Benchmarking Deep Learning Frameworks and Investigating FPGA Deployment for Traffic Sign Classification and Detection

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

Abstract— **We benchmark several widely-used deep learning frameworks and investigate the FPGA deployment for performing traffic sign classification and detection. We evaluate the training speed and inference accuracy of these frameworks on the GPU by training FPGA-deployment-suitable models with various input sizes on GTSRB, a traffic sign classification dataset. Then, selected trained classification models and various object detection models that we train on GTSRB's detection counterpart (i.e., GTSDB) are evaluated with inference speed, accuracy, and FPGA power efficiency by varying different parameters such as floating-point precisions, batch sizes, etc. We discover that Neon and MXNet deliver the best training speed and classification accuracy on the GPU in general for all test cases, while TensorFlow is always among the frameworks with the highest inference accuracies. We observe that with the current OpenVINO release, the performance of lightweight models (e.g., MobileNet-v1-SSD, etc) usually exceeds the requirement of real-time detection without losing much accuracy, while other models (e.g., VGG-SSD, ResNet-50-SSD) generally fail to do so. We also demonstrate that we can adjust the precision of bitstreams and the batch sizes to balance inference speed and accuracy of the applications deployed on the FPGA. Finally, we show that for all test cases, the FPGA always achieves higher power efficiency than the GPU.**

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 then transferred to an optimized on-vehicle embedded system for real-time inference. The high parallelism and power efficiency of FPGAs make them a preferred embedded platform for performing autonomous driving tasks. However, embedded systems, like FPGAs, have limited resources, and meanwhile automotive tasks require real-time results with high accuracy to protect the driver's and passengers' safety. Thus, it is important to investigate methodologies for training the FPGA-suitable models which require less resources and for deploying these models to the FPGA that achieve fast inference with high accuracy.

In this research, we use traffic sign classification and detection as two different applications of autonomous driving tasks, and study how the best result of training and deployment can be realized for both. Based on this idea, our paper is logically divided into two parts, namely, classification and detection. In the classification part, we investigate the tradeoffs between the five deep learning frameworks and benchmark their training speed and inference accuracy on the GPU for performing a traffic-sign-classification task using three different models. Then, we verify the effectiveness of two selected models on the FPGA by evaluating their speed and accuracy with varied parameters such as floating point precisions, batch sizes and data types. In the detection part, we train six models for traffic sign detection using three different frameworks. We then deploy these models to the FPGA and investigate their speed, accuracy, and power efficiency by varying the same parameters used in the classification part.

CNNs have been used in traffic sign classification since the start of this decade, when the model designed by Ciresan et al. [6] (referred to by the name of the institution, IDSIA, in the rest of the paper) ranked first in the IJCNN competition in 2011. Impressive results of this study and other similar studies lead to rapid development of various alternative tools, frameworks, and models in the area of deep learning. However, there are only few studies that compare these alternative proposals. Prior to our research, Chu et al. [23] comprehensively evaluated five mainstream deep learning frameworks over a series of metrics on training MNIST and CIFAR datasets. For our benchmark in classification, we worked with a similar set of frameworks including CNTK [17], MXNet [2], Neon [18], PyTorch [21], and TensorFlow [8]. We use a different dataset, the German Traffic Sign Recognition Benchmark (GTSRB) [25] dataset that contains traffic sign images. In addition to IDSIA, we also utilize two deep residual neural networks designed by He et al. [10], ResNet-20 and ResNet-32 for our classification experiments. All three models are proven to be capable of achieving high classification accuracy results while having a reasonable size for FPGA deployment. In the detection part, we use GTSRB's detection counterpart, the German Traffic Sign Detection Benchmark (GTSDB) [20] as the training dataset. We train six SSD models with different base networks, namely, VGG [16], MobileNet-v1 [12], MobileNet-v2 [22] (with SSDLite), ResNet-18, ResNet-50 [10], and SqueezeNet-v1.1 [13] on GTSDB . We select these detection models since they are all known to have high efficiency and accuracy in performing object detection and OpenVINO [14]

*Corresponding authors

[1]Zhongyi Lin, Matthew Yih and John D. Owens are with the Department of Electrical and Computer Engineering, University of California, Davis, CA 95616; {zhylin,myih,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.

(i.e., the tool we have utilized in our FPGA deployment) has a strong support for SSD models. We train these detection models using Caffe [3], MXNet [4], and TensorFlow [1], all of which are currently supported by OpenVINO.

Deployment of deep learning models on different hardware has been an interesting but difficult topic in both academia and industry. FPGAs are one of the most challenging target hardware, as FPGAs do not have a fixed architecture as GPUs or CPUs and thus different models are usually deployed with different hardware configuration to exploit FPGAs' parallelism. Tools for FPGA deployment are proposed by studies including Suda et al. [26] and Wang et al. [27]. Although these tools have produced impressive results, very few of them are open source and most of them lack a user-friendly interface that can support development of new applications. OpenVINO, which was released at Intel's AI Dev Con 2018, provides support for FPGA development and effectively addresses the aforementioned shortcomings of the existing deployment tools. In our research, we use OpenVINO for FPGA deployment of both classification and detection models, and use various features it provides to analyze the tradeoffs in these deployments.

This paper extends our prior study [15]. Our contributions can be summarized into four main points where we provide:

- a benchmark of five popular frameworks on the training speed and inference accuracy of performing traffic sign classification;
- a thorough investigation of how the choice of CNN models and some critical parameters—e.g., floating point precision, batch sizes, data types, etc.—will affect the performance of the FPGA on performing both traffic sign classification and detection;
- optimization techniques we apply to classification experiments and detection model training which can be utilized by future studies targeting similar frameworks, models, and datasets; and
- a power efficiency comparison of the FPGA and GPU on traffic sign detection.

We believe our discussions and analysis can guide engineers and researchers in training models on the GPU and deploying them on the FPGA to perform traffic sign classification, detection, and other autonomous driving applications.

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, experiment specifications, and optimizations in Section III. Then, we present our results and analysis in Section IV. Finally, we discuss potential future work and provide conclusions in the last two sections.

II. Background

A. Frameworks/Tools

In the classification part, we use the native APIs of CNTK (v2.5.1), Neon (v2.6.0), MXNet (v1.3.0), PyTorch (v0.4.0), and TensorFlow (v1.12), which are all currently popular and actively used in both industry and academia.

We use OpenVINO for the deployment of both the classification and detection models on the FPGA. With a pre-trained model, users start with OpenVINO's model optimizer to convert them into intermediate representation (IR), and consequently deploy this model's IR to FPGA which has been flashed with a pre-compiled bitstream (.aocx) file as the hardware configuration.

