Early Experience in Benchmarking Edge AI Processors with Object Detection Workloads

Yujie Hui[1(✉)], Jeffrey Lien[2], and Xiaoyi Lu[1]

[1] Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA
{hui.82,lu.932}@osu.edu
[2] NovuMind Inc., Santa Clara, CA, USA
jlien@novumind.com

Abstract. Nowadays, GPGPU plays an important role in data centers for Deep Learning training. However, GPU might not be suitable for many Deep Learning inference applications, especially for Edge Computing scenarios, due to its high power consumption and high cost. Thus, researchers and engineers have spent a lot of effort on designing edge-side artificial intelligence (AI) processors recently. Because of different edge-side application requirements, edge AI processors are designed with different approaches, which make these processors very diversified. This scenario makes it hard for customers to decide what kind of processors may be more beneficial for their requirements. To provide a selection guidance, this paper proposes a three-dimensional benchmarking methodology and shares the early experience of evaluating three different kinds of edge AI processors (i.e., Edge TPU, NVIDIA Xavier, and NovuTensor) with object detection workloads (i.e., Tiny-YOLO and YOLOv2 with Microsoft COCO dataset). We also characterize a GPU platform (i.e., GTX 1080 Ti) from the three dimensions of accuracy, latency, and energy efficiency. Based on our experimental observations, we find that edge AI processors are able to deliver better energy efficiency (e.g., Edge TPU has the highest energy efficiency in our experiments.), while NovuTensor and Xavier, can also provide comparable performance in latency as GPU. Further, all these edge AI processors can achieve similar accuracy as GPU. The differences among these processors and GPU are less than 3%.

Keywords: Benchmarking · Edge Computing · AI Processor · Deep Learning

1 Introduction

The advancement of Deep Learning has been significantly taking advantage of high-performance computing technologies, such as multi-/many-core processors

This research is supported in part by National Science Foundation grant CCF# 1822987.

© Springer Nature Switzerland AG 2020
W. Gao et al. (Eds.): Bench 2019, LNCS 12093, pp. 32–48, 2020.
https://doi.org/10.1007/978-3-030-49556-5_3

and accelerators, high-speed interconnects, etc. General Purpose GPUs (GPGPUs) recently have become the most popular platforms for Deep Learning training workloads. Many modern data centers in the world provide efficient solutions for Deep Learning training with GPUs. However, Deep Learning infrastructures in data centers might not be suitable for Deep Learning inference workloads, due to the high power consumption and high cost of modern data centers, especially with many of the GPGPUs on cloud servers. Thus, Edge Computing based AI platforms are replacing GPGPUs for many Deep Learning inference applications [29]. The convergence of AI and Edge Computing provides more opportunities to businesses by efficiently running Deep Learning inference applications on edge-side AI platforms.

Edge-side AI platforms are typically equipped with specialized edge AI processors, which can support diverse Deep Learning inference applications, eliminate the high response time from data centers, and provide much cheaper solutions than GPU-based solutions for end users. Thus, more and more research and development activities have been spent on designing edge-side AI processors for Deep Learning inference applications and scenarios [11].

Due to different requirements from diversified Deep Learning applications, edge AI processors are typically designed with different approaches. For example, an edge AI processor may have a larger design space to have more memory. Or an edge AI processor may be designed to support only some specific Deep Learning operations in different requirements of performance, accuracy, and cost. Different designs of edge AI processors may lead to different performances. Thus, it is hard for customers to select an edge AI processor that may be most beneficial for their requirements. This scenario implies that the Deep Learning community needs more standard benchmarks, data-sets, and open research studies to evaluate and compare different edge AI processors for diverse applications.

However, we find that such kind of benchmark-oriented studies for edge AI processors are still not yet prevalent in the community. For example, the study in [1] has run several deep learning models on multiple edge devices, but the work mainly measures the latencies of running object detection workloads on those devices, which does not cover other important aspects for evaluating edge-side AI processors, such as accuracy and energy efficiency.

In order to provide more guidance to the community about how to select more appropriate edge AI processors for end users, this paper proposes a three-dimensional benchmarking methodology (i.e., accuracy, latency, and energy efficiency) on evaluating and comparing different edge AI processors. Deep Learning inference applications are usually customer-facing, which means the inference response time (i.e., latency) and the response accuracy may be more important than other performance metrics. In addition, since these AI processors are designed for Edge Computing platforms, energy efficiency will also be a very important factor for customers in selecting a device.

