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]Nanyang Technological University [2]Georgia Institute of Technology [3]Inmobi [4]Facebook
`utsav002@e.ntu.edu.sg` `ramramrakhya81@gmail.com`
`{virajp,deshraj,harsh.agrawal,dbatra}@gatech.edu`

Abstract

We present Fabrik, a neural network editor that provides tools to visualize, 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 interact with, build, 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 is available at `https://github.com/Cloud-CV/Fabrik`.

1 Introduction

In recent years, long strides have been made in AI, primarily propelled by the advent of deep learning [LeCun et al., 2015]. This progress has been catalyzed by the development of well-designed and well-documented open source frameworks for deep learning. Over 23 such frameworks have been released as of last year [KDNuggets2017], several of which have distinct strengths, such as deployment and serving [Abadi et al., 2016], inference on mobile devices [Caffe2, 2018], or extensibility for rapid prototyping [Paszke et al., 2017]. However, keeping up with the latest frameworks presents a daunting challenge to newcomers and experienced researchers alike, and having to constantly learn new frameworks (say, to run a model written using an unfamiliar framework), regularly proves to be a significant time sink.

Moreover, neural networks developed in such frameworks are often designed *collaboratively* over the course of several iterations. However, the process of this model design and development is still largely restricted to whiteboard discussions, and scaling such collaboration to remote and online settings remains an open problem. Considering the popularity of deep learning, a tool for collaborative design of neural networks is likely to have widespread utility.

To overcome these challenges, we present Fabrik, an open source platform to visualize, edit, and share neural networks. See Figure 1.

Visualize. Fabrik provides a simple GUI to create networks via drag and drop or import neural networks written in popular frameworks such as Caffe, Keras, and TensorFlow.[1]

Edit. Fabrik supports editing layers and parameters of the neural network right in the browser. Once completed, 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. A researcher can generate a shareable link for a model, and can see real time updates and comments from their collaborators.

Fabrik is targeted at newcomers and experienced deep learning practitioners alike. For newcomers, it lowers the barrier to entry to the field by enabling them to interact with neural networks online without

[1]Note that these are the top three frameworks by popularity based on Github metrics [Zacharias et al., 2018].

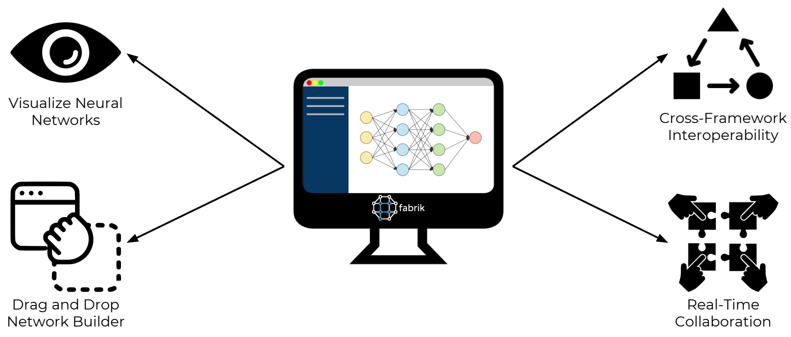

Visualize Neural Networks
Drag and Drop Network Builder
fabrik
Cross-Framework Interoperability
Real-Time Collaboration

Figure 1: We present Fabrik, an online collaborative neural network editor.

having to grapple with syntax. For researchers, it provides a tool to easily visualize the depth and breadth of their network and verify correctness of implementation. Further, it allows porting between frameworks to enable them to work in their preferred one. Finally, it provides collaborative features to brainstorm about model design remotely and at scale, and to share interactive visualizations of their models to aid in reproducibility.

2 Related Work

Deep learning in the browser. A number of browser-based deep learning tools have been released recently. Some are geared towards providing a platform to manage and run deep learning experiments [Aetros, 2018, KerasJS, 2017]. However, these are either paid [Aetros, 2018], or restricted to a single framework [KerasJS, 2017]. Yet others, like [ConvnetJS, 2016] have been proposed primarily as teaching tools and do not plug into existing popular frameworks. In this work we propose Fabrik as a tool for easy visualization of models from a range of popular frameworks.

