International Journal of Electrical and Computer Engineering (IJECE)
Vol. 10, No. 5, October 2020, pp. 5479∼5486
ISSN: 2088-8708, DOI: 10.11591/ijece.v10i5.pp5479-5486

Benchmarking open source deep learning frameworks

Ghadeer Al-Bdour, Raffi Al-Qurran, Mahmoud Al-Ayyoub, Ali Shatnawi
Jordan University of Science and Technology, Jordan

Article Info

Article history:

Received Jul 23, 2019
Revised Apr 25, 2020
Accepted May 6, 2020

Keywords:

CNTK
Performance comparison
TensorFlow
Theano

ABSTRACT

Deep Learning (DL) is one of the hottest fields. To foster the growth of DL, several open source frameworks appeared providing implementations of the most common DL algorithms. These frameworks vary in the algorithms they support and in the quality of their implementations. The purpose of this work is to provide a qualitative and quantitative comparison among three such frameworks: TensorFlow, Theano and CNTK. To ensure that our study is as comprehensive as possible, we consider multiple benchmark datasets from different fields (image processing, NLP, etc.) and measure the performance of the frameworks' implementations of different DL algorithms. For most of our experiments, we find out that CNTK's implementations are superior to the other ones under consideration.

Corresponding Author:

Mahmoud Al-Ayyoub,
Jordan University of Science and Technology, Irbid, Jordan.
Email: maalshbool@just.edu.jo

1. INTRODUCTION

Deep learning (DL) is the hottest trend in machine learning (ML). Although the theoretical concepts behind DL are not new, it has enjoyed a surge of interest over the past decade due to many factors. One example is that DL approaches have significantly outperformed state-of-the-art (SOTA) approaches in many tasks across different fields such as image processing, computer vision, speech processing, natural language processing (NLP), etc. Moreover, the scientific community (from both the academia and the industry) has quickly and massively adopted DL. Open source implementations of successful DL algorithms quickly appeared on code sharing websites, and were subsequently used by many researchers in different fields.

Several DL frameworks exist, such as TensorFlow, Theano, CNTK, Caffe and PyTorch, each with different features and characteristics. Furthermore, each framework utilizes different techniques to optimize its code. Although the same algorithm is implemented in different frameworks, the performance of the different implementations can vary greatly. A researcher/practitioner looking to use such an algorithm in his/her work would face a difficult choice, since the number of different implementations is high and the effort invested by the research community in scientifically comparing these implementations is limited.

In this work, we aim at providing qualitative and quantitative comparisons between three popular open source DL frameworks: TensorFlow, Theano and CNTK. These frameworks support multi-core CPUs as well as multiple GPUs. All of them import cuDNN, which is a DL library from NVIDIA that supports highly tuned implementations for standard routines such as forward and backward convolution, normalization, pooling and activation layers. We compare these frameworks by training different neural network (NN) architectures on five different standard benchmark datasets for various tasks in image processing, computer vision and NLP. Despite their importance, comparative studies like ours that focus on performance issues are rare.

Limited efforts have been dedicated to conducting comparative studies between SOTA DL frameworks running on different hardware platforms (CPU and GPU) to highlight the advantages and limitations for each framework for different deep NN architectures. These efforts included papers [1-9] as well as online blogs

(https://github.com/soumith/convnet-benchmarks). Due to space constraint, we do not discuss the details of these works here. Interested readers are referred to earlier versions of this work [10, 11] for such details. However, we do note that, in previous studies, the comparison goal focused only on processing time. None of those comparative studies dealt with CPU and GPU utilization or memory consumption. This work covered these metrics to find which of the considered frameworks achieve the best performance. Finally and most importantly, the comparisons involved more datasets from more fields compared with previous studies.

The rest of this paper is organized as follows. Section 2. discusses the frameworks, the way they were used to train the datasets and a brief comparison between them. The methodology we follow is discussed in Section 3. Experimental results and the discussion are detailed in Section 4. The work is concluded with final thoughts presented in Section 5.