To train our detection models which were mainly used for collecting FPGA deployment results, we utilize three OpenVINO-supported frameworks: Caffe, MXNet, and TensorFlow. The other three frameworks used in classification experiments—CNTK, Neon and PyTorch—are not used as they are not directly supported. We do not add Caffe to the training benchmark for performance comparison because it is currently inactive and thus out-of-date in some operations such as depthwise convolution. However, the Caffe community has a huge amount of code for legacy projects (e.g., SSD), which still makes it a good choice for academic research including ours.

B. Datasets

The GTSRB dataset we use in the classification part contains more than 50,000 ppm-format images with sizes varying from 15×15 to 250×250. Each image contains a traffic sign that belongs to one of the 43 different classes. GTSRB's detection counterpart, the GTSDB dataset which we use in the detection part, contains 900 images of size 800×1360, among which 159 images contain no traffic signs while rest of the images contain one or multiple of them. All the traffic signs in GTSDB comprise the same 43 classes of traffic signs as GTSRB, but are further classified into 4 main classes: danger, mandatory, other, and prohibitory. The bounding boxes of traffic signs in each image range from 16×16 to 128×128.

C. Models

1) Classification: The models we train on include the CNN model designed by Ciresan et al. [6] (IDSIA), which is memory-friendly on FPGAs and achieves greater than 99% classification accuracy with proper preprocessing. We also investigate ResNets (ranked first in the ImageNet competition in 2015) introduced by He et al. [10] to address the vanishing/exploding gradient problem and liberate CNNs from the limit of depth. In addition to having high accuracy, ResNets are also considered FPGA-friendly for its easy construction with repetitive basic/bottleneck building blocks that allow resizing the model when resources are limited, as well as its flexible input sizes that are lacking in many other models that originated from ImageNet competitions.

2) Detection: The models we used for detection are all Single Shot MultiBox Detectors (SSDs). SSD (shown in Fig. 1) is introduced by Liu et al. [16] and consists of a base image classification network, which is usually pre-trained, and several multi-box layers (connected to either feature maps in the base network or the previous layer), which detect objects in different scales. SSD is an end-to-end model that requires only one pass for object detection on a

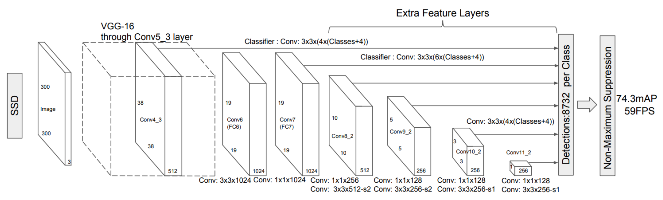


Fig. 1: Structure of the SSD model

given image, which brings a higher detection speed compared to its ascendants. SSD is also very flexible because different base networks can be used for applications with different requirements.

In this study, following the study of Liu et al., we use VGG-reduced (with VGG-16's [24] fully-connected layers replaced by convolutional layers and layer fc6 dilated) as one of the base networks since this model is known to produce high inference accuracy in object detection. We also use MobileNet-v1 [12], -v2 [22], and SqueezeNet-v1.1 [13] as alternative base networks since they are all suitable for embedded systems including FPGAs. These models utilize techniques such as using depthwise convolution layers and removing fully connected layers to reduce computation burden and parameter count while making a negligible sacrifice in accuracy compared to using normal convolutional layers. In particular, the additional convolution layers in the MobileNet-v2-SSDLite model are also replaced by depthwise separable convolutional layers, which accounts for the name "SSDLite". Finally, we also include two ResNet models—ResNet-18 and ResNet-50—as additional base networks in the detection experiments due to their advantages highlighted above in the description of classification models (Section II.C.1).

III. Methodology

Our experiments are completed on a Ubuntu 16.04 LTS machine with an Intel i7-7700K 4.2 GHz CPU, an NVIDIA GTX 1050 Ti GPU with 4 GB memory and an Arria 10 FPGA development board. CUDA (v9.0) is used as packages to support GPU acceleration. OpenVINO Release 5 for FPGA is used for FPGA deployment experiments. The experiment code is programmed with Python 3.5 and libraries including PyCuda (v2017.1.1) and PyGPU (v0.7.5).

A. Image Preprocessing

We crop the images in GTSRB 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) [28], which resulted in a minimum error rate among four preprocessing techniques used by Ciresan et al.

For the GTSDB dataset, we simply remove all 159 images that contain no traffic signs, and split the 741 remaining images into train-val and test datasets with a ratio of roughly 4:1. In our experiments, this results in 588 images in the train-val set and 153 in the test set. The reason we do not follow the original 600 + 300 train-val/test split is that the train-val image ratio (2:1) is too low and tends not to give good results. Except for this step, we do not perform any other preprocessing on GTSDB since we follow the default data augmentation techniques that are introduced by Liu et al. [16].

Model Name	1st Feature Map	2nd Feature Map
VGG	conv4_3	fc7
MobileNet-v1	conv5_5/sep	conv6/sep
MobileNet-v2	conv5_3/expand	conv6_4
ResNet-18	block2	block4
ResNet-50	block2	block4
SqueezeNet-v1.1	fire5	fire9

TABLE I: 1st and 2nd feature maps of the base networks which are connected to multi-box layers in SSD detection models

B. Construction of ResNets for Experiments

We follow He et al.'s way of constructing ResNets for the CIFAR-10 and ImageNet datasets to construct our ResNet classification models and SSD detection models that use ResNet base networks. In the classification part, 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. In the detection part, we construct ResNet-18 and ResNet-50 as described by He et al. [10]. 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. Experiment Specifications

In our classification 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 [11], which claims to have better convergence. We select two models, namely, ResNet-20 (with input size 32 by 32, referred to as 32x32 below) and ResNet-32 (64x64) trained by TensorFlow for the FPGA deployment in the detection part. The reasoning for these selections is provided in Section IV-B. These models are converted to IR representations in XML format using OpenVINO's model optimizer, and then passed to the validation application for classification accuracy and speed evalutation. We do not present the results of classification models trained by MXNet, as the speed and accuracy are low and unstable, which may indicate an existing bug of the model converter of the current release of OpenVINO. Since CNTK, Neon, and PyTorch are not directly supported by OpenVINO as mentioned in Section II-A, we do not test classification models trained by them in our FPGA deployment experiments.

