Effective Visualizations for Training and Evaluating Deep Models

Luke Yeager LYEAGER@NVIDIA.COM
Greg Heinrich GHEINRICH@NVIDIA.COM
Joe Mancewicz JMANCEWICZ@NVIDIA.COM
Michael Houston MHOUSTON@NVIDIA.COM
NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, CA 95050, USA

Abstract

Training and deploying a deep neural network model can be a long and complicated ordeal. It involves gathering and processing a large amount of data, iterating over various network architectures and optimization hyperparameters, and evaluating trained models with various complex techniques. Throughout the process, visualizations can help users understand information and debug problems. In this paper, we survey some visualization techniques and discuss which are effective in decreasing the time required to reach a working solution.

1. Introduction

Over the past few years, deep learning has proved itself to be a powerful, viable tool for solving real problems in a wide range of applications. As deep neural networks (DNNs) win competitions (Krizhevsky et al., 2012), revolutionize common interfaces (Hannun et al., 2014) and even produce art (Gatys et al., 2015), the number of companies and individuals investing time and money in deep learning steadily grows. And yet, as our dependence on this technology grows, in some ways we still know very little about why it works or even what it is doing under the hood. This leads to some of the best minds in the field saying things like "I can't explain this discrepancy between theory and practice" (Krizhevsky, 2014). Thus, many researchers have invested time in developing new techniques for analyzing deep models and exploring what they have "learned."

In this paper we will survey some of the different areas of opportunity for visualization and list some common techniques. In addition to analyzing trained models, we will explore different ways to analyze datasets and training runs which can provide just as much if not more value.

Where possible, we will present examples of these techniques in action inside of DIGITS. NVIDIA provides DIGITS under the BSD license, available at http://github.com/NVIDIA/DIGITS.

2. Opportunities for Visualization

The deep learning pipeline provides opportunities for visualizations at every step. Representing data visually enables users to catch errors more quickly and interpret results more accurately. As we enumerate some of the opportunities and analyze different techniques, we will be showing examples of data and models related to images. Though DNNs can be used for many other types of data, images are the simplest to visualize and among the most used in practice.

2.1. Data Preparation

Real-world data comes in various sizes and formats, while most networks have rigid requirements for input tensors (Krizhevsky et al., 2012) (Hannun et al., 2014). So, the first task of many would-be deep learning users is to convert their data into the required format. For example, when choosing how to resize images the user may choose to crop them in order to preserve aspect ratio, or they may instead want to squash them into the required size so as not to throw away any regions of interest at the edges. When choosing between options such as these, simply generating and displaying an example image explains the trade-offs and enables the user to make the correct choice for his or her application. See Figure 1 for an example of this in DIGITS.

2.2. Dataset Analysis

Pre-processed data is often collected into a single database or folder before training begins. Providing some simple visualizations of the data and an interface for accessing the contents can help avoid configuration errors which, if

Proceedings of the 33^{rd} *International Conference on Machine Learning*, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

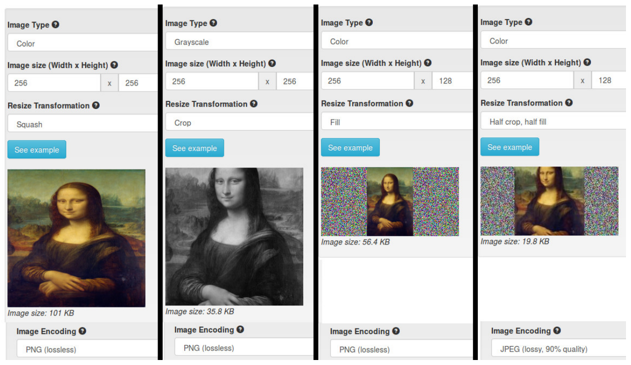


Image Type: Color | Image size (Width x Height): 256 x 256 | Resize Transformation: Squash | See example | Image size: 101 KB | Image Encoding: PNG (lossless)

Image Type: Grayscale | Image size (Width x Height): 256 x 256 | Resize Transformation: Crop | See example | Image size: 35.8 KB | Image Encoding: PNG (lossless)

Image Type: Color | Image size (Width x Height): 256 x 128 | Resize Transformation: Fill | See example | Image size: 56.4 KB | Image Encoding: PNG (lossless)

Image Type: Color | Image size (Width x Height): 256 x 128 | Resize Transformation: Half crop, half fill | See example | Image size: 19.8 KB | Image Encoding: JPEG (lossy, 90% quality)

Figure 1. Example images and file sizes for different data transformation options

not caught at this stage, can lead to cryptic errors and odd behavior later during training or inference. Figures 2 and 3 show some examples of useful dataset visualizations in DIGITS.