2. DEEP LEARNING FRAMEWORKS

The frameworks considered in this comparative study are: CNTK, TensorFlow and Theano. Moreover, we use Keras on top of these frameworks as discussed later. All of these frameworks provide flexible APIs and configuration options for performance optimization. Software versions of the frameworks are shown in Table 1 and their properties are shown in Table 2.

Table 1. Frameworks used for this comparative study

Framework	Major Version	Github Commit ID
CNTK	2.0	7436a00
TensorFlow	1.2.0	49961e5
Theano	0.10.0.dev1	8a1af5b
Keras	2.0.5	78f26df

Table 2. Properties of the considered frameworks

Property	CNTK	TensorFlow	Theano	Keras
Core	C++	C++	Python	Python
CPU	✓	✓	✓	✓
Multi-Threaded CPU	✓	Eigen	Blas, conv2D, Limited OpenMP	✓
GPU	✓	✓	✓	✓
Multi-GPU	✓	✓	X (experimental version)	✓
NVIDIA cuDNN	✓	✓	✓	✓

2.1. Microsoft cognitive toolkit (CNTK)

CNTK is an open source DL framework developed by Microsoft Research [12] for training and testing many types of NN across multiple GPUs or servers. CNTK supports different DL architectures like Feedforward, Convolutional, Recurrent, Long Short-Term Memory (LSTM) and Sequence-to-Sequence NN. In CNTK, a Computational Network learns any function by converting it to a directed graph, where leaf nodes consist of an input values or learning parameters while other nodes represent matrix operation applied to its children. In this case, CNTK has an advantage as it can automatically find the derive gradients for all the computations which are required to learn the parameters. In CNTK, users specify their networks using a configuration file that contains information about the network type, where to find input data and the way to optimize parameters [13]. CNTK interface supports different APIs of several languages such as Python, C++ and C# across both GPU (CUDA) or CPU platforms. According to its developers (https://docs.microsoft.com/en-us/cognitive-toolkit/cntk-evaluation-overview), CNTK was written in C++ in an efficient way, where it removes duplicated computations in forward and backward passes, uses minimal memory and reduces memory reallocation by reusing them.

2.2. Theano

Theano is an open source Python library developed at MILA lab at the University of Montreal as a compiler for mathematical expressions that lets users and developers optimize and evaluate their expressions using NumPy's syntax (a Python library that supports a large and multi-dimensional arrays) [3, 14]. Theano starts performing computations automatically by optimizing the selection of computations, translates them into other machine learning languages such as C++ or CUDA (for GPU) and then compiles them into Python

modules in an efficient way on CPUs or GPUs. Theano's development started in 2008 and it is more popular on a research and ecosystem platform than many DL libraries. Several software packages have been developed to build on top of Theano, with a higher-level user interface which aims to make Theano easier to express and train different architectures of deep learning models, such as Pylearn2, Lasagne and Keras.

2.3. TensorFlow

TensorFlow is an open source framework developed by Google Brain Team [15]. It uses a single data flow graph, expressing all numerical computations, to achieve excellent performance. TensorFlow constructs large computation graphs where each node represents a mathematical operation, while the edges represent the communication between nodes. This data flow graph executes the communication between sub-computations explicitly, which makes it possible to execute independent computations in parallel or to use multiple devices to execute partition computations [15]. Programmers of TensorFlow define large computation graphs from basic operators, then distribute the execution of these graphs across a heterogeneous distributed system (can deploy computation to one or more CPUs or GPUs on a different hardware platforms such as desktops, servers, or even mobile devices). The flexible architecture of TensorFlow allows developers and users to experiment and train a wide variety of NN models; It is used for deploying ML systems into production for different fields including speech recognition, NLP, computer vision, robotics and computational drug discovery. TensorFlow uses different APIs of several languages such as Python, C++ and Java for constructing and executing a graph. Python API is the most complete and the easiest to use (https://www.tensorflow.org/).

