Fabrik: An online collaborative neural network editor

Utsav Garg[1] **Viraj Prabhu**[2] **Deshraj Yadav**[2] **Ram Ramrakhya**[3]
Harsh Agarwal[2] **Dhruv Batra**[2,4]
[1]SAP Asia Pte Ltd [2]Georgia Institute of Technology [3]Inmobi [4]Facebook

Abstract

We present Fabrik, a neural network editor that provides tools to visualize, create, edit, and share networks from within a browser. Fabrik provides an intuitive GUI to import neural networks written in popular deep learning frameworks and allows users to build, interact with, and edit models via simple drag and drop. It is designed to be framework agnostic and support high interoperability and can be used to export models back to any supported framework. Finally, it provides powerful collaborative features for users to iterate over model design remotely and at scale. Code on GitHub[1].

1 Introduction

In recent years, long strides have been made in AI, primarily propelled by the advent of deep learning [6]. This progress has been catalyzed by the development of well-designed and documented open-source frameworks for deep learning. Over 23 such frameworks exist [5], each having a distinct strength, such as deployment and serving [1], inference on mobile devices [2], or extensibility for rapid prototyping [7]. However, keeping up with the latest frameworks presents a daunting challenge to newcomers and experienced researchers alike, and having to constantly learn new frameworks (for example, to run some other researchers work) proves to be a significant time sink. Moreover, these networks are developed iteratively and collaboratively, but the process is still largely restricted to whiteboards discussions, which does not scale well to global teams.

To address these obstacles, we present Fabrik, an open-source platform to visualize, create, edit, and share neural networks.

Visualize: Fabrik provides a simple GUI to import existing networks written in popular frameworks such as Caffe, Keras, and TensorFlow.[2]

Create: Fabrik has tools to create neural networks from scratch by dragging and dropping layers.

Edit: Fabrik supports editing layers and parameters of the neural network. Once done, the network can be exported to a target framework, providing easy interoperability.

Share: Fabrik provides powerful collaborative features for researchers to brainstorm and design models remotely. Multiple researchers can concurrently work on a single model and see all updates to it.

These features pose Fabrik to become a great model introspection tool for all neural network practitioners.

[1]https://github.com/Cloud-CV/Fabrik

[2]Note that these are the top-3 frameworks by popularity based on Github [9].

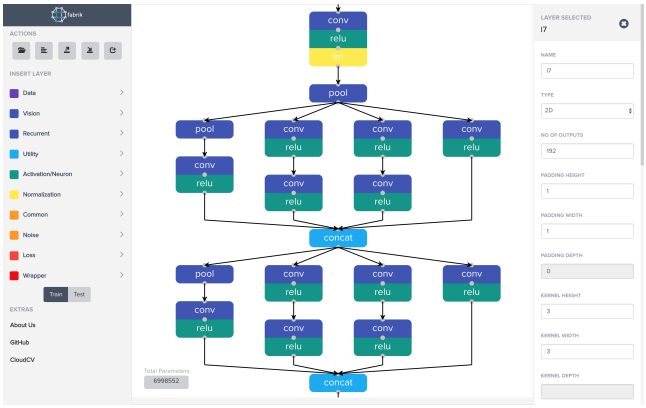

Figure 1. Visualizing the GoogLeNet [8] architecture with Fabrik.

2 Fabrik System Design

Fabrik can be represented by a 3-tier architecture, with the tiers being broadly classified as Presentation, Intermediate Representation, and Back-End. The following subsections discuss each of these components in detail.

2.1 Presentation Layer

The user-facing Presentation layer is one of the most important components as all user interactions depend on it. Figure 1 shows the landing screen displayed with the GoogLeNet [8] network being visualized. It has four distinct sections:

Canvas: At the centre of figure 1 is the main scroll-able canvas which shows the visualized network, allows the user to move the layers(nodes of a neural network) around, connect and disconnect layers, and add or remove layers.

Dashboard: Present on the left, it provides all of the core functions, such as searching and adding layers, importing or exporting models, sharing models with collaborators, or accessing the model zoo (A built-in collection of popular models available to import on a click).

Layer Parameter Pane: Shown on the right, it displays the parameters of the selected layer and allows the user to overview or edit any of these parameters as required.

Parameter Overview: A tool-tip is rendered (not shown here) on layer hover, providing a quick overview of that layers' configuration.

2.2 Intermediate Representation Layer

The intermediate representation allows us to decouple the display portion of the application from framework-specific back-ends. On importing a model from any of the supported

frameworks, the layers are stored as the corresponding layers in the intermediate representation. We use the prototxt-based model configuration protocol employed by Caffe [4] as our intermediate representation as it is clean and extensible.

