# No! Not Another Deep Learning Framework

Linh Nguyen*
lvnguyen@umich.edu
University of Michigan

Peifeng Yu*
peifeng@umich.edu
University of Michigan

Mosharaf Chowdhury
mosharaf@umich.edu
University of Michigan

## ABSTRACT

In recent years, deep learning has pervaded many areas of computing due to the confluence of an explosive growth of large-scale computing capabilities, availability of datasets, and advances in learning techniques. While this rapid growth has resulted in diverse deep learning frameworks, it has also led to inefficiencies for both the users and developers of these frameworks. Specifically, adopting useful techniques across frameworks – both to perform learning tasks and to optimize performance – involves significant repetitions and reinventions.

In this paper, we observe that despite their diverse origins, many of these frameworks share architectural similarities. We argue that by introducing a common representation of learning tasks and a hardware abstraction model to capture compute heterogeneity, we might be able to relieve machine learning researchers from dealing with low-level systems issues and systems researchers from being tied to any specific framework. We expect this decoupling to accelerate progress in both domains.

## CCS CONCEPTS

• **Software and its engineering → Software architectures**;

**ACM Reference format:**
Linh Nguyen, Peifeng Yu, and Mosharaf Chowdhury. 2017. No! Not Another Deep Learning Framework. In *Proceedings of HotOS '17, Whistler, BC, Canada, May 08-10, 2017,* 6 pages.
https://doi.org/10.1145/3102980.3102995

## 1 INTRODUCTION

Significant progress in deep learning techniques in recent years has led to its broad adoption in many data-driven applications [20, 21, 36]. Because deep learning models typically perform better with more data, advances in system efficiency and scalability often directly translate to model quality. This development has led to the proliferation of deep learning frameworks in the industry – e.g., Google's TensorFlow [5], Microsoft's CNTK [53], and Facebook's Caffe2 [1] – and in academia [16, 40, 48, 51].

We observe that, except for a few outliers [17, 31], the majority of deep learning frameworks [5, 10, 16, 53] are converging to a common pipeline design (Figure 1). A framework's *model construction API* is the front-end that takes user inputs. The *symbolic computational graph* serves as the internal representation of the model that goes through a series of *transformations*. Finally, the *execution engine* is the back-end that carries out the computations.

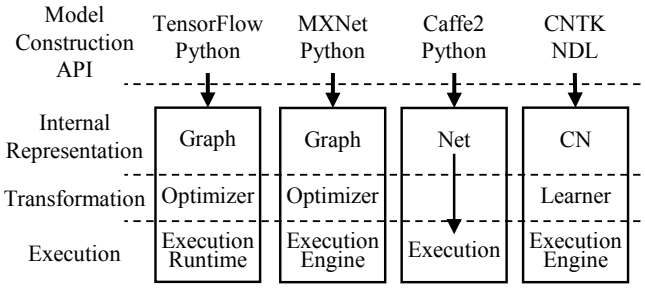


**Figure 1: Architectures of common deep learning frameworks. TensorFlow and MXNet are quite similar to each other. Caffe2 does not have a transformation phrase. CNTK uses Network Definition Language (NDL) for network specification and Computation Network (CN) as the internal representation.**

Unfortunately, these frameworks have little in common beyond these high-level similarities. Each proposes its own API, representation, optimizations, and execution engine. This diversity often stems from the need to solve a specialized set of problems and then attempting to generalize as an afterthought, and it hinders interoperability, deployability, and reusability of techniques. Users who want to use complex deep learning models are often restricted to one specific implementation, because porting models across frameworks requires significant efforts.

Given the diversity of deep learning frameworks and their divergent user bases, we call for a respite before building yet another deep learning framework and ask a simple question: *do we really need one more?* We postulate that there are already many deep learning frameworks; what we do need is interoperability and sharing between them.

In this paper, we articulate our vision for designing a deep learning software stack with *interoperability* and *sharing* as the two guiding principles. Drawing key insights from real-world challenges and similarities among existing frameworks, we argue that such a design is essential to reduce the efforts needed for machine learning researchers to reuse existing work and to provide a common platform for system researchers to develop techniques for better resource management. We conclude by discussing potential challenges and open research directions enabled by such a common platform.