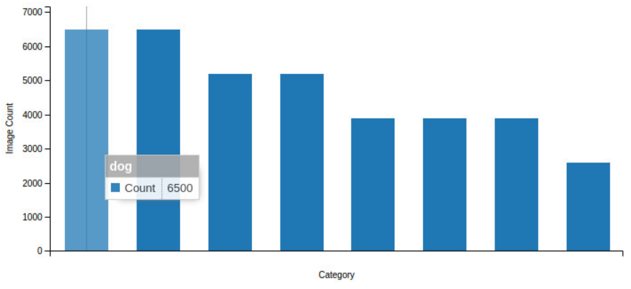
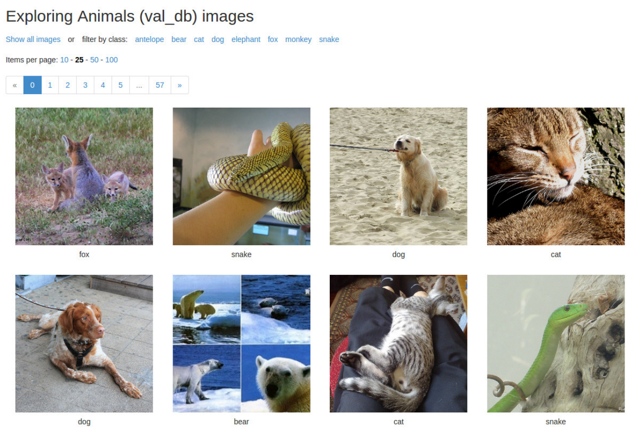


Figure 2. Graph of the distribution of classes in a dataset

2.3. Designing a Network Architecture

A neural network is essentially a dataflow graph of nodes (layers) passing data (tensors) to other nodes in the graph. Designing these networks is an obvious candidate for visualizations because these networks are usually specified with code or some structured text, but can be displayed graphically as a dataflow graph. Figure 4 shows an example of DIGITS displaying a Caffe network as a graph.

Some tools exist for designing architectures with a graphical interface, but this is not necessarily an ideal solution for all use cases. Networks with more than 100 layers (He et al., 2015) may be easier to design programmatically than graphically. The best solution would likely be a side-by-side combination of these two methods. As with some IDEs, the user could edit the network by either directly editing the code or by clicking and dragging the UI elements around and the tool would automatically update the graph based on the code, or vice versa.



Exploring Animals (val_db) images

Show all images or filter by class: antelope bear cat dog elephant fox monkey snake

Items per page: 10 - 25 - 50 - 100

« 0 1 2 3 4 5 ... 57 »

Figure 3. Browsing through entries in a dataset

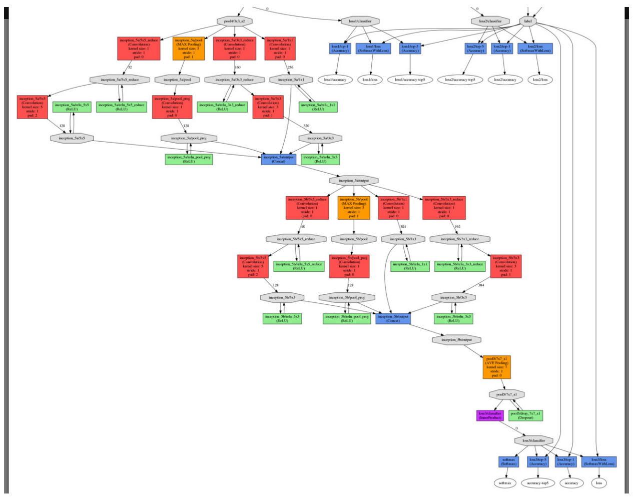


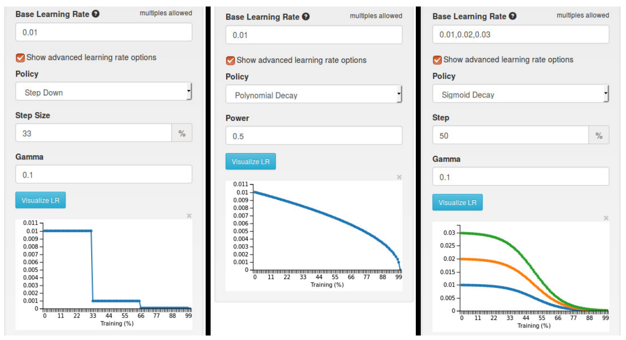
Figure 4. Visualization of a Caffe network architecture