2.4. Keras

Keras is an open source DL library developed in Python. It runs on top of CNTK, Theano or TensorFlow frameworks. Keras was founded by Google engineer Chollet in 2015 as a part of the research project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). Keras is designed in a way that allows fast expression with DNN and easy and fast prototyping (modularity and extensibility) [16].

3. METHODOLOGY

The goal of this experimental study is to compare the aforementioned frameworks (Theano, TensorFlow and CNTK) by using them to train Convolutional NN (CNN) and Recurrent NN (RNN) models on standard benchmark datasets of classical problems in image processing (MNIST, CIFAR-10 and Self-driving Car) and NLP (Penn TreeBank and IMDB). Specifically, we aim at comparing the resources consumed by each framework to reach a certain accuracy level for each problem. Thus, we experiment with different epoch counts in order to make sure the accuracy for all frameworks are close to each other. Each framework's performance is evaluated using running time, memory consumption and CPU and GPU utilization. We use a laptop that has an Intel Core i7-6700HQ CPU @ 2.60GHz (4 cores) with 16 GB RAM, 64-bit operating system (Windows 10), and NVIDIA GEFORCE GTX 960m graphics card with PCI Express 3.0 bus support, equipped with 4 GB GDDR5 memory and 640 CUDA cores.

3.1. Benchmark datasets

In this subsection, we discuss the datasets used in our experiments.

3.1.1. MNIST

The MNIST (Mixed National Institute of Standards and Technology) dataset for handwritten digits is widely used in ML [17]. It has 60,000 training images and 10,000 testing images. Each image is 28×28 pixels which is flattened into a 784-value vector. The label of each image is a number between 0 and 9 representing the digit appearing in the image.

3.1.2. CIFAR-10

The CIFAR-10 dataset is one of the datasets collected by Krizhevsky et al. [18, 19]. It consists of 60,000 32×32 color images evenly distributed over ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. There are 50,000 training images and 10,000 test images. The classes are completely mutually exclusive. I.e., there is no overlap between them. For instance, the "Automobile" class includes sedans, SUVs, etc. On the other hand, the "Truck" class includes only big trucks. To avoid overlap, neither one of these two classes includes pickup trucks.

3.1.3. Penn TreeBank

In 1993, Marcus et al. [20] wrote a paper on constructing a large annotated corpus of English called the Penn TreeBank (PTB). They reviewed their experience with constructing one large annotated corpus that consists of over 4.5 million words of American English. It was annotated for part-of-speech (POS) tag information. Moreover, half of the corpus was annotated for skeletal syntactic structure. The dataset is large and diverse. It includes the Brown Corpus (retagged) and the Wall Street Journal Corpus, as well as Department of Energy abstracts, Dow Jones Newswire stories, Department of Agriculture bulletins, Library of America texts, MUC-3 messages, IBM Manual sentences, WBUR radio transcripts and ATIS sentences.

3.1.4. IMDB

The IMDB dataset [21] is another example of applying CNN, which is an online dataset of information regarding films, TV programs and video games. It consists of 25,000 reviews labeled by the sentiment (positive/negative) of each review. The reviews have been preprocessed and encoded as integers in a form of a sequence of word indexes. Words are indexed by overall frequency in the dataset, so that the index i encodes the ith most frequent word in the data in order to allow operations of quick filtering.

3.1.5. Self-Driving Car

This dataset uses a Udacity's Self-Driving Car simulator as a testbed for training an autonomous car. This work started in the 1980s with Carnegie Mellon University's Navlab and ALV projects [22]. The training phase starts with activating the simulator which is an executable application. A user initiates the service of collecting the data for training followed by collecting the data as images and saving them locally on the computer. So, the framework can take these images and train them. The training is done via distinguishing the image edges, which are taken by the three cameras laying on the front of the car in the simulator. After the training phase is done, the testing phase begins by taking the file generated whenever the performance in the epoch is better than the previous best. Finally, the last generated file is executed in order to make the car drive autonomously to observe the testing phase results.