*Both authors contributed equally to this work.


*HotOS '17, May 08-10, 2017, Whistler, BC, Canada*

ACM ISBN 978-1-4503-5068-6/17/05…$15.00
https://doi.org/10.1145/3102980.3102995

## 2 BACKGROUND

This section identifies the gap between what current systems can offer and the expectations of machine learning and systems researchers as deep learning continues to evolve. We focus on concerns that motivate the need for a common infrastructure for current and future frameworks.

### 2.1 Existing Frameworks

Each new deep learning framework justifies its contributions based on historical or commercial reasons. Early artifacts from the academia [10, 31] originated from in-house libraries, long before any mature deep learning frameworks were open-sourced. From the industry side, companies such as Google, Microsoft, Facebook, and Amazon incubate their own frameworks [1, 5, 16, 53] to fulfill internal needs of different products. Oftentimes, these frameworks gain rapid popularity thanks to their code quality and documentation.

Regardless of their origins, the interfaces of these frameworks can broadly be categorized as either declarative or imperative. Frameworks following the former allow users to specify their computations in advance, usually in some symbolic fashion, and internally represent them as dataflow graphs [1, 5, 10, 51, 53]. In case of the latter, computation happens as soon as it is defined [3, 17]. While most frameworks choose one, a few – for example, MXNet [16] – try to combine both.

Existing frameworks lack resource management capabilities for large-scale deployments. Each assumes that it has complete control of the hardware device (e.g., a GPU), making resource sharing nearly impossible. Users can work around by enabling CUDA Multi-Process Service (MPS) [2] in case of Nvidia GPUs and ensuring that the applications' collective memory requirements do not exceed the GPU's capacity. Prior work has shown that this approach leads to little performance penalty for a small number of applications [25] on a single server; however, all applications must be written in the same framework, and this does not extend to distributed settings. Overall, the lack of inherent support for resource sharing places burden on users and creates scalability challenges.

### 2.2 Challenges and Expectations

In this section, we consider the concerns of the users and developers of deep learning frameworks.

#### 2.2.1 *Machine Learning Practitioners.*

The front-end users of deep learning frameworks are machine learning practitioners, who develop new models for distinct problems. There are currently three trends in this community that call for a common representation across deep learning frameworks.

**Larger models and data**: Recent developments in deep learning have demonstrated that deeper networks tend to outperform shallow networks in many cases (e.g., in vision tasks [26]). In addition, the wide adoption of deep learning techniques in various applications such as image, video, and natural language processing [33, 34, 47] has led to an explosive growth of datasets available to deep learning models.

Naturally, this has led to different parallelization strategies with *model parallelism* and *data parallelism* being the two most prominent ones. Data parallelism trains separate model replicas with their own data chunks and updates a global model. This technique has become the de facto method in distributed training, with streamlined support in modern frameworks [1, 3, 5, 16]. Model parallelism, on the other hand, seeks to split a model into different devices, and it often requires manual management of the dataflow graph in many frameworks. Some recent efforts, such as SINGA [51], provide support for both model and data parallelism.

Nevertheless, because the implementation of one framework cannot be reused for another, deep learning frameworks provide varying degrees of support for these parallelization techniques.

**Comparing learning models**: Because of the availability of many frameworks, improvements to an existing algorithm may not always be implemented in the same system as the original. For instance, when proposing a new object detection algorithm, it is natural to benchmark it against a popular algorithm in this field, Faster R-CNN [42], which is implemented in Caffe [31]. A fair comparison is only feasible if the new proposal is also implemented in Caffe, but that restricts the flexibility of choosing a deep learning framework. In contrast, when implemented using a different framework, head-to-head comparison leads to uncertainty about performance: it is difficult to determine how much of it comes from algorithmic improvements vs. that from the underlying implementation.

This lack of interoperability also hinders researchers from quickly assessing new models proposed in different deep learning frameworks. They have to either spend time setting up new environments or put efforts toward reimplementing them in their own frameworks.

