenance database. Due to DNNProv's flexibility, a modification was made to the p-prov and this activity was included, which means that in the next training executions, the data defined by the user for this activity started to be captured. Likewise, other extensions can be done to the relational schema showing the flexibility of the approaches.

Table 1 defines provenance queries based on the most frequent queries from [35, 24, 40, 41, 12]. We categorize the set of possible provenance queries as illustrated in Table 2. Queries are classified according to the provenance data processing needed to answer them. For instance, queries in class C1 get entity attributes from a single provenance graph, while queries in class C2 access multiple provenance graphs. Queries in class C3 require relating entity attributes and data derivation path on one graph, while C4 queries multiple provenance graphs. ML frameworks like Tensorflow or Keras mention their provenance support through logs. Despite the possibility of extracting provenance from logs [13], it is far from trivial. It requires repetitive log post-processing for every trial, with no flexibility on defining what to capture. Running aggregation queries through epoch iterations from logs, like Q6 and Q8, is also time-consuming and demands a significant effort from the user.

Table 1. Example of provenance queries.

#	**Queries**	**Class**
Q1	What is the loss value of epoch 10 of a specific training *t'* of a specific model *m'*?	C1
Q2	What are the layers of model *m'*?	C1
Q3	Retrieve the time consumed and loss by each epoch in the training *t'* of model *m'*.	C1
Q4	What is the initial value of the learning rate when the training for model *m'* was more accurate?	C2
Q5	Retrieve the combinations of hyperparameters where was obtained the 3 best accuracy values in previous training for model *m'*?	C2
Q6	Was there an adaptation of the learning rate in the training of the model *m'* that obtained the best accuracy? If so, at what epoch did it occur and what technique was used for this adaptation? And what is the new value?	C3
Q7	What filter was applied to model *m'* when the dropout value was 0.4?	C4
Q8	Which filter was applied in the training executions that showed the best and the worst accuracy for model *m'*? What was the number of epochs in these executions?	C4

Table 2. Classification of queries.

Class	**Entity Attributes**	**Derivation Path**	**Single Graph**	**Multiple Graphs**
C1	Yes	No	Yes	No
C2	Yes	No	No	Yes
C3	Yes	Yes	Yes	No
C4	Yes	Yes	No	Yes

Using the queries presented in Table 1, during the training of the DNN, the user is able to monitor metrics by epoch and steer the training at runtime. If, for instance, the loss value is not meeting the criteria defined by the user, one may decide to stop training or modify the learning rate. In this case, these adaptations are also saved (time and date when it happened) since they are important for *a posteriori* analysis, even at the end of the training process. The user may want to discover if the training with adaptation at runtime produced better results than the training that used the same value for the learning rate without modifications at runtime. It is worth noticing that DNNProv and Keras- Prov can also be connected to data visualization tools, as a setup option, such as Kibana [8], to create dashboards and other resources to support the user's analysis.

To evaluate the potential of DNNProv and Keras-Prov, queries Q3, Q6, and Q8 were submitted to the provenance database that was populated with the different training data of AlexNet (from multiple training runs). The results con-

[8] `https://www.elastic.co/kibana`

sider the number of epochs, if a gray-scale filter was applied to the input dataset and the use of LearningRateScheduler (LRS) for adaptations in the learning rate during training. These results are presented in Tables 3, 4 and 5.

From the result of Q3 (Table 3), it is possible to investigate, for example, if any epoch is taking longer than usual. In addition, if the loss value attribute is selected along with the identification of the epoch, the specialist can verify whether the increase in the number of epochs no longer contributes to better accuracy after a certain epoch. The result of Q6 (Table 4) shows the impact of learning rate adaptations in model convergence. Though decreasing the learning rate is a known technique, the registered values help the user trace the cause of changes when analyzing different models. These data are also important if the user is evaluating the impact of different learning rate adaptation techniques or different parameters for decreasing learning rate functions, such as Step Decay. From the result of Q8 (Table 5), we observed that the application of the filter that converts the images to a gray-scale presented a worse accuracy in the test set, 0.37, than the training without this filter, which presented an accuracy of 0.59.

