**Deep Learning Hub: A Novel Open Access Graphical Programming Environment for Developing Reliability and Predictive Maintenance Solutions**

Mohammad Pishahang, Enrique López Droguett  
Mechanical Engineering Department, University of Chile, Santiago, Chile

**ABSTRACT**

The rapid evolution of artificial intelligence landscape, besides the global trend toward Industry 4.0, made Deep Learning an inevitable part of the reliability, predictive maintenance and digital twin areas. Although the development of integrated and mature frameworks like Tensorflow and Keras made the implementation and deployment of Deep Learning based fault diagnostics and prognostics models less challenging than ever, still “coding” remains the main barrier for many non-computer-scientists in the reliability and maintenance community. In order to tackle these challenges, here we introduce Deep Learning Hub (DLHub): an open access and web-based graphical programming environment for developing Deep Learning solutions in general, and for reliability and maintenance areas in particular. In DLHub, implementation of the data flow diagrams in an intuitive drag-and-drop manner not only offers a fast prototyping and code-free platform by helping the users to focus on the concepts rather than the syntax; but also makes the opportunity to guide the users toward error-free prototyping in the upcoming versions. Another contribution of DLHub is the standardization on the structure of Deep Learning projects by offering a comprehensive path from preparing datasets all through the predictions and necessary assessments, besides the standard auto generated Tensorflow code for each step. DLHub has been designed as an extendible and progressive platform, which means that by the development of new concepts and models in Deep Learning, it is easy to add new modules and blocks to it. Finally, adopting the Jupyter technology provides a unique opportunity for the same online tool, to be used for local or cloud computing. Furthermore, we demonstrate the flexibility and usefulness of DLHub in the reliability and predictive maintenance by means of one case study in civil aviation industry.

**KEYWORDS:** Deep Learning, Tensorflow, Keras, Graphical Programming

**1. INTRODUCTION**

With the advent of the Industry 4.0, Industrial Internet of Things (IoT), the proliferation of inexpensive sensing technology and the advances in prognostics and health management, customers are not only requiring their physical assets investment be reliable, but are also requiring their assets diagnose and prognose faults and alert their maintenance staff when components need to be replaced. These assets often have substantial sensor systems capable of generating millions of data points a minute. Artificial Intelligence in general, and Deep Learning in particular, arises as a powerful approach for smart, autonomous and on-line early fault detection and prognostics of remaining useful life as it provides tools for processing and analyzing Big Machinery Data, i.e., massive and multidimensional sensor data.

The application of deep learning in reliability and predictive maintenance is getting quite mature during the recent years. Deep learning can be used in a wide spectrum of subjects from automatic fault detection and identification [1-3], to damage localization and quantization [4, 5], to predictions of the remaining useful life of the physical assets [4, 6]. The huge consequences of deep learning methodologies in increasing safety and reliability and decreasing costs motivate the vibrant community to adopt cutting-edge technologies in this area.

The development of comprehensive and integrated frameworks like Tensorflow [7], Keras, PyTorch [8] and many more, made working in deep learning less challenging than ever. However, “coding” is what prevents a soft transition zone, and rapid residency to this new world for the non-computer-scientists. On the other hand, for the ones who have passed this obstacle, problem prototyping via coding is a time-consuming process, and it is even more challenging when various people with different levels of proficiency in deep learning or coding want to share their model prototypes in a team.

The solution presented in this paper is Deep Learning Hub (DLHub), an open access web-based visual programming framework that can be used by engineers, medical and finance people for a fast code-free prototyping of their deep learning problems. In this paper, we shall exemplify the power and flexibility of DLHub in the context of smart reliability and predictive maintenance.

The remaining of this paper is organized as follows. In section 2, a general description of DLHub and its background technologies are presented. Section 3 explains the general map of a deep learning project in DLHub and describes the standard steps that the user must go through to have a complete project. Section 4 provides a detailed case study on prognostics methods for CMAPSS turbofan dataset. Section 5 discusses the implementation of more sophisticated models in DLHub. Section 6 ends the paper up with some conclusion remarks.

