Benchmarking Deep Learning Frameworks with FPGA-suitable Models on a Traffic Sign Dataset

Lin, Zhongyi
Ota, Jeffrey M.
Owens, John D.
et al.

2018-06-26

Benchmarking Deep Learning Frameworks with FPGA-suitable Models on a Traffic Sign Dataset

Zhongyi Lin*[1], Jeffrey M. Ota[2], John D. Owens[1], and Pınar Muyan-Özçelik*[3]

Abstract— **We benchmark several widely used deep-learning frameworks for performing deep-learning-related automotive tasks (e.g., traffic sign recognition) that need to achieve real-time and high accuracy results with limited resources available on embedded platforms such as FPGAs. In our benchmarks, we use various input image sizes on models that are suitable for FPGA deployment, and investigate the training speed and inference accuracy of selected frameworks for these different sizes on a popular traffic sign recognition dataset. We report results by running the frameworks solely on the CPU as well as by turning on GPU acceleration. We also provide optimizations we apply to fine-tune the performance of the frameworks. We discover that Neon and MXNet deliver the best training speed and inference accuracy in general for all our test cases, while Tensorflow is always among the frameworks with the highest inference accuracies. We also observe that on the particular dataset we tested on (i.e., GTSRB), the image size of the region of interest does not necessarily affect the inference accuracy, and that using deep models, e.g., ResNet-32, which have longer training times, might not provide improvements to inference accuracy.**

I. Introduction

Deep learning is the driving power behind many automotive computing tasks involved in driverless cars that are on our horizon. This technique involves deep Convolutional Neural Network (CNN) models that are usually trained offline with powerful computers then transferred to an optimized embedded system available in cars for real-time inference. Since embedded systems have limited resources and automotive tasks require real-time results with high accuracy to protect the driver's and/or passengers' safety, we need deep-learning-based systems that can achieve real-time high-accuracy inference with limited resources. As shown by Muyan-Özçelik et al. [9], we can exploit the inherent parallelism in automotive tasks to make efficient use of the limited computation resources of embedded systems in order to achieve real-time results with high accuracy. Since FPGAs provide high parallelism and power efficiency, they may be a preferred embedded platform for performing autonomous driving tasks, such as real-time traffic sign recognition with a pre-trained model. Several different popular deep-learning frameworks are suitable to train such models for FPGAs. In order to choose the most suitable framework, we investigate the tradeoffs between the selected frameworks and benchmark them for performing a traffic-sign-recognition task using models that are suitable for performing real-time and high-accuracy inference on the limited resources of FPGAs.

CNNs have been used in traffic sign recognition since the start of this decade [2]. One restriction of CNNs is that the input image size must be fixed after the network is defined and trained. However, in real applications, the detection phase of the traffic sign recognition task performed prior to the classification phase usually proposes regions of interest (ROIs) with different sizes. It is critical to explore the size that ROIs are resized to, which also determines the input image size of the CNN due to three reasons: 1) Inferencing a big image might not be affordable on FPGAs that have limited resources; 2) The low level implementation and optimization of computation of a framework is usually unclear to users. For instance, some specific sizes might be optimized on some frameworks (e.g., Neon's GPU kernels are optimized for sizes being multiples of 4); 3) An image size being too big or too small might affect the inference accuracy. In addition, we would also like to minimize the time we spend on training the model. Therefore, we benchmark the performance, i.e., training speed and inference accuracy, of several popular deep learning frameworks on a popular traffic sign recognition dataset by primarily focusing on different image sizes. We collect results by running the frameworks solely on the CPU and with GPU acceleration. Although currently using GPUs for training CNNs is the preferred method, we also report CPU results since training with novel CPU backends (e.g., an Intel Xeon Phi coprocessor) may also be preferred in the future if they deliver more power-efficient high-performance computation than GPUs.