3.2. Networks architecture

CNN is used for the MNIST, CIFAR-10, IMDB and Self-Driving Car datasets, where a different network architecture is used for each dataset. The architecture of each CNN is shown in [23]. For the MNIST and CIFAR-10 datasets, two convolutional layers with ReLU activation function are used after the input layer. The activation function is used to reduce the training time and to prevent vanishing gradients. After each CNN layer, a max-pooling layer is added in order to down-sample the input and to reduce overfitting. In the max-pooling layer, the stride value must be specified with which the filter is slid. When the stride is x, the filter (window) is moved x pixels at a time. This will produce smaller output volumes spatially. After each max-pooling layer, the dropout method is used in order to reduce overfitting by forcing the model to learn many independent representations of the same data through randomly disabling neurons in the learning phase.

For the Self-Driving Car dataset, the CNN has the same components as the ones used with the MNIST and CIFAR-10 datasets, but with deeper model that consists of five convolutional layers with Exponential Linear Unit (ELU) activation function. The convolutional layers are used for feature engineering. The fully connected layer is used for predicting the steering angle (final output). The dropout avoids overfitting and, finally, the ELU activation function is used to solve the problem of the vanishing gradient.

In the IMDB dataset, the movie reviews are composed of sequences of words of different lengths. These words are encoded by mapping movie reviews to sequences of word embeddings where words are mapped to vectors of real numbers; the network architecture consists of an embedding layer followed by a 1D convolutional layer which is used for temporal data followed by a global max-pooling operation. These sequences are padded to have the same size as the largest sequence because they have different lengths.

The other NN type we consider is RNN with LSTM. One of the most popular uses of LSTM is for text analysis tasks such as the ones associated with the Penn TreeBank (PTB) dataset. Word-level prediction experiments on PTB was adopted, which consists of 929k training words, 73k validation words and 82k test words. It has 10k words in its vocabulary. We trained models of two sizes (small and medium) using the same architecture presented in [24]. To evaluate language models of the PTB implementation, a special metric called a perplexity is used, where better prediction accuracy is achieved when perplexity value is as low as possible. Perplexity is the inverse of probability definition. This means that minimizing perplexity value is the same as maximizing probability. The goal of applying PTB dataset is to match a probabilistic form which

assigns probabilities to sentences. This process is done by predicting the next words in a text given a history of previously located words. LSTM cells represent the core of the model which processes one word at a time and computes probabilities of the possible values for the next word in the sentence. A vector of zeros is used for the memory state of the network to get initialized and updated after reading each word. In the small model, two hidden layers (with 200 LSTM units per layer) are used with Tanh activation function. The weights are initialized to 0.1. We trained it for four epochs with a learning rate of one (number of epochs trained with initial learning rate) and then the learning rate is decreased by a factor of two after each epoch (the decay of the learning rate for each epoch after four epochs), for a total of 13 training epochs. The size of each batch is 20, then the network is unrolled for 20 steps. This model's architecture is shown in [23].

4. RESULTS AND DISCUSSION

In this section we discuss the results of our experiments. For each model on each dataset, Table 3 shows the CPU and GPU processing times while Tables 4–8 show the utilization levels of the CPU, GPU and their memories. For the image classification datasets (MNIST and CIFAR-10), one can observe the superiority of CNTK over TensorFlow and Theano in terms of GPU and CPU multithreading; however, in CIFAR-10 using 8, 16 and 32 threads in CPU, TensorFlow was faster than CNTK. On the other hand, Theano revealed to be more time consuming than other frameworks. Transitioning to sentiment analysis dataset (IMDB), CPU multithreading was not performed because CNTK is written in Python in which multithreading is not supported. Without CPU multithreading (CPU uses the default number of existing physical cores which are equal one thread per core), the superiority of TensorFlow is revealed in both CPU and GPU environments. The results for the text analysis dataset (Penn TreeBank) shows the superiority of TensorFlow over CNTK and Theano, for CPU with 8 threads as well as the case in GPU. Moving forward to video analysis dataset (Self-Driving Car), the superiority of TensorFlow is revealed in both CPU and GPU environments, while CNTK showed to be more time consuming than the other two frameworks.

