Saga: An Open Source Platform for Training Machine Learning Models and Community-driven Sharing of Techniques

Rune Johan Borgli, Håkon Kvale Stensland
Simula Research Laboratory, Norway
University of Oslo, Norway

Pål Halvorsen, Michael Alexander Riegler
SimulaMet, Norway

***Abstract*—With the increasing popularity of machine learning in areas such as multimedia indexing and social media analysis, comes an increasing number of tools for developing and training the machine learning models. These tools assist in the selection and optimization of hyperparameters, but the user becomes locked into the platform's built-in solutions. Therefore, this demo presents an open-source, community-driven platform for sharing machine learning models, training techniques, and datasets. The platform, called Saga, provides a machine learning training pipeline such as those found in machine learning services provided by cloud providers. However, Saga also provides a store where users can upload and share their machine learning training methods. Additionally, the store allows users to rate and comment on other users' uploaded methods, as well as download and run them without any additional setup. In our demo, we will run a scenario where a user wants to train an image classifier using Saga. The demonstration will involve downloading methods from the store and displaying the pipeline provided by the Saga platform.**

***Index Terms*—machine learning; training pipeline; platform; plugin store; method sharing**

I. Introduction

The field of machine learning has expanded rapidly after deep learning started producing impressive results. In the multimedia community, machine learning is now one of the most commonly used tools [1]–[4], and is used in applications such as social media analysis [5], image analysis and generation [6], and multimedia data indexing [7]. As part of this expansion, frameworks have been developed, making machine learning more accessible. These frameworks, such as PyTorch [8] and TensorFlow [9], are complex but try to retain a simple application programming interface (API). However, in-depth knowledge of machine learning and programming is needed for advanced functionality. While one can get a classifier trained relatively quickly with a framework such as Keras [10], adding, for instance, hyperparameter optimization requires special programming and knowledge.

Working with these frameworks alone can limit a user as they often mainly focus on the creation and training of machine learning models. The reason why this is limiting is that a full machine learning workflow should include data pre-processing and analysis of the training in order to improve the machine learning model. Many opt for a solution where they create their own tools for these jobs. As a response, in recent years, several tools and services covering a full machine learning training pipeline have been created for the public.

Cloud computing providers, such as Amazon Web Services [11], Google Cloud Platform [12], and Microsoft Azure [13], have invested heavily into tools and services for developing and deploying machine learning applications. These tools cover the entire pipeline, and one of the major selling points is their access to powerful GPUs and other hardware useful for large machine learning tasks such as pre-processing or training. However, these services can be expensive, and they are often bound to vendors and specific tools. There is room for an open source solution which can be deployed both in the cloud and locally.

As part of their pipelines, the major cloud computing providers have automated training of machine learning models implemented in their platforms. Automated training allows users to optimize their models by, for example, using hyperparameter optimization [14], [15]. However, you are bound to the cloud providers implementation of the automated training and training tools and have to trust that they use the best solution. Aside from that, creating and training machine learning models involves a lot more than what the automated machine learning tools can cover. Different techniques for machine learning are invented all the time. Sharing the implementation of these techniques are often done through Git repositories or using Docker. Allowing methods to be shared without any additional setups and requirements would not only be a significant contribution to the development of machine learning methods, but also to the applications in where we see machine learning used today, such as in media processing, content adaptation, and multimedia data indexing.

In this demo, we present the Saga platform. Saga is an open source, community-driven platform for sharing and running methods of training machine learning models. The platform's store allows users to download community-created methods for training models, analyzing and pre-processing datasets, and analyzing, visualizing, and evaluating trained models. The plugin store also allows users to upload their methods and rate and comment on others' work. Each method contains source code and Docker images, allowing the method to be used in a

978-1-7281-4673-7/19/$31.00

plug-and-play fashion. Saga also allows each plugin to provide a user interface to compliment their method optionally. Thus, Saga allows for training of models using both a web-interface and a terminal interface.

The platform introduces a three-step pipeline where each step uses plugins for its features. The pipeline has two types of data flowing through it, datasets and models. The step decides what data is taken as input and what is given as output. Datasets can be converted to different formats, which are also available as plugins. Our ambitions for Saga is not only to be a platform for training your own models but to be an ecosystem for sharing ideas and testing and giving feedback on the ideas of others. It is built on the idea of open software and open ideas, and with it, we hope users can more easily understand the state-of-the-art and be inspired to create new ideas while creating more powerful machine learning models.

This paper is structured as follows: First, we introduce services that are similar to Saga in that they allow for a workflow for training machine learning models. We also discuss the differences between Saga and these systems. Next, we describe the platform and its implementation, and we discuss design decisions. Lastly, we describe the demo.