Based on our benchmarking methodology, we deploy Tiny-YOLO and YOLOv2, which are two popular object detection applications, on three different edge AI platforms (i.e., Edge TPU, NVIDIA Xavier, and NovuMind's NovuTensor). Accurate object detection is one of the most essential challenges for the

Deep Learning community to solve. YOLO-based Deep Learning solutions are extremely fast one-stage object detection systems in the Convolutional Neural Networks (CNN) architecture to detect objects efficiently from an image [37]. To mimic real object detection scenarios, we choose an open data-set from industry, i.e., Microsoft COCO [30] , which provides both training and validation images. We also compare these edge AI processors with a GPU platform (i.e., GTX 1080 Ti) from the three dimensions. These edge AI processors and GPUs typically support different machine learning frameworks and provide deployment tools individually. Note that Edge TPU and NovuTensor do not support some Deep Learning operations (e.g., leaky ReLU) in YOLO's neural network. We retrain a neural network model that is fully supported by these processors in our experiments.

Through our benchmarking experience, the major observations we find include: 1) All edge AI devices can provide similar-accuracy inference results with only 1% to 3% accuracy differences due to lower precision arithmetic. 2) All edge AI processors have better energy efficiency than the GTX 1080 Ti GPU. 3) NovuTensor and Xavier have good and comparable performance in latency as well as energy efficiency. 4) Edge TPU can achieve 6.7X higher energy efficiency but may be 14.79X slower than the GTX 1080 Ti GPU.

Overall, this paper makes the following specific contributions:

- We successfully deploy Tiny-YOLO and YOLOv2 inference applications on multiple edge AI platforms by leveraging deployment tools and modifying standard models to be hardware friendly. Then we compare multiple edge AI processors' performance through running representative object detection workloads (Sect. 5).
- Through our experimental results and observations (Sect. 5.4), we provide guidance to select edge AI platforms for consumers with our proposed three-dimensional benchmarking methodology.
- We share our early experience in benchmarking edge AI processors to the community and encourage more benchmarking efforts to promote the evolution of edge AI processors.

The rest of this paper is organized as follows. Section 2 provides the necessary background for this paper. Section 3 gives a high-level overview of modern edge AI processors. The benchmarking methodology is stated in Sect. 4. Section 5 gives our experiments results and observations. Section 6 introduces related work. Section 7 concludes the paper.

2 Background

In this section, we introduce inference and object detection task in deep learning as well as the edge AI platforms in our experiments.

2.1 Inference in Deep Learning

Training and inference are two important steps in Deep Learning. A Deep Learning model usually has millions of parameters to train. Modern GPGPUs can execute billions of floating-point operations per second (FLOPS), which has made GPGPUs the most popular training platforms for deep learning models. Deep Learning models are usually trained with Big Data in tens of hours to even hundreds of hours. Using multiple GPUs with large training batch sizes can accelerate training time. Trained models can be loaded on edge AI processors or GPUs to do real-time inference. Inference takes new input data to infer results using trained models. Deep Learning training requires high throughput while inference requires low latency. Thus, GPU-based clusters are used to train models because of their parallel computing capabilities, while a single edge device can be used to do inference. Inference brings Deep Learning models to many aspects in our real life. For example, a face recognition system takes a small batch of face images as input each time and infers the identities of the images in a short response time.

2.2 Object Detection and YOLO-Based Systems

Object detection is a typical inference workload that combines localization and classification tasks. For example, an autonomous driving system needs to detect objects in a short time using deep learning models. An object detection system can correctly infer several bounding boxes, which contain the object's location and category in the input image. Each training image is labeled with rectangular bounding boxes that annotate the locations and categories of the objects. Inference will predict multiple bounding boxes. A predicted bounding box with location and category information represents an inferred object. Gradient-based learning approach is used in CNNs to solve object detection tasks [27]. CNN has stronger expressive capability since it has deeper architecture compared with traditional models [41]. Multiple methods on object detection tasks have been proposed in these years such as R-CNN [19], Fast R-CNN [18], YOLO [36], YOLOv2 [37], and SSD [31].

