Benchmarking Open Source Machine Learning Frameworks and Development of Auto-coding Interface for Entry-Level Engineers (Extended Abstract)

Louis Yu
AE 8900 MAV – Special Problems, Spring 2017

ABSTRACT. The implementation of machine learning into multiple domains has renewed the interest of artificial intelligence enthusiasts to develop multiple frameworks to best resolve a problem. One of the most prevalent subsets, Neural Networks, has had a growing community of research enthusiasts. This paper covers a brief comparative study on four frameworks—Tensorflow, Theano, MXNet, and Caffe on several aspects: speed, utilization, and scalability onto commercial platforms. Tensorflow achieved the overall best score and was utilized for the language of choice for an interface between entry level engineers and an open source framework. Many aerospace engineers do not formally use lower level programming languages (Java, C, C++, etc.) and typically Matlab (versus Python) is the academic-program of choice for high-level interpreted language. However as machine learning permeates into the aerospace domain, immediate work may be stifled by some delay due to picking up a new syntax. The aim of this paper and likewise product, is to devise an interface that allows for problems to be devised in Matlab and then properly ported over to Tensorflow to utilize the extensive benefits this open-source tool provides for either GPU/CPU based hardware configuration.

Nomenclature

ANN	=	artificial neural networks
NN	=	neural networks
ML	=	machine learning
TF	=	tensorflow
GiB	=	Gigabyte
VM	=	Virtual Machine
FLOPS	=	Floating Point Operations
GPU	=	Graphical Processing Units
OS	=	Operating System

I. Introduction

ARTIFICIAL intelligence has been migrating towards new forms of knowledge representation and processing capabilities that emulate the human brain. There now exists new computational paradigms that have been established for new applications, namely artificial neural networks. An artificial neural network is best defined as a biologically inspired computational model that maintains a network architecture composed of artificial neurons. Each of these neurons and the network architecture layout are customizable with different mathematical parameters to tune in order to have the network perform a certain task. Neural networks today follow a production-like based scenario typically designated in a two-step process: development and deployment. Figure 1 shows that the neural network is first trained, i.e. labeled examples of inputs and desired outputs are passed through a neural network model. Second, the compiled and trained neural network is deployed to run inferences on typically unknown variations of inputs. This

deployment allows a previously trained neural network to classify data at a certain confidence rate based off of its prior training and validation performance.

Artificial neural networks have begun to permeate throughout various industry and domains as a useful way of solving complex issues in a more humanlike fashion. In addition, the more complex neural networks have begun a new sensational wave known as deep learning where the neurons and networks are utilizing learning methods based off of learning how data is represented. Deep learning itself has begun to be the frontrunner consideration for solutions to self-driving cars, automatic image captioning and real-time language translation.

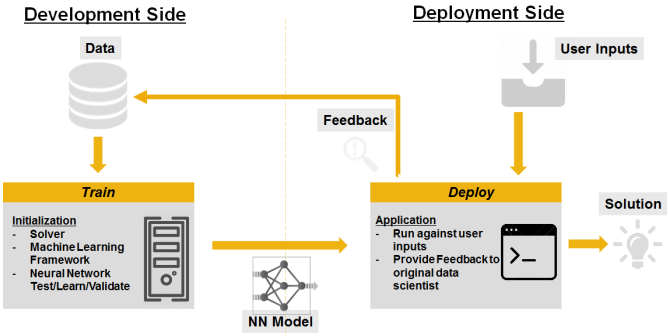


Figure 1. Production Pipeline of Neural Networks. A graphical adaptation of NVIDIA Production Pipeline showing how neural networks follow a two-stage feedback cycle.

Former state of the art algorithms and heuristic methods to tackle some of the largest problems have seen significant performance improvements based off of the commercial sectors momentum in publicly showcasing their deep learning projects. For instance, Google's AlphaGo deep learning network was the first computer program to beat a Human Go Professional and the results were showcased in late 2015 and early 2016[14]. In another instance, Baidu Deep Speech 2 became a well versed neural network that could perform English and mandarin speech recognition at a higher accuracy rate than humans (error rate of 3.7% vs 4% for humans)[15]. These two cases and many others like them provide continual public and research interest in realizing the potential that neural networks have for the future.

