Reproducing Deep Learning

Demystifying Parallel and Distributed Concurrency Analysis

TAL BEN-NUN and TORSTEN HÖFFLER, ET AL.

Deep Neural Networks (DNNs) are becoming an important part of our lives. In this survey, we describe the problem from the perspective of parallelism. Specifically, we present trends in parallelization strategies. We discuss the different asynchronous stochastic gradient descent distributed performance models. Based on these approaches, we present learning.

Additional Key Words and Phrases: Deep Learning, DNN


I. INTRODUCTION

Machine learning, and in particular Deep Learning, taking over a variety of aspects in our daily lives. A Network (DNN), a construct inspired by the interplay between the expressiveness of DNNs provides a network of neural networks (DNNs) provides a network of neural networks. Formal definitions of DNNs vary across different domains, ranging from superficial recognition [Amodei et al., 2016] autonomous driving [Bui et al., 2016] and deep learning in general [see Fig. 1 for more examples].

https://www.arxiv.org/abs/1802.09941


A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning

Tal Ben-Nun, Maciej Basta, Simon Huber, Alexandros Nikolos Zierpitis, Daniel Peter, Torsten Höfler
Department of Computer Science, ETH Zurich

Abstract—We introduce DeepPath: the first customizable benchmarking infrastructure that enables fair comparisons of the plethora of deep learning frameworks, algorithms, hyperparameters, and techniques. The key idea behind DeepPath is to modularly design, implement, and deploy benchmarking algorithms to enable a full-fledged system under the hood. In this paper, we design and implement DeepPath as a benchmarking framework for reproducibility. Since DL is converging in terms of procedures, it is possible to design a white-box abstraction that covers key functionalities of the problem, enabling arbitrary metric measurement and full integration of the different software stacks. We use the following five pillars: Customizability, Metrics, Performance, Validation, and Reproducibility. Customizability indicates that DeepPath enables benchmarking of arbitrary combinations of DL systems, such as various frameworks running on different platforms, and executing custom algorithms. To achieve this, we design DeepPath to be a meta-framework that can be straightforwardly extended to benchmark any DL code. Table I illustrates how various DL frameworks, libraries, and frameworks can be integrated into DeepPath to enable more and faster DL programming.

Metrizes” indicates that DeepPath enables the integration of more DL code that, unlike benchmarks such as TPUv2 [28], makes a single metric such as FLOPS as insufficient measure. To this end, we propose metrics that consider the accuracy-related difficulty and the implementation effort. These metrics include accuracy improvement, time-to-solve, and accuracy-to-time ratio. These metrics are then compared to the accuracy of the DL code that is run in DeepPath.

I. INTRODUCTION

Deep Learning (DL) has transformed the world and is now ubiquitous in areas such as speech recognition, image classification, or autonomous driving [1]. Its core concept is a Deep Neural Network (DNN), a structure modeled after the human brain. Thanks to rigorous training, DNNs are able to solve various problems, previously deemed unsolvable.

Recent years saw exponential growth in the number of approaches, schemes, algorithms, applications, platforms, and frameworks for DL. This diversity adds value to the inference and training. Second, hardware platforms can vary significantly, including CPUs, GPUs, or FPGAs. Third, the problem may be solved in a number of ways, e.g., through [28] or [29]. This flexibility may open up many parallel and distributed optimizations, such as data, model, and pipeline parallelism. Finally, DL workloads are executed in highly varying environments, such as mobile phones, multi-GPU chassis, or large-scale supercomputers.

This richness of the DL domain raises an important question: how can we ensure a level playing field for comparison, competition, and benchmarking in Deep Learning? The key issue is that, due to the complex nature of DL workloads, there is no single metric by which one DNN or one framework can be deemed better than another. To this end, we propose DeepPath as a modular benchmarking framework that enables fair analysis and comparison of diverse DNN frameworks.
What is Deep Learning good for?

- Digit Recognition
- Object Classification
- Image Captioning
- Gameplay AI
- Neural Computers

A very promising area of research!