**Combinations of learning models**: Modern data analytics workflows, which utilize machine learning extensively, rarely use one single end-to-end model. For instance, image detection algorithms are often part of a video analytics pipeline [33, 35, 50]. Faster R-CNN [42] is the state-of-art method for object detection in static images and is often referenced in such projects [7, 34]. However, researchers can only use Caffe, the framework in which Faster R-CNN was implemented. Otherwise, they would have to spend time porting Faster R-CNN to another framework.

An ad-hoc solution is to write a driver that calls components of the workflow written in different frameworks and transfers data between them. However, mixing several learning frameworks in the system is often complex and error-prone. Shoumik et al. showed that this method significantly slows down the performance of the whole data analytics pipeline [41]. Additionally, when contending for GPU memory, multiple frameworks may cause out-of-memory errors.

***Expectations**: Ideally, machine learning practitioners should not have to deal with the differences among deep learning frameworks. Parallelization, inter-framework operations, and other low-level details should be handled transparently inside the framework. From a users' perspective, any framework should just work with models and data at arbitrary scales.*

#### 2.2.2 *Systems Developers.*

Modern deep learning frameworks make heavy use of accelerators such as GPUs and FPGAs [5, 55, 56]. However, many of these devices lack proper OS-level abstractions. This often forces deep learning frameworks to directly use low-level interfaces, leading to various performance and resource management issues.

**Heterogeneous hardware**: GPUs are known to perform well for data-parallel applications. However, other types of accelerators such as FPGAs and ASICs have started to show promises as well [9, 15, 30]. Consequently, chip makers are developing their own accelerators in anticipation of the deep learning market's growth: Google has developed their own Tensor Processing Unit [32] for TensorFlow; Intel, through a recent acquisition is showing interest in a tensor-based architecture, among others.[1] While it is expected that these specialized hardware would outperform GPUs in specific deep learning tasks, specializations for one framework are also unlikely to extend to other frameworks.

**Distributed execution**: While the computational graph used by current frameworks technically allows arbitrary partitioning among devices, the increasing popularity of distributed training poses significant challenges in effectively scheduling them. A distributed model would need to ensure that the model would be updated globally when each node trains on its own replica and data. Distributed implementations are often complex, and only a handful of frameworks support multiple computation devices across multiple machines [5, 16], along with several makeshift solutions [4, 8].

**Applying optimizations**: Even though many of the learning frameworks are open-source and share similar conceptual designs, there is little code sharing between these frameworks. Any system-level optimization technique would require sizable engineering work to port to different frameworks. Besides, advanced features such as automatic differentiation[2] or distributed execution are only supported by a small number of newer frameworks.

**Resource sharing**: Most existing frameworks simply assume they have full access to the hardware resources in a cluster [5, 16]. Some frameworks even have their own cluster abstraction and management component in their codebases [5]. In practice, however, it is desirable to share resources across all applications in the same cluster to improve utilization and to ensure fairness.

***Expectations**: There should be a shared infrastructure that eases the process of supporting new accelerators, supports distributed execution, enables system-level optimizations across multiple frameworks, and ensures resource sharing. System developers should be able to apply their expertise without being locked into any particular framework.*

## 3 TOWARD A COMMON STACK

The two expectations from Section 2 suggest ample opportunities for consolidation across different deep learning frameworks. In this section, we identify two design goals that guide our vision to achieve them:

(1) For machine learning practitioners, a backward compatible interface that integrates well with existing frameworks will help significantly. They can focus solely on their efforts on implementing and evaluating their ideas.[3]

[1] https://www.nextplatform.com/2016/09/07/next-wave-deep-learning-architectures

[2] Instead of manually differentiating the layer function to get the gradient needed for training, the system can automatically calculate the gradients, which significantly reduces the efforts for the researcher to experiment with different neural network structures.

[3] Note that currently our focus is on the training phase of a deep learning model, where the model structure and data can change dynamically, and fast iteration of evaluation is needed.