It would be unfair to not also attribute the major success of neural networks if it was not for innovative solutions on existing hardware, namely Graphical Processing Units. GPUs have brought the most influential shifts in this industry by providing a reduced computational and economical footprint for neural networks to train on. GPU computing contain characteristics that are perfectly aligned with neural networks: (1) they are inherently parallel, (2) filled with matrix operations and (3) contain FLOPS. The GPU footprint has already drawn down 80% of the typical run-time associated with neural networks that were usually represented as 5% of the code[4]. As many pieces have begun to come together for neural networks and subsequently deep learning methods to succeed, it's evident that the community it has fostered and the projects that have come to fruition will create a foundation where neural networks will become pervasive in throughout many industries.

II. Motivation

As attention continues to grow over neural networks and deep learning architectures, a wide variety of tools have broken ground and commenced gaining massive traction over their open source features and content. However each tool differs in their approach of programmatically building neural networks even though the underlying mathematics remains the same. The primary differences such as API language, syntax, and compilers utilized provide each machine learning framework the ability to reside isolated in a crowded space of open source frameworks. This however has garnered users to become specialist in a specific framework and require a deeper understanding of the intricacies of a toolbox versus being able to construct a neural network and tune the parameters to best suit a specific dataset.

Engineers throughout industry and academia that are not in tune with the computer science open-source community may see this as an inhibiting factor that is not easily mitigated. The number of computer languages used and content sharing procedures pervasive throughout an open-source community make it very intimidating to enter into this continually changing space. These inhibiting factors not only turn away engineers from the ability to characterize or classify datasets using open source neural network frameworks, but also prevent engineers from tapping into the most up-to-date technical capabilities available in this field. As these barriers may likely not be drawn down in the next few years due to a lack of standardization throughout open source communities, a need was found to allow engineers to utilize these tools without a heavy learning curve in the start-up phase and allow for a focus on implementing solutions for a wide variety of datasets. Thus in order to ensure the most optimal framework was used based off of several key metrics, this study aims to provide a clear and concise overview of four prevalent frameworks (Theano, Caffe, Tensorflow, and MXNet) and also develop a means of autocoding the syntax of one selected framework via a GUI interface in Matlab. In other words, the goal is to let the engineer focus on the mathematics and technicalities of neural networks versus engaging with intricacies of a specific framework.

III. Related Work

As over 29 machine learning frameworks have been pushed out to the open source community through GitHub, it is evident that many of these frameworks have been compared against one another in some form or fashion. Benchmarking over selections of frameworks have covered hardware/network configurations[1], evaluations over forward time and gradient computation time[2], and even operations based off of syntax for computational speed-up[3]. It has become apparent though that the various publications all focus on specific areas and each provide different recommendations for the overall top contender—as everything is relative to a specific hardware and software configuration. However these benchmarks do provide insight on the typical time factors associated with running a certain framework—as some frameworks are heavily dependent on CPUs to scale and be optimized[1].

Other frameworks tend to be heavily favoring GPU speed-ups with standard CPU performance relative to the other frameworks investigated[2]. Among all of the machine learning frameworks available, the most pervasive static configuration always references the GPU-based deep learning cudNN engine provided by NVIDIA. NVIDIA has maintained the forefront of standards for GPU performance improvements for training neural networks[4]. NVIDIA also offers their own graphical interface to build neural network models, known as DIGITS. DIGITS is a web-based application specifically for image classification, segmentation and object detection.

ASDL also holds a graphical interface known as BRAINN[17] (Basic Regression Analysis for Integrated Neural Networks) which is a Matlab-based interface utilizing the neural network toolbox. The tool allows for a single layer architecture mapped to a set of inputs, user settings on epochs to iterate, convergence criteria to stop training, and many more options. Some inhibiting factors to using BRAINN and Matlab's Neural Network toolbox application are the license controls on Matlab and GPU-based processing requires the Parallel Computing Toolbox to run. This study aims to draw on the capability to build neural networks via open source frameworks (that have CPU/GPU capabilities) and also utilize a familiar tool, Matlab, to act as the interface for a framework that is not native to Matlab. The benefits that entail this study is to identify the top open-source candidate and allow open sharing of neural networks through this framework while minimizing the ramp up time associated with an open source framework.

IV. Benchmarking Setup

In order to obtain a holistic grading criteria of the four machine learning frameworks chosen, several benchmarks were configured to evaluate the performance against four datasets spanning different applications. The four datasets evaluated were: (1) MNIST[5], (2) Airfoil Self-Noise Data Set[6,7], (3) Naval Propulsion Plants[6,8], and (4) Household Power Consumption[6,9]. Non-performance related benchmarking was undertaken qualitatively by inspecting code syntax, adoption into commercial services and popularity amongst open-source communities (GitHub) & research institutions.