**2. APP ARCHITECTURE**

The general architecture of DLHub is similar to Jupyter notebooks. It consists of two separated components that work together: a web application and a Jupyter Kernel Gateway. The web app makes a visual environment with facilities and tools that offer an intuitive way to implement and develop a deep learning project. More details on this subject are discussed in the following sections. The kernel gateway, on the other hand, is the official product of the Jupyter open source project with minor changes. It has been rebranded as “dlhub-gateway” to be distinct from the official version and can be installed simply via standard Python package management system. This gateway is responsible to run the codes on the client’s machine.

The front-end is implemented as a web application (https://dlhub.app), and that brings many advantages by itself. First, a web application works cross-platform on many operating systems (Windows, Mac and Linux) that have a browser installed. Theoretically, using DLHub on any modern browser should make the same experience for the users, but Google Chrome is the preferred one to be used, and many of the debugging processes in the current version have been performed on that. Second, upgrading the web application on the server brings the last features for every user without any further installations. Third, being a Single Page Application (SPA) helps the app to work completely inside the browser and without any further communications with the server. It makes the app more stable on weak internet connections, and even lets the application to work offline while it has been loaded completely. Finally, it is possible to open DLHub on various tabs and to work simultaneously on different projects. This could help the user to compare various models on the same dataset at the same time.

All the communications between the web app and the kernel gateway are done via HTTP and WebSocket protocols. HTTP is used to perform some operations like starting a new kernel or interrupting or shutting down the existing kernels. On the other hand, WebSocket is used to transfer the execution results in real time. These connections offer the following advantages: the user develops deep learning projects in an online tool and runs them on her/his local machine. That way, she/he gets all the benefits of an online tool, while keeping 100% privacy of her/his data.

As it is mentioned before, the kernel gateway is a part of Python ecosystem that the user needs to have installed on her/his computer. One can install and configure Python manually, but it is preferable to use Anaconda to install and configure it automatically. Once Python is installed, the user can install DLHub Gateway using the following snippet:

pip install dlhub-gateway

DLHub lets the application to be used anonymously. However, signing-in to the app makes it possible to save all the projects in the user’s account and to work on them later, on any other computer. For the sake of simplicity and security, for the moment the users are permitted to sign-in to the app by any Google accounts (Gmail, or any other kind of Google accounts). Navigating to the app’s dashboard, the user can decide if she/he wants to get connected to a kernel gateway or not. If the app does not get connected to the gateway, still it can be used to edit existing projects or to work on a new project. But in this case, the app does not run any code, and it works only as a code generator. Copying and pasting the following snippet in a command prompt runs a new kernel gateway on the client’s computer:

dlhub-gateway --KernelGatewayApp.allow\_origin='\*' --KernelGatewayApp.allow\_headers='\*' --KernelGatewayApp.allow\_methods='\*'

As it is shown in Figure 1, running this snippet starts a new Kernel Gateway App on the client’s localhost, and on a specified port. By default, the port is 8888, but if this port is occupied by another application like a Jupyter notebook, another port number will be selected for the kernel gateway. Now, by setting this port number in the web app, both parts of the app could be get connected (Figure 2).



Figure 1- Running kernel gateway on the user’s computer

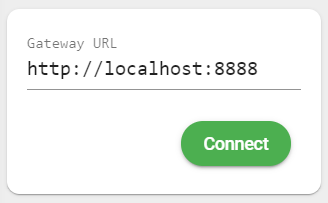


Figure 2- Connecting the web app to the kernel gateway

**3. PROJECTS STRUCTURE**

Any project in DLHub has five main sections. By starting a project, the user automatically navigates to the first section. In each step the app generates a standard Tensorflow/Keras code that will be run via kernel gateway. These steps are as follows:

**3-1. Preparing Datasets**

In the first section, the user works in a graphical programming module to construct a fluxgram which typically starts with reading a file containing data and, after few very common preprocessing blocks, ends up with one or more datasets. It is important to mention that the preprocessing blocks that already exist in this step are limited to simple data structure manipulations, few scaling blocks, and a couple of data label preprocessing. More sophisticated preprocessing steps usually must be performed outside the app (typically in a Jupyter notebook). In DLHub, a dataset is a block containing two matrices: the first one is the features, and the second one is the targets (labels). A dataset necessarily does not have both features and targets. They also may have only features or only targets.