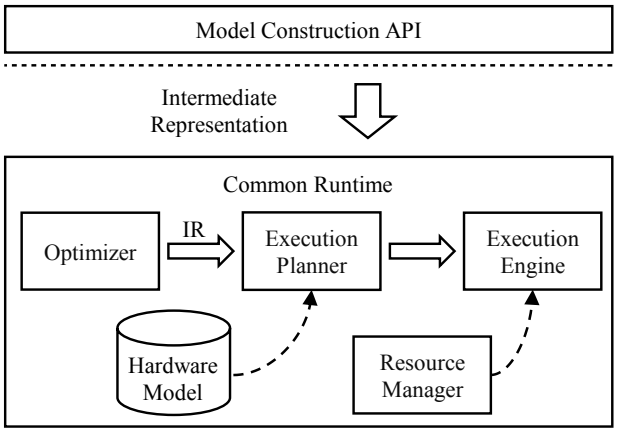


**Figure 2: Key components of a deep learning software stack. The optimizer transforms the intermediate representation (IR) of the computation graph, the execution planner breaks it down into operations, and the execution engine schedules them by interacting with the resource manager.**

(2) For systems developers, a stable but flexible interface would encourage contribution and is crucial to enable an ecosystem around the infrastructure. Any advances in optimizations will also become available to all frameworks.

While we are proposing a common infrastructure to achieve separation of concerns for different users and to facilitate future research, this does not mean that every framework will become the same. Application-level goals can still vary, and frameworks can still provide domain-specific languages. At the same time, system-level design decisions and trade-offs can be made by systems experts without knowing application-level specifics. Figure 2 depicts a high-level architecture of the envisioned software stack.

### 3.1 Intermediate Representation (IR)

The key to enabling interoperability and backward compatibility is enabling an IR that will be the standardized exchange format across different frameworks and their individual components. Different components can define their interfaces based on the IR, and this exchange format can be annotated with information such computation attributes and dependencies. We consider the following properties to be necessary for any potential IR.

**Expressiveness**: First, machine learning researchers will continue to develop new models in the future at different granularities. One may combine different models in various ways or modify existing models. The IR should be able to capture the computations and dependencies of these architectures.

Second, an expressive IR will convey enough information to the runtime to assist in sophisticated optimizations and execution planning. For instance, automatic scheduling of both model and data parallelism for arbitrary models would require the optimizer to consider device placement constraints for specific operations.

Third, serving as a medium of exchange, the IR will allow separate components to evolve independently. They will only require the information contained in the IR to perform their functionalities.

**Hardware-agnostic**: Similar to program IRs in compilers, a hardware-agnostic IR for deep learning will abstract hardware-level details away from the users. As long as the input to a framework can be represented by a valid IR, it can be executed on any hardware platform that supports that IR.

## 3.2 Runtime

The runtime will consist of loosely coupled components that communicate using the aforementioned IR.

*3.2.1 Optimizer.*

Most deep learning frameworks involve an optimization phase that transforms the IR based on pre-defined rules or cost metrics. However, instead of having framework-specific optimizers, this phase should provide enough functionalities to help developers and system researchers develop new optimization algorithms.

First, the optimizer must be modularized and pluggable so that users can easily introduce new optimizations. In the case of a multi-pass optimizer, users should be able to isolate the effect of specific passes without affecting other parts of the system.

For simplicity and modularity, the IR will be the only input to the optimizer. By limiting the data structure the optimizer can access, we can enforce better abstractions and greatly reduce the management burden and learning curves for beginners. In contrast, providing more information may result in better optimizations. Finding the best tradeoff in scenarios like this is a well-known interface design problem.

*3.2.2 Execution Planner.*

Given an optimized IR, the execution planner decides where and when the actual computation should happen. This decision includes, for example, which executor gets the task to run and where should a memory blob be allocated. Much like the optimizer, the execution planner makes decisions based on IR, but it also have access to hardware-specific information provided by the hardware models. However, we envision that while the optimizer is allowed to change what computations will happen, the execution planner will work on a read-only version of the IR. This division of work between the optimizer and the execution planner is inspired by that in logical and physical planning in SQL query execution.

The execution planner exposes opportunities for more system-level optimizations. When combined with the distributed executor, it may improve network communication characteristics. Even lower level optimizations such as fine-grained GPU sharing and better utilization of heterogeneous accelerators should also be possible.