Prior to our research, Chu et al. [16] comprehensively evaluated five mainstream deep learning frameworks over a series of metrics on MNIST and CIFAR datasets. For our benchmarks, we worked with a similar set of frameworks including CNTK [8], MXNet [1], Neon [12], PyTorch [15], and Tensorflow [3]. However, since we focus on traffic sign recognition, we have used a widely-used dataset, the German Traffic Sign Recognition Benchmark (GTSRB) [17] dataset that contains traffic sign images. In addition, we have utilized three models in our study. These are the model designed by Ciresan et al. [2] (referred to by the name of the institution, IDSIA, in the rest of the paper) and two Deep Residual Neural Networks designed by He et al. [4], ResNet-20 and

*Corresponding authors

[1] Zhongyi Lin and John D. Owens are with the Department of Electrical and Computer Engineering, University of California, Davis, CA 95616 {zhylin, jowens}@ucdavis.edu

[2] Jeffrey M. Ota is with the Autonomous Driving and Sports Research Group, Intel Labs, Santa Clara, CA 95054 jeffrey.m.ota@intel.com

[3] Pınar Muyan-Özçelik is with the Department of Computer Science, California State University, Sacramento, CA 95819 pmuyan@csus.edu

- The source code of this benchmark can be accessed from https://github.com/owensgroup/TrafficSignBench, which is under construction.

ResNet-32. Given that real-time traffic sign recognition on autonomous cars needs to achieve high accuracy results with limited resources, these three models are considered to be suitable for real-time inference on the FPGA due to their small sizes and high inference accuracy.

Hence, the contributions of our research include a benchmark of five popular frameworks over three CNN models that are suitable for FPGA inference with three different input sizes, a combination that was not investigated in previous studies. We have specifically investigated the effect of using different image sizes on training time and accuracy since input size is important for utilizing the selected models for the traffic sign recognition task that involves different ROI sizes. In addition to providing in-depth discussion of these benchmarking results, we also provide optimization techniques that are used for each framework that can be utilized by future studies that target these frameworks. We believe our discussions and optimization tips can guide engineers and researchers in choosing and fine-tuning the most suitable frameworks in terms of runtime and accuracy for training CNN models for traffic sign recognition on FPGAs.

The rest of the paper starts with a section of background information, introducing the frameworks, dataset, and models we use in the experiments. Following that, we explain our methodology, including image preprocessing, implementation of ResNets, and specifications and optimizations in Section III. Results and analysis are presented in Section IV, while in the last two sections we present potential future work and conclusions.

II. Background

A. Frameworks

The frameworks we use include CNTK, MXNet, Neon, PyTorch and Tensorflow, which are all currently popular and actively used in both industry and academia.

[**CNTK**] The Microsoft Cognitive Toolkit (CNTK) is a unified deep-learning toolkit that allows constructing popular model types like CNNs, feed-forward Deep Neural Networks (DNNs), Recurrent Neural Networks (RNNs), and Long Short Term Memory networks (LSTMs) via directed computational graphs. It is considered a framework with top-tier performance for training neural networks across multiple CPUs/GPUs.

[**MXNet**] Amazon-developed Apache MXNet is a deep learning framework with high portability among multiple programming languages. With MXNet, one can mix symbolic and imperative programming to maximize efficiency and productivity, since both types of execution are parallelized and optimized. MXNet also claims to have great scalability across multiple machines.

[**Neon**] Intel Nervana's reference neural network framework, Neon, includes optimizations for various hardware but specifically for Intel CPUs. It is tightly integrated with the Intel MKL library for machine learning as well as the latest GPU kernel libraries. Neon claims to have the fastest performance among deep learning libraries for fast iteration and model exploration.

[**PyTorch**] Developed by Facebook AI, PyTorch is a deep learning Python package that provides both GPU-accelerated tensor computation and deep neural network implementation on a tape-based autograd system that remembers all operations it executed in the forward phase and repeats them in the backward phase for automatic differentiation. Unlike many other frameworks such as Tensorflow that rely on static computational graphs, PyTorch adopts dynamic computational graphs, which are valuable when computation cannot be determined and memory must be dynamically allocated to carry out tasks such as the ones involved in natural language processing (NLP).

[**Tensorflow**] Introduced as Google Brain's tool for machine learning and deep neural network research, Tensorflow soon became a widely used deep learning framework and currently owns one of the largest communities among deep learning frameworks. It is known for its flexible architecture and easy deployment on various hardware.

In our research, we use the native APIs of CNTK (v2.5.1), Neon (v2.6.0), MXNet (v1.1.0), PyTorch (v0.4.0), and Tensorflow (v1.7).

B. Dataset

The GTSRB dataset is a popular multi-class traffic sign dataset with detailed annotation for classification and/or detection purpose. It originates from the International Joint Conference on Neural Networks (IJCNN) 2011 and is maintained by Ruhr University of Bochum. It contains more than 50000 images in total comprising 43 different classes of traffic signs. Images are in ppm format and have sizes varying from 15×15 to 250×250. Each image contains a traffic sign that belongs to one of the 43 classes. A tightly bounded ROI of each image is annotated in the dataset for further cropping. GTSRB provides RGB images as well as Haar, HueHist, and HOG representations of images.