**3-2. Designing Network Architecture**

The second step also offers a visual programming module that lets the user to construct the network architecture. This architecture starts with one or more input layers and can end up with one or more outputs. It is possible to have multiple separated architectures. All the official Keras layers have been created in DLHub, and the user can construct whatever network that can be done via coding. Networks with shared layers, Multi- input/output networks, and “time distributed” layer wrappers are also available in DLHub. Moreover, by selecting a layer, all its parameters can be modified in a very user-friendly manner. After each single modification on the blocks in this step, the application automatically generates a standard Tensorflow code using Keras functional API.

**3-3. Defining Models**

The third section provides the user with tools that help forming a list of models based on the previous architectures. Each model can be defined based on its inputs/outputs or can be a stack of previously defined models. In the case of a stack, it is possible to freeze whatever part of the model. If the model is chosen to be compiled, the user has options to select loss function and metrics for each output, as well as the optimizer for the model.

**3-4. Listing the Tasks to Run**

The fourth step is to make a list of tasks to run. There are four main tasks for each model: fit (train) the model, evaluate the trained model with test data, make predictions using the trained model, and finally, assess the quality of the predictions. In each task, the user can select the model, and its inputs/outputs from some lists which are populated based on the previous steps. Moreover, many other hyper-parameters can be modified for each function. Especially, for the “fit” task one can activate “early stopping” and “model checkpoint” callbacks.

**3-5. Running the Tasks**

Finally, by navigating to the last step, the user can run each task and analyze the results. Each task, based on its characteristics, generates a visual and user-friendly way to represent the results. The fit task shows a dynamic graph which plots the loss as well as all the other metrics in real time while the model is training. In addition, an interactive table shows the numerical values for all the metrics. The assess task, which is used to evaluate the quality of the predictions of a model, based on the problem’s characteristics can generate an interactive confusion matrix (for classification cases) or a scatter plot (for regression problems).

**4. CASE STUDY**

The workflow of DLHub would be understood more intuitively through an example. The CMAPSS dataset [9] has been chosen to exemplify a prognostics model. This dataset contains simulated turbofan run-to-failure data and was published first by NASA’s Prognostics Center of Excellence (PCoE) in 2008 [10]. CMAPSS contains various datasets, but here the FD001 is used. This dataset simulates the Remaining Useful Life (RUL) of turbofans in one work conditions and one fault mode.

The dataset has been shuffled and divided to the training and testing subsets. There are 15071 data for training, and 100 data for testing. Each data is a time window containing 30 consecutive time steps for 14 different variables (sensors) which are stored in a flattened format and are already scaled between 0 and 1. The dataset is stored in Python “.npz” file, which contains four matrices: “x\_train”, “y\_train”, “x\_test” and “y\_test”, which are features and targets of the training and testing subsets, respectively.

As shown in Figure 3, the first step in DLHub is to read the file, extract matrices and construct two datasets for training and test subsets. Each dataset consists of two matrices: features (x) and targets (y). All the process is being performed by putting proper pre-defined blocks together by drag-and-drop, and then making the data flow diagram by linking them together. During this process, DLHub automatically generates the python code corresponding to the data flow.

By navigating to the next step, the previously generated Python code runs, and the app keeps the datasets for further uses. In the second step, the architecture must be constructed. The same visual programming environment is used to form the fluxogram and all the Keras layers are available. Typically, the diagram starts with input layers, and in this case the app helps the user by setting the dimensions of the input layer (Figure 4).