The processing times clearly show the advantage of GPU over CPU for training CNN and RNN. The advantage of fast GPU would be more significant when training complex models with larger data as in the Self-Driving Car dataset. From the CPU results, the best performance occurred when the number of threads is equal to the number of physical CPU cores, where each thread possesses a single core. In our work, we use a laptop with 8 cores. Thus, in each dataset, the best performance in terms of processing time was achieved while using 8 threads. The metrics measurement of each framework was conducted to explain the failure of one of the selected frameworks.

We notice poor performance of Theano at most datasets comparing to CNTK and TensorFlow. This could be attributed to its low CPU utilization compared to the other frameworks. CNTK outperformed both TensorFlow and Theano while training MNIST and CIFAR-10 datasets. This achievement is highly likely due to the use of BrainScript format (https://docs.microsoft.com/en-us/cognitive-toolkit/BrainScript-Network-Builder), which is a custom network description language that makes CNTK more flexible for NN customization. On the other hand, TensorFlow uses Eigen (http://eigen.tuxfamily.org/index.php?title=Main_Page), which is a C++ template library (BLAS library) for linear algebra including matrices, vectors, numerical solvers and related algorithms. It is used to make TensorFlow perform better than CNTK and Theano in RNN.

In addition to processing time, we also report the utilization levels of the CPU, GPU and their memories for each model on each framework under consideration. These results are shown in Tables 4–8. The utilization levels for both CPU and GPU are high for all models. The only supersizing numbers are the CPU utilization for Theano, which were very low. The tables also show that the utilization levels are rather small for both types of memory. This applies to all models for all frameworks. However, the tables also show that, in most cases, CNTK had the lowest memory utilization while TensorFlow had the highest. Surprisingly, the case is almost reversed for the video analysis dataset (the Self-Driving Car dataset), where CNTK had the highest utilization and Theano had the lowest. Another unexpected finding of these experiments is that the models of the IMDB generally needed the largest portions of memory.

Comparing our work to previous works [1, 2], we reveal the following findings. Bahrampour et al. [1] based their comparative study on three main aspects including speed, hardware utilization and extensibility. Besides, they used three NN types: CNN, AutoEncoder (AE) and LSTM to train MNIST, ImageNet [25] and IMDB datasets on Caffe, Neon, TensorFlow, Theano and Torch frameworks. They came up with the following results. Training on CPU, Torch performed the best followed by Theano while Neon had the worst

performance. Moreover, Theano and Torch are the best in terms of extensibility, as well as TensorFlow and Theano were very flexible and Caffe was the easiest to find the performance. Regarding training datasets on GPU and for larger convolutional and fully connected networks (FCN), Torch was the best followed by Neon. For smaller networks Theano was the best. For LSTM, Theano's results were the best, while TensorFlow's performance was not competitive compared with the other studied frameworks.

On the other hand, Shi et al. [2] based their comparative study on two metrics: processing time and convergence rate. The NN used are fully connected NN, CNN and RNN to train ImageNet, MNIST and CIFAR-10 datasets on Caffe, CNTK, MXNet, TensorFlow and Torch frameworks. The results of TensorFlow were the best using CPU. While using single GPU; on FCN, Caffe, CNTK and Torch performed better than MXNet and TensorFlow. As for small CNN, Caffe and CNTK achieved a good performance and for RNN (LSTM), however, CNTK was the fastest (5-10x faster than other frameworks). Using multi-GPU implementation, all frameworks had higher throughput and accelerated the convergence speed compared with single-GPU implementation.