Visualizing Neural Networks. A few tools designed for visualizing neural networks have been proposed. For example, [Netscope, 2017] supports visualizing arbitrary networks available in Caffe's prototxt format. However, it works only with Caffe models, and only supports import of existing models. [Tensorboard, 2018] is a web application designed for debugging and visualizing models written in TensorFlow. While a powerful tool, it requires writing code to set up, and is thus not accessible for newcomers. Moreover, it only works with TensorFlow (and related frameworks such as Keras). Fabrik supports interactively building models from within a browser, collaborating with other researchers, and exporting to popular frameworks, without having to write a line of code.

Cross-framework support. A few tools [ONNX, 2018, MMdnn, 2018] for deep learning framework interoperability have been recently proposed. [ONNX, 2018] provides an open source format for AI models, and is supported by many frameworks, but has not yet been adopted by TensorFlow (currently the most popular deep learning framework). We view ONNX as an alternative intermediate format that be integrated into our application in the future. Similarly, [MMdnn, 2018] is a cross-framework solution to convert between and visualize deep neural network models written in several frameworks. However, it does not provide tools for building models from within the browser. Additionally, it is not an online service and is not designed for remote collaboration.

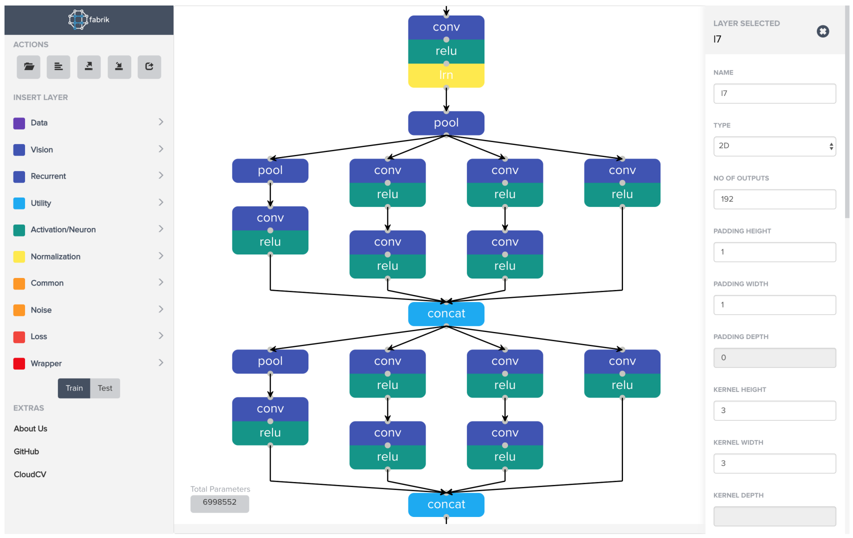
3 Application Overview

3.1 Presentation Layer

Figure 2 shows the landing screen displayed with the GoogLeNet [Szegedy et al., 2015] network being visualized. We can see two distinct sections – on the right is the *Canvas* where the neural network is displayed, and on the left is the *Dashboard*, which provides all of Fabrik's core functionalities.

3.1.1 Canvas

The canvas displays the neural network being visualized. This uses the jsPlumb [jsPlumb] library that allows for a simple interface for connecting UI elements. The layers of the network are represented



Total Parameters
6998552

LAYER SELECTED
l7
NAME
l7
TYPE
2D
NO OF OUTPUTS
192
PADDING HEIGHT
1
PADDING WIDTH
1
PADDING DEPTH
0
KERNEL HEIGHT
3
KERNEL WIDTH
3
KERNEL DEPTH

Figure 2: Visualizing the GoogLeNet [Szegedy et al., 2015] architecture with Fabrik.

as nodes of a graph which are connected based on layer dependencies. These nodes can be dragged and re-positioned on the canvas. Upon hovering over a layer, an overview of layer parameters is displayed as a tool-tip. These layer parameters can be further edited in the side-pane (show on right).