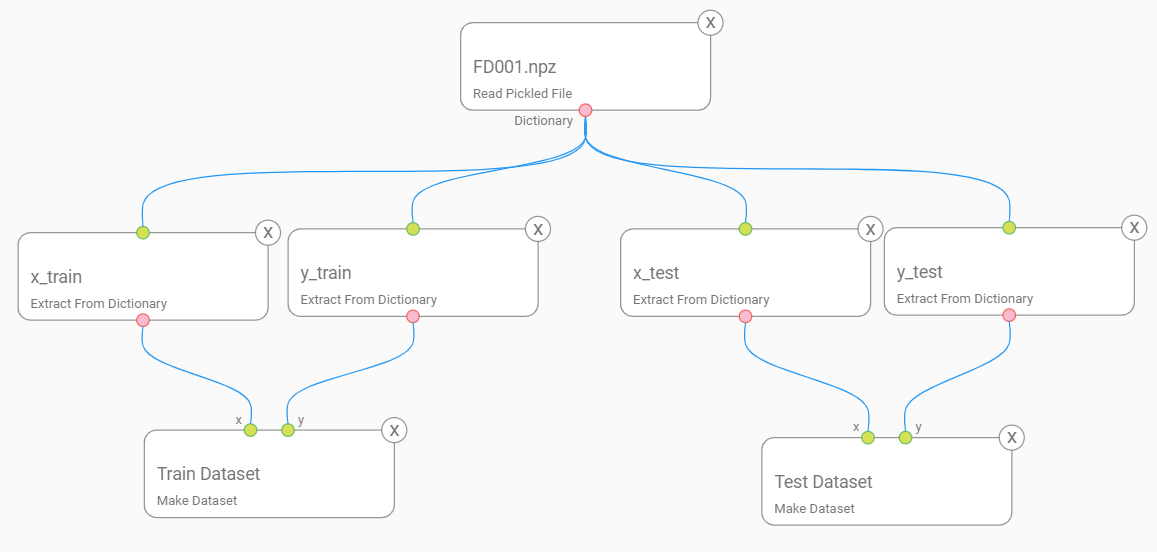


Figure 3- Forming train and test datasets

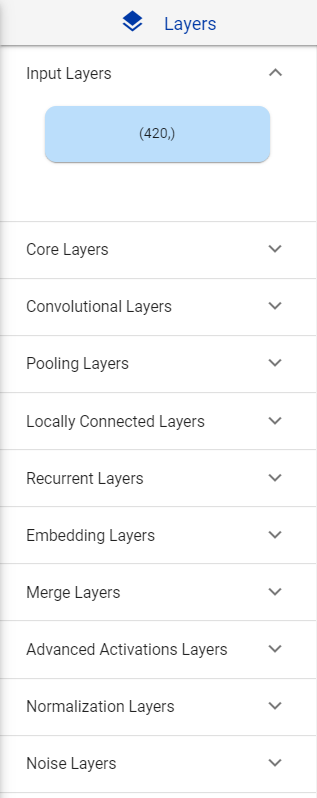


Figure 4- The custom input layers based on the user’s dataset

Building the layers architecture, and consequently the model, is straight forward. Figure 5 shows a simple stack of layers to be used for the fault prognostics. The input layer is followed by reshape, simple RNN, flatten, and two dense layers. All the parameters for any layer can be modified visually. Figure 6 shows how easy is to define a model based on the layers. It is only needed to define the input and the output of the model, as well as the loss function for the output. Moreover, the optimizer and other advanced features can be modified in the same page. As mentioned in the previous sections, any visual modification in the app is converted to a standard Python code. Figure 7 shows an example of such a code which is generated for this model.

|  |  |
| --- | --- |
| Figure 5- Network Architecture | Figure 6- Making a model based on the architecture |