2.4. Learning Rate

In addition to specifying a network architecture, some optimization hyperparameters must be chosen when training a DNN. Options include things like different optimization algorithms, the number of epochs through the training data, and the initial learning rate. Typically for gradient descent, the learning rate is chosen to decrease over time according to some formula. A simple but useful visualization tool is to graph the projected learning rate based on the selected policy. See Figure 5 for some examples.

2.5. Training Outputs

While training a model, certain metrics like loss and accuracy are used to measure the quality of the model over time. Plotting these values is perhaps the quintessential vi-



Base Learning Rate — multiples allowed: 0.01
✓ Show advanced learning rate options
Policy: Step Down
Step Size: 33 %
Gamma: 0.1
Visualize LR

Base Learning Rate — multiples allowed: 0.01
✓ Show advanced learning rate options
Policy: Polynomial Decay
Power: 0.5
Visualize LR

Base Learning Rate — multiples allowed: 0.01,0.02,0.03
✓ Show advanced learning rate options
Policy: Sigmoid Decay
Step: 50 %
Gamma: 0.1
Visualize LR

Figure 5. Projected curves for the selected learning rate options

sualization for machine learning. Everyone does it because it is trivial to display and extremely informative. Figure 6 shows loss curves which let the trained eye distinguish a good training session from a bad one with a mere glance.

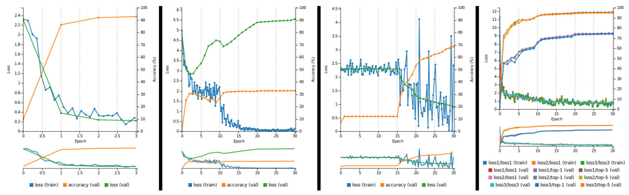


Figure 6. Graphs of training outputs - a well behaved network, an overfit network, an oddly behaved network, and a network with many outputs

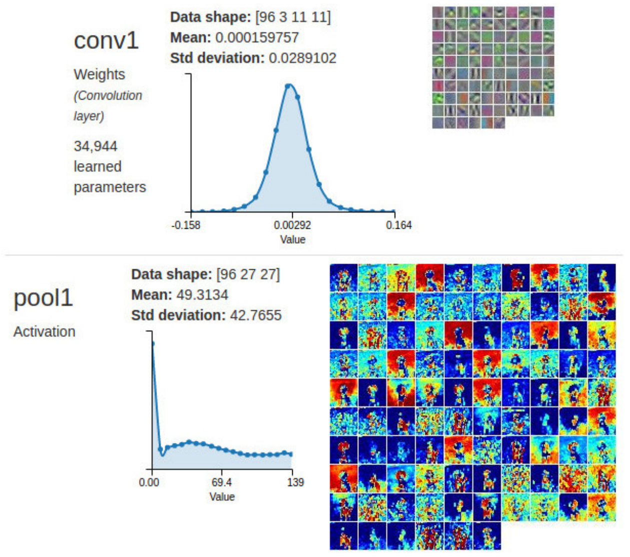
Another metric that can be useful to display while training is the change in weights and gradients for each layer over time. For large networks, this can be a large amount of data to display, and it my not always be clear what "good" behavior looks like. But it can help identify the "vanishing gradient" and "exploding gradient" problems that tend to plague back-propagation networks.

2.6. Model Analysis

Training a network produces a series of trained models. The most straightforward way to test a model is to perform inference on it with some test data. We will address some of those techniques in the next section, but first, we will look at a few ways to analyze the model without using any test data.

For convolutional networks trained on image data, it can be instructive just to view the weights displayed as images (see Figure 7 for an example). This technique is not particularly useful for smaller convolution layers or inner product layers, however.