A. System Configuration

All experiments are performed on a single machine running on Windows 10 Pro (64 Bit) with Intel® Core™ i7-4790K CPU @ 4.00GHz 3.60 GHz; Nvidia GeForce GTX 750 Ti (Ver. 378.66); 16 GiB DDR3 memory; and WD Blue HDD. Table 1 below shows the software framework and versions utilized for the evaluation.

Table 1 Table of Machine Learning Frameworks Investigated

Framework	Tensorflow[10]	Theano[11]	MXNet[12]	Caffe[13]
Version	R1.00	0.8.2	0.9.3	1.0.0.rc3
Core Language	C++, Python	Python	C++	C++
Interface Language	C++, Python	Python	C++, Python, Julia, Matlab, Javascript, Go, R, Scala	C, C++, Python, Matlab
cuDNN Support	V5.1	V5	v.5.1	V5.0
Python Version	V3.5	V2.7	V2.7	V2.7
Docker Image Pulled	Latest:py3	Latest	Latest	Latest

Docker v.1.13.1 for Windows was chosen as the deployment software configuration controller for each set of experiments to be run in isolated containers via HyperV. Figure 2 shows how Docker allows for each ML framework pre-compiled image to be invoked on separate Docker containers on the VM and separately monitored. Docker was allocated (of the original machine) 8 GiB of memory and all 8 CPU Cores. No GPU runs were conducted since Nvidia Docker currently does not have a windows interface. Each Machine Learning Framework in Table 1 were run separately from one another (e.g. using all CPU resources on its own and not sharing between other frameworks).

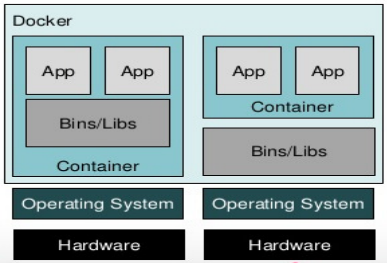


Figure 2. Docker Image over hardware. Docker containers are isolated, but share the same OS, and where appropriate, bins/libraries.

B. Datasets investigated

The datasets evaluated were intended to provide a swath of values that go from as low as thousands of data points up to over 2 million data points. Certain datasets were not provided with a specific training & testing dataset (like MNIST), so either a 70/10/20 or 60/10/30 training/validation/testing split was utilized to separate the data. Table 2 below covers a brief summary of all the different datasets covered and certain attributes they contained.*

Table 2 Table of Datasets Investigated and Benchmarked

	MNIST	Airfoil Self-Noise	Naval Propulsion Plant	Household Power Consumption
Attribute Characteristics	Integer	Real	Real	Real
# of Instances	60000 (Training) 10000 (Testing)	1503	11934	2075259
# of Attributes*	10 (28 x 28 image)	6	16	9
Testing/Training Data	Testing & Training Provided	Data Only	Data Only	Data Only
Data Provider	NYU/Google Labs	UCI ML Repository	UCI ML Repository	UCI ML Repository

C. Monitoring Solution

A blend of 5 different Docker plugins were utilized to capture each containers specific CPU usage and memory usage. Prometheus, AlertManager, Grafana, NodeExporter, and cAdvisor were compiled in separate Docker containers and communicated via localhost proxies. These overlaying containers were segregated from one another when collecting each of the performance metrics over each set of runs. Figure 3 shows how each container reports its own metrics. These metrics allow for timestamps to be utilized in calculating the duration of how long each network took to complete the training of a network.

Figure 3. Docker Monitoring Solution. Docker containers are isolated and each container reports back its CPU and memory usage.

V. Benchmark Results

The experiment results are presented in two subsections: quantitative results and qualitative results. For quantitative results, the focus was on running a specific dataset over a chosen neural network framework in each

* Attributes for MNIST refer to the classes of objects definable (e.g. digits from 0 to 9), whereas for each of the other datasets the attributes refer to a feature of the data collected for a certain configuration or timestamp

separate tool. The Machine Learning framework was monitored for hardware utilization (CPU usage and memory usage), as well as time elapsed from start to finish for the entire network to be trained. This format was conducted to take a systemic view of an entire training session for each dataset versus metrics concerning the highest contributing Pareto factors in neural network training (forward time, gradient computation time)[1,2]. For qualitative results, the focus on this interpretation was to identify key characteristics of each network as they related to syntax, utilization, and extensibility.