C. Models

The models we train on include the CNN model designed by Ciresan et al. [2] (IDSIA), which is memory-friendly on FPGAs and achieves greater than 99% classification accuracy with proper preprocessing (ranked first in the IJCNN competition in 2011). Ciresan et al. showed that with appropriate preprocessing, CNNs can achieve high accuracy even with only a small number of layers. The structure of the IDSIA model is shown in Table I:

We also investigate models that are developed more recently. He et al. [4] introduced ResNets (ranked first in the ImageNet competition in 2015) to address the vanishing/exploding gradient problem, which liberated CNNs from the limit of depth. In addition to having high accuracy, ResNets are also considered FPGA-friendly for three reasons: 1) Different from other models that originated from ImageNet competitions, their input sizes are flexible. This

Layer	Type	# maps & neurons	Kernel
0	input	3 maps of 48x48 neurons	
1	convolutional	100 maps of 46x46 neurons	3x3
2	max pooling	100 maps of 23x23 neurons	2x2
3	convolutional	150 maps of 20x20 neurons	4x4
4	max pooling	150 maps of 10x10 neurons	2x2
5	convolutional	250 maps of 8x8 neurons	3x3
6	max pooling	250 maps of 4x4 neurons	2x2
7	fully connected	200 neurons	
8	fully connected	43 neurons	

TABLE I: Structure of the IDSIA model

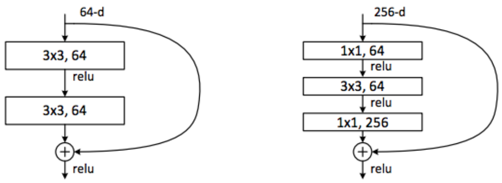

Fig. 1: ResNet's basic/bottleneck building blocks

means ResNets can be used in many different applications; 2) ResNets consist of repetitive basic/bottleneck building blocks depicted in Table I, which makes them suitable to be deployed on FPGAs; 3) The number of parameters of a ResNet model is comparatively small. For instance, ResNet-32 with input size 32 by 32 has only 0.46 million parameters, while AlexNet and VGG, both of which are used in the acceleration of CNNs on FPGAs by researchers like Suda et al. [18] and Wang et al. [19], have 60 million and 138 million parameters respectively.

III. Methodology

Our experiments are completed on a Ubuntu 16.04 LTS machine with an Intel i7-7700K 4.2 GHz CPU and an NVIDIA GTX 1050 Ti GPU with 16 GB memory. CUDA (v9.0) and MKLML (v20171227) are used as packages to support GPU and CPU acceleration. The experiment code is programmed with Python 3.5 and libraries including MKL (v2018.0.0), PyCuda (v2017.1.1), and PyGPU (v0.7.5).

A. Image Preprocessing

Generally, datasets for traffic sign recognition have small but varying image size, and a medium number of classes, i.e., fewer than 100. These images usually suffer from various illumination conditions that require some preprocessing. For instance, histogram equalization can be used as a preprocessing step to improve the image quality. Hence, we perform image preprocessing on the dataset images before they are used to construct a data source for the CNN model training. The original images are first cropped according to the annotated bounding boxes, then resized to 32 by 32, 64 by 64 (only for ResNets), and 48 by 48 (for IDSIA and ResNets). These images are further processed with contrast limited adaptive histogram equalization (CLAHE) [20]. We have decided to apply CLAHE, since it resulted in a minimum error rate among four preprocessing techniques used by Ciresan et al. We did not conduct more complicated preprocessing on the current color channels (BGR) or other channels since while performing real-time inference, extra preprocessing will add an additional burden to FPGA's performance.

B. Construction of ResNets for Experiments

We follow He et al.'s way of constructing ResNets for the CIFAR-10 dataset to construct our ResNet models for different input sizes. The total number of layers of a ResNet model, l, can be specified as $l = 6n + 2$, where n is conventionally an odd number starting from 3. In our experiments, we pick $n = 3$ and $n = 5$ to build two models, ResNet-20 and ResNet-32. Since the side length of the feature maps will be halved twice in a ResNet model, which requires the input size to be a multiple of 4, it is appropriate for us to use 32, 48, and 64 as sizes of the input images after resizing. The implementations of ResNets are ported from the official model "zoo" of each framework. We make necessary changes to the code to guarantee images of different sizes can be fed to each model correctly, and the global average pooling layer at the end works properly.