Table 3. Results for the Query Q3

Epoch	Time (seconds)	Loss Value
1	22.075	3.484
2	20.560	2.870
3	19.996	2.542
4	20.478	2.188
5	20.378	2.015
6	20.006	1.784
7	20.486	1.600
8	20.238	1.466
9	20.395	1.246
10	20.318	0.977

Table 4. Results for the Query Q6

Epoch	Learning rate	Technique
10	0.001	LRS
30	0.00025	LRS

Table 5. Results for the Query Q8

Filter	Accuracy	Epochs
None	0.59	100
Gray-scale	0.37	100

It is worth mentioning that for the results presented in this paper, AlexNet was trained with a maximum of 100 epochs. AlexNet was also trained a few times with a larger number of epochs, such as 500 and 1000, which took about three and six hours, respectively. Due to this training time, we chose to train with fewer epochs to show provenance helping in evaluating different variations of hyperparameters. Furthermore, provenance queries like Q6 show that the highest accuracy reached by AlexNet with 500 epochs was around 65%, which is consistent with the top accuracy of 68.68% presented for AlexNet in flower categorization [16].

Moreover, we observed the overhead introduced by the proposed approach. The purpose of this measurement was to assess the impact of capturing provenance on training time. We observed that the time overhead corresponds to an increase of 2% in the worst case over the total workflow time. This overhead can be considered negligible, especially in longer executions, considering that the user will have the benefit of queries and visualizations to the captured provenance data. In addition, the size of the provenance database was 5MB and with such a size it is already possible to answer relevant questions.

5 Conclusions

The approach presented in this paper aims at supporting the analysis of hyperparameter configurations and adaptations in the training of DNNs by capturing relevant provenance data. We present a provenance-based user steering approach that allows for capturing and storing data to query during and after the training. This approach is implemented in two modes, independent and dependent of the DNN library or framework. By adopting the W3C PROV recommendation, both modes aim at reducing the diversity of data representation and the effort in modeling and querying DL training data related to PROV. The first, DNNProv, is not specific to programming languages or libraries, and does not require the use of a particular ML execution environment. However, DNNProv requires the user to instrument the script code of the DNN workflow, which may require some effort. To provide a solution that