Another popular technique for analyzing a model without



conv1
Weights *(Convolution layer)*
34,944 learned parameters
Data shape: [96 3 11 11]
Mean: 0.000159757
Std deviation: 0.0289102

pool1
Activation
Data shape: [96 27 27]
Mean: 49.3134
Std deviation: 42.7655

Figure 7. Inference weights and activations

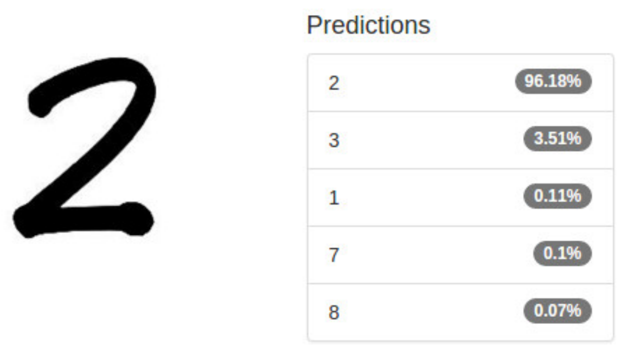
using test data is to generate synthetic inputs. For each neuron in the network, there are various techniques (Zeiler & Fergus, 2013), (Erhan et al., 2009), (Yosinski et al., 2015), (Le et al., 2011) to generate the input which maximally stimulates the neuron. Each of the proposed techniques has its own downfalls in terms of image quality, complexity, or speed of computation. Also, without a well-designed interface, the results can be difficult to navigate. But digging through the network can lead to some interesting discoveries about what the network has "learned."

2.7. Single Inference

Performing inference on a test input is a great way to test a deep model and there are some intriguing visualizations that come as a result. The simplest thing to do is to simply simulate the model as it would be used in production - run inference on a single image and check the answer. Figure 8 shows an example of single inference on an image classification model in DIGITS, and Figure 9 shows one for an object detection model.

While performing inference, the network is passing tensors of data in-between the various layers. Those intermediate tensors, called activations, can be also useful to visualize. As with the layer weights (Section 2.6), this technique is useful for some layers in some networks, but not in general. See Figure 7 for an example of activation visualization.

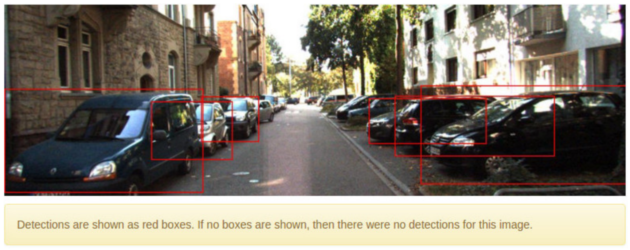
With techniques very similar to those used to generate maximally stimulating images (Section 2.6), it is also possible to visualize which part of the input image contributed to the activation of each neuron (Samek et al., 2015). This



Predictions

2	96.18%
3	3.51%
1	0.11%
7	0.1%
8	0.07%

Figure 8. Inference on a classification network



Detections are shown as red boxes. If no boxes are shown, then there were no detections for this image.

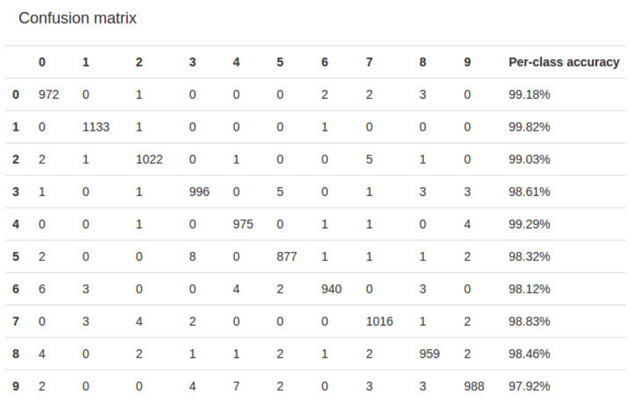
Figure 9. Inference on an object detection network

lets you say things like "This neuron is activated by the ears on a cat." It is possible to use techniques like this to explore whether higher layer neurons may be activated by higher level features like complex shapes while lower layer neurons may be activated by lower level features like textures. As discussed previously, some of these techniques are difficult to implement or are computationally expensive. But in general, the results from this type of visualization are more interpretable than the images generated for maximally stimulating inputs since they highlight portions of real-world data.

2.8. Multiple Inference

There are other opportunities for visualization related to performing inference on multiple inputs and aggregating the results. A canonical example is the confusion matrix (see Figure 10). For classification networks, it can tell you which classes are more difficult for the network to accurately classify. This is a very useful result because it can translate into concrete actions like gathering more data for difficult categories, for example.