C. Specifications and Optimizations

In our experiments, we use a Stochastic Gradient Descent (SGD) learner with 0.01 learning rate and 0.9 momentum as the optimizer, and set batch size to 64 and epoch number to 25. We also use BGR channel order and image size $(3, n, n)$, where n = 32, 48, and 64. Weights of convolutional layers are initialized with the *he_normal* (also named the *kaiming_normal*) initializer [5], which claims to have better convergence.

To obtain the best performance for each framework, especially on the CPU, we optimize them individually by building them from source or installing optimized Python wheels. Usually doing these optimizations will improve the performance by adding MKL support or taking advantages of the local machine (e.g., utilizing multiple CPUs and/or advanced instruction sets like SSE and AVX). We test each framework with a single epoch (to reduce the testing time) with the IDSIA model on the GTSRB dataset with training set size 31367. The input size we use in these tests is 48 by 48 and the batch size is 64.

We apply system optimizations to improve performance of the frameworks when they solely run on the CPU as well as when GPU acceleration was turned on. We also perform code optimizations to enhance the speed and accuracy of these frameworks. All these optimizations are explained in the following sub-sections.

1) System optimizations to improve the CPU performance: We improve the speed of MXNet running on the CPU from **219.9 s/epoch** to **114.4 s/epoch** by setting *USE_MKL=1* and *USE_MKL_EXPERIMENTAL=1* before building, or alternatively, installing the mxnet-cu90mkl wheel. We boost the CPU performance of Neon from **198.8 s/epoch** to **141.5 s/epoch** by setting *OMP_NUM_THREADS* to the number of physical cores of the local machine (i.e., 4 in this

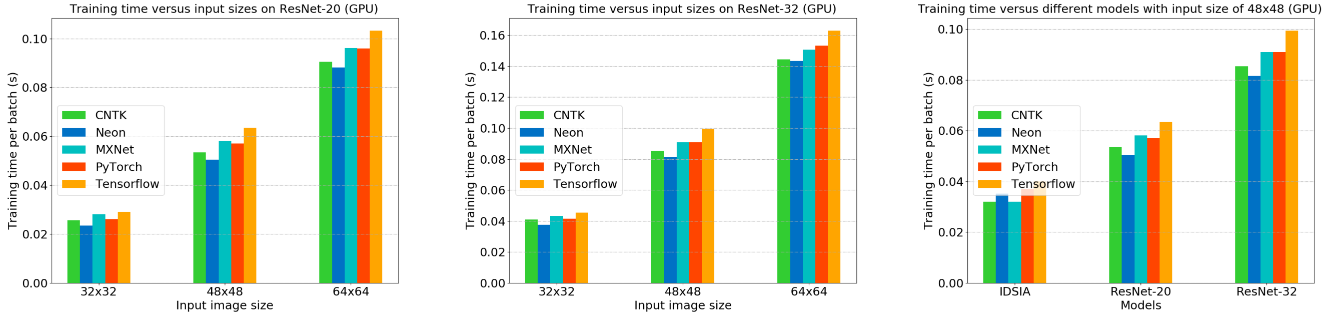

Fig. 2: GPU average batch training time versus (a) input sizes, on ResNet-20, (b) input sizes, on ResNet-32 and (c) all models, with the same input size 48x48.