<table>
<thead>
<tr>
<th>Year</th>
<th>2012</th>
<th>2013</th>
<th>2014</th>
<th>2015</th>
<th>2016</th>
<th>2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs.AI</td>
<td>1,081</td>
<td>1,765</td>
<td>1,022</td>
<td>1,105</td>
<td>1,929</td>
<td>2,790</td>
</tr>
<tr>
<td>cs.CV</td>
<td>577</td>
<td>852</td>
<td>1,349</td>
<td>2,261</td>
<td>3,627</td>
<td>5,693</td>
</tr>
</tbody>
</table>

number of papers per year

23 papers per day!
How does Deep Learning work?

- ImageNet (1k): 180 GB
- ImageNet (22k): A few TB
- Industry: Much larger

- 100-200 layers deep
- ~100M-2B parameters
- 0.1-8 GiB parameter storage

Deep Learning is Supercomputing!

- 10-22k labels
- growing (e.g., face recognition)
- weeks to train
A brief theory of supervised deep learning

labeled samples $x \in X \subset \mathcal{D}$

$f(x) : X \rightarrow Y$

network structure (fixed) weights $w$ (learned)

$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$

$f(x) = f_n \left( f_{n-1}(f_{n-2}(\ldots f_1(x) \ldots)) \right)$

$\ell_{sq}(w, x) = (f(x) - l(x))^2$

$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$

$\ell_{ce}(w, x) = - \sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$

layer-wise weight update

label domain $Y$ true label $l(x)$
**Stochastic Gradient Descent**

Layer storage = \( |w_l| + |f_l(o_{l-1})| + |\nabla w_l| + |\nabla o_l| \)

\[
\begin{align*}
    w^{(t+1)} &= w^{(t)} - \eta \cdot \nabla \ell(w^{(t)}, x) \\
    f_{1}(x) &= \text{convolution 1} \\
    f_{2}(f_{1}(x)) &= \text{convolution 2} \\
    \cdots &= \text{pooling} \\
    f(x) &= \text{fully connected}
\end{align*}
\]

\[
\begin{align*}
    \text{Learning Rate} \\
    \text{Adaptive Learning Rate} \\
    \text{Momentum [Qian 1999]} \\
    \text{Nesterov Momentum [Nesterov 1983]} \\
    \text{AdaGrad [Duchi et al. 2011]} \\
    \text{RMSProp [Hinton 2012]} \\
    \text{Adam [Kingma and Ba 2015]}
\end{align*}
\]

---

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, arXiv Feb 2018
Minibatch Stochastic Gradient Descent (SGD)

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, arXiv Feb 2018
Common Problems in ML Research

- “I have a new optimization algorithm, how do I test it on actual data/models?”
- “We can fully utilize a supercomputer, what are the commonly used DL workloads?”
- “I designed a new ASIC, how do I convince ML researchers to use it?”
- “Mathematical theory states that algorithm X is optimal for max-pooling, how fast is it compared to the others? How accurate is it?”
- “If we use compression method Y, how drastic is the communication reduction?”
Deep Learning Breakdown

- Individual operators
- Network evaluation
- Optimization algorithm
- Distributed training
### Computing convolutional layers

#### Direct

<table>
<thead>
<tr>
<th>Method</th>
<th>Work (W)</th>
<th>Depth (D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>$N \cdot C_{out} \cdot H' \cdot W' \cdot C_{in} \cdot K_y \cdot K_x$</td>
<td>$\left[ \log_2 C_{in} \right] + \left[ \log_2 K_y \right] + \left[ \log_2 K_x \right]$</td>
</tr>
<tr>
<td>im2col</td>
<td>$N \cdot C_{out} \cdot H' \cdot W' \cdot C_{in} \cdot K_y \cdot K_x$</td>
<td>$\left[ \log_2 C_{in} \right] + \left[ \log_2 K_y \right] + \left[ \log_2 K_x \right]$</td>
</tr>
<tr>
<td>FFT</td>
<td>$c \cdot HW \log_2(HW) \cdot (C_{out} \cdot C_{in} + N \cdot C_{in} + N \cdot C_{out}) + HWN \cdot C_{in} \cdot C_{out}$</td>
<td>$2 \left[ \log_2 HW \right] + \left[ \log_2 C_{in} \right]$</td>
</tr>
</tbody>
</table>