A. Quantitative Results

The first benchmark dataset was over the MNIST dataset that holds a database of handwritten digits containing a training set of 60,000 examples and a test set of 10,000 examples. A validation set of 5000 examples was taken out of the training sets' 60,000 examples. The MNIST digits are centered in a 28x28 image with the center at the mass of the pixels. This dataset was trained on a LeNet5 convolutional neural network developed by Yann Lecun[16]. This network shown in Figure 4 contains two convolutional layers (containing both an activation function and subsampling frame) and three fully connected hidden layers that draw down the outputs from 120 to a one-hot vector of 10 to classify the 10 attributes. The input parameters for the network was a batch size of 64, 2 filters with a filter width and height of 5x5. The learning rate was set to 0.05 initially and allowed to change with momentum factors in each tool.

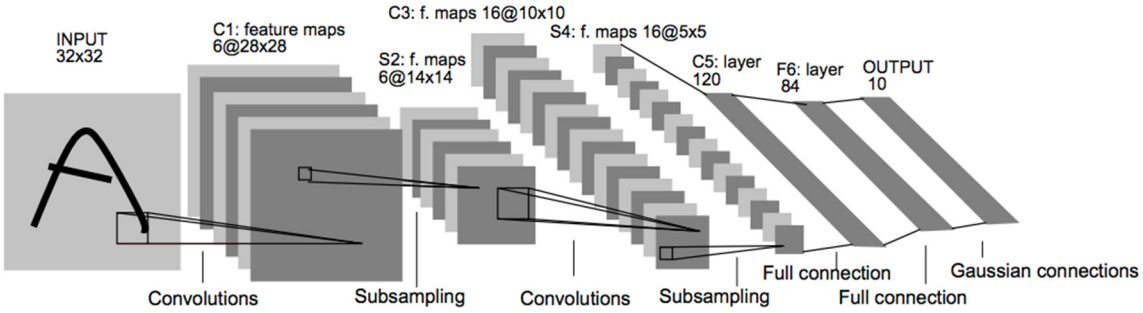
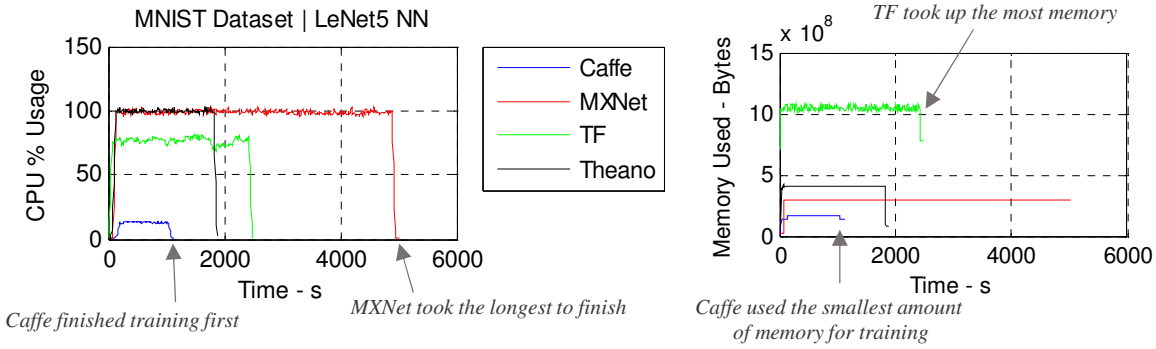


Figure 4. LeNet5 Neural Network Architecture. The architecture shows 2 convolutional layers with subsampling and 3 fully connected output layers to drive the output to 10.