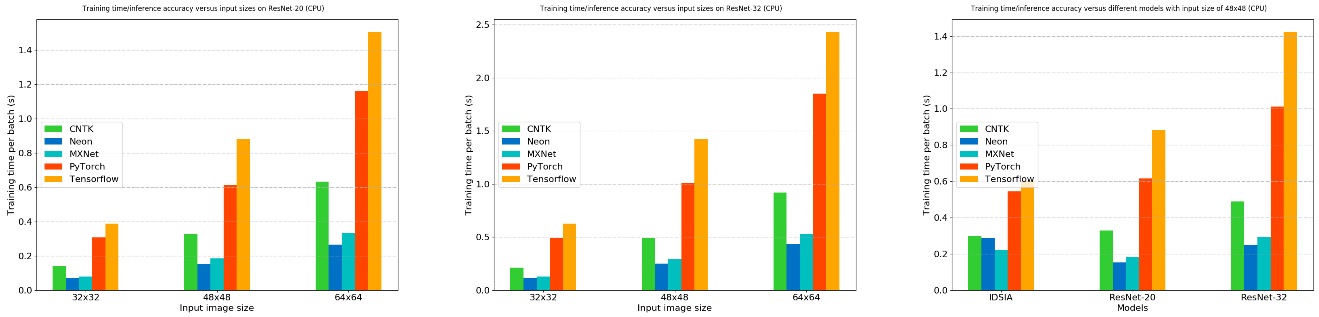

Fig. 3: CPU average batch training time versus (a) input sizes, on ResNet-20, (b) input sizes, on ResNet-32, and (c) all models, with the same input size 48x48.

Framework	GPU				CPU			
	ResNet-20		ResNet-32		ResNet-20		ResNet-32	
	t_{train}	acc	t_{train}	acc	t_{train}	acc	t_{train}	acc
CNTK	313.80	96.93%	502.60	97.24%	1731.30	96.63%	2582.35	97.12%
Neon	**288.65**	**97.68%**	**461.50**	**97.72%**	**888.14**	97.54%	**1434.08**	97.78%
MXNet	344.82	96.61%	531.36	96.26%	983.94	97.30%	1560.08	96.40%
PyTorch	319.53	94.00%	509.17	97.00%	3372.83	96.00%	5420.62	97.00%
Tensorflow	357.01	97.32%	558.48	97.60%	3964.77	**97.63%**	6260.26	**97.85%**

TABLE II: Training time (in seconds) and inference accuracy for input size 32 by 32.

Framework	GPU				CPU			
	ResNet-20		ResNet-32		ResNet-20		ResNet-32	
	t_{train}	acc	t_{train}	acc	t_{train}	acc	t_{train}	acc
CNTK	1109.80	96.49%	1771.54	96.79%	7763.12	96.86%	11297.63	96.90%
Neon	**1081.47**	97.35%	**1760.56**	97.76%	**3263.59**	96.80%	**5309.61**	**97.40%**
MXNet	1179.94	96.73%	1849.95	97.34%	4084.95	97.48%	6494.11	97.35%
PyTorch	1177.42	95.00%	1877.46	96.00%	12944.11	96.00%	21551.92	97.00%
Tensorflow	1267.75	**98.00%**	2001.53	**98.34%**	12747.86	**97.96%**	19912.17	96.40%

TABLE III: Training time (in seconds) and inference accuracy for input size 64 by 64.

Framework	GPU						CPU					
	IDSIA		ResNet-20		ResNet-32		IDSIA		ResNet-20		ResNet-32	
	t_{train}	acc	t_{train}	acc	t_{train}	acc	t_{train}	acc	t_{train}	acc	t_{train}	acc
CNTK	393.4	**96.78%**	655.72	96.37%	1047.60	96.64%	3657.14	96.45%	4041.97	96.63%	5994.42	96.77%
Neon	432.2	95.87%	**618.46**	97.35%	**1001.42**	**98.27%**	3557.70	96.07%	**1877.23**	97.49%	**3057.35**	**98.02%**
MXNet	**392.7**	96.55%	712.59	96.80%	1116.57	97.32%	**2711.96**	**96.76%**	2268.31	97.36%	3605.97	97.14%
PyTorch	455.5	96.00%	697.94	97.00%	1114.56	96.00%	6029.23	96.00%	7037.74	97.00%	11368.58	97.00%
Tensorflow	491.92	96.37%	778.71	**97.36%**	1220.48	97.26%	6355.15	96.29%	8086.93	**97.77%**	12268.18	97.49%

TABLE IV: Training time (in seconds) and inference accuracy for input size 48 by 48.

case), and *KMP_AFFINITY* to *compact,1,0,granularity=fine* as Intel Nervana suggests [10]. We improve Tensorflow's CPU training speed from **298.6 s/epoch** to **167.8 s/epoch** after installing a Python wheel that is prepared by TinyMind and optimized with SSE4.1, SSE4.2, AVX, AVX2, FMA, and MKL support available on the local machine. Notice that *OMP_NUM_THREADS* and *KMP_AFFINITY* should also be set with the same values we set for Neon to achieve this speedup. Also note that this wheel may not contain the latest Intel optimizations to Tensorflow. Since CNTK and PyTorch all come with MKL support by default, there are no such optimizations to be done on them.