3.1.2 Dashboard

The dashboard provides controls for all core functionalities – export, import, building, and sharing. Via the top menu, the user can access the Model Zoo, a collection of state-of-art deep learning models that we provide inbuilt support for (Table 1 provides a list). Other buttons allow the user to initiate import / export (with a prompt to select source/target framework) of their model, and to share the current model with other users through a unique URL. Below the top menu are layers that can be dragged and dropped onto the canvas to build or edit a model. These are collapsed into categories, each of which drills down to a set of layers. The added layers are initialized with default parameters which can be edited by the user.

3.1.3 Visualization Algorithm

The visualization algorithm plots the layer objects on the canvas. The main goal of the algorithm is to assign coordinates and establish connections between layer objects to yield an accurate and aesthetic visualization, while minimizing the number of overlapping layers and connections.

Minimizing overlap: To achieve non-overlapping position allocation, we implement a hashing technique: maintaining a constant height and width for each layer object, an initial position allocation is computed. Each time a layer has been assigned co-ordinates, the immediate area surrounding the layer is searched with the help of hashed co-ordinates of visited layers to detect overlap. If overlap is detected, a constant displacement is added to the y-coordinate of the layer, and we then check for overlap recursively. The search space for detecting overlap is reduced significantly via hashing, leading to much faster visualization. Further, to ensure non-overlapping layer connections various parameters are considered, e.g. the number of child nodes of a layer, position of the source layer etc. Based on these parameters, the slope of the connection arrow is calculated, and if needed the line is divided into sub parts to ensure that connection overlap and cutting connections are minimized.

3.2 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 [Jia et al., 2014] as our intermediate representation as it is clean and extensible.

As different frameworks might have slightly varying parameters or parameter names, we first 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 an intermediate representation that serves all supported frameworks well is a challenging task. For example, the Caffe framework uses numerical padding, to pad the inputs before applying operations such as Convolution or Pooling. But frameworks like Keras and Tensorflow use 'SAME' or 'VALID' padding, where the padding required is automatically determined based on input and output sizes. To map from both types of frameworks to the same intermediate representation, we first need to convert the 'SAME'/'VALID' type of padding to its numerical counterpart by determining the input and output sizes for each layer, and perform the reverse when converting back. Note that while we provide a single example here, many such corner cases exist and are appropriately handled by the back-end layer.

3.3 Back-End Layer

3.3.1 Model conversion

The backend layer supports conversion from one framework to another. Since we use a single intermediate representation, export logic is only required to be written from the intermediate representation to each of the supported framework, rather than between each pair of frameworks. For n different frameworks, this reduces the problem complexity from $O(n^2)$ to $O(n)$.

In some cases, model conversion is restricted by major differences between frameworks. For example, the *Local Response Normalization (LRN)* layer is only available in Caffe, and so it is not natively possible to port an AlexNet [Krizhevsky et al., 2012] network defined in Caffe (which uses this layer) to Keras or Tensorflow. In some cases we complete such conversions by using third-party implementations, e.g. for *LRN*, we use such implementations to allow export to these frameworks. However, there still remain cases where even this is not possible. For example, Caffe supports a *Python* layer that allows users to define custom logic. This can clearly not directly be ported to other frameworks, and in this case we report an appropriate error message. Note that since TensorFlow is a supported backend of Keras, we define porting logic for just two frameworks, Caffe and Keras. For TensorFlow conversion, we first export the model as Keras and then internally use the Keras backend to port to TensorFlow.

3.3.2 Asynchronous tasks for Model Export

For a model to be exported to a target framework, a back-end task needs to be executed which handles the parsing of the intermediate model representation. Waiting for the task to complete might prevent a user from being able to perform any operations on the canvas. To overcome this, asynchronous tasks for model export are employed.