In the detection experiments, we train the VGG-SSD and SqueezeNet-v1.1-SSD models using the SGD learner with 0.9 momentum. We use RMSProp as the optimizer and follow the parameter settings provided in the related repository [5] and the study by Sandler et al. [22] to train MobileNet-v1-SSD and MobileNet-v2-SSDLite. We also use SGD for ResNet-18 and ResNet-50, but tune our own parameter settings, which can be viewed in our GitHub repository [7]. The batch size is set to 2 for all models except for SqueezeNet-v1.1-SSD (batch size 16). The input images are all resized to 300 by 510 which both retains the aspect ratio of the original images and has a height that matches the size of images in the original SSD implementations for convenient source code reuse. Table I shows which feature maps of the base networks are connected to first two multi-box layers in SSD detection models. We follow the layer naming convention in the studies these models are created [10,12,13,23,25]. We test models trained by Caffe, MXNet, and TensorFlow, with batch size 1 or 4 for detection inference. We do not measure the speed, accuracy, and power measurement on the GPU with batch size 4 for TensorFlow due to the lack of this option running on the GPU. We also skip running ResNet-50-SSD with batch size 4 on the GPU since it exceeds the memory limit of the GPU used in this study. Similar to the classification experiments, we use OpenVINO's model optimizer and validation application to obtain the mean average precision (mAP) and speed. We use the FP11 and FP16 pre-compiled FPGA bitstreams for the tests. Runtime power of the FPGA and GPU is measured by Arria 10's power monitor and NVIDIA's nvidia-smi tool, respectively.

D. *Optimizations*

In the classification part of our study, we apply both system and code optimizations to improve performance of the frameworks. In these optimizations, we demonstrate potential speed change by running a simple single-epoch test on frameworks with the IDSIA model on the GTSRB dataset with training set size 31,367. The input size we use in these tests is 48 by 48 and the batch size is 64. In addition to optimizations used in the classification part, to obtain better object detection results, we also conduct parameter tuning during detection model training. All the optimization techniques will be introduced in the following subsections:

1) System optimizations for classification model training on the GPU: 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.

2) Code optimizations for classification model training: Code optimization can also be done to improve speed and accuracy. For CNTK, before being passed to the 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 measurements 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 have 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.

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 classification accuracy.

3) Parameter tuning for detection model training: SSD requires a good match between the aspect ratios of the

Parameter Name	Original Value(s)	New Value(s)
Anchor box aspect ratio	[1,1/2,1/3,2,3]	[1]
Prior box scale range	[0.2, 0.9]	[0.05, 0.6]
NMS bounding box number	400	100
Post-detection bounding box number	100	40
RGB mean values across the dataset	[123,117,104]	[125,127,130]

TABLE II: Parameters tuned for SSD models trained with GTSDB

Size Scale	0.02	0.04	0.06	0.08	0.10	0.20	0.50
Percentage(%)	0.0	33.3	68.1	83.5	92.7	100.0	100.0

TABLE III: Cumulative percentage of bounding boxes based on their size scales.

bounding boxes in the dataset, and the aspect ratios that are set for anchor boxes of different multi-box layers. It also requires a good match between the sizes of the bounding boxes in the dataset, and the sizes of objects that each multi-box layer can detect depending on which feature map layer these layers are connected to. In Table II we summarize the parameters we tuned for the GTSDB dataset. In the following paragraphs, we explain the reason for these modifications.

Compared to the VOC PASCAL and COCO datasets that were used in the original development of SSD, in the GTSDB dataset, the bounding box sizes are relatively small (at most 16% of the image), and the aspect ratios of the bounding boxes are relatively fixed, which are close to 1. This is because traffic signs usually appear to be small in the image, and those marked in GTSDB usually have a square bounding box. After calculating the aspect ratios of all region of interests in GTSDB, we see that 287 and 912 out of 1024 boxes have aspect ratios between (5/6, 1) and (1, 6/5), respectively, which represent 98.8% of the total bounding boxes and coincide with our initial guess of nearly-square bounding boxes. Thus, there is no need to keep anchor boxes with aspect ratios of 1/2, 2, 1/3, 3, etc. in the SSD model.

Table III shows the cumulative percentage of bounding boxes based on their size scales, where size scale indicates the ratio of bounding box size to overall image size. We see that 92.7% of the bounding boxes have a size scale smaller than 0.1, and all the bounding boxes have a size scale smaller than 0.2. Therefore, we tend to focus on small sizes, and we reset the prior boxes scale range to [0.05, 0.6]. Since the majority of default bounding boxes are generated from the first two multi-box layers which are responsible for the detection of objects with size scale between 0.05 and 0.2, this design will satisfy our need of detecting traffic signs which are all smaller than 20% of the size of the overall image.

The tuning of the last three parameters shown in Table II is for better inference speed and accuracy performance.

IV. Results and Analysis

Fig. 2 provides plots that indicate the batch training time on the GPU versus the following variables: two ResNets with different input sizes and all models with a fixed input size 48 by 48. We also present the total training time and inference accuracy of all classification models, frameworks, and input sizes on the GPU in Table IV, where the best results are shown in bold. Table V shows the inference and accuracy of ResNet-20 (32x32) and ResNet-32 (64x64) on the FPGA versus different batch sizes, data types, and bitstreams with different precisions, as well as the same type of results on the GPU for reference. We present inference speed and accuracy of six detection models on the FPGA and the GPU (for reference), which were trained with Caffe (Table VI), MXNet (Table VII), and TensorFlow (Table VIII). At the end of this section, we present the power efficiency of running detection models with batch size 1 on the FPGA and the GPU as shown in Fig. 4. The power efficiency is represented by the number of processed images per Joule (img/J), which is calculated by dividing FPS by the average runtime power.

A. Classification Training Speed on the GPU

As shown in Table IV, the classification experiments, training speed results on the GPU show that Neon is the fastest in most cases, which could be explained by the utilization of the Winograd-based algorithm for convolution [19]. 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. Classification Inference Accuracy on the GPU

Results from classification experiments also show that in terms of classification accuracy, Neon and TensorFlow provide the best performance, which could possibly be accredited to their optimized low-level implementation. 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 classification 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 dominant classification accuracy, the best performance on average was reported with the 64 by 64 input size on ResNet-32. Hence, we conclude that ResNet-20 (32x32) constitutes the most computationally economical option that does not compromise inference accuracy and ResNet-32 (64x64) is the option that delivers the best accuracy. Thus, for the FPGA deployment, we choose to work with these two particular classification models.