does not require instrumentation, Keras-Prov adds DNNProv components into Keras to capture the provenance data automatically. In addition, the approach is flexible since it allows for the inclusion of new types of data to be captured, like the DL domain application data. Experiments show the adequacy of the use of provenance in the analysis throughout the training of DNNs, including extensions for capturing data related to pre-processing. As future work, we plan to extend the approach to the domain of Physics Informed Neural Networks (PINN) [33]. PINNs define the neural network loss function based on partial differential equations that inform physics. Analyzing loss function data in PINNs increases the complexity of provenance data management.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
2. Agrawal, P., Arya, R., Bindal, A., Bhatia, S., Gagneja, A., Godlewski, J., Low, Y., Muss, T., Paliwal, M.M., Raman, S., et al.: Data platform for machine learning. In: Proceedings of the 2019 International Conference on Management of Data. pp. 1803–1816 (2019)
3. Almeida, R.F., da Silva, W.M.C., Castro, K., de Araújo, A.P.F., Walter, M.E.T., Lifschitz, S., Holanda, M.: Managing data provenance for bioinformatics workflows using aprovbio. Int. J. Comput. Biol. Drug Des. **12**(2), 153–170 (2019). https://doi.org/10.1504/IJCBDD.2019.099761, `https://doi.org/10.1504/IJCBDD.2019.099761`
4. Cheney, J., Chapman, A., Davidson, J., Forbes, A.: Data provenance, curation and quality in metrology. arXiv preprint arXiv:2102.08228 (2021)
5. Corrigan, D., Curcin, V., Ethier, J., Flynn, A.J., Sottara, D.: Challenges of deploying computable biomedical knowledge in real-world applications. In: AMIA 2019, American Medical Informatics Association Annual Symposium, Washington, DC, USA, November 16-20, 2019. AMIA (2019), `http://knowledge.amia.org/69862-amia-1.4570936/t002-1.4575206/t002-1.4575207/3201770-1.4575319/3203261-1.4575316`
6. Davison, A.: Automated capture of experiment context for easier reproducibility in computational research. Computing in Science & Engineering **14**(4), 48–56 (2012)
7. Fairweather, E., Wittner, R., Chapman, M., Holub, P., Curcin, V.: Non-repudiable provenance for clinical decision support systems. CoRR **abs/2006.11233** (2020), `https://arxiv.org/abs/2006.11233`
8. Fekete, J., Freire, J., Rhyne, T.: Exploring reproducibility in visualization. IEEE Computer Graphics and Applications **40**(5), 108–119 (2020). https://doi.org/10.1109/MCG.2020.3006412, `https://doi.org/10.1109/MCG.2020.3006412`
9. Freire, J., Koop, D., Santos, E., Silva, C.T.: Provenance for computational tasks: A survey. Computing in Science & Engineering **10**(3), 11–21 (2008)
10. Gehani, A., Tariq, D.: Spade: Support for provenance auditing in distributed environments. In: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing. pp. 101–120. Springer (2012)
11. Gharibi, G., Walunj, V., Alanazi, R., Rella, S., Lee, Y.: Automated management of deep learning experiments. In: Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning. p. 8. ACM (2019)
12. Gharibi, G., Walunj, V., Rella, S., Lee, Y.: Modelkb: towards automated management of the modeling lifecycle in deep learning. In: Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. pp. 28–34. IEEE Press (2019)
13. Ghoshal, D., Plale, B.: Provenance from log files: a bigdata problem. In: Proceedings of the Joint EDBT/ICDT 2013 Workshops. pp. 290–297 (2013)
14. Gil, Y., Garijo, D., Khider, D., Knoblock, C.A., Ratnakar, V., Osorio, M., Vargas, H., Pham, M., Pujara, J., Shbita, B., Vu, B., Chiang, Y.Y., Feldman, D., Lin, Y., Song, H., Kumar, V., Khandelwal, A., Steinbach, M., Tayal, K., Xu, S., Pierce, S.A., Pearson, L., Hardesty-Lewis, D., Deelman, E., Ferreira da Silva, R., Mayani, R., Kemanian, A.R., Shi, Y., Leonard, L., Peckham, S., Stoica, M., Cobourn, K., Zhang, Z., Duffy, C., Shu, L.: Artificial intelligence for modeling complex systems: Taming the complexity of expert models to improve decision making. ACM Transactions on Interactive Intelligent Systems (2021)
15. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT press Cambridge (2016)
16. Gurnani, A., Mavani, V., Gajjar, V., Khandhediya, Y.: Flower categorization using deep convolutional neural networks. arXiv preprint arXiv:1708.03763 (2017)
17. Herschel, M., Diestelkämper, R., Lahmar, H.B.: A survey on provenance: What for? what form? what from? The VLDB Journal **26**(6), 881–906 (2017)
18. Huynh, T.D., Stalla, S., Moreau, L.: Provenance-based explanations for automated decisions: final iaa project report (2019)
19. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
20. Lourenço, R., Freire, J., Shasha, D.: Debugging machine learning pipelines. In: Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning. pp. 1–10 (2019)
21. Mattoso, M., Dias, J., Ocaña, K.A., Ogasawara, E., Costa, F., Horta, F., Silva, V., de Oliveira, D.: Dynamic steering of hpc scientific workflows: A survey. Future Generation Computer Systems **46**, 100–113 (2015)