Figure 5 shows the computational results gathered on a complete run of the MNIST database over each of the ML frameworks. It can be seen that of all the frameworks, MXNet and Theano both consumed the most CPU usage throughout the entire training and validation phase. Tensorflow allocated the most memory during the entire training phase. MXNet took the longest to complete and Caffe was the quickest. In addition, Caffe also only took a fraction of the time as all of the other frameworks took upwards of 30 minutes to complete.



a) b)

Figure 5. MNIST Computational Results. Results show a) CPU usage and b) Memory used for each of the four frameworks investigated on the MNIST LeNet5 Neural Network

(More Quantitative Results coming soon)

B. Qualitative Results

The performance of a machine learning framework may provide great insight, but one of the biggest challenges is having the library installed and all accompanying dependencies propagated throughout the system. Throughout the beginning of this study, the windows-based installations of Theano, MXNet, and Caffe were filled with error messages that were machine-independent (e.g. certain system configurations cause different error messages). Thus, community-based support of installation issues were limited and typically unrelated to problems a user may have when attempting to install each framework. These three packages were more suited for preconfigured software package files that Linux Machines utilize to install packages on a system (aka apt-get) versus the Windows format of manipulating batch scripts to suit the needs of the individual's computer configuration. Tensorflow out of all 4 had the smoothest installation package for Windows.

Beyond the installation phase, a deeper dive into the community was investigated and a general gauge for the community inertia was captured via GitHub for commits and forks of the official databases. In GitHub, commits are software updates & revisions versus forks being community-led projects that started from the original database and modified on their own. It was evident that Tensorflow was the most popular amongst the community for forks and came in second for number of commits—thus high reliability and community engagement was evident. In addition, cloud services were all available to be ported onto Amazon's AWS service with Tensorflow having the additional Google Cloud as an option. Higher level APIs designed to provide easier guidance in building neural networks in a specific framework were only evident in both Tensorflow and Theano. These two both had an internal high-level API (TFLearn and Lasagne respectively) while also sharing an independent open-source high-level API known as Keras.

For the implementation phase of editing Python code and working in the context of generic classification problems, it was found that MXNet had the most simplistic and compact form of neural network definitions and defining how to begin training. Caffe was found to have the hardest syntax to comprehend as it not only involved the Python API but also heavily depended on understanding a JSON format used for defining how a neural network was configured. Several configuration scripts in batch or bash format also had to be edited for the network to run appropriately in Caffe. A summary of qualitative inspection of each network is found in Table 3[†].

Table 3 Qualitative Assessment of Neural Network Frameworks

	Tensorflow	Theano	MXNet	Caffe
GitHub Commits|Forks	15049 | 23183	25221 | 2014	4931 | 3226	3914 | 10105
Cloud Service	Google Cloud, Amazon AWS	Amazon AWS	Amazon AWS	Amazon AWS
Higher-Level APIs	Keras, TFLearn	Keras, Lasagne	---	---
Syntax	Moderate	Moderate	Simple	Complex (JSON + Python)
Graphical APIs	Tensorboard	matplotlib	matplotlib	matplotlib

VI. GUI Interface Development

A. Functional Model

For the development of a development package to suit the needs of a data scientist, four main phases will occur in the interface: (1) environment set up and data loading, (2) neural network creation, (3) interface to python, and (4) neural network training as summarized in Figure 6. The goal of each of these steps is to maintain modularity in the interface to allow for future modifications in a given phase as need be. Data differentiation and neural network configuration is highly important to the user and are the most configurable features. As some research projects may need to classify text or labels (e.g. [Bat, Cat Dog, Bat] would be [1, 2, 3, 1] respectively), a basic text encoding feature will be made available. Large datasets will be managed appropriately by maintaining a datastore variable that doesn't immediately read the entire file into memory. All neurons/layers that will be used will be based on a pre-defined library with editable parameters to be adjusted based upon user preferences. This library can then be added onto by other users to share different configurations.