From a systems perspective, we expect continued research on optimization and execution policies. Therefore, the execution planner should provide a pluggable policy interface to make it possible to adapt to different situations, where the objective metrics might be different. Furthermore, this will encourage the development of advanced placement algorithms that will be readily available to all frameworks instead of being locked to a specific one.

*3.2.3 Hardware Model.*

In current deep learning frameworks, users often have to know significant details of the underlying hardware whenever they have to port their algorithms to a new platform. To address this challenge, we envision a hardware model that can represent different hardware platforms (e.g., CPUs, GPUs, and other type of accelerators) via a common queryable interface. Both single-node and distributed clusters should be supported through the same interface as well.

Beyond an abstraction, these models should also provide sufficient information to the optimizer and execution planner. For example, a cost-based optimization algorithm needs to know the cost of operations on all available compute devices as well as the communication cost between them; it can then make decisions on whether using replicas for hot data is more efficient than simply recalculating them on demand.

Finally, registration of new accelerators and hardware configurations should be straightforward. This is even more important in a distributed setting, because a cluster is likely to have some churn in terms of new devices being added and old ones being retired.

*3.2.4 Execution Engine.*

The final element of the runtime is the execution engine that carries out the output of the execution planner on the actual single-device, multi-device, or distributed hardware platform. To enable fine-grained resource sharing and to achieve high utilization, the execution engine needs to be able to execute multiple operations on the same device. Note that this seemingly simple goal is difficult today because of little or no support in existing hardware (e.g., fine-grained GPU sharing does not work well even with CUDA MPS). At the same time, it is essential for the execution engine to be compatible with existing libraries employed in deep learning frameworks such as cuDNN, Intel MKL, or similar libraries.

## 3.3 Backward Compatibility

To alleviate the hurdles to backward compatibility, we hope to achieve graceful degradation via a separation between the full API and the legacy API, as shown in Figure 3.

Existing frameworks can first adopt the legacy API for easy integration. However, due to certain limitations in these frameworks' device abstraction layers, the benefits of the components outlined above are likely to be limited. For example, many frameworks only submit computations independently to the GPU; this can restrict the optimizer from making decisions that use both the dependencies and computations. Therefore, the legacy API is directly connected to the execution planner.

We believe that the least-resistant path toward including existing frameworks is writing plugins for them to convert their current intermediate representations to the common IR and then replacing their optimizers, execution planners, and execution engines with the shared runtime components.

# 4 CONCLUSIONS

Our overarching goal is enabling consolidation of efforts across many areas of computing to better implement, leverage, and optimize deep learning frameworks. If successful, this would foster a rich ecosystem, where one can interoperate across a variety of workloads, frameworks, and techniques, while still benefiting from well-developed optimization techniques.

So far, we have highlighted some important guidelines to enable interoperability and sharing, but their concrete implementations remain open research challenges. We conclude this paper by highlighting some of the pressing questions that must be answered to realize our proposal.

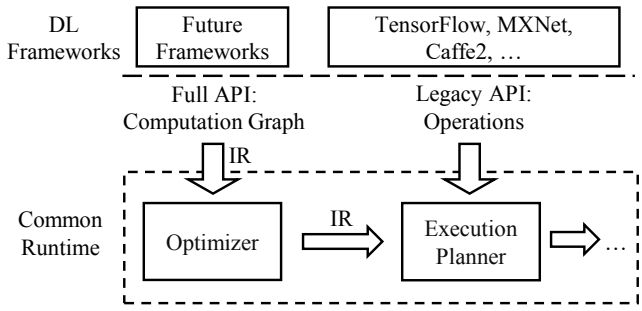


**Figure 3: Two sets of APIs should be provided for high-level frameworks to integrate with the common runtime.**

**What is a good intermediate representation**? Since most deep learning frameworks already use an internal computational graph representation, it is natural to first examine them and perhaps conclude that this is the common intermediate representation we are looking for. Computational graph is flexible enough to support not only deep neural networks [19, 28, 54], but also other types of networks such as convolutional neural networks [6, 14, 37] and recurrent neural networks [29, 44, 46]. The ability to transform the graph also brings automatic differentiation [22, 24] and various optimizations like constant folding [52] or common sub-expression elimination [38].