C. Classification Inference Speed and Accuracy on the FPGA

For the reasons explained in Section III.C, we do not include classification models trained by CNTK, Neon, MXNet, and PyTorch in the FPGA deployment experiments. Hence, Table V shows FPGA deployment results of selected classification models that are trained by TensorFlow. As can be seen from this table, ResNet-20 (32x32) has inference speed higher than ResNet-32 with only a slightly lower accuracy than ResNet-32 (64x64). This result is in accordance with the GPU results, and indicates that ResNet-20 (32x32) is also computationally economical on the FPGA. In both models, if we use an FP11 pre-compiled bitstream instead of an FP16 one, we always get a better inference speed with slightly lower accuracy. If the pre-compiled bitstream being used is fixed, changing the data type between half or full precision does not significantly affect the inference speed or accuracy. We also observe that using batch size 32 always results in faster inference on the FPGA than using batch size 1.

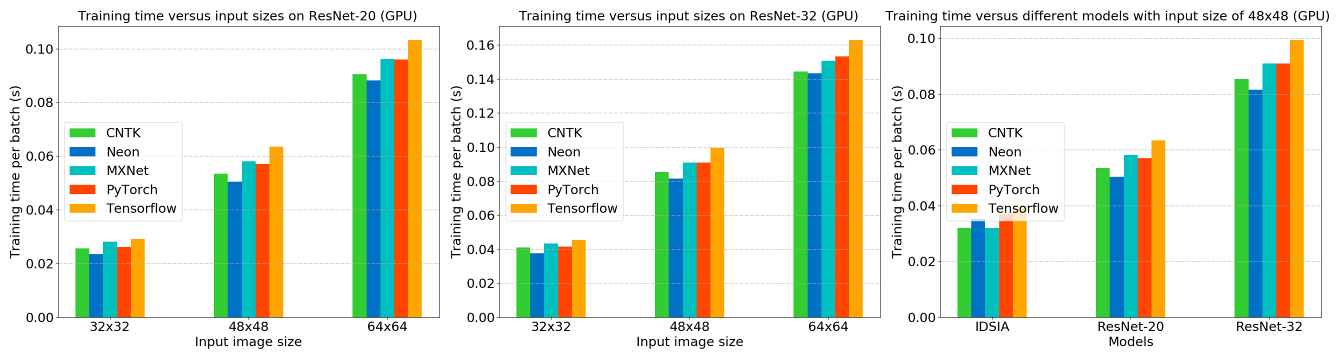


Fig. 2: GPU average batch training time versus (a) input sizes, on ResNet-20, (b) input sizes, on ResNet-32 and (c) all models, with the same input size 48x48.

Framework	32x32				48x48						64x64			
	ResNet-20		ResNet-32		IDSIA		ResNet-20		ResNet-32		ResNet-20		ResNet-32	
	t_{train}	acc	t_{train}	acc	t_{train}	acc	t_{train}	acc	t_{train}	acc	t_{train}	acc	t_{train}	acc
CNTK	313.80	96.93%	502.60	97.24%	393.4	**96.78%**	655.72	96.37%	1047.60	96.64%	1109.80	96.49%	1771.54	96.79%
Neon	**288.65**	**97.68%**	**461.50**	**97.72%**	432.2	95.87%	**618.46**	97.35%	**1001.42**	**98.27%**	**1081.47**	97.35%	**1760.56**	97.76%
MXNet	344.82	96.61%	531.36	96.26%	**392.7**	96.55%	712.59	96.80%	1116.57	97.32%	1179.94	96.73%	1849.95	97.34%
PyTorch	319.53	94.00%	509.17	97.00%	455.5	96.00%	697.94	97.00%	1114.56	96.00%	1177.42	95.00%	1877.46	96.00%
TensorFlow	357.01	97.32%	558.48	97.60%	491.92	96.37%	778.71	**97.36%**	1220.48	97.26%	1267.75	**98.00%**	2001.53	**98.34%**

TABLE IV: Total trainning time (in seconds) and accuracy of all classification models on the GPU for all sizes.

Models	FP11				FP16				GPU		
	Half-precision		Full-precision		Half-precision		Full-precision				
	FPS	acc	FPS	acc	FPS	acc	FPS	acc	FPS	acc_{train}	acc_{test}
ResNet-20 (32x32) (batch=1)	1787.41	97.23%	1728.87	97.23%	1820.37	97.30%	1827.78	97.27%	533.82	100%	97.32%
ResNet-20 (32x32) (batch=32)	4726.92	97.23%	4680.44	97.23%	4637.72	97.30%	4627.19	97.27%	7782.22	100%	97.32%
ResNet-32 (64x64) (batch=1)	838.80	98.13%	844.69	98.21%	1062.65	98.26%	1055.13	98.29%	393.74	100%	98.34%
ResNet-32 (64x64) (batch=32)	1223.33	98.13%	1216.87	98.21%	1673.56	98.26%	1671.77	98.29%	1555.34	100%	98.34%

TABLE V: Inference speed (in FPS) and accuracy of TensorFlow classification models on the FPGA and GPU

We also compare the results on the FPGA and GPU, on which we overfit the models and reach 100% training accuracy. We see that in the case of ResNet-20 (32x32) with batch size 32, the speed of the GPU is faster than the speed of the FPGA. On the other hand, in the case of ResNet-32 (64x64) with batch size 32, the speed of the GPU is slower than the speed of the FPGA when the FP16 bitstream stream is used, but it is faster if the FP11 bitstream is used. In all other cases, the FPGA reaches an inference speed higher than the GPU, possibly because of higher utilization of compute units. In addition, in each test case, the classification accuracy on the FPGA is slightly lower than accuracy on the GPU which has a higher FP32 floating point precision, but the discrepancy is negligible. Therefore, we conclude that for classification tasks on small-size images (e.g., 32x32 or 64x64), with most of the models we investigated, the FPGA reaches higher inference speed than the GPU, with a negligible sacrifice in accuracy.