II. Related Work

In the tool space for the training of machine learning models, we find a myriad of different tools: There are the cloud providers, such as Google [12], Amazon [11], and Microsoft [13], who provides a streamlined tool for training and deploying machine learning methods. With all the heavy competition in this space, these tools are rapidly improving and claimed to be state-of-the-art. In addition to cloud services, we have platforms created by smaller companies such as Valohai [16] and Determined AI [17]. These platforms allow the users to deploy similar services like those provided by the cloud providers on a local resource. Also, smaller companies often provide extra features, such as a focus on reproducible results. Lastly, there exist several tools at a lower level for training. An example of this is Auto-Keras [18] and other, similar automated training approaches [14].

While these tools are excellent and powerful for their purposes, Saga separates from them by allowing community-created features such as automated machine learning, while still providing a robust pipeline for developing and training machine learning models. By using the store, advanced and experimental methods not found in conventional tools could be deployed. To the best of our ability, we could not find a platform allowing for the same sharing capability as Saga while also providing a pipeline for running and training machine learning models.

III. The Saga Platform

Saga is a platform based on the web-framework React.js [19], communicating using WebSockets to a Python back-end. The back-end provides an API for plugins to enable them to access datasets and models. Additionally, the API facilitates the communication between the front-end of plugins to their back-end. We have opted for a WebSocket based solution because of its full duplex communication. This communication type is useful for our use case as the server can push updates to the clients without the need for the clients to poll the server. The communication is done through the JSON format to a single endpoint, where a plugin identifier redirects the request to the correct plugin.

When using Saga, the user is met with a dashboard, including categories and plugins. Each category has enabled plugins displayed underneath. A plugin can have a 64x64 pixel icon, a name, and a description. The description can include links to relevant sources. By clicking a plugin's title or icon, the view of the dashboard is substituted with that of the selected plugin. However, the navigation bar at the top and the menu-bar to the left will still be displayed.

A. Datasets and models

The Saga platform supplies users with two types of input and output, datasets and models. A user can add datasets to the platform by either uploading a dataset through the user interface, by copying a dataset into the designated dataset folder, or by running a plugin from the pre-processing category which outputs a dataset. Datasets can be accessed by plugins in the training category and the analysis category, but these categories cannot create new datasets.

Datasets in the Saga platform are accessed through API calls. From these API calls, the user can specify a converter. Converters are plugins separate from categories, where their sole intent is to convert a given dataset into a requested format. We separate between converters and pre-processing although they could be merged. The reason is to make it possible for plugin creators to get the expected format without additional user interactions.

Models in the Saga platform refer to machine learning models. These models hold the weights of a model, information about its structure, and information about its training. Additional information from the training is stored in log-files, and the format of the log-file is up to the creator of the plugin. This means that different plugins in the training and analysis category might be incompatible. We hope to remedy this with proper documentation and a few required fields such as the number of epochs trained, iterations, and specific hyperparameters. However, if this configuration proves problematic, we will have to change it in the future.

B. The Pipeline

Saga provides three-stage pipeline covering pre-processing of datasets, training of machine learning models, and analysis and visualization of training models. There are no hard restrictions on what the contents of each category are, but each category has different access to the API. The pipeline is designed as follows, and shown in Figure 1:

1) **Pre-processing:** The first step of the pipeline is pre-processing. Plugins in this category take datasets as input and produce new datasets based on the functionality of the selected plugin. Plugins in this category should

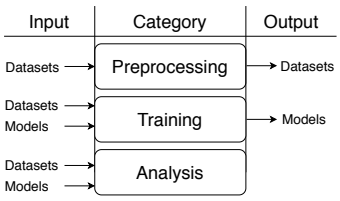

Fig. 1. Visualization of the Saga pipeline.

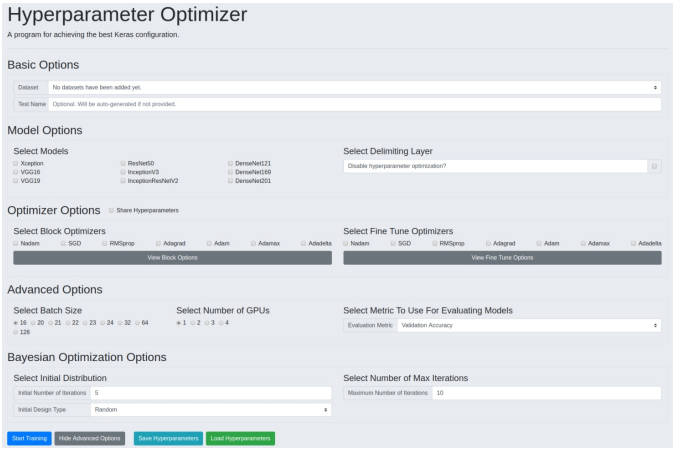

Fig. 2. Plugin-example from the training category performing hyperparameter optimization as described in [15].

be focused on dataset preparation, including sanitation, estimating the difficulty of predicting given classes for a dataset [20], and augmentation [21].