However, we note that this is not necessarily the best option. Palkar et al. [41] recently presented an intermediate representation design close to monad comprehensions [23] and multiloop construct [11, 43], which achieved significant improvement for logistic regression in TensorFlow, among other data-intensive applications.

**How to model the hardware abstraction**? Very few of the current deep learning frameworks consider devices other than CPUs and Nvidia GPUs. The current solution to support a different architecture often boils down to implementing a new framework: for example, CNNLab [56] supports FPGA in addition to a Nvidia GPU. How to develop a model based on the costs of its computation and communication with other devices remains an open question. Whether computation and communication can sufficiently capture the complexities of modern accelerators may deserve a reassessment as well.

**What (cost-based) optimizations and execution policies are possible**? This question depends heavily on the choice of an intermediate representation. While there is rich literature in optimizing traditional programming languages based on their intermediate representations [39, 45] as well as rule- and cost-based optimizations of SQL queries [12, 13], little attention has been paid toward developing a symbolic representation for deep learning workflows: Currey et al. [18] did some initial work on optimal device allocation using a graph cut-based algorithm given the computational graph. Ideally, the intermediate representation should enable optimizations across different learning models used together, similar to inter-procedural and program optimizations in compilers. Given the prevalence of distributed systems, how to efficiently schedule the devices and keep them fully utilized during application runtime is also an open question.

**How to enforce execution policies in multi-tenant scenarios**? Resource sharing is fundamental to achieving high utilization and fair allocation between multiple applications and tenants. Sharing can happen both within a single compute device such as a GPU and across an entire cluster. While it is easy to share GPUs inside a cluster in a coarse-grained per-GPU fashion – existing cluster managers such as Apache Mesos [27] allocate an entire GPU to a task [49] – it may be desirable to share a single GPU across multiple tenants with certain QoS or fairness guarantees to improve GPU utilization and to reduce overall execution times.

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their insightful comments and feedback. This work was supported by the National Science Foundation under Grant No.: CNS-1617773, CCF-1629397 and CNS-1563095.

## REFERENCES

[1] Caffe 2. https://caffe2.ai. Accessed: 2017-04-21.
[2] Cuda multi-process service. https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf. Accessed: 2017-04-27.
[3] PyTorch. http://pytorch.org. Accessed: 2017-04-21.
[4] Tensorflowonspark. https://github.com/yahoo/TensorFlowOnSpark. Accessed: 2017-04-21.
[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A system for large-scale machine learning. In *OSDI*, 2016.
[6] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *ICASSP*, 2012.
[7] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*, 2016.
[8] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda. S-Caffe: Co-designing MPI runtimes and Caffe for scalable deep learning on modern GPU clusters. In *PPoPP*, 2017.
[9] R. Bekkerman, M. Bilenko, and J. Langford. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.
[10] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron, and Y. Bengio. Theano: Deep learning on GPUs with Python. In *BigLearn, NIPS Workshop*, 2011.
[11] K. J. Brown, H. Lee, T. Rompf, A. K. Sujeeth, C. De Sa, C. Aberger, and K. Olukotun. Have abstraction and eat performance, too: Optimized heterogeneous computing with parallel patterns. In *CGO*, 2016.
[12] S. Chaudhuri. An overview of query optimization in relational systems. In *SIGMOD/PODS*, 1998.
[13] S. Chaudhuri and V. R. Narasayya. An efficient, cost-driven index selection tool for Microsoft SQL server. In *VLDB*, 1997.
[14] K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. In *ICFHR*, 2006.
[15] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM SIGPLAN Notices*, 2014.
[16] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.
[17] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A Matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.
[18] J. Currey, A. Eversole, and C. Rossbach. Scheduling dataflow execution across multiple accelerators. In *SFMA Workshop*, 2014.
[19] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *TASLP*, 20(1):30–42, 2012.
[20] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *NIPS*, 2012.
[21] L. Deng and D. Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(3–4):197–387, 2014.

[22] A. Griewank and A. Walther. *Evaluating derivatives: Principles and techniques of algorithmic differentiation*. SIAM, 2008.

