NNView: Automating the Creation of Neural Network Architecture Diagrams from Code

Eunice Jun and Matthew Conlen
Paul G. Allen School of Computer Science & Engineering
Seattle, WA
{emjun,mconlen}@cs.washington.edu

ABSTRACT

Machine learning researchers and practitioners are confronted with several challenges when trying to create or consume neural network architecture diagrams. These diagrams are presented in a diversity of forms and often are lacking information, making them difficult to use practically. These issues are exacerbated by a lack of available tooling. NNView is an open source Python library that automatically generates neural network architecture diagrams from model code. NNView provides a consistent visualization format for Pytorch and Keras models based off of a common format used in the machine learning literature.

KEYWORDS

visualizations, neural network architecture

ACM Reference Format:
Eunice Jun and Matthew Conlen. 2018. NNView: Automating the Creation of Neural Network Architecture Diagrams from Code. In *Proceedings of Au. 2018 (CS 599).* ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.475/123_4

1 INTRODUCTION AND MOTIVATION

Visualizations of neural network architectures, such as those that show up in academic papers [2, 4], are vital to the communication of deep learning advances. However, the diagrams are difficult to compare across publications because they are not standardized, partly due to the lack of tools. Current tools for creating these diagrams are difficult and time-consuming to use. NN SVG [3], for example, provides a GUI to create visualizations in different styles found in highly cited papers. A key flaw of current visualization tools is that they are separate from the development process. Programmers need to re-type significant amounts of information about each of the layers every time they create or want to edit their diagrams. As a result, the benefit of these diagrams for building and debugging models is also under-realized.

These diagrams are also lacking key information that would allow a programmer to write code implementing a model based on an architecture diagram. A programmer may run into issues during this implementation process, and because there are no standard tools for digram creation, it is often unclear the programmer has made an error or if some crucial information was missing from the (manually created) diagram, leading to a painful debugging process.

NNView, a Python library, aims to ease the creation of neural network architecture diagrams by visualizing the architectures based on the code programmers already write to define deep learning models. NNView also works toward standardizing the diagrams to facilitate quick comparison between user-defined models and state-of-the-art models already implemented in frameworks such as Pytorch and Keras. With the creation of a standard tool for generating these diagrams, any innovations in the visual forms that these diagrams take can be incorporated into the tool and immediately benefit the broader machine learning community.

NNView makes the following contributions:

- NNView provides a workflow for extracting key layer information from Pytorch and Keras models and then rendering the layers in a consistent, standard form.
- NNView supports in-situ architecture visualization during development in notebook environments.
- NNView provides a platform for the dissemination of future improvements and innovation of diagram design.
- NNView is open source and freely available for use. The library can be downloaded on github.

2 NNVIEW: IMPLEMENTATION

NNView[1] is an open source library intended for use in iPython notebook environments. NNView currently works with Pytorch and Keras models, two of the most popular frameworks for machine learning today. The target user group is deep learning novices although experts would likely benefit from the library as well.

There are two key components to NNView: (1) Layer information extraction and (2) Rendering of the visualization.

2.1 Layer Data Extraction

NNView provides a common abstraction layer for capturing relevant layer data for Pytorch and Keras models.

For Pytorch models, users must provide a model and any input. Because the order of layer declaration can differ from the order of invocation in Pytorch models, NNView does a single forward pass through the network. At each layer, NNView stores the layer type, input/output dimensions, number of channels, kernel size, stride, and activation function in a standardized format. After the final layer, NNView passes the layer log to the renderer.

For Keras models, users must provide NNView with only the model declaration. Because Keras invokes the layers in the order they are declared, NNView does not need to do a single forward