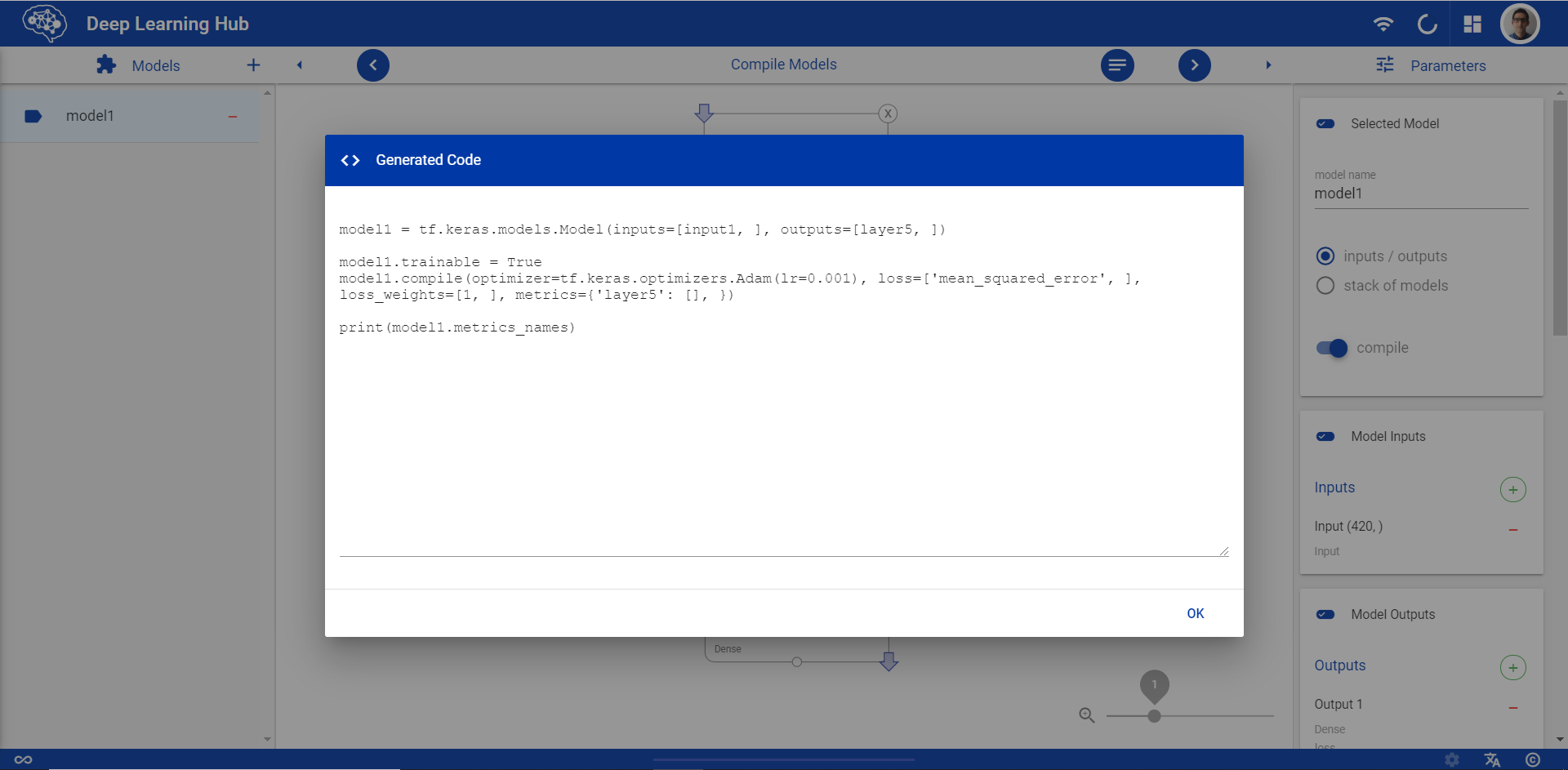


Figure 7- An example of auto-generated code in the app

In the “Timeline” page, the user can make a list of different tasks to run. Standard task types have been defined in the app. For each task, there are many configurations to be considered (Figure 8). In this example, the tasks are as follows:

1. Train the model using the train dataset, keeping 10% of data as the validation set.
2. Evaluate the trained model using the test dataset.
3. Predict the remaining useful life (RUL) value for the test dataset.
4. Assess the quality of the predictions through a comparison between the real RUL values and the predicted ones.