Table 3. Processing time for each dataset (measured in seconds), For the Environment columns, CPU (x) denotes CPU with x threads

	Env	CNTK	TensorFlow	Theano
MNIST	CPU (1)	847	5130	3560
	CPU (2)	630	3180	2500
	CPU (4)	574	2070	2260
	CPU (8)	560	1740	2060
	CPU (16)	567	1920	2050
	CPU (32)	588	2010	2050
	GPU	66.67	328.93	377.86
CIFAR-10	CPU (1)	20196	25905	26700
	CPU (2)	14520	16610	18700
	CPU (4)	13662	11550	17250
	CPU (8)	11484	9955	15800
	CPU (16)	11550	10340	15850
	CPU (32)	11649	10835	15750
	GPU	926	2166.4	2386.1
IMDB	CPU (1)	-	1244	538
	CPU (2)	-	642	412
	CPU (4)	-	390	380
	CPU (8)	486	290	368
	CPU (16)	-	249	368
	CPU (32)	-	302	384
	GPU	73.1	62.4	220.
Self-Driving Car	CPU (1)	-	∼33.3 hours	∼50 hours
	CPU (2)	-	∼19.8 hours	∼44.2 hours
	CPU (4)	-	∼15 hours	∼42.6 hours
	CPU (8)	∼47.6 hours	∼14.1 hours	∼43.5 hours
	CPU (16)	-	∼16.4 hours	∼43.5 hours
	CPU (32)	-	∼16.4 hours	∼43.5 hours
	GPU	∼8.7 hours	∼6 hours	∼6.8hours
PTB	CPU (1)	-	40560	27066
	CPU (2)	-	26819	23244
	CPU (4)	-	18733	21541
	CPU (8)	4290	16407	21450
	CPU (16)	-	16848	21476
	CPU (32)	-	18369	21541
	GPU	2106	1342.28	1630

Table 4. Performance metrics of all models on the MNIST dataset

Metrics	Environment	CNTK	TensorFlow	Theano
Accuracy	CPU	99.27	99.14	99.10
	GPU	99.26	99.11	99.17
CPU%	-	99.6	92.2	14.7
GPU%	-	92	77	95
Memory%	CPU	1.7	2.2	2.1
	GPU	3.6	5.2	4.9
Epochs#	CPU	7	15	10
	GPU	7	15	10

Table 5. Performance metrics of all models on the the CIFAR-10 dataset

Metrics	Environment	CNTK	TensorFlow	Theano
Accuracy	CPU	82.68	82.26	82.29
	GPU	82.57	82.33	82.30
CPU%	-	99.8	87.3	15.3
GPU%	-	97	73	94.5
Memory%	CPU	3.2	5.3	5.1
	GPU	4.5	7.4	7.5
Epochs#	CPU	33	55	50
	GPU	33	55	50

Table 6. Performance metrics of all models on the IMDB dataset

Metrics	Environment	CNTK	TensorFlow	Theano
Accuracy	CPU	88.87	88.68	88.72
	GPU	88.93	88.83	88.48
CPU%	-	94.8	92.2	14.6
GPU%	-	76	76	88
Memory%	CPU	5.7	6.6	5.1
	GPU	6.6	9.1	7.6
Epochs#	CPU	2	3	2
	GPU	2	3	2

Table 7. Performance metrics of all models on the Self-Driving Car dataset

Metrics	Environment	CNTK	TensorFlow	Theano
Accuracy	CPU	99.93	99.96	99.71
	GPU	99.97	99.97	99.73
CPU%	-	93.2	85	22
GPU%	-	32.4	34	31
Memory%	CPU	5.3	4.3	3.2
	GPU	6.6	6.2	5.3
Epochs#	CPU	10	10	10
	GPU	10	10	10

Table 8. Performance metrics of all models on the Penn TreeBank dataset

Metrics	Environment	CNTK	TensorFlow	Theano
Perplexity	CPU	113.7	114.79	114.57
	GPU	113.2	113.21	113.3
CPU%	-	94	91.8	18.3
GPU%	-	76.6	77	81
Memory%	CPU	1.3	2.3	4.1
	GPU	2.2	4.5	5.4
Epochs#	CPU	13	13	13
	GPU	13	13	13