D. Detection Inference Speed and Accuracy

From Table VI, VII and VIII, we clearly observe that lightweight models including MobileNet-v1-SSD, MobileNet-v2-SSDLite, ResNet-18-SSD, and SqueezeNet-v1.1-SSD achieve an inference speed much higher than the speed required for real-time performance (24 FPS), regardless of the framework that trains them. In particular, SqueezeNet-v1.1-SSD achieves extremely high inference speed on the FPGA when the model is trained by Caffe and MXNet, but a much lower speed when trained by TensorFlow. We conjecture that this might be because OpenVINO's model optimizer is not fully optimizing the TensorFlow model. On the contrary, VGG-SSD fails to have real-time performance due to the computation burden of its last fully-connected layers. Also due to its computation latency, ResNet-50-SSD fails all of the real-time tests on the FPGA but one, where it is trained by TensorFlow and run with FP11 precision. We see that models trained by MXNet suffer an accuracy degradation on the FPGA, which might again indicate an existing bug in OpenVINO's model

Models	FP11				FP16				GPU		
	Half-precision		Full-precision		Half-precision		Full-precision				
	FPS	mAP	FPS	mAP	FPS	mAP	FPS	mAP	FPS	mAP$_{\text{train}}$	mAP$_{\text{test}}$
VGG-SSD (batch=1)	16.16	0.806	16.19	0.800	12.16	0.802	12.13	0.801	18.87	0.810	0.801
VGG-SSD (batch=4)	16.73	0.803	16.65	0.795	12.05	0.796	12.33	0.796	20.41	0.810	0.799
MobileNet-v1-SSD (batch=1)	98.74	0.720	99.67	0.763	81.30	0.796	83.17	0.795	48.26	0.837	0.808
MobileNet-v1-SSD (batch=4)	103.07	0.719	102.92	0.752	86.92	0.793	86.58	0.793	48.42	0.837	0.808
MobileNet-v2-SSDLite (batch=1)	67.33	0.743	68.49	0.742	62.57	0.753	62.52	0.753	29.37	0.782	0.756
MobileNet-v2-SSDLite (batch=4)	70.06	0.743	70.06	0.743	64.02	0.753	63.83	0.753	37.05	0.782	0.757
ResNet-18-SSD (batch=1)	37.49	0.787	37.33	0.788	28.69	0.786	28.77	0.789	17.61	0.837	0.833
ResNet-18-SSD (batch=4)	39.28	0.789	39.27	0.790	30.12	0.789	30.02	0.791	20.70	0.837	0.832
ResNet-50-SSD (batch=1)	13.59	0.724	13.62	0.718	9.95	0.736	9.95	0.736	15.77	0.808	0.801
ResNet-50-SSD (batch=4)	13.83	0.727	13.81	0.723	10.12	0.741	10.14	0.742	N/A	N/A	N/A
SqueezeNet-v1.1-SSD (batch=1)	218.28	0.699	220.93	0.704	160.35	0.701	158.37	0.701	17.51	0.738	0.738
SqueezeNet-v1.1-SSD (batch=4)	238.34	0.695	236.23	0.702	167.86	0.697	168.45	0.697	19.15	0.738	0.734

TABLE VI: Inference speed (in FPS) and accuracy (in mAP) of Caffe detection models on the FPGA and GPU.

Models	FP11				FP16				GPU		
	Half-precision		Full-precision		Half-precision		Full-precision				
	FPS	mAP	FPS	mAP	FPS	mAP	FPS	mAP	FPS	mAP$_{\text{train}}$	mAP$_{\text{test}}$
VGG-SSD (batch=1)	16.23	0.561	16.18	0.543	12.07	0.556	12.08	0.556	18.55	0.753	0.730
VGG-SSD (batch=4)	16.57	0.567	16.68	0.549	12.28	0.565	12.82	0.565	19.80	0.753	0.729
MobileNet-v1-SSD (batch=1)	92.76	0.529	90.93	0.525	81.31	0.523	81.08	0.519	106.76	0.712	0.701
MobileNet-v1-SSD (batch=4)	101.48	0.531	100.46	0.524	85.64	0.527	85.40	0.522	135.53	0.712	0.701
MobileNet-v2-SSDLite (batch=1)	65.80	0.514	65.88	0.489	59.69	0.516	60.14	0.517	78.85	0.688	0.665
MobileNet-v2-SSDLite (batch=4)	68.19	0.515	68.24	0.490	62.23	0.525	62.32	0.526	95.93	0.688	0.665
ResNet-18-SSD (batch=1)	55.82	0.574	55.35	0.574	31.14	0.570	31.20	0.571	88.94	0.708	0.702
ResNet-18-SSD (batch=4)	57.85	0.575	58.05	0.574	31.81	0.570	31.94	0.571	120.49	0.708	0.700
ResNet-50-SSD (batch=1)	20.68	0.589	20.59	0.562	12.64	0.575	12.64	0.580	37.22	0.737	0.729
ResNet-50-SSD (batch=4)	21.07	0.589	21.04	0.563	12.77	0.576	12.75	0.580	45.14	0.737	0.732
SqueezeNet-v1.1-SSD (batch=1)	184.76	0.199	183.41	0.202	147.68	0.279	151.14	0.281	37.89	0.468	0.445
SqueezeNet-v1.1-SSD (batch=4)	189.75	0.199	197.97	0.202	157.18	0.280	161.49	0.281	71.05	0.468	0.445

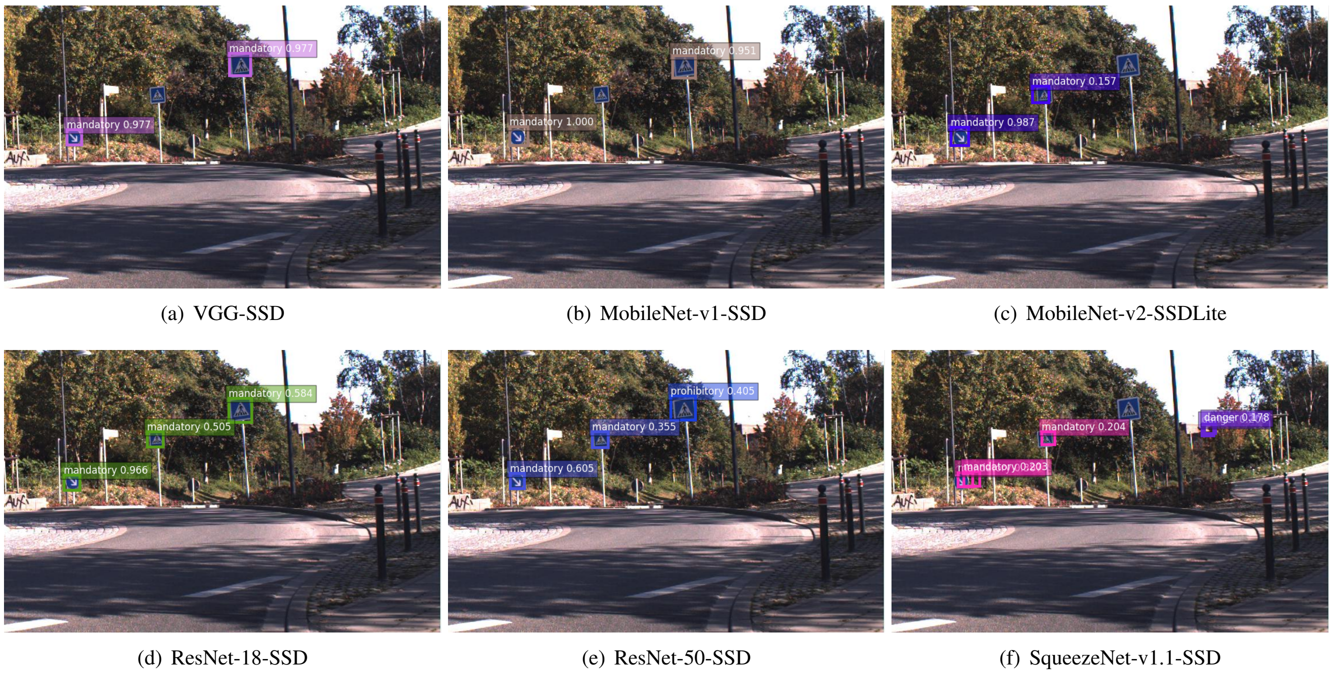
TABLE VII: Inference speed (in FPS) and accuracy (in mAP) of MXNet detection models on the FPGA and GPU.