2) System optimizations to improve the GPU performance: Since currently almost all the mainstream frameworks rely on the cuDNN library for CUDA-based GPU computation, as long as cuDNN and CUDA are correctly installed and configured, building frameworks from source will not affect their performance on the GPU. One exception is PyTorch. We boost its GPU performance from **22.4 s/epoch** to **21.1 s/epoch** after adding LAPACK support by installing the magma-cuda90 library.

Notice that the optimization techniques mentioned above might affect the training convergence rate. However, since the number of epochs we use (e.g., 25) guarantees the models to be overfitted in our tests, it is unnecessary to consider the effect of different convergence rates on inference accuracy.

3) Code optimizations to improve speed and accuracy: Code optimization can also be done to improve speed and accuracy. For CNTK, before being passed to minibatch source constructor, numpy arrays of data are converted to contiguous arrays if they were not in order to improve computation efficiency. Since delicate time measurement cannot be done merely with the callbacks passed to the common *training_session* function, we create a training loop for more flexible time measurement. This might sacrifice some of CNTK's speed advantage (especially on the GPU). In fact, it is suggested that rather than creating a customer training loop for each epoch, the *training_session* function should be used for a higher training speed. Another difference of CNTK is that it sums the gradient of each minibatch for parameter updates instead of averaging the minibatches like other frameworks such as Tensorflow. CNTK also differs from other frameworks in the sense that it normalizes the gain for SGD learner with momentum. These two features of CNTK usually decrease the inference accuracy of CNTK compared to other frameworks. To resolve this problem we set *unit_gain* to *False* and *use_mean_gradient* to *True* for the SGD learner, and bring CNTK back to the same evaluation standard with other frameworks. We show the related results in Section IV.

We also explore code optimization opportunities on PyTorch. One optional parameter of the *Dataloader* function, *num_workers*, can be set to use multiple threads for data loading. We try 0 (default), 4, and 16, but find that the training speed per batch is decreased from **21.1 s/epoch** to **22.3 s/epoch** and **23.4 s/epoch**, which contradicts our expectation. One possible reason is that compared to huge datasets like ImageNet, the datasets we use has a much smaller size, for which the overhead of multithreading dominates the code running time. As a result, we use the default value 0 for our experiments.

IV. Results and Analysis

In Fig. 2, we present plots that provide the batch training time on the GPU vs. the following variables: two ResNets with different input sizes and all models with a fixed input size 48 by 48. In Fig. 3, we show the CPU counterparts of the same plots. We also present the total training time and inference accuracy of all frameworks, models, and input sizes on the GPU and CPU in Tables II, III, and IV, where the best results are shown in bold.

A. Training Speed on the GPU

Training speed results on the GPU show that Neon is the fastest in most cases. There is only one exception: MXNet ranks first on IDSIA when the input size is 48 by 48. In general, the runtime of all frameworks are fairly close to each other. In most of the test cases, the training speed of the frameworks has the following descending order: Neon, CNTK, PyTorch, MXNet, and Tensorflow.

B. Training Speed on the CPU

Training speed results on the CPU show that Neon and MXNet have the fastest speed. Similar to the GPU results, Neon ranks first in most of the cases, except for the IDSIA case, where it is surpassed by MXNet. An interesting discovery is that the CPU runtimes of MXNet and Neon training for IDSIA are longer than that of for ResNet-20, which is deeper than IDSIA. The reason could be that the massive computation at the two final fully-connected layers may not be optimized for CPUs as effectively as for GPUs on these two frameworks.

Compared with the GPU results where CNTK has a runtime that is very close to Neon and MXNet, on the CPU, CNTK does not scale as well as Neon and MXNet do when the model becomes deeper and the input size becomes larger. Similar to CNTK, PyTorch's and Tensorflow's runtime on the CPU is not as close to Neon and MXNet as it is on the GPU. In general except for the IDSIA case, the training speed of the frameworks has the following descending order: Neon, MXNet, CNTK, PyTorch, and Tensorflow.

C. Inference Accuracy on the GPU

In terms of inference accuracy, Neon and Tensorflow provide the best performance. Over the seven cases we test, Neon and Tensorflow rank first in three cases each, while CNTK ranks first in the remaining case. However, the gap between their inference accuracies and that of other frameworks is not very big. For all frameworks, the two ResNet models have a clear advantage against the IDSIA model as expected. However, one ResNet model does not dominate over the other. Among all the three input sizes we tested, although we do not observe any of these sizes resulting in a dominating inference accuracy, the best performance on average was reported with the 64 by 64 input size on ResNet-32.