YOLOv2 is a state-of-the-art object detection system [37]. Tiny-YOLO is a lite version of YOLOv2. Tiny-YOLO is faster but less accurate than YOLOv2, since YOLOv2 has more convolutional layers than Tiny-YOLO. Applications of YOLO are able to predict the location and category of objects from any input images. YOLOv2 has two main training components. The first pre-trained model has trained on ImageNet [16] dataset for the classification task. The second component is trained from the previous pre-trained model in the first component. The output of YOLO's neural networks is a feature map with multiple grid cells. Each grid cell predicts five bounding boxes with the probabilities of each class.

3 Overview of Edge AI Processors

This section provides a high-level overview of three different kinds of modern Edge AI processors.

3.1 Edge TPU

Edge TPU from Google provides an end-to-end edge AI solution. Edge TPU integrates onboard TPUs to execute neural networks (e.g., CNNs). TPU consists of Matrix Multiplier Unit (MXU), Unified Buffer (UB), and Activation Unit (AU) to execute convolutional calculations [25]. The key technology in TPU is called Systolic Array. Multiple ALUs are combined together in the systolic array and input data can be reused from one single register. But TPU can only execute a few specific operations in a trade-off between performance and energy efficiency. The workflow to create an Edge TPU compatible model is shown in Fig. 1. A quantized TensorFlow Lite model is converted from a TensorFlow model using some calibration data. TensorFlow Lite is a lightweight version of TensorFlow designed for mobile and embedded devices. Then this quantized model needs to be compiled to a hardware compatible model via the Edge TPU compiler before deployment on the board. Edge TPU only supports INT8 or INT16 based quantization models and the quantization method can be found in this paper [24]. Some pre-trained and pre-compiled models for image classification and object detection tasks are provided in this project [5], which can be deployed on Edge TPU directly.

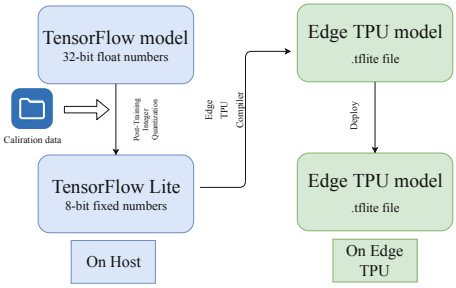


Fig. 1. Workflow to Create an Edge TPU Model

3.2 NVIDIA Xavier

NVIDIA Jetson AGX Xavier is an embedded system on a module, containing a Volta GPU, dual Deep Learning accelerators, a Carmel ARMv8.2 CPU, and 16GB memory [7]. Jetson AGX Xavier is an edge computing device for deploying AI applications and providing end-to-end AI solutions. Xavier allows users to configure operating modes at 10 W, 15 W, and max 30 W. Xavier supports AI software libraries like CUDA [35], cuDNN [12], and TensorRT to improve the inference performance. With the speeding up by TensorRT, Xavier can achieve high performance according to our experiments. NVIDIA TensorRT is a runtime for high-performance Deep Learning inference. Developers can use TensorRT to optimize their Deep Learning models and deploy them on any platforms that have TensorRT runtime from NVIDIA. TensorRT supports INT8 and FLOAT16 optimizations for Deep Learning models as well as the original FLOAT32 data type. TensorRT supports almost all the popular frameworks like TensorFlow, PyTorch, and Caffe. TensorRT takes trained models from those popular frameworks, optimizes the neural network models, and generates light-weight runtime engines for GPUs. Developers only need to deploy the generated runtime engines on the NVIDIA platforms like Jetson Xavier. Figure 2 illustrates the workflow to deploy a model on an NVIDIA platform using TensorRT.