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have provided a qualitative and quantitative comparison between three of the most popular and most comprehensive DL frameworks (namely Microsoft's CNTK, Google's TensorFlow and University of Montreal's Theano). The main goal of this work was to help end users make an informed decision

about the best DL framework that suits their needs and resources. To ensure that our study is as comprehensive as possible, we have used multiple benchmark datasets namely MNIST, CIFAR-10, Self-Driving Car and IMDB which were trained via multilayer CNN network architecture and Penn TreeBank dataset which was trained via RNN architecture. We have run our experiments on a laptop with windows 10 operating system. We have measured performance and utilization of CPU multithreading , GPU and memory. For most of our experiments, we find out that CNTK's implementations are superior to the other ones under consideration.

REFERENCES

[1] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of deep learning software frameworks," *arXiv preprint arXiv:1511.06435*, 2015.

[2] S. Shi, et al., "Benchmarking state-of-the-art deep learning software tools," *arXiv preprint arXiv:1608.07249*, 2016.

[3] R. Al-Rfou et al., "Theano: A python framework for fast computation of mathematical expressions," *arXiv preprint arXiv:1605.02688*, 2016.

[4] P. Goldsborough, "A tour of tensorflow," *arXiv preprint arXiv:1610.01178*, 2016.

[5] V. Kovalev et al., "Deep learning with theano, torch, caffe, tensorflow, and deeplearning4j: Which one is the best in speed and accuracy?," *Pattern Recognition and Information Processing (PRIP)*, 2016.

[6] F. Bastien, et al., "Theano: new features and speed improvements," *arXiv preprint arXiv:1211.5590*, 2012.

[7] W. Ding, R. Wang, F. Mao, and G. Taylor, "Theano-based large-scale visual recognition with multiple gpus," *arXiv preprint arXiv:1412.2302*, 2014.

[8] W. Dai and D. Berleant, "Benchmarking contemporary deep learning hardware and frameworks: A survey of qualitative metrics," *International Conference on Cognitive Machine Intelligence*, pp. 148-155, 2019.

[9] C. Coleman et al., "Dawnbench: An end-to-end deep learning benchmark and competition," *31st Conference on Neural Information Processing Systems*, vol. 100, no. 101, 2017.

[10] A. Shatnawi et al., "A comparative study of open source deep learning frameworks," *9th International Conference on Information and Communication Systems*, pp. 72-77, 2018.

[11] G. Al-Bdour, R. Al-Qurran, M. Al-Ayyoub, and A. Shatnawi, "A detailed comparative study of open source deep learning frameworks," *arXiv preprint arXiv:1903.00102*, 2019.

[12] D. Yu et al., "An introduction to computational networks and the computational network toolkit," *Microsoft, Tech. Rep. MSR-TR-2014-112*, 2014.

[13] D. Yu, K. Yao, and Y. Zhang, "The computational network toolkit [best of the web]," *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp. 123-126, 2015.

[14] J. Bergstra et al., "Theano: A cpu and gpu math compiler in python," *Proc. 9th Python in Science Conference*, vol. 1, pp. 3-10, 2010.

[15] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 265-283, 2016.

[16] F. Chollet et al., "Keras," 2015.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, pp. 1097-1105, 2012.

[19] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *University of Toronto, Tech. Rep.*, 2009.

[20] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313-330, 1993.

[21] A. Maas et al., "Learning word vectors for sentiment analysis," *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

[22] R. Wallace et al., "First results in robot road-following," *IJCAI*, pp. 1089-1095, 1985.

[23] G. Al-Bdour, "Comparative study between deep learning frameworks using multiple benchmark datasets," *Master's thesis, Jordan University of Science and Technology*, 2017.

[24] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[25] J. Deng, et al., "Imagenet: A large-scale hierarchical image database," *IEEE conference on computer vision and pattern recognition*, pp. 248-255, 2009.