[1] https://github.com/mathisonian/nn-view

CS 599, Final Project Report, NNView

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

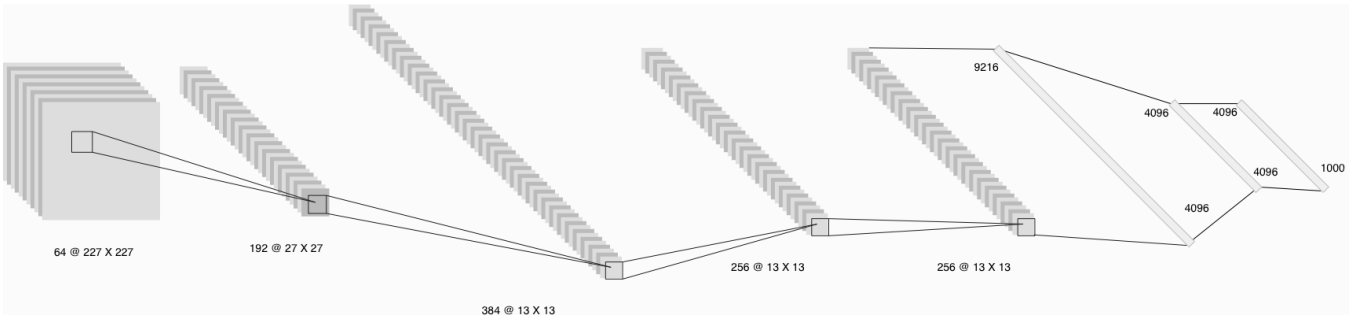

Figure 1: This diagram of a simple convolutional network was created by NNView. Convolutional layers are shown as stacked rectangles, and fully connected layers are shown as solid lines. Metadata about each layer is presented as text where appropriate, including information such as input shape, output shape, and number of channels.

pass to extract the layer information as it did for Pytorch. Instead, NNView directly introspects on the Keras model configuration, finds and stores the relevant data (same as above), and then passes the data to the renderer.

2.2 Rendering

NNView must subsequently parse the extracted layer information and render a graphic on the user's screen. Because we are targeting computational notebook environments, this restricts us to using web rendering technologies. We chose to use Scalable Vector Graphics (SVG) because of its cross-platform support and because it allows users the ability to subsequently import the generated graphic into a visual design program such as Adobe Illustrator and make any desired touch ups before publication.

The rendering algorithm performs two iterations over all the layers: the first pass is responsible for inspecting the dimensions of each layer, and calculating how that will correspond to the size of that layer in the resulting output; the second pass then generates the actual SVG markup, using the information computed in the first pass to scale the output so that it will fit in a user's browser window.

We use a Python implementation [1] of a Virtual Document Object Model (VDOM) in order to produce the rendered markup. This allows us to generate the markup in Python alongside the rest of our library's code, rather than writing it using JavaScript, which is how SVGs are typically generated on the web. This makes our code much easier to reason about and will enable us to directly save the generated images to a user's filesystem if they desire to use NNView outside of a notebook environment.

2.3 Diagram Design

After surveying the many different forms [2, 4, 5] used to depict neural network model architectures, we chose to render our graphics in a style reminiscent of that used in the LeNet [2] paper. The LeNet style strikes the optimal balance of factors: clarity, information density, and the ability for a reader to implement models based on the diagram. While this is the only type of diagram design that our library currently supports, in the future we could add support for additional customization or incorporate multiple output templates that a user could choose.

3 LIMITATIONS AND FUTURE WORK

We look forward to expanding NNView's coverage of frameworks, continuing development on NNView, and evaluating its utility.

NNView currently only supports Pytorch and Keras models. Additional frameworks such as MXNet are gaining popularity, so we plan to support MXNet models in our library as well. The goal for NNView is to provide a consistent and abstracted workflow that supports a wide range of frameworks and renderings.

From the poster session, we received feedback on additional details that potential users would want to visualize, specifically recurrences, forward or skip connections, residual layers, and max-pooling. Although the current literature does not visually encode different kinds of layers, connections, or optimizations in a consistent manner, we plan to create initial versions of these additional elements with the hope that the open source community will also contribute. Our aim is to use NNViews as a platform for exploring and supporting standardization of neural network architecture diagrams.

Although we are excited about the visualizations from our library for helping novices learn and debug their models, we do not yet understand how these visualizations could affect the mental models and intuitions people develop about deep learning. We hope to conduct preliminary user evaluations to see how experts and novices may use the diagrams differently.

REFERENCES

[1] Kyle Kelly. 2018. VDOM. https://github.com/nteract/vdom.
[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[3] Alex Lenail. 2018. NN SVG. https://github.com/zfrenchee/NN-SVG.
[4] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[5] 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.