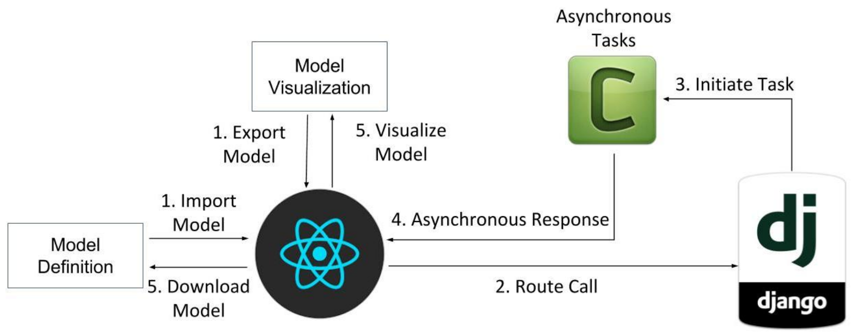


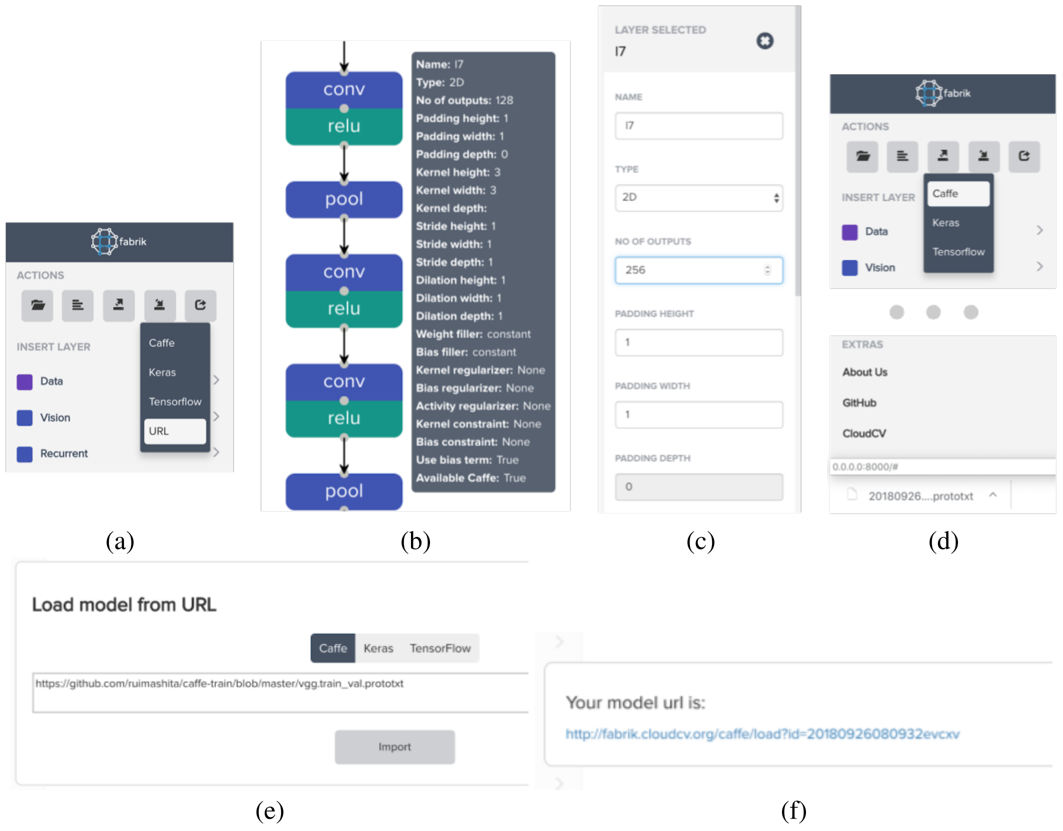
Figure 3: Export / Import Workflow

The asynchronous tasks are implemented using Celery [Celery, 2018], which allows for distributing work across threads and machines. Figure 3 shows how celery workers are used in the Fabrik infrastructure. As the call to the export event is implemented in an asynchronous manner, there is no latency at the front-end and the user can continue their operations. Once the task completes at the back-end it sends a response containing the result of the task with the help of web-sockets and the user is notified of the response on the front-end.