As different frameworks might have slightly varying parameters or parameter names, we map them to corresponding names in this representation. Note that we extend the protocol to hold the union of all parameters required by all supported frameworks so that no information is lost. Creating such an intermediate representation is a challenging task. As each framework has its own peculiarities for handling layers. Take an example of the *padding* parameter (of convolution layers for instance), some frameworks use a numerical value for this while others use a text form ('SAME'/'VALID'). Such differences have to be appropriately handled so that converting between frameworks does not alter the model.

2.3 Back-End Layer

The back-end layer implements Fabrik's business logic. It consists of three main pillars:

Model conversion and export: The intermediate representation helps support conversion from one framework to another. Without it, separate logic would be required to port between each pair of supported frameworks. But now, the export logic is only required from the intermediate representation to each of the supported frameworks. This reduces the problem complexity significantly from $O(n^2)$ to $O(n)$, for n different frameworks.

For export, a back-end task is created which converts the intermediate representation to a model object of the respective framework. This task is made asynchronous through Celery [3] so that it does not block the user from performing actions on the canvas.

Real-time collaboration: For achieving real-time collaboration, a global copy of the shared model is maintained in the database and all updates over multiple sessions are performed on this, using communication via web-sockets. Fabrik also employs an efficient implementation of operational transformation for the intermediate representation of a model. These are categorised as – parameter update, layer addition, layer deletion, and layer highlight. Whenever one of these is performed, a corresponding update event is triggered and propagated to all active sessions for maintaining consistency in the collaborative session.

3 Use Cases

Visualizing Neural Networks: The prime objective of Fabrik is to provide users with the tool to visualize their network structure. To enable this, there are a few core functionalities built into the framework. Firstly, the user can import existing networks, this serves as a sanity check confirming if the network is built as intended. Networks' can be visualized by importing the network file, directly pasting the network configuration into the available text editor or by providing a GitHub link for it. Secondly, Fabrik provides a drag and drop neural network creator. Eliminating the need for a user to be familiar with the syntax of particular frameworks, and allows to simply drag and drop layers and edit their parameters to build a network from scratch. Lastly, Fabrik maintains and displays the total parameter count for the visualized network. The count is updated whenever a layer is modified. This provides an idea of the layers' impact on the parameter count and thus model complexity. The parameter count can be effectively used in teaching to show students the impact of a particular layer on the parameter count.

Cross Framework Model Conversion: Deep learning researchers tend to have a preferred framework in which they have expertise in and use to publish code for their research. When others wish to build upon that work but are unfamiliar with a framework, it proves to be a hindrance. Fabrik can bridge this gap as models can easily be ported to a framework the user is comfortable in. Any network being visualized in Fabrik can be exported to any of the supported frameworks, irrespective of it being imported from a different framework or being created in the application itself via drag and drop. This, however, can sometimes be bottle-necked by the absence of some layers in a framework.

Real-time collaboration: Fabrik provides users with a platform where they can build deep learning models collaboratively in real-time. Real-time collaboration allows users to share a model with others through a unique link, which can be used to access and edit it. Furthermore, users can view the history of updates performed on the model in a collaborative session. They can also revert to a specific state of the model at a particular point of time, which assists in debugging. Collaborators can also comment on various aspects of the model architecture to provide feedback and make working remotely efficient and transparent.

4 Conclusion and Future Work

In this work, we propose Fabrik, a simple GUI for visualizing, creating, editing, and sharing neural network models in the browser. Fabrik provides novel real-time collaboration features to assist researchers in brainstorming and design models remotely and at scale. It provides a framework-agnostic visualization scheme based on a novel intermediate representation that can be used to quickly and efficiently visualize model architectures. While of today it supports three popular deep learning frameworks, as future work we plan on adding support for additional popular frameworks, such as PyTorch. Further, we plan to incorporate ONNX as our intermediate representation as a step towards long term interoperability and maintainability. Finally, Fabrik will be available as an online service which provides instant access on any browser supporting device for the purposes of debugging, reproducibility, learning, and collaboration.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning.. In *OSDI*, Vol. 16. 265–283.

[2] Caffe2. [n. d.]. Caffe2: A New Lightweight, Modular, and Scalable Deep Learning Framework. https://caffe2.ai/.

[3] Celery. 2018. Celery: Distributed Task Queue. https://github.com/celery/django-celery.

[4] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 675–678.

[5] KDNuggets2017. [n. d.]. Ranking Popular Deep Learning Frameworks for Data Science. https://www.kdnuggets.com/2017/10/ranking-popular-deep-learning-libraries-data-science.html.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[7] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. 2017. PyTorch.

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[9] Jan Zacharias, Michael Barz, and Daniel Sonntag. 2018. A Survey on Deep Learning Toolkits and Libraries for Intelligent User Interfaces. *arXiv preprint arXiv:1803.04818* (2018).