D. Inference Accuracy on the CPU

Similar to the GPU results, Neon and Tensorflow still have the best inference accuracy on the CPU. One or the other ranks first in all cases, except for the IDSIA case where MXNet ranks first. Again, the two ResNet models outperform the IDSIA model as expected, while their accuracies are almost the same. In addition, just like on the GPU, no input size results in a dominating inference accuracy.

E. Summary of Findings

Based on these results, we see that among the five frameworks Neon has the fastest average training speed. Its inference accuracy is also one of the best, along with Tensorflow, which has slower training speed on both the CPU and GPU. MXNet also provides outstanding performance on the two metrics, which is comparable with that of Neon. CNTK, PyTorch, and MXNet all have training speeds very close to that of Neon and inference accuracy close to Tensorflow on the GPU. However, CNTK, PyTorch, and Tensorflow suffer from a training speed degradation on the CPU. Overall, in terms of both training speed and inference accuracy, we consider Neon and MXNet to be the most suitable framework for training CNNs on both GPUs and CPUs.

We observe that on both the CPU and GPU, while the two ResNet models achieve higher accuracies than the IDSIA model as expected, ResNet-32 does not dominate over ResNet-20 and instead, they both have similar accuracies. This could be explained by overfitting, as ResNet-32 might be unnecessarily big for our dataset. Thus, for datasets similar to GTSRB, we do not recommend using models deeper than ResNet-20 since they would require longer training time and in return might not bring any improvements to inference accuracy. We also observe that changing the input size does not necessarily affect the inference accuracy nor the scalability of the training runtime of ResNet-20 and ResNet-32 on the GTSRB dataset. Hence, we expect that reshaping the ROIs to a larger size might not improve the inference accuracy on FPGAs either. In our experiments, we observe that 32 by 32 is a computationally economical input size for training traffic sign dataset without compromising inference accuracy.

In addition, for each framework, when the input size, model architecture and other settings are fixed, we notice that the inference accuracy on the CPU and the GPU are different. This is because the CPU training process is generally deterministic (by fixing a random seed) and the GPU training process is generally not (at least in part because of scheduling thread blocks to processors is non-deterministic). By fixing the random seed in CPU training, we can reproduce training results on the CPU, but cannot do the same on the GPU, and thus we cannot generate identical pre-trained models across devices. In fact, in our experiments, no framework can reproduce training results on the GPU. CNTK is the only framework that can reproduce the training result of the initial 20–30 batches by forcing deterministic algorithms, but the training loss deviates after this point. Forcing deterministic algorithms also reduces training speed: we run the one-epoch test introduced previously and find that this setting slows down CNTK's training speed from **16.6 s/epoch** to **29.4 s/epoch** on the GPU.

Finally, given the fact some of the frameworks have a longer CPU runtime while training IDSIA than training ResNet-20, possibly due to the non-optimized feature map sizes and massive computation at fully-connected layers, we expect CNNs that are not tested in this study and that have these two features might also go through the same bottlenecks, especially when training on the CPU using these frameworks.

V. Future Work

Our ultimate goal is to deploy the models we train onto the FPGA using OpenCL and perform real-time inference on the FPGA. There are several OpenCL frameworks for DNN deployment on FPGAs. Among these available frameworks, we are using PipeCNN created by Wang et al. [19], which is the only open-source one. Our next step is to deploy pre-trained models to the FPGA using PipeCNN and investigate the real-time inference performance of the FPGA for different models and input image sizes. Although currently we only focus on classification, in the future we would like to perform detection and classification of objects in one pass. We plan to investigate alternative solutions for realizing this extension (e.g., YOLOv2 [14], SSD [7] and MobileNets [6]) and deploy the best alternative for performing real-time detection and classification on the FPGA. Because of the high inference accuracy, the two ResNet models could be used as the base network of SSD in the future. For traffic sign detection and classification purpose, the neural network could be trained on GTSDB dataset [13], the "detection" counterpart of GTSRB.

Training on larger traffic sign datasets and benchmarking each framework's data loading speed is another future research direction we plan to explore. The GTSRB dataset we train on has a relatively small size, which causes no problems for data loading. Large datasets like ImageNet challenge the framework for a better data loading solution. Fortunately, many frameworks have developed data loaders for large raw datasets. For instance, Intel Nervana's Aeon loader [11] has support on reading images, videos and audios, and performing specific preprocessing like cropping, deformations, and shuffling before feeding the data to computation on deep learning frameworks including Neon. Hence, we plan to explore the Aeon loader's capabilities for larger traffic sign datasets that we plan to utilize in the future.