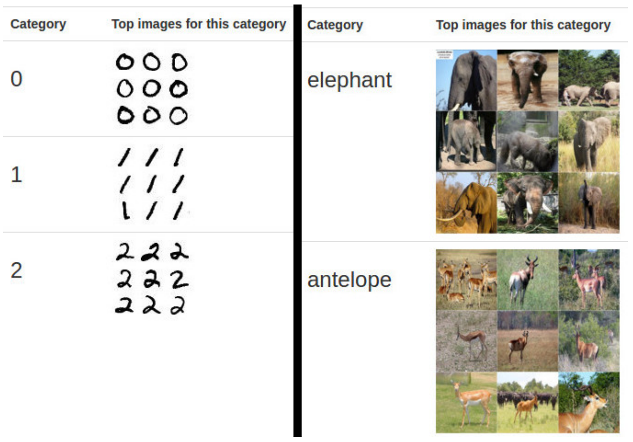
Another technique is to display a list of inputs from the test set which maximally stimulated a specific neuron during inference. This is in some ways similar to the technique of generating maximally stimulating images discussed in Section 2.6, but it does not require any special algorithms and produces a list of real images. This simple technique has been used in many papers to great effect (Le et al., 2011) (Yosinski et al., 2015) (Zeiler & Fergus, 2013), and is a useful tool in the training pipeline. Figure 11 shows examples in DIGITS of maximally stimulating images for output neurons in classification models.



Confusion matrix

	0	1	2	3	4	5	6	7	8	9	Per-class accuracy
0	972	0	1	0	0	0	2	2	3	0	99.18%
1	0	1133	1	0	0	0	1	0	0	0	99.82%
2	2	1	1022	0	1	0	0	5	1	0	99.03%
3	1	0	1	996	0	5	0	1	3	3	98.61%
4	0	0	1	0	975	0	1	1	0	4	99.29%
5	2	0	0	8	0	877	1	1	1	2	98.32%
6	6	3	0	0	4	2	940	0	3	0	98.12%
7	0	3	4	2	0	0	0	1016	1	2	98.83%
8	4	0	2	1	1	2	1	2	959	2	98.46%
9	2	0	0	4	7	2	0	3	3	988	97.92%

Figure 10. Confusion matrix



Category	Top images for this category	Category	Top images for this category
0		elephant	
1			
2		antelope	

Figure 11. Top activations from a test set for output neurons of classification models

3. Conclusion

Despite the wide range of tools and frameworks available for deep learning, successfully training and deploying a DNN model is still pretty complicated and error-prone. Throughout the process, well-placed visualizations can help to avoid errors and interpret results, leading to a quicker solution.

We have surveyed several of these opportunities for visualization and presented some popular solutions, but the list is far from comprehensive. We have also presented a product which attempts to put these lessons into practice. DIGITS builds on the experience and innovations of many researchers to provide a solution which uses visualizations to aid users on their quest to harness deep learning in their applications.

References

Erhan, D., Bengio, Y., Courville, A., and Vincent, P. Visualizing higher-layer features of a deep network. Technical Report 1341, University of Montreal, June 2009. Also presented at the ICML 2009 Workshop on Learning Feature Hierarchies, Montréal, Canada.

Gatys, L. A., Ecker, A. S., and Bethge, M. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015. URL http://arxiv.org/abs/1508.06576.

Hannun, A. Y., Case, C., Casper, J., Catanzaro, B. C., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014. URL http://arxiv.org/abs/1412.5567.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014. URL http://arxiv.org/abs/1404.5997.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C.J.C., Bottou, L., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Le, Q. V., Monga, R., Devin, M., Corrado, G., Chen, K., Ranzato, M., Dean, J., and Ng, A. Y. Building high-level features using large scale unsupervised learning. *CoRR*, abs/1112.6209, 2011. URL http://arxiv.org/abs/1112.6209.

Samek, W., Binder, A., Montavon, G., Bach, S., and Müller, K. Evaluating the visualization of what a deep neural network has learned. *CoRR*, abs/1509.06321, 2015. URL http://arxiv.org/abs/1509.06321.

Yosinski, J., Clune, J., Nguyen, A. Mai, Fuchs, T., and Lipson, H. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015. URL http://arxiv.org/abs/1506.06579.

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. URL http://arxiv.org/abs/1311.2901.