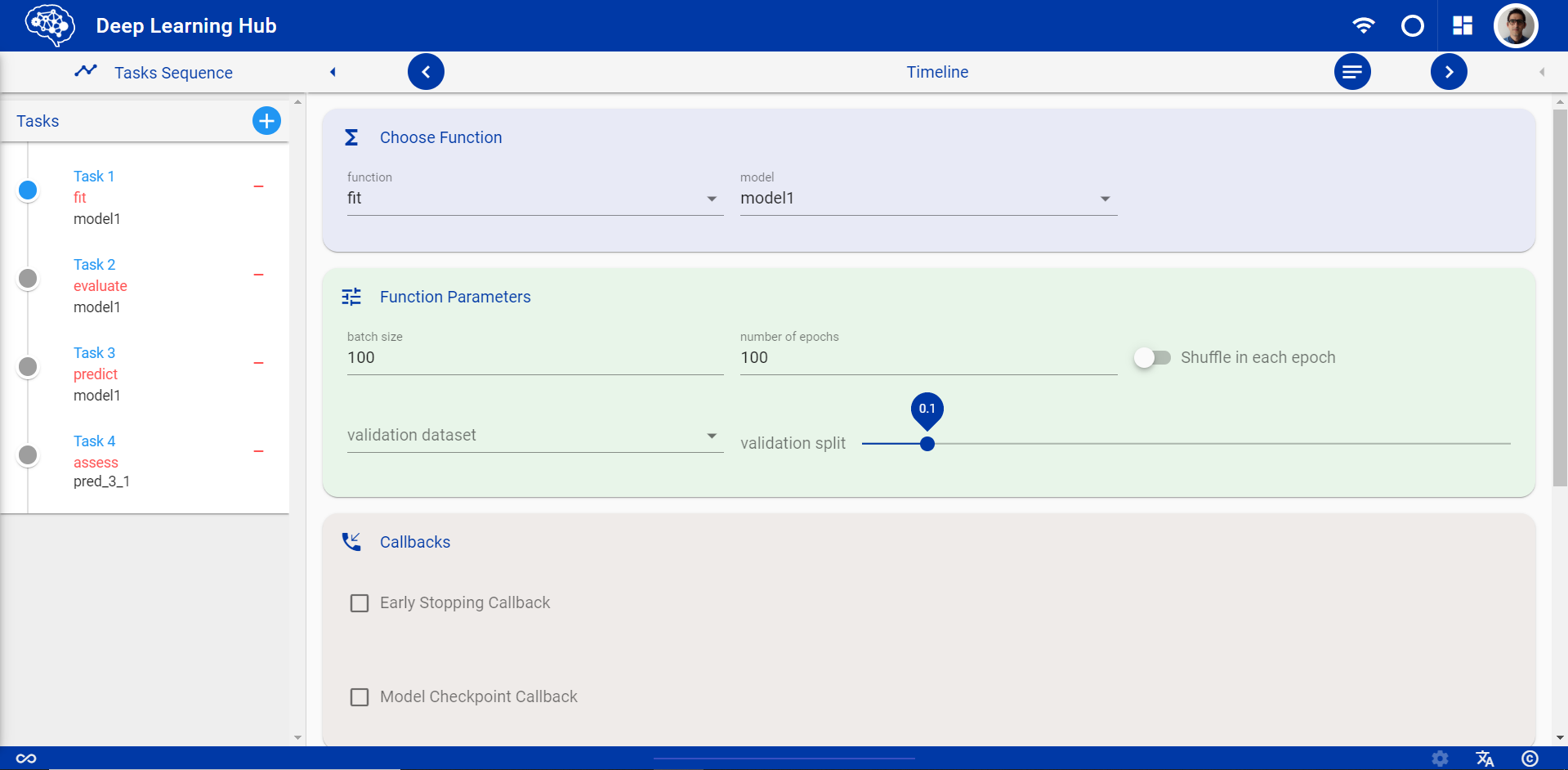


Figure 8- Configuring the tasks list

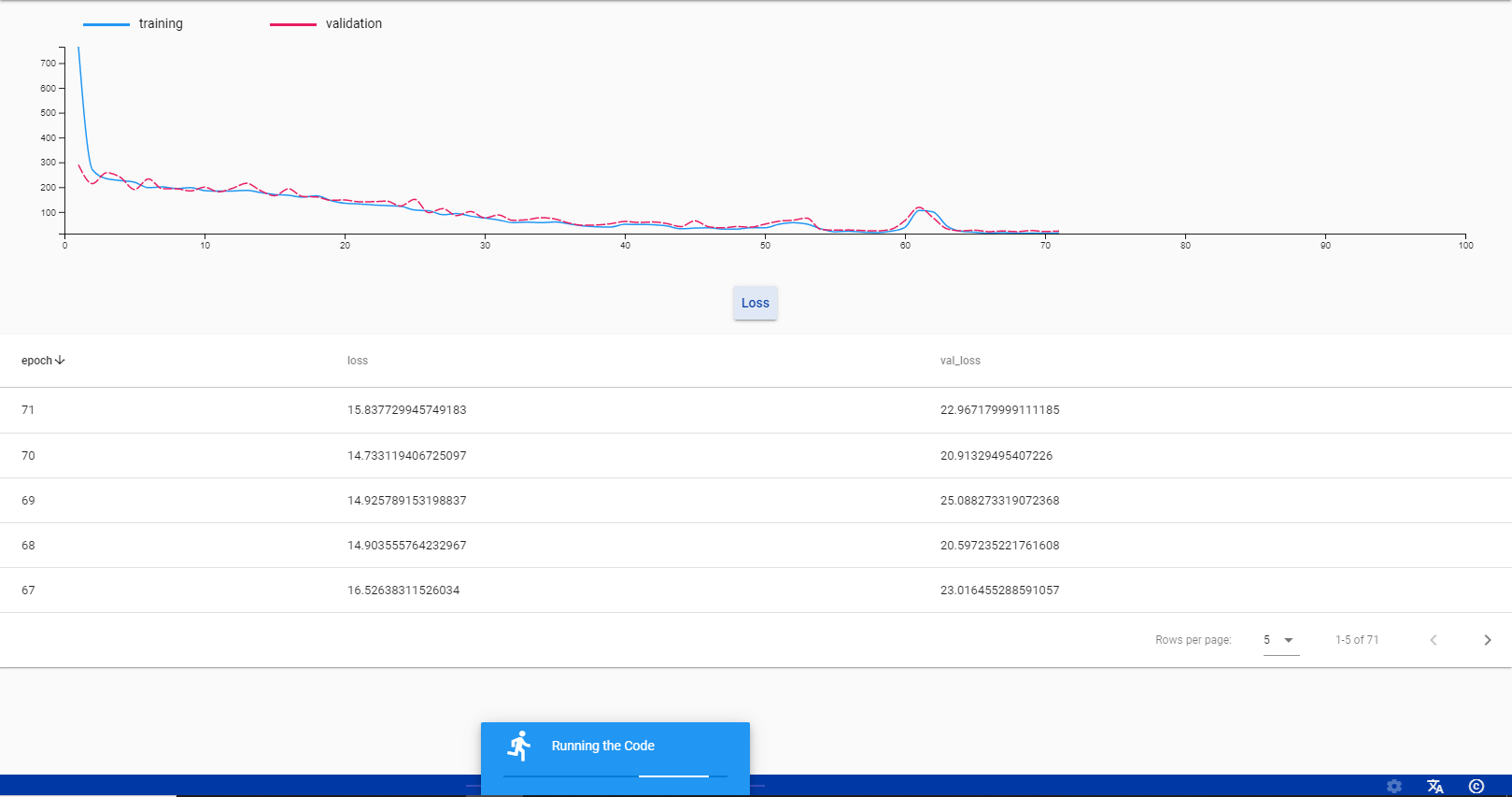


Figure 9- Training the model and receiving real time logs

Running each task generates user friendly and interpretable graphical results. Figure 9 shows the training curves while training the model, and Figure 10 shows the assessment process via a scatter plot. Note that in this example the model is performing a regression task, and that is why the assessment is being performed by a scatter plot. On the other hand, in case of a classification task (such as fault diagnostics of discrete health states), DLHub performs the assessment through an interactive confusion matrix.