#### Indirect

<table>
<thead>
<tr>
<th>Method</th>
<th>Work (W)</th>
<th>Depth (D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winograd (m × m tiles, r × r kernels)</td>
<td>$\alpha(r^2 + \alpha r + 2\alpha^2 + \alpha m + m^2) + C_{out} \cdot C_{in} \cdot P$</td>
<td>$2 \left[ \log_2 r \right] + 4 \left[ \log_2 \alpha \right] + \left[ \log_2 C_{in} \right]$</td>
</tr>
</tbody>
</table>

M. Mathieu et al.: Fast Training of Convolutional Networks through FFTs, ICLR’14
A. Lavin and S. Gray: Fast Algorithms for Convolutional Neural Networks, CVPR’16
Data parallelism

- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Generalization is affected by minibatch size

Model parallelism

- Parameters can be distributed across processors
- Mini-batch has to be copied to all processors
- Backpropagation requires all-to-all communication every layer

U.A. Muller and A. Gunzinger: Neural Net Simulation on Parallel Computers, IEEE Int’l Conf. on Neural Networks 1994
Pipeline parallelism

- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Complicated management in code

G. Blelloch and C.R. Rosenberg: Network Learning on the Connection Machine, IJCAI'87
Hybrid parallelism

- Layers/parameters can be distributed across processors
- Can distribute minibatch
- Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)
  - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

J. Dean et al.: Large scale distributed deep networks, NIPS’12.
T. Ben-Nun, T. Hoeffer: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, arXiv Feb 2018
Hyperparameter and Architecture search

- Meta-optimization of hyper-parameters (momentum) and DNN architecture
  - Using Reinforcement Learning [1] (explore/exploit different configurations)
  - Genetic Algorithms with modified (specialized) mutations [2]
  - Particle Swarm Optimization [3] and other meta-heuristics


---

[3] P. R. Lorenzo et al.: Hyper-parameter Selection in Deep Neural Networks Using Parallel Particle Swarm Optimization, GECCO’17
Communication optimizations

- **Different options how to optimize updates**
  - Send $\nabla w$, receive $w$
  - Send FC factors ($o_{l-1}, o_l$), compute $\nabla w$ on parameter server
    
    *Broadcast factors to not receive full $w*
  - Use lossy compression when sending, accumulate error locally!

- **Quantization**
  - Quantize weight updates and potentially weights
  - Main trick is stochastic rounding [1] – expectation is more accurate

    *Enables low precision (half, quarter) to become standard*
  - TernGrad - ternary weights [2], 1-bit SGD [3], …

- **Sparsification**
  - Do not send small weight updates or only send top-k [4]

    *Accumulate them locally*

[3] F. Seide et al. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs, In Interspeech 2014