2) **Training:** The second step of the pipeline is the training of machine learning models. Plugins in this category take datasets, as well as optional models as their input. Models are optional because plugins can choose to retrain certain models or make intelligent decisions based on previous runs. Plugins in this category can also create new models from scratch. In this category, plugins should focus only on the creation of models and training of these. This spans from fully automated training to very basic framework-specific code. Figure 2 shows an example of a training plugin doing automatic hyperparameter optimization.
3) **Analysis:** The final step of the pipeline is the analysis and visualization of the previous machine learning model training. Plugins in this category take datasets and models as their input. This category is focused on understanding the behavior of the trained model and the effects of the training. Plugins should focus on visualization and analytics, and can include for example visualization of network activation per layer per image as described in [3].

C. Plugins

Plugins in Saga have an optional front-end and a back-end. The front-end of Saga utilizes the HyperText Markup Language (HTML) tag *iframe* to show a plugin. The iframe tag embeds an HTML document into the front-end. This approach allows plugins full freedom in their choice of styling and JavaScript without colliding with the styling and JavaScript already present in the Saga user interface. The back-end is written using Python and has access to an API. The API allows plugins access to datasets and models based on their category. The front-end of the plugin can only communicate directly to the back-end of the plugin. For instance, if the front-end wants a list over available datasets, it must communicate directly with its back-end, where the back-end can make an API request for the list and send it to the front-end.

All the features of Saga used for the training of machine learning models are provided through plugins. There are several reasons for this approach:

1) Plugins allows for community-created features, which would allow for more advanced and varied features than the platform creators could create themselves.
2) Plugins can be created by the user, allowing users to add their own features and prototype new functionality in a simple manner.
3) Plugins can be added and removed as needed, allowing users to customize the experience to only show functionality they actually use.
4) With a community creating plugins, users can find new and interesting features they otherwise would not find. These features are ready for the platform and can be run with no additional setup, in contrast to features found on the web where the user must make changes to the code to make it fit into their application.

Saga includes a plugin store. The plugin store is an online service where users can upload their own plugins and download and run community-created plugins. A plugin consists of the method, an optional front-end, and a Docker file or hosted Docker image containing the run time environment. This approach allows users to download and run plugins without any additional setup or requirements.

The store can also be used for managing all currently installed plugins, their descriptions, and their settings. Additionally, the user can enable or disable a plugin without uninstalling it. Each plugin has its own entry with a download button, where descriptions and screenshots of the plugin can be viewed. Additionally, users can rate and comment on each plugin. The plugin store will include search functionality and browsing based on popularity, new, and top rated. We have ambitions of keeping the plugins in the plugin store open source and free.

D. Processing

Saga supports scaling processing over different architectures and configurations. The user of the platform can configure what hardware is being used. This hardware includes cloud computing resources, remote hardware, and local hardware. These resources can be combined, but there are limitations on the available concurrency. Processing nodes cannot work together, but a processing node can contain multiple resources which can work together. A job manager takes jobs from the

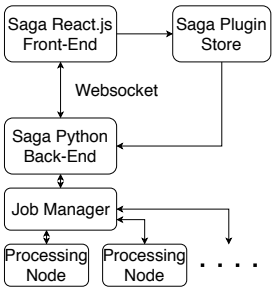

Fig. 3. Overview of the whole Saga platform, including how jobs are distributed for processing.

plugin back-end and distributes it to the requested available resources. If no processing node with the requested resource is available, the job manager adds the job to a queue.

Figure 3 gives an overview of the Saga system, including an overview of how the Saga back-end communicates with the job manager, which in turn communicates with its assigned processing nodes. Jobs can be managed in the Saga user interface from a dedicated job manager page. On this page, a user can see the status of jobs. The five statuses a job can have are:

1) **Running -** the job is running as normal.
2) **Queued -** the job manager is waiting for an available processing node.
3) **Stopped -** the user has cancelled the job.
4) **Finished -** the job has finished successfully.
5) **Error -** the job has been cancelled because of some error.

Besides seeing the statuses of jobs, the user can cancel, move priority, and read descriptions of the job. Additionally, plugins can run while a job is running and spawn additional jobs. Accessing the plugin is useful because the plugin can provide the user with a visualization of the job and detailed updates on the progress.