[†] All assessments are made as of 03/9/2017

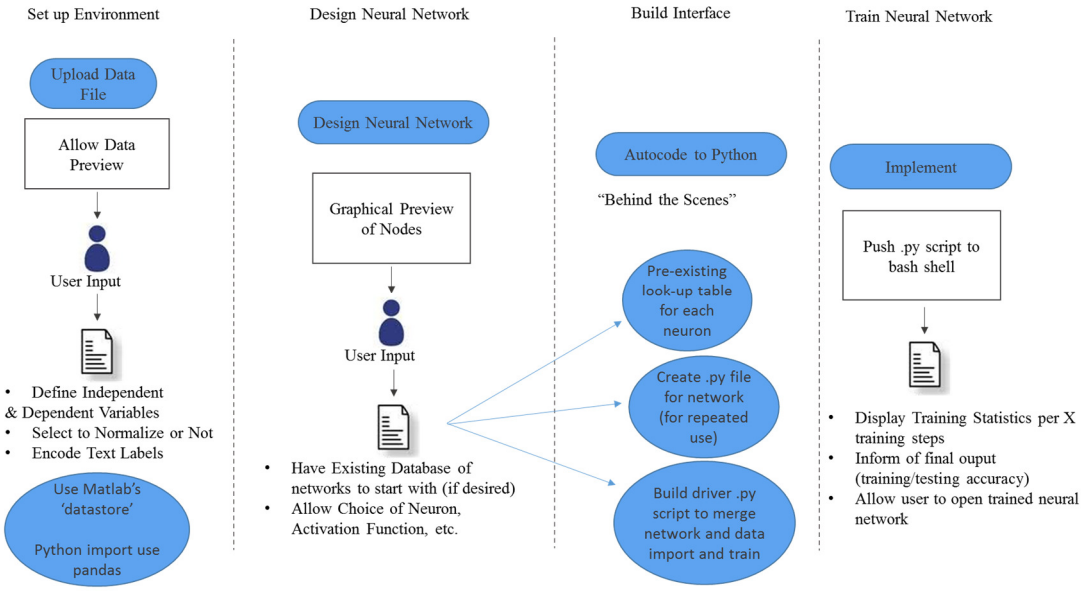


Figure 6. GUI Interface Functional Layout. Functional design of the GUI Interface package detailing four phases that will be modularized in Matlab code

Additional configurations to be considered for implementation are GPU/CPU switching, ability to push python scripts onto Docker images/containers, and a queue capability for several different networks to be selected to run against a specific dataset.

VII. Conclusion

(To Be Filled Later)

Acknowledgments

The author would like to thank Dimitri Mavris and Burak Bagdatli for their support of this special problems project.

References

[1]Shi, S., Xu, P., Wang, Q., and Chu, X., "Benchmarking State-of-the-Art Deep Learning Software Tools," Department of Computer Science, Hong Kong Baptist University, Jan. 2017.

[2]Bahrampour, S., Ramakrishnan, N., and Schott, M. S. Lukas, "Comparative Study of Deep Learning Frameworks," Research and Technology Center, Robert Bosch LLC, Mar. 2016.

[3]Vanhoucke, V., Senior, A., and Mao, M. Z., "Improving the speed of neural networks on CPUs," Google Inc., 2011.

[4]NVIDIA, "GPU-Based Deep Learning Inference: A Performance and Power Analysis," NVIDIA Whitepaper, 2015.

[5]Lecun, Y., Cortes, C. and Burges, C. J.C., "MNIST Database," *Special Database* [http://yann.lecun.com/exdb/mnist/], NYU/Google, New York, NY, 1998.

[6]Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science

[7]Brooks, T.F., Pope, D.S., and Marcolini, A.M. "Airfoil Self-Noise Data Set," [https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise], NASA, NASA RP-1218, July 1989.

[8]Coraddu, A., Oneto, L., Ghio, S., Savio, S., Anguita, D., and Figari, M. "Machine Learning Approaches for Improving Condition Based Maintenance of Naval Propulsion Plants," Journal of Engineering for the Maritime Enviornment [https://archive.ics.uci.edu/ml/datasets/Condition+Based+Maintenance+of+Naval+Propulsion+Plants], (In Press), 2014.

[9]Habrail, G., and Bacrard, A., "Individual Household Electric Power Consumption," EDF R&D [https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption], Clamart, France, 2012.

[10]TF, Tensorflow, Software Package, Ver. 1.0, Google Brain Team, Mountain View, CA, 2017.

[11]Theano, Software Package, Ver. 0.8.2, Universite de Montreal, Quebec, Canada, 2016.

[12]MXNet, Software Package, Ver. 0.9.3, University of Washington, Seattle, Washington, 2016.

[13]Caffe, Software Package, Ver. 1.0.0.rc3, Berkeley Vision and Learning Center, Berekley, CA, 2017

[14]Google, "AlphaGo," https://deepmind.com/research/alphago/, Mountain View, CA, 2016.

[15]Baidu, "Deep Speech 2," http://usa.baidu.com/tag/deep-speech/, Beijing, China, 2016.

[16]Lecun, Y., Bottou, L., Haffner, P., "Gradient-based learning applied to document recognition," Proceedings of the IEEE, November 1998.

[17]Johnson, C. and Schutte, J., "Basic Regression Analysis for Integrated Neural Networks (BRAINN) Documentation," Georgia Institute of Technology (2011).