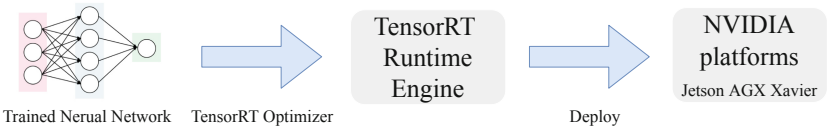


Fig. 2. Deployment using TensorRT

3.3 NovuTensor

NovuMind's NovuTensor uses domain specific architecture focusing on performing 3D tensor computations. Traditional tensor processors like TPU implement convolution layer using 2D matrix multiplication. TPU and GPU calculate 3D convolution multiplication by unfolding the matrix into a 2D matrix and then multiplying the 2D matrix with convolution kernels. NovuTensor's architecture includes a patented design that natively performs 3D tensor computations on the chip, which can avoid the overhead of unfolding 3D tensors into 2D matrices that is inherent in other chips [32]. Processing natively 3D tensor computations achieves high performance and less memory usage. The native tensor processors inside NovuTensor chip convolve 3D tensors of input feature maps with 3×3 convolution kernels. 3×3 convolution kernels are very common in CNNs in previous work [13]. Thus, native tensor processors in NovuTensor can gain much bigger performance improvements for CNN models.

4 Benchmarking Methodology

This section presents workload, platform, and metric selection in our three-dimensional benchmarking methodology.

4.1 Workload Selection

A representative workload for benchmarking AI systems needs a real dataset. Microsoft COCO [30] (MS COCO) is one of the most popular open dataset in the community, which has been widely used in the computer vision field. MS COCO contains hundreds of thousand images from the real world for training, validation, and testing. The training and validation images are labeled with segmentation and bounding boxes. Annotations of MS COCO are arranged in the JSON format. MS COCO mainly contains three tasks (i.e., object detection, key-point detection, and segmentation). MS COCO has 80 categories, which is larger than other datasets like PASCAL VOC [17]. Object instances in MS COCO are also annotated more than those in PASCAL VOC. Developers can use C++ or Python APIs provided by MS COCO to load images and calculate the accuracy [6]. Based on these advantages of MS COCO, this paper chooses it to construct our object detection workloads with YOLOv2 and Tiny-YOLO.

4.2 Platform Selection

We choose GTX 1080 Ti, which is a general-purpose GPU, as our baseline to compare with Edge TPU, Xavier, and NovuTensor, which are edge-side AI platforms using application-specific integrated circuit (ASIC) for inference. These platforms are popular in the industry and use innovative technologies in their designs. Developers can deploy Deep Learning applications on these platforms. They have similar low power consumption (e.g., 20 W for NovuTensor and 30 W for Xavier). We also consider the software support when we select platforms in our experiments. These platforms support different Deep Learning frameworks (e.g., Caffe, TensorFlow, TensorFlow Lite, and PyTorch).

4.3 Metrics and Dimensions

To evaluate the performance of different edge AI processors, we record real-time statistics in our experiments. These statistics help us compare and explore characteristics of these platforms from the three dimensions we have selected in this paper, which include:

– Accuracy: This is typically the most important factor (i.e., how well the processor can infer correct answers) to consider for end users to select appropriate edge AI processors for their applications. We take the accuracy into account since the accuracy may be influenced by different hardware restrictions on Deep Learning operations, such as supporting lower-precision data types. A standard metric for the accuracy dimension is called mean Average Precision (i.e., mAP). mAP has been popularly used to evaluate the accuracy with object detection workloads [17]. mAP is the mean value of Average Precision (AP). One AP value is calculated for one category of images. To introduce mAP further, we first introduce the basic concepts of precision and recall in the following equations.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{1}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2}$$

Based on Eq. 1 and Eq. 2, we can draw a precision-recall curve for all predictions from validation dataset. In the curve, we can choose 11 points in the axis of recall, ranging from 0, 0.1, 0.2 until to 1.0. Then the corresponding 11 precision values will be used to calculate the Average Precision value. Equation 3 calculates an AP value for one category of images based on the 11 interpolated precision values, where P_r is the interpolated precision value in Eq. 1 and r corresponds to the recall in Eq. 2.

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots 1.0\}} P_r \tag{3}$$

$$mAP = \frac{\sum_{i=1}^{N} AP}{N} \tag{4}$$

Then, mAP can be calculated by Eq. 4, where N is the total number of categories. The MS COCO dataset contains 80 categories, which means N equals 80 in Eq. 4. The mAP for MS COCO is the mean value of these 80 average precision values.

- Latency: The time to complete an inference on the input images in one batch is defined as latency. Latency is one of the most important and critical dimension for inference since Deep Learning inference applications are usually customer-facing.
- Energy Efficiency: This means the number of input images can be fully processed per unit-power, which is usually expressed as performance/watt or images/second/watt. Energy efficiency is another key factor to be considered for choosing edge AI devices, since Edge Computing environments usually need lower energy-consumption technologies.