Models	FP11				FP16				GPU		
	Half-precision		Full-precision		Half-precision		Full-precision				
	FPS	mAP	FPS	mAP	FPS	mAP	FPS	mAP	FPS	mAP$_{train}$	mAP$_{test}$
VGG-SSD (batch=1)	20.61	0.754	20.59	0.748	11.32	0.761	11.33	0.761	26.24	0.788	0.786
VGG-SSD (batch=4)	20.50	0.754	20.57	0.748	11.37	0.761	11.33	0.761	N/A	N/A	N/A
MobileNet-v1-SSD (batch=1)	105.46	0.626	105.14	0.603	88.02	0.709	88.38	0.709	66.81	0.723	0.719
MobileNet-v1-SSD (batch=4)	108.48	0.627	108.49	0.626	90.37	0.709	90.39	0.709	N/A	N/A	N/A
MobileNet-v2-SSDLite (batch=1)	68.78	0.559	66.16	0.552	56.44	0.731	56.84	0.732	63.88	0.775	0.769
MobileNet-v2-SSDLite (batch=4)	77.00	0.557	76.73	0.558	63.13	0.730	61.99	0.731	N/A	N/A	N/A
ResNet-18-SSD (batch=1)	50.36	0.669	48.28	0.667	42.81	0.682	42.62	0.680	45.00	0.751	0.748
ResNet-18-SSD (batch=4)	48.20	0.669	49.97	0.667	42.59	0.682	42.46	0.680	N/A	N/A	N/A
ResNet-50-SSD (batch=1)	26.15	0.692	25.80	0.686	19.35	0.684	19.30	0.672	35.83	0.751	0.745
ResNet-50-SSD (batch=4)	25.84	0.692	26.28	0.685	19.32	0.684	19.17	0.672	N/A	N/A	N/A
SqueezeNet-v1.1-SSD (batch=1)	66.54	0.618	66.14	0.617	62.74	0.609	61.74	0.609	48.26	0.663	0.661
SqueezeNet-v1.1-SSD (batch=4)	67.20	0.618	64.27	0.617	62.63	0.609	63.07	0.609	N/A	N/A	N/A

TABLE VIII: Inference speed (in FPS) and accuracy (in mAP) of TensorFlow detection models on the FPGA and GPU.

converter. With Caffe and TensorFlow, we reach the same conclusions as in the classification part that in most of the cases the FP16 bitstream provides higher accuracy yet lower speed than the FP11 one, and that the datatype does not make a big difference with the fixed bitstream. Exceptions include VGG-SSD and some ResNet-18/50 cases, where FP11 produces almost the same or even higher accuracy as FP16. We also observe that except for the case of VGG-SSD with FP16, using batch size 4 always results in slightly faster detection on the FPGA than using batch size 1.

Notice that although MXNet models suffer an accuracy degradation on the FPGA, in terms of inference speed, models trained by the three frameworks achieve results essentially very close to each other. This indicates that the use of different frameworks during model training does not make a big difference on the inference speed of the deployment on the FPGA. However, we should remember that different frameworks have different model representations and Open-



(a) VGG-SSD (b) MobileNet-v1-SSD (c) MobileNet-v2-SSDLite

(d) ResNet-18-SSD (e) ResNet-50-SSD (f) SqueezeNet-v1.1-SSD

Fig. 3: Qualitative inference results across detection models. Detected traffic signs marked by bounding boxes and notated by category and confidence.

VINO's optimization techniques on these representations are different. Hence, it is impractical to expect identical performance across copies of the same model trained by different frameworks.

Compared to the reference results on the GPU, we see that in most of the cases inference speed on the GPU is higher than the FPGA. One exception is running MobileNet-v1-SSD and MobileNet-v2-SSDLite with Caffe, because Caffe does not have a good support for depthwise separable convolution. We can also observe that with both Caffe and TensorFlow, VGG-SSD and ResNet-50-SSD has a lower inference speed on the FPGA than on the GPU, while the opposite happens for other models. This is because VGG-SSD and ResNet-50-SSD might be computation-bound on the FPGA, whereas other models are relatively lightweight and the FPGA's flexible configuration and optimization enable them to have a shorter runtime compared to the GPU. However, with MXNet, all models run faster on the GPU than on the FPGA except for SqueezeNet-v1.1-SSD, which could be credited to MXNet's efficient implementation on the GPU. This is also in line with our conclusion in the classification part. Our results also show that the accuracy on the GPU using the same model and different batch size is slightly different which is caused by the test set size (i.e., 153) not being a multiple of the batch size 4. We also observe that with Caffe and TensorFlow trained models, the FPGA reaches the accuracy of the GPU using the FP16 bitstream. In addition, the FPGA reaches the inference speed close to that of the GPU by using an FP11 datatype, which leads to some sacrifice in accuracy.

When we train the classification models we keep all specifications we use for training as well as their implementations across different frameworks exactly the same to obtain fair benchmarking results. However, although we keep the specifications the same while training the detection models, we do not guarantee that the implementations are the same. Therefore, the detection models' accuracy on the FPGA and the training/testing accuracy on the GPU should only be used for comparison within a framework, not for comparison across frameworks. This means the above results does not suggest Caffe as a better framework in training SSD models than MXNet or any conclusions alike.