source: ai.intel.com
## Existing Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Operators</th>
<th>Networks</th>
<th>Training</th>
<th>Dist. Training</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sta</td>
<td>Cus</td>
<td>Def</td>
<td>Eaq</td>
</tr>
<tr>
<td>(L) cuDNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(L) MKL-DNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) TensorFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) Caffe, Caffe2 [20]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) [Py]Torch [10, 34]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) MXNet [6]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) CNTK [45]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) Theano [4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) Chainer [43]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) Darknet [37]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) DL4J [42]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) DSSTNE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) PaddlePaddle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F) TVM [7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E) Keras [8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E) Horovod [41]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E) TensorFlow [14]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E) Lasagne</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E) TFlearn [11]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE I:** An overview of DL frameworks, related systems that can be integrated within Deep500, and the advantages of such integration. Each column is a specific feature/functionality; they are explained in more detail in Background (§ II). **Sta:** Standard Operators, **Cus:** Customizable (without full recompilation), **Def:** Deferred Execution Mode, **Eaq:** Eager Execution Mode (also called “define-by-run”), **Com:** Network Compilation, **Tra:** Transformable, **Dat:** Dataset Network Integration, **Opt:** Standard Optimizers, **PS:** Parameter Server, **Dec:** Decentralized, **Asy:** Asynchronous SGD, **UR:** Update Rule Optimizers, □: A given system does not offer a given feature, □: A given system offers a given feature in a limited way, □□: A given system does not offer a given feature, (L): a library, (F): a framework, (E): a frontend. Native system support for a category of features: none, partial, full.
## Existing Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Focus</th>
<th>Metrics</th>
<th>Criteria</th>
<th>Customizability</th>
<th>DL Workloads</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Perf</td>
<td>Con</td>
<td>Acc</td>
<td>Tim</td>
<td>Cos</td>
</tr>
<tr>
<td>DeepBench [39]</td>
<td></td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBD [47]</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fathom [2]</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAWN Benchmark</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Kaggle [21]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>ImageNet [13]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLP [30]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### TABLE II: An overview of available DL benchmarks, focusing on the offered functionalities.
- **Perf:** Performance, **Con:** Convergence, **Acc:** Accuracy, **Tim:** Time, **Cos:** Cost, **Ene:** Energy, **Util:** Utilization, **Mem:** Memory Footprint, **Tput:** Throughput (Samples per Second), **Brk:** Timing Breakdown, **Sca:** Strong Scaling, **Com:** Communication and Load Balancing, **TTA:** Time to Accuracy, **FTA:** Final Test Accuracy, **Lat:** Latency (Inference), **Clo:** Closed (Fixed) Model Contests, **Ope:** Open Model Contests, **Inf:** Fixed Infrastructure for Benchmarking, **Ops:** Operator Benchmarks, **Img:** Image Processing, **Obj:** Object Detection and Localization, **Spe:** Speech Recognition, **Txt:** Text Processing and Machine Translation, **RL:** Reinforcement Learning Problems. 
- ✔: A given benchmark does offer the feature.
- ✗: A given benchmark does not offer the feature.

- Very structured (e.g., open vs. closed competitions)
- Includes both convergence and performance tests
- Seems to be heavily backed by industry
- Unclear: requirements from a potential competitor, how results are presented, reproducibility guarantees, hardware differences, HPC/distributed aspects
Deep500

- Deep learning **meta-framework**: a framework for frameworks to reside in

**Metrics**
## Deep500 Component Overview

### Level 0: Operators
- **Python Operators**
- **Custom Operators**
- **ONNX Test Parser**
- **CMake Interface**
- **Python, C++ References for LeNet, AlexNet, ResNet**
- **Operator Validation**
- **Gradient Checker**

### Level 1: Network Processing
- **Network**
- **ONNX Parser + Visitor**
- **Graph Executor**
- **Reference Graph Executor**
- **TensorFlow, Pytorch, Caffe2**
- **ONNX Models (LeNet, ResNet)**
- **Reference Optimizers (SGD, Momentum, Adagrad, Adam)**
- **L1 Framework Integration**
- **Training Accuracy**
- **Inference Validation**
- **Overhead Analysis**

### Level 2: Training
- **Dataset**
- **Optimizer**
- **WUR, 3-Step Opt.**
- **Reference Optimizers (SGD, Momentum, Adagrad, Adam)**
- **L1 Framework Integration**
- **Training Accuracy**
- **Performance, Energy Metrics**

### Level 3: Distributed Training
- **Distributed Dataset**
- **Distributed Optimizer**
- **Model Consistency**
- **MPI Reference Optimizer (Decentralized, Param. Server)**
- **Horovod, Distributed Caffe2**
- **Communication Volume**
- **Distributed Training Accuracy**
- **Centralization**
- **Model Consistency**
- **Distributed Optimizer**
- **MPI Reference Optimizer (Decentralized, Param. Server)**
- **Horovod, Distributed Caffe2**
- **Communication Volume**
- **Distributed Training Accuracy**