3.3.3 Real-time collaboration

Fabrik provides support for collaboratively building deep learning models with multiple users working in real time on a single model. For sharing a model with multiple users, a unique link is generated and any user with access to it can view, edit, or comment on a model. A global copy of a single shared model is maintained in the database and all updates over multiple sessions are performed on this copy, using communication via web-sockets. Whenever an update operation is performed, an event message is broadcast by the back-end layer which is used by the presentation layer to re-render the canvas on all active sessions. Additionally, the layer being updated is highlighted with the name of the user performing the update.

In order to achieve real-time collaboration, we employ an efficient implementation of operational transformation for the intermediate representation of a model. By storing it in JSON format, the size of each update is minimized. Based on the type of operation being performed, there are 4 types of operational update events – parameter update, layer addition, layer deletion, and layer highlight, each of which trigger an update event and the change is propagated through all sessions.



Name: l7
Type: 2D
No of outputs: 128
Padding height: 1
Padding width: 1
Padding depth: 0
Kernel height: 3
Kernel width: 3
Kernel depth:
Stride height: 1
Stride width: 1
Stride depth: 1
Dilation height: 1
Dilation width: 1
Dilation depth: 1
Weight filler: constant
Bias filler: constant
Kernel regularizer: None
Bias regularizer: None
Activity regularizer: None
Kernel constraint: None
Bias constraint: None
Use bias term: True
Available Caffe: True

LAYER SELECTED
l7
NAME
l7
TYPE
2D
NO OF OUTPUTS
256
PADDING HEIGHT
1
PADDING WIDTH
1
PADDING DEPTH
0

Load model from URL
https://github.com/ruimashita/caffe-train/blob/master/vgg.train_val.prototxt

Your model url is:
http://fabrik.cloudcv.org/caffe/load?id=20180926080932evcxv

Figure 4: Screen-shots of the application depicting the stages in the life-cycle of loading(a,e), visualizing(b), editing(c), saving(d), and sharing(f) a neural network model with Fabrik.

4 Walk-through and Use Cases

We now provide a complete walk-through of the visualizing, editing, and sharing functionalities via a concrete example. Consider that we want to visualize and modify the well known VGG-16 network proposed in [Simonyan and Zisserman, 2014]. VGG is a 16-layer convolutional neural network proposed for image classification on the ImageNet dataset that is primarily composed of convolutional, pooling, and fully connected layers.

Figure 4 walks through this use case: The network is first loaded into the application through the Import option which supports a) direct upload of model configuration file (as *prototxt* for Caffe, *JSON* for Keras, or *pbtxt* for Tensorflow) b) directly pasting the configuration as text into an input form-field c) Sharing a URL on GitHub or as a Gist. Note that for TensorFlow and Keras that do not decouple

Models	**Caffe**	**Keras**	**TensorFlow**
Recognition			
Inception V3 [Szegedy et al., 2016]	✓	✓	✓
Inception V4 [Szegedy et al., 2017]	✓	✓	✓
ResNet 101 [He et al., 2016]	✓	✓	✓
VGG 16 [Simonyan and Zisserman, 2014]	✓	✓	✓
GoogLeNet [Szegedy et al., 2015]	✓	✗	✗
SqueezeNet [Iandola et al., 2016]	✓	✗	✗
AllCNN [Springenberg et al., 2014]	✓	✗	✗
DenseNet [Huang et al., 2017]	✓	✗	✗
AlexNet [Krizhevsky et al., 2012]	✓	✓	✓
Detection			
YoloNet [Redmon et al., 2016]	✓	✓	✓
FCN32 Pascal [Long et al., 2015]	✓	✗	✗
Seq2Seq			
Pix2Pix [Isola et al., 2017]	✓	✗	✗
Visual Question Answering			
VQA [Antol et al., 2015]	✓	✓	✓