[23] T. Grust. Monad comprehensions: A versatile representation for queries. In *The Functional Approach to Data Management*, 2004.

[24] B. Guenter. Efficient symbolic differentiation for graphics applications. *TOG*, 26(3):108, 2007.

[25] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang. DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers. In *ISCA*, 2015.

[26] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[27] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, 2011.

[28] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[29] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[30] K. Irick, M. DeBole, V. Narayanan, and A. Gayasen. A hardware efficient support vector machine architecture for FPGA. In *FCCM*, 2008.

[31] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACMMM*, 2014.

[32] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a Tensor Processing Unit. In *ISCA*, 2017.

[33] K. Kang, H. Li, J. Yan, X. Zeng, B. Yang, T. Xiao, C. Zhang, Z. Wang, R. Wang, X. Wang, and W. Ouyang. T-CNN: Tubelets with convolutional neural networks for object detection from videos. *arXiv preprint arXiv:1604.02532*, 2016.

[34] K. Kang, W. Ouyang, H. Li, and X. Wang. Object detection from video tubelets with convolutional neural networks. In *CVPR*, 2016.

[35] C. Kim and J.-N. Hwang. Fast and automatic video object segmentation and tracking for content-based applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(2):122–129, 2002.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[37] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[38] T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM TOPLAS*, 1(1):121–141, Jan. 1979.

[39] N. P. Lopes, D. Menendez, S. Nagarakatte, and J. Regehr. Provably correct peephole optimizations with alive. *ACM SIGPLAN Notices*, 50(6):22–32, 2015.

[40] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan. SparkNet: Training deep networks in Spark. *arXiv preprint arXiv:1511.06051*, 2015.

[41] S. Palkar, J. J. Thomas, A. Shanbhag, D. Narayanan, H. Pirk, M. Schwarzkopf, S. Amarasinghe, and M. Zaharia. Weld: A common runtime for high performance data analytics. In *CIDR*, 2017.

[42] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[43] T. Rompf, A. K. Sujeeth, N. Amin, K. J. Brown, V. Jovanovic, H. Lee, M. Jonnalagedda, K. Olukotun, and M. Odersky. Optimizing data structures in high-level programs: New directions for extensible compilers based on staging. *ACM SIGPLAN Notices*, 48(1):497–510, 2013.

[44] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 2011.

[45] Y. Sui and J. Xue. SVF: Interprocedural static value-flow analysis in llvm. In *CC*, 2016.

[46] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

[47] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[48] L. Truong, R. Barik, E. Totoni, H. Liu, C. Markley, A. Fox, and T. Shpeisman. Latte: A language, compiler, and runtime for elegant and efficient deep neural networks. In *PLDI*, 2016.

[49] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet another resource negotiator. In *SOCC*, 2013.

[50] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[51] W. Wang, G. Chen, T. Dinh, J. Gao, B. Ooi, and K. Tan. SINGA: A distributed system for deep learning. Technical report, NUS Tech Report, 2015.

[52] M. N. Wegman and F. K. Zadeck. Constant propagation with conditional branches. *ACM TOPLAS*, 13(2):181–210, Apr. 1991.

[53] D. Yu, A. Eversole, M. Seltzer, K. Yao, O. Kuchaiev, Y. Zhang, F. Seide, Z. Huang, B. Guenter, H. Wang, J. Droppo, G. Zweig, C. Rossbach, J. Gao, A. Stolcke, J. Currey, M. Slaney, G. Chen, A. Agarwal, C. Basoglu, M. Padmilac, A. Kamenev, V. Ivanov, S. Cypher, H. Parthasarathi, B. Mitra, B. Peng, and X. Huang. An introduction to computational networks and the computational network toolkit. Technical report, Microsoft Research, October 2014.

[54] D. Yu and M. L. Seltzer. Improved bottleneck features using pretrained deep neural networks. In *Interspeech*, pages 237–240, 2011.

[55] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *FPGA*, 2015.

[56] M. Zhu, L. Liu, C. Wang, and Y. Xie. CNNLab: a novel parallel framework for neural networks using GPU and FPGA-a practical study with trade-off analysis. *arXiv preprint arXiv:1606.06234*, 2016.