IV. Demo

In our demo, we will demonstrate the platform from a user's perspective, where the user wants to train an image classifier based on an eight-class dataset containing images of findings from the gastrointestinal tract called Kvasir [22]. First, we activate the plugins we want to demonstrate through the plugin store. We use one plugin from each category. After adding the plugins, we show how a user can upload the image dataset into the platform and augment the dataset with a pre-processing plugin. Then, we show an example of a training plugin to create and train the image classifier. The training plugin is an implementation of automated hyperparameter optimization based on previous work [14], [15]. Finally, we analyze the training through a plugin from the analysis category, which will show a table of predictions and visualize the weights of the model. We can from this plugin see how the classifier performs per image. This information can be used to understand the training and dataset better and help make informed improvements to the classifier training.

References

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

[2] K. Pogorelov, M. Riegler, S. L. Eskeland, T. de Lange, D. Johansen, C. Griwodz, P. T. Schmidt, and P. Halvorsen, "Efficient disease detection in gastrointestinal videos - global features versus neural networks," *Multimedia Tools and Applications*, vol. 76, no. 21, pp. 22 493–22 525, 2017.

[3] S. Hicks, M. Lux, T. de Lange, K. R. Randel, M. Jeppsson, K. Pogorelov, P. Halvorsen, and M. Riegler, "Mimir: An automatic reporting and reasoning system for deep learning based analysis in the medical domain," 2018, pp. 369–374.

[4] R. Mekuria, M. J. McGrath, V. Riccobene, V. Bayon-Molino, C. Tselios, J. Thomson, and A. Dobrodub, "Automated profiling of virtualized media processing functions using telemetry and machine learning," in *Proceedings of the ACM Multimedia Systems Conference*, 2018, pp. 150–161.

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[6] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *arXiv preprint arXiv:1812.04948*, 2018.

[7] L. Liu, Y. Yang, M. Hu, X. Xu, F. Shen, N. Xie, and Z. Huang, "Index and retrieve multimedia data: Cross-modal hashing by learning subspace relation," in *International Conference on Database Systems for Advanced Applications*. Springer, 2018, pp. 606–621.

[8] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *Proceedings of Conference on Neural Information Processing Systems*, 2017.

[9] M. Abadi and et. al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[10] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[11] Amazon, "Compute Services - Overview of Amazon Web Services," 2019. [Online]. Available: https://docs.aws.amazon.com/aws-technical-content/latest/aws-overview/aws-overview.pdf

[12] Google Cloud, "Google Cloud including GCP & G Suite," accessed: 2019-02-25. [Online]. Available: https://cloud.google.com/

[13] H. Li, *Introducing Windows Azure*. Berkely, CA, USA: Apress, 2009.

[14] R. J. Borgli, H. K. Stensland, M. A. Riegler, and P. Halvorsen, "Automatic hyperparameter optimization fortransfer learning on medical image datasets usingbayesian optimization," in *In Proceedings of the 13th International Symposium on Medical Information and Communication Technology*, 2019.

[15] R. Borgli, P. Halvorsen, M. Riegler, and H. K. Stensland, "Automatic hyperparameter optimization in keras for the mediaeval 2018 medico multimedia task." CEUR Workshop Proceedings (CEUR-WS.org), 2018.

[16] Valohai, "Deep Learning Management Platform — Valohai," accessed: 2019-02-25. [Online]. Available: https://valohai.com/

[17] D. AI, "Determined AI Platform," accessed: 2019-05-21. [Online]. Available: https://determined.ai/

[18] H. Jin, Q. Song, and X. Hu. (2018) Auto-keras: Efficient neural architecture search with network morphism.

[19] F. Inc, "React - A JavaScript library for building user interfaces," 2016, accessed: 2019-02-25. [Online]. Available: https://reactjs.org/https://facebook.github.io/react/index.html

[20] F. Scheidegger, R. Istrate, G. Mariani, L. Benini, C. Bekas, and A. C. I. Malossi, "Efficient image dataset classification difficulty estimation for predicting deep-learning accuracy," *CoRR*, vol. abs/1803.09588, 2018. [Online]. Available: http://arxiv.org/abs/1803.09588

[21] M. Kirkerød, V. Thambawita, M. Riegler, and P. Halvorsen, "Using preprocessing as a tool in medical image detection." CEUR Workshop Proceedings (CEUR-WS.org), 2018.

[22] K. Pogorelov, K. R. Randel, C. Griwodz, S. L. Eskeland, T. de Lange, D. Johansen, C. Spampinato, D.-T. Dang-Nguyen, M. Lux, P. T. Schmidt, M. Riegler, and P. Halvorsen, "Kvasir: A multi-class image dataset for computer aided gastrointestinal disease detection," in *Proceedings of the ACM Multimedia Systems Conference*, 2017.