Table 1: List of tested models on Fabrik

model definition like Caffe does, serializing the model configuration is easily achieved via native API calls. As seen in Figure 4(a), the URL option is selected for import. 4(e) shows how we directly load the VGG-16 model from a prototxt file hosted on GitHub. If the import is successful, the network will be visualized as shown in 4(b). Once loaded, the user can scroll to explore different parts and zoom in/out to see a more local/global view. The user can also hover over a layer to get an overview of its parameters (4(b)). If the user wishes to edit a layer, they can click on it and modify values in the parameter pane launched on the right, as shown in 4(c). In this case, the number of outputs for the selected convolutional layer are being modified. Note that since the layer is a 2D convolution, edits to the depth dimension are disabled.

The user can also add a new layer to an existing network by dragging it, or by clicking on it in the dashboard. The latter attaches the layer to the deepest node in the network. Further, Fabrik displays and updates a real time parameter count of the network on the bottom-left.

Finally, the user can choose to export their network to a supported framework, or to share a link to it with collaborators. Figure 4(d) shows the network being exported back to the Caffe framework from which it was originally loaded (the downloaded prototxt file is visible on the bottom-left). Lastly, figure 4 (f) shows a unique link generated for this model which can be used to access it in the future. Table 1 shows the results of model conversion between currently supported frameworks. As seen, certain models cannot be converted to certain frameworks. For example, exporting GoogLeNet to TensorFlow and Keras is not possible due to lack of support for the LRN layer.

5 Conclusion and Future Work

In this work we propose Fabrik, a simple GUI for visualizing, creating, editing, and sharing neural network models from within a browser. Fabrik provides novel real-time collaboration features to assist researchers to brainstorming and design models remotely and at scale. Further, it provides a framework agnostic visualization scheme based on a novel intermediate representation that can be used to quickly and efficiently visualize model architectures.

As of today Fabrik supports three popular deep learning frameworks. As future work, we plan on adding support for additional popular frameworks such as PyTorch and Caffe2. Further, we plan to incorporate ONNX as our intermediate representation as a step towards long term interoperability and maintainability. Finally, Fabrik is maintained and by an active and vibrant community of open source developers that are committed to building more powerful features for debugging, reproducibility, learning, and collaboration.

References

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

Aetros. Aetros: Deep learning experiment management platform. `https://aetros.com/`, 2018.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

Caffe2. Caffe2: A new lightweight, modular, and scalable deep learning framework. `https://caffe2.ai/`, 2018.

Celery. Celery: Distributed task queue. `https://github.com/celery/django-celery`, 2018.

ConvnetJS. ConvnetJS: Deep learning in javascript. `https://github.com/karpathy/convnetjs`, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017.

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

jsPlumb. jsPlumb: Visual connectivity for webapps. `https://github.com/jsplumb/jsplumb`.

KDNuggets2017. Ranking popular deep learning frameworks for data science. `https://www.kdnuggets.com/2017/10/ranking-popular-deep-learning-libraries-data-science.html`.

KerasJS. KerasJS: Run keras models in the browser. `https://github.com/transcranial/keras-js`, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

MMdnn. MMdnn: A comprehensive, cross-framework solution to convert, visualize and diagnosis deep neural network models. `https://github.com/Microsoft/MMdnn`, 2018.

Netscope. Netscope: A web-based tool for visualizing neural network topologies. `https://github.com/ethereon/netscope`, 2017.

ONNX. ONNX: Open neural network exchange format. `https://onnx.ai/`, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. Pytorch, 2017.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

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

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

Tensorboard. Tensorboard: Suite of web applications for inspecting and understanding your tensorflow runs and graphs. `https://github.com/tensorflow/tensorboard`, 2018.

Jan Zacharias, Michael Barz, and Daniel Sonntag. A survey on deep learning toolkits and libraries for intelligent user interfaces. *arXiv preprint arXiv:1803.04818*, 2018.