---

### Level 0: Operators
- **Python Operators**
- **Custom Operators**
- **ONNX Test Parser**
- **CMake Interface**
- **Python, C++ References for LeNet, AlexNet, ResNet**
- **Operator Validation**
- **Gradient Checker**

---

### Level 1: Network Processing
- **Network**
- **ONNX Parser + Visitor**
- **Graph Executor**
- **Reference Graph Executor**
- **TensorFlow, Pytorch, Caffe2**
- **ONNX Models (LeNet, ResNet)**
- **Reference Optimizers (SGD, Momentum, Adagrad, Adam)**
- **L1 Framework Integration**
- **Training Accuracy**
- **Inference Validation**
- **Overhead Analysis**

---

### Level 2: Training
- **Dataset**
- **Optimizer**
- **WUR, 3-Step Opt.**
- **Reference Optimizers (SGD, Momentum, Adagrad, Adam)**
- **L1 Framework Integration**
- **Training Accuracy**
- **Performance, Energy Metrics**

---

### Level 3: Distributed Training
- **Distributed Dataset**
- **Distributed Optimizer**
- **Model Consistency**
- **MPI Reference Optimizer (Decentralized, Param. Server)**
- **Horovod, Distributed Caffe2**
- **Communication Volume**
- **Distributed Training Accuracy**
- **Centralization**
- **Model Consistency**
- **Distributed Optimizer**
- **MPI Reference Optimizer (Decentralized, Param. Server)**
- **Horovod, Distributed Caffe2**
- **Communication Volume**
- **Distributed Training Accuracy**
# Existing Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Operators</th>
<th>Networks</th>
<th>Training</th>
<th>Dist. Training</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sta</td>
<td>Cus</td>
<td>Def</td>
<td>Eag</td>
</tr>
<tr>
<td>cuDNN (L)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>MKL-DNN (L)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>TensorFlow (F)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Caffe, Caffe2 (F)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>[PyTorch](10, 34) (F)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>MXNet (F) [6]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CNTK (F) [45]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Theano (F) [4]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Chainer [MN] [43] (F)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Darknet [37] (F)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>DL4J [42] (F)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>DSSTNE (F)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>PaddlePaddle (F)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>TVM (F) [7]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Keras (E) [8]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Horovod [41] (E)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>TensorFlowLayer [14] (E)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Lasagne (E)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>TF Learn [11] (E)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Use-Case 1: New Operator

```python
class IPowOp(CustomPythonOp):
    def __init__(self, power):
        super(IPowOp, self).__init__()
        self.power = power
        assert int(power) == power  # integral

    def forward(self, inputs):
        return inputs[0] ** self.power

    def backward(self, grads, fwd_inputs, fwd_outputs):
        return (grads[0] * self.power *
                (fwd_inputs[0] ** (self.power - 1)))
```

```cpp
template<
    typename T>

class ipowop : public deep500::CustomOperator {
  protected:
    int m_len;
  public:
    ipowop(int len) : m_len(len) {}
    virtual ~ipowop() {}

    void forward(const T *input, T *output) {
      #pragma omp parallel for
      for (int i = 0; i < m_len; ++i)
        output[i] = std::pow(input[i], DPOWER);
    }

    void backward(const T *nextop_grad,
                  const T *fwd_input_tensor,
                  const T *fwd_output_tensor,
                  T *input_tensor_grad) {
      #pragma omp parallel for
      for (int i = 0; i < m_len; ++i) {
        input_tensor_grad[i] = nextop_grad[i] * DPOWER *
                                std::pow(fwd_input_tensor[i], DPOWER - 1);
      }
    }

};
```
Use-Case 2: Intel FPGA DNN Executor

- Use ONNX visitor to construct OpenCL kernels per operator and compile
- Network executor invokes FPGA through .so file
Use-Case 3: New Optimization Algorithm