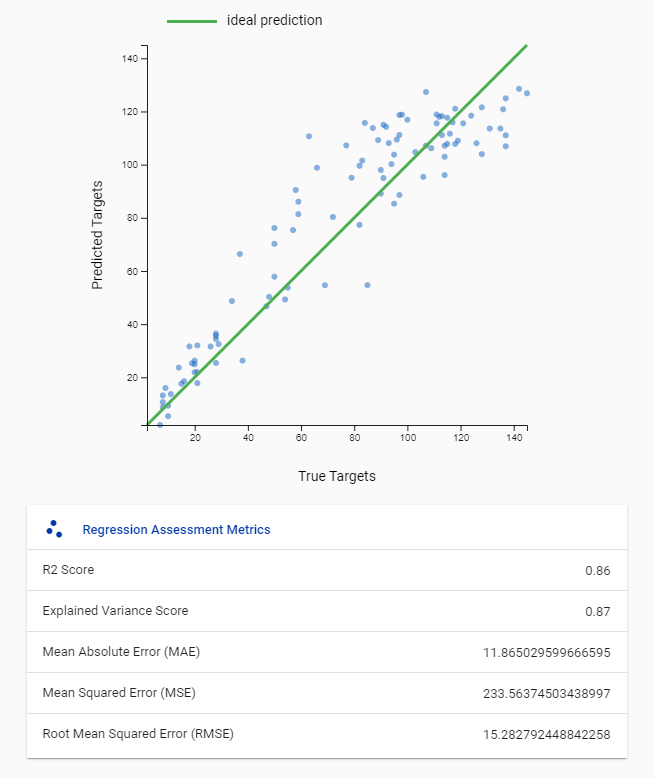


Figure 10- Assessment of the predictions via a scatter plot

**5. DISCUSSION**

Section 4 of the paper provided a simple example of a deep learning prognostics. In this example one model has been defined on a simple stack of layers, and then the model has been trained and evaluated. However, DLHub lets the user to develop more sophisticated models like autoencoders, variational autoencoders, and generative adversarial networks (GANs). Such models usually are collections of simpler models, and some more complicated procedures are implemented in the app out of the box so to make the whole process more intuitive for the users. For instance, a “sampling” layer has been added to DLHub. This layer is not officially a part of Keras but using this layer the implementation of variational autoencoders can be done very simply. Another example could be the training process for the GANs, where DLHub just requires recognizing three models - the “generator”, “discriminator”, and the “GAN” - and the whole training process is done automatically.

On the other hand, some more specific cases are being added to DLHub. The main goal is to add some problem schemes in the field of reliability and maintenance. An example is the procedure of anomaly detection using autoencoders and variational autoencoders. Such very specific modules can help the users in extremely fast prototyping of their diagnostics and prognostics problems.

**6. SUMMARY & CONCOLUSIONS**

Deep Learning Hub is a web based application which helps the users with different levels of proficiency for a fast and code-free prototyping of deep learning models. Although by using DLHub one can almost do anything that can be done in Keras, there is still a long way ahead. New features are constantly being added to DLHub such as data dimension tracking in diagrams, three-dimensional visualization of the architecture, new reliability-specific problem schemes, among others.

**REFERENCES**

[1]. Modarres, C.; Astorga, N.; Lopez Droguett, E.; Meruane, “V. Convolutional Neural Networks for Automated Damage Recognition and Damage Type Identification”. Structural Control and Health Monitoring, v.25, 2018.

[2]. Verstraete, D.; Lopez Droguett, E.; Meruane, V.; Modarres, M.; Ferrada, A. “Deep Semi-Supervised Generative Adversarial Fault Diagnostics of Rolling Element Bearings”. Structural Health Monitoring, 2019.

[3]. San Martin, G.; Lopez Droguett, E.; Meruane, V.; Moura, M. “Deep Variational Auto-Encoders: A Promising Tool for Dimensionality Reduction and Ball Bearing Elements Fault Diagnosis”. Structural Health Monitoring, v. 18, n. 4, 2019.

[4]. Aria, A.; Lopez Droguett, E.; Modarres, M.; Azarm, S. “Estimating Damage Size and Remaining Useful Life in Degraded Structures Using Deep Learning-Based Multi-source Data Fusion”. Structural Health Monitoring, 2019.

[5]. Cofre, S.; Kobrich, P.; Lopez Droguett, E.; Meruane, V. Deep Convolutional Neural Network Based Structural Damage Localization and Quantification Using Transmissibility Data. Journal of Shock and Vibration, 2019.

[6]. Ruiz-Tagle, A.; Lopez Droguett, E.; Pascual, R. A Novel Deep Capsule Neural Networks for Remaining Useful Life Estimation. Journal of Risk and Reliability, 2019.

[7]. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. “TensorFlow: Large-scale machine learning on heterogeneous systems”, 2015. Software available from *tensorflow.org*.

[8]. Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam, “Automatic differentiation in PyTorch”, 2017. NIPS Autodiff Workshop

[9]. Saxena, A., & Goebel, K., “C-mapss data set”, 2008. NASA Ames Prognostics Data Repository.

[10]. Emmanuel Ramasso, Abhinav Saxena, “performance benchmarking and analysis of prognostic methods for cmapss datasets”, 2014. international journal of prognostics and health management, ISSN2153-2648, 2014 014