VI. Conclusions

In our research, we benchmark training speed and inference accuracy of CNTK, MXNet, Neon, PyTorch, and Tensorflow on three models that are suitable for achieving real-time and high accuracy inference on the limited resources of the FPGA, namely IDSIA, ResNet-20, and ResNet-32, with different input sizes. We present optimization techniques that we have used for each framework and provide in-depth discussion of CPU and GPU results. In our experiments, we investigate the effect of using different image sizes on results, which is important for models that can be utilized in traffic sign recognition tasks involving different ROI sizes. Our study provides insights on identifying and optimizing the most suitable framework in terms of runtime and accuracy for training models that are suitable for performing traffic sign recognition on FPGAs.

We conclude that generally Neon and MXNet have the best training speed and inference accuracy in all our test cases. In addition, we observe that Tensorflow has one of the highest accuracies among the five frameworks on both the GPU and CPU. Since changing the input size does not necessarily affect the inference accuracy, while training ResNet-20 and ResNet-32 on the GTSRB dataset, we propose 32 by 32 as the input size for traffic sign classification, which is computationally economical without sacrificing inference accuracy. We also compare the models and see that ResNet-32 has almost the same inference accuracy as ResNet-20. This indicates that although ResNet-32 and other deeper models have much longer training times than shallower models, in return they might not provide improvement on the inference accuracy for datasets that have similar characteristics to GTSRB. Our future plan includes both benchmarking frameworks on larger and/or augmented traffic sign datasets as well as testing the pre-trained models deployed on FPGAs.

ACKNOWLEDGMENT

Thanks to Intel Labs for the funding and the technical support of this project.

REFERENCES

[1] Amazon. MXNet's official page. `https://mxnet.apache.org/`. Accessed: 2018-01-28.

[2] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. A committee of neural networks for traffic sign classification. In *International Joint Conference on Neural Networks*, pages 1918–1921, 2011.

[3] Google. Tensorflow's official page. `https://www.tensorflow.org/`. Accessed: 2018-01-28.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Computing Research Repository*, abs/1512.03385, 2015.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *Computing Research Repository*, abs/1502.01852, 2015.

[6] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *Computing Research Repository*, abs/1704.04861, 2017.

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *Computing Research Repository*, abs/1512.02325, 2015.

[8] Microsoft. CNTK's official page. `https://www.microsoft.com/en-us/cognitive-toolkit/`. Accessed: 2018-01-28.

[9] Pınar Muyan-Özçelik, Vladimir Glavtchev, Jeffrey M. Ota, and John D. Owens. Real-time speed-limit-sign recognition on an embedded system using a GPU. In Wen-mei W. Hwu, editor, *GPU Computing Gems*, volume 1, chapter 32, pages 497–516. Morgan Kaufmann, February 2011.

[10] Intel Nervana. Neon's github. `https://github.com/NervanaSystems/neon`. Accessed: 2018-01-28.

[11] Intel Nervana. Nervana aeon's official page. `http://aeon.nervanasys.com/index.html/`. Accessed: 2018-01-28.

[12] Intel Nervana. Nervana neon's official page. `https://neon.nervanasys.com/index.html/`. Accessed: 2018-01-28.

[13] Ruhr University of Bochum. GTSRB's official page. `http://benchmark.ini.rub.de/`. Accessed: 2018-01-28.

[14] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *Computing Research Repository*, abs/1612.08242, 2016.

[15] Facebook AI Research. PyTorch's official page. `http://pytorch.org/`. Accessed: 2018-01-28.

[16] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. *Computing Research Repository*, abs/1608.07249, 2016.

[17] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, (0):–, 2012.

[18] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 16–25, New York, NY, USA, 2016. ACM.

[19] Dong Wang, Jianjing An, and Ke Xu. PipeCNN: An OpenCL-based FPGA accelerator for large-scale convolution neuron networks. *Computing Research Repository*, abs/1611.02450, 2016.

[20] Karel Zuiderveld. Graphics Gems IV, Contrast Limited Adaptive Histogram Equalization. pages 474–485. Academic Press Professional, Inc., San Diego, CA, USA, 1994.