We present a brief qualitative comparison of inference results using different models in Fig. 3. We can see that with ResNet-18/50 as base networks, SSD models can accurately capture all traffic signs appear in the image. The other models either miss or misdetect one or more traffic signs. This matches the theoretical fact that among all models we tested, ResNets provide the best feature extraction performance. Overfitting and suboptimal training strategy might explain the reason why ResNets-based models do not achieve the highest inference accuracy in our experiments.

E. Power Efficiency Comparison of the FPGA and GPU

One of the expected advantages of FPGAs against GPUs is power efficiency. In Fig. 4 we show the comparison of power efficiency of the detection models in img/J running on the FPGA and the GPU respectively, with batch size set to 1. The names of all models are following the order in the above tables and abbreviated. We can see that in all the cases we test, the FPGA has better power efficiency compared to the GPU. Among the six models we tested,

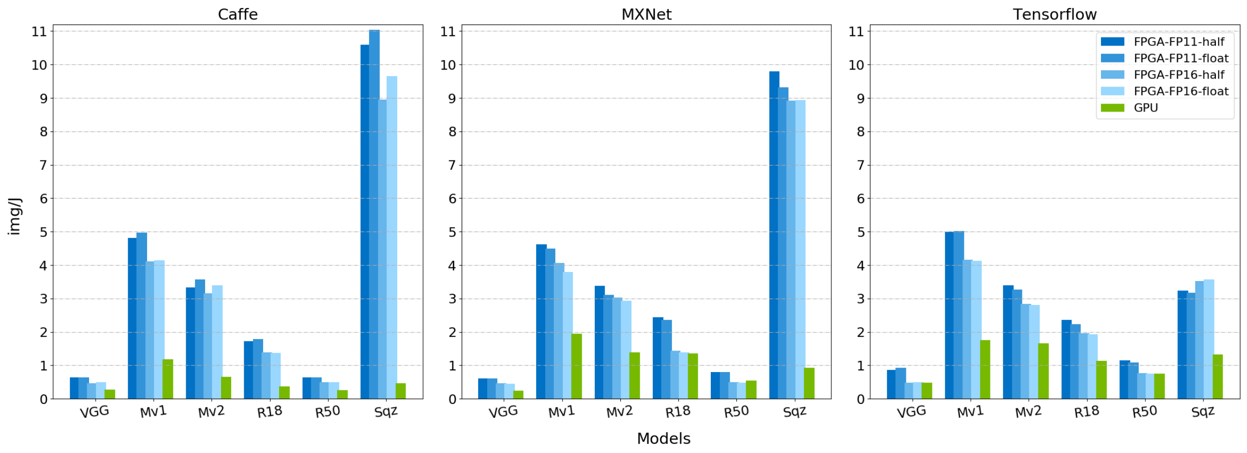


Fig. 4: Power efficiency (in image/J) of different models running on the FPGA and GPU with batch size 1.

SqueezeNet-v1.1-SSD has the highest power efficiency in the Caffe and MXNet test cases, while MobileNet-v1-SSD has the highest values in the TensorFlow test cases because of the TensorFlow with SqueezeNet-v1.1-SSD case, where we observe unexpected low inference speeds as explained in Section IV-D. Among the FPGA test cases, FP11 is always more power-efficient than FP16, while the data type does not have a clear impact on the power efficiency. The comparison across frameworks shows that there is no clear difference on power efficiency running on the FPGA except for the TensorFlow with SqueezeNet-v1.1-SSD case mentioned above. We also observe that Caffe models always have a lower power efficiency when running on the GPU. This could again be explained by Caffe's weak support for high performance of depthwise separable convolution.

F. Summary of Findings

Based on these results, we see that among the five frameworks we have utilized in classification experiments, Neon has the fastest average training speed, and one of the best inference accuracies on the GPU, along with TensorFlow, whose speed is slower. 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. Overall, in terms of both training speed and inference accuracy, we consider Neon and MXNet to be the most suitable framework for training classification models. We also conclude that ResNet-20 (32x32) constitutes the most computationally economical option that does not compromise inference accuracy for the specific task, i.e., traffic sign classification on GTSRB, on both the GPU and FPGA. This means that ResNet-20 (32x32) would probably be one of the most economical choices for other similar classification tasks on small size images as well.

In the classification part, the FPGA obtains a higher speed than the GPU in most of the cases, which can possibly be explained by better utilization of compute units. In all test cases the FPGA reaches almost the same inference accuracy as the GPU with a negligible gap. In the detection part, although in most of the cases the FPGA cannot reach the GPU's speed, the FPGA is still able to provide detection speeds that exceed real-time performance requirements with lightweight models like MobileNet-v1, etc. The FPGA does not get an accuracy as high as the GPU with models trained by TensorFlow (only in FP11 cases) and MXNet. We believe this may be caused by the model optimizer of OpenVINO instead of the FPGA itself, since with Caffe-trained models we see that the inference accuracies of the FPGA and GPU are very close to each other. In addition, we observe that with the same model, the FPGA always has a higher power efficiency compared to the GPU.

In both the classification and detection experiments, we discover that using FP11 bitstreams and bigger batch sizes always results in higher speed on the FPGA compared to using FP16 ones and smaller batch sizes, respectively. However, using FP11 bitstreams results in some decrease in accuracy. Finally, if the bitstream being used is fixed, the data type makes only a very tiny difference on both inference speed and accuracy.

V. Future Work

From the results we can see that the power efficiency of the FPGA is higher than the GPU. However, we can still try to further raise it on the FPGA. This could be critical to autonomous driving applications in real life as image processing consumes a significant portion of the vehicle's energy and directly contributes to cooling requirements. We plan to explore additional detection models given that OpenVINO will provide official support for them in the future. In addition, we plan to explore additional techniques like deep compression, to seek the opportunity of enhancing the power efficiency of the FPGA. Using deep compression, Han et al. [9] claims to achieve 3x to 7x better energy efficiency with models like AlexNet and VGG on the CPU, GPU, and mobile GPU.

Finally, in the future we plan to investigate whether it would be possible to deploy models trained by frameworks that support the ONNX format but are not currently included in our FPGA deployment experiments. To accomplish this deployment, we plan to first convert these models to ONNX format and then explore whether OpenVINO can be used to deploy them since OpenVINO is supporting the ONNX format.

VI. Conclusions

In this research, we benchmark deep learning frameworks and investigate FPGA deployment using traffic sign classification and detection as two application contexts.