22. McPhillips, T., Bowers, S., Belhajjame, K., Ludäscher, B.: Retrospective provenance without a runtime provenance recorder. In: 7th {USENIX} Workshop on the Theory and Practice of Provenance (TaPP 15) (2015)
23. Miao, H., Li, A., Davis, L.S., Deshpande, A.: Modelhub: Lifecycle management for deep learning. Univ. of Maryland (2015)
24. Miao, H., Li, A., Davis, L.S., Deshpande, A.: Towards unified data and lifecycle management for deep learning. In: 2017 IEEE 33rd International Conference on Data Engineering (ICDE). pp. 571–582. IEEE (2017)
25. Missier, P., Belhajjame, K., Cheney, J.: The w3c prov family of specifications for modelling provenance metadata. In: Proceedings of the 16th International Conference on Extending Database Technology. pp. 773–776 (2013)
26. Moreau, L., Groth, P.: Provenance: an introduction to prov. Synthesis Lectures on the Semantic Web: Theory and Technology **3**(4), 1–129 (2013)
27. Murta, L., Braganholo, V., Chirigati, F., Koop, D., Freire, J.: noworkflow: capturing and analyzing provenance of scripts. In: International Provenance and Annotation Workshop. pp. 71–83. Springer (2014)
28. Nilsback, M.E., Zisserman, A.: A visual vocabulary for flower classification. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). vol. 2, pp. 1447–1454. IEEE (2006)
29. Ocaña, K.A.C.S., Silva, V., de Oliveira, D., Mattoso, M.: Data analytics in bioinformatics: Data science in practice for genomics analysis workflows. In: 11th IEEE International Conference on e-Science, e-Science 2015, Munich, Germany, August 31 - September 4, 2015. pp. 322–331. IEEE Computer Society (2015). https://doi.org/10.1109/eScience.2015.50, `https://doi.org/10.1109/eScience.2015.50`
30. Ormenisan, A.A., Ismail, M., Haridi, S., Dowling, J.: Implicit provenance for machine learning artifacts. Proceedings of MLSys **20** (2020)
31. Orr, G.B., Müller, K.R.: Neural networks: tricks of the trade. Springer (2003)
32. Pimentel, J.F., Freire, J., Murta, L., Braganholo, V.: A survey on collecting, managing, and analyzing provenance from scripts. ACM Comput. Surv. **52**(3), 47:1–47:38 (2019). https://doi.org/10.1145/3311955, `https://doi.org/10.1145/3311955`
33. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. arXiv preprint arXiv:1711.10561 (2017)
34. Sáenz-Adán, C., Moreau, L., Pérez, B., Miles, S., García-Izquierdo, F.J.: Automating provenance capture in software engineering with uml2prov. In: International Provenance and Annotation Workshop. pp. 58–70. Springer (2018)
35. Schelter, S., Böse, J.H., Kirschnick, J., Klein, T., Seufert, S.: Automatically tracking metadata and provenance of machine learning experiments. In: Machine Learning Systems workshop at NIPS (2017)
36. Scherzinger, S., Seifert, C., Wiese, L.: The best of both worlds: Challenges in linking provenance and explainability in distributed machine learning. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). pp. 1620–1629. IEEE (2019)
37. Ferreira da Silva, R., Casanova, H., Chard, K., Laney, D., Ahn, D., Jha, S., Goble, C., Ramakrishnan, L., Peterson, L., Enders, B., Thain, D., Altintas, I.., Babuji, Y., Badia, R., Bonazzi, V., Coleman, T., Crusoe, M., Deelman, E., Di Natale, F., Di Tommaso, P., Fahringer, T., Filgueira, R., Fursin, G., Ganose, A., Gruning, B., Katz, D.S.., Kuchar, O., Kupresanin, A., Ludascher, B., Maheshwari, K., Mattoso, M., Mehta, K., Munson, T., Ozik, J., Peterka, T., Pottier, L., Randles, T., Soiland-Reyes, S., Tovar, B., Turilli, M., Uram, T., Vahi, K., Wilde, M., Wolf, M., Wozniak, J.: Workflows community summit: Bringing the scientific workflows research community together (Mar 2021)
38. Silva, V., Campos, V., Guedes, T., Camata, J., de Oliveira, D., Coutinho, A.L., Valduriez, P., Mattoso, M.: Dfanalyzer: Runtime dataflow analysis tool for computational science and engineering applications. SoftwareX **12**, 100592 (2020)
39. Silva, V., de Oliveira, D., Valduriez, P., Mattoso, M.: Dfanalyzer: runtime dataflow analysis of scientific applications using provenance. Proceedings of the VLDB Endowment **11**(12), 2082–2085 (2018)
40. Tsay, J., Mummert, T., Bobroff, N., Braz, A., Westerink, P., Hirzel, M.: Runway: machine learning model experiment management tool (2018)
41. Vartak, M., Subramanyam, H., Lee, W.E., Viswanathan, S., Husnoo, S., Madden, S., Zaharia, M.: Model db: a system for machine learning model management. In: Proceedings of the Workshop on Human-In-the-Loop Data Analytics. p. 14. ACM (2016)
42. Wang, D., Churchill, E., Maes, P., Fan, X., Shneiderman, B., Shi, Y., Wang, Q.: From human-human collaboration to human-ai collaboration: Designing ai systems that can work together with people. In: Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems. pp. 1–6 (2020)
43. Wang, D., Weisz, J.D., Muller, M., Ram, P., Geyer, W., Dugan, C., Tausczik, Y., Samulowitz, H., 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)
44. Warnke, T., Helms, T., Uhrmacher, A.M.: Reproducible and flexible simulation experiments with ml-rules and SESSL. Bioinform. **34**(8), 1424–1427 (2018). https://doi.org/10.1093/bioinformatics/btx741, `https://doi.org/10.1093/bioinformatics/btx741`
45. Xin, D., Ma, L., Liu, J., Macke, S., Song, S., Parameswaran, A.: Accelerating human-in-the-loop machine learning: challenges and opportunities. In: Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning. pp. 1–4 (2018)
46. Zhang, Z., Sparks, E.R., Franklin, M.J.: Diagnosing machine learning pipelines with fine-grained lineage. In: Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing. pp. 143–153 (2017)