In the classification part of this study, we conclude that generally Neon and MXNet have the best training speed and inference accuracy on the GPU for the cases we test. In addition, we observe that TensorFlow has one of the highest accuracies among the five frameworks. Finally, we conclude that ResNet-20 with input size 32x32 is an economical choice for performing small-image classification, e.g., GTSRB on both the GPU and FPGA.

In the detection part of this study, we observe that with the current OpenVINO release, lightweight models including MobileNet-v1-SSD, MobileNet-v2-SSDLite, ResNet-18-SSD, and SqueezeNet-v1.1-SSD achieve detection speeds on the FPGA that exceed the real-time performance requirements. All the six models we test are observed to be more power-efficient running on the FPGA than on the GPU, without compromising much accuracy.

As the common conclusion from the FPGA deployment of both the classification and detection models, we discover that with FP11 bitstreams, models achieve higher inference speed than with FP16 bitstreams with some sacrifice in accuracy. In addition, we observe that when using the same bitstream, a bigger batch size usually brings higher speed but the data type of the model does not make a difference for these two metrics.

Our future plan includes exploring the effects of using non-SSD models and/or techniques like deep compression on the performance of the FPGA and GPU, as well as deploying models trained by frameworks which are ONNX-friendly but not directly supported by OpenVINO.

Acknowledgment

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

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA, 2016. USENIX Association.

[2] Amazon. MXNet's official page. `https://mxnet.apache.org/`. Accessed: 2019-04-05.

[3] UC Berkeley. Caffe's official page. `http://caffe.berkeleyvision.org/`. Accessed: 2019-04-05.

[4] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[5] chuanqi305. Mobilenet-v1-ssd. `https://github.com/chuanqi305/MobileNet-SSD`. Accessed: 2019-04-05.

[6] 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.

[7] Zhongyi Lin et al. Traffic sign classification and detection benchmark. `https://github.com/owensgroup/TrafficSignBench`. Accessed: 2019-04-05.

[8] Google. TensorFlow's official page. `https://www.tensorflow.org/`. Accessed: 2019-04-05.

[9] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *CoRR*, abs/1510.00149, 2015.

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

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

[12] 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. *CoRR*, abs/1704.04861, 2017.

[13] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR*, abs/1602.07360, 2016.

[14] Intel. OpenVINO's official page. `https://software.intel.com/en-us/openvino-toolkit`. Accessed: 2019-04-05.

[15] Zhongyi Lin, Jeffrey M. Ota, John D. Owens, and Pınar Muyan-Özçelik. Benchmarking deep learning frameworks with FPGA-suitable models on a traffic sign dataset. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1197–1203, Jun 2018.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. *CoRR*, abs/1512.02325, 2015.

[17] Microsoft. CNTK's official page. `https://www.microsoft.com/en-us/cognitive-toolkit/`. Accessed: 2019-04-05.

[18] Intel Nervana. Nervana Neon's official page. `https://neon.nervanasys.com/index.html/`. Accessed: 2019-04-05.

[19] Intel Nervana. Not so fast, FFT: Winograd. `https://ai.intel.com/winograd/`. Accessed: 2019-04-05.

[20] Ruhr University of Bochum. GTSDB's official page. `http://benchmark.ini.rub.de/?section=gtsdb&subsection=news`. Accessed: 2019-04-05.

[21] Facebook AI Research. PyTorch's official page. `http://pytorch.org/`. Accessed: 2019-04-05.

[22] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

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

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

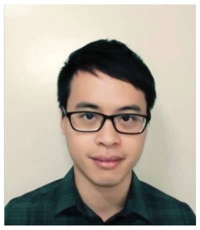
[25] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 2012.

[26] 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 *In Proceedings of the 2018 IEEE Intelligent Vehicles (IV) Symposium*, FPGA '16, pages 16–25, New York, NY, USA, 2016. ACM.

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

[28] Karel Zuiderveld. Graphics Gems IV, contrast limited adaptive histogram equalization. pages 474–485. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

Zhongyi Lin is currently a first-year Ph.D. student and a graduate research assistant in John D. Owens' lab at University of California, Davis. He earned his M.S. in electrical and computer engineering in 2017 from UC Davis and his B.S. in electronics engineering in 2014 from Sun Yat-sen University, China and The Hong Kong Polytechnic University, Hong Kong SAR, China. He is interested in and mainly working on machine learning deployment on heterogeneous compute devices, e.g., GPU and FPGA, as well as on novel architectures and mobile devices.



Matthew Yih recently completed a master's degree as a graduate student researcher in John Owens' lab at UC Davis, California, working on GPU and FPGA accelerator programming. He graduated from National Central University, Taiwan with a B.S. degree in Electrical Engineering in 2015. After serving in the army for one year, he enrolled at UC Davis and earned the M.S. degree in electrical and computer engineering in 2018. He is interested in heterogeneous computing and machine learning; his master's thesis focuses on the comparison of programming FPGAs and GPUs. He has internship experience at an autonomous vehicle company working on the deployment of CNN models on GPU.

Jeffrey M. Ota is the program lead of Autonomous Driving and Sports Research at Intel Labs, Santa Clara. He was previously senior principal engineer and the director of Sports Technology, Science, Innovation and Wearable Innovation and Technologies at Intel. Before joining Intel, he worked as R&D lead of Digital Sport at Nike, senior applications engineer at NVIDIA, and senior advanced technology engineer at BMW Technology Office USA. He received his M.S. in aeronautical and astronautical engineering in 1995, and B.S. in engineering in 1994, both from Stanford University.

John Owens is the Child Family Professor of Engineering and Entrepreneurship in the Department of Electrical and Computer Engineering at the University of California, Davis, where he leads a research group with a focus on GPU computing, both fundamental primitives and applications. At UC Davis, he earned the Department of Energy Early Career Principal Investigator Award (2004) and is a NVIDIA CUDA Fellow. He taught the 90,000+-student Udacity course "Introduction to Parallel Processing". He is a Distinguished Member of the ACM and Senior Member of the IEEE. John earned his Ph.D. in electrical engineering in 2003 from Stanford University and his B.S. in electrical engineering and computer sciences in 1995 from the University of California, Berkeley.

Pınar Muyan-Özçelik is an assistant professor at the Department of Computer Science of the California State University, Sacramento. She earned her PhD degree in Computer Science from the University of California, Davis. She received her MSc degree from the Department of Computer Science at the University of British Columbia and completed her BSc degree in Computer Engineering at the Ege University. She is interested in conducting research on the use of GPU computing in various domains such as automotive computing, embedded systems, combinatorial optimization, and medical imaging. She has also been conducting studies in the fields of artificial intelligence, machine learning, and comparison of parallel architectures.