The dimensions and corresponding metrics mentioned above can help us benchmark different edge AI processors. Through these metrics, we are able to evaluate the performance of these edge AI processors and provide selection guidance for end users.

4.4 Experimental Methodology

To capture the real inference execution time on hardware, our experiments split the whole execution flow of running an inference application into three steps:

- Pre-processing time: is the time of the pre-processing step (e.g., normalization of input images).
- Execution time: measures the time of transferring the input feature maps into devices, execution, and receiving the output feature maps from devices.
- Post-processing time: is the time of the post-processing step (e.g., parse the output tensors to get readable prediction results).

To evaluate the performance of edge AI processors, only the execution time should be accounted. As for YOLO-based applications, the pre-processing step includes getting input images and normalization. The post-processing step gets a 3-dimensional tensor from neural network's output and decodes this tensor to get the prediction results.

5 Experiments

This section presents our experimental configuration and setup to perform the benchmarking on edge AI processors.

5.1 Hardware Configuration

The specifications of edge AI processors in our experiments are shown in Table 1. Table 1 illustrates the theoretical metrics for comparing these edge AI processors. Tera Operations per Second (TOPs) is a common metric for evaluating the throughput of AI processors, which represents the processing capability of an AI processor.

5.2 Setup

We deploy Tiny-YOLO and YOLOv2 applications on three edge AI platforms as shown in Table 1. The YOLO-based applications include three components, which are image pre-processing, trained CNN models, and inference post-processing. We select five thousands images in MS COCO's 2014 validation dataset as the inference workload for all AI platforms. We evaluate the AI devices with input images of both regular and large sizes. Thus, input images will be re-sized to 416×416 and 1024×1024 before feeding them into the neural network models. Every edge AI platform processes the same five thousands validation images and generates a JSON file with all the inferred bounding boxes. We use MS COCO API to parse the JSON file and calculate the mAP, which is the common accuracy evaluation metric for object detection task [6].

Edge TPU: We first convert Tiny-YOLO and YOLOv2 applications from Darknet framework to TensorFlow framework using DarkFlow [2], as Egde TPU only supports TensorFlow Lite. Darknet is the framework that supports original YOLO-based applications [3]. A model in Darknet includes two components, a configuration file and a binary file. The configuration file defines the architecture of the neural network and the binary file is the trained weights of convolutional kernels. We combine the configuration file and binary file into a TensorFlow SavedModel format file via DarkFlow. TensorFlow provides two ways to do quantization (i.e., post-training quantization and quantization-aware training). Post-training quantization creates a small model by using 8-bit values, but during inference, it is converted back to 32-bit floats. Quantization-aware training creates fake quantization nodes containing the minimum and maximum values of layers' weights during training. Edge TPU utilizes the quantized values to do inference. However, developers have to modify the training source code to do training-aware quantization. We use a new quantization toolkit without modifying training source code [8]. We feed 80 calibration images into the models, which can quantize and generate a TensorFlow Lite model. Then we compile the TensorFlow Lite model to an Edge TPU model that has instructions supported by Edge TPU.

Edge TPU does not support dynamic tensor sizes and only supports 3-dimensional tensors. If a tensor has more than three dimensions, only the three innermost dimensions can have a size greater than one. In this case, the batch size for inference could only be one when the input image has three channels.

YOLOv2 and Tiny-YOLO use leaky rectification (leaky ReLU) [26] as the activation function for all the convolutional layers. But leaky ReLU is not supported by Edge TPU. We replace leaky ReLU with ReLU and retrain the models, using four GTX 1080 Ti GPUs. Reorganization and route layers are also not supported by Edge TPU. Before deploying the models on Edge TPU, it is necessary to make sure that all the operations are supported with INT8 data type, because operations that do not have quantized implementations will not work with Edge TPU.

In addition, Edge TPU only supports 8-bit input data. As a consequence, the input image's pixels can not be normalized to Float32 data type between 0 and 1. So we modify the weights of the first convolutional layer to additionally transform input tensors to appropriate forms without normalization, which can match with the input requirements of Edge TPU.

Table 1. Specifications of platforms

Specification	Edge TPU	Xavier	NovuTensor	1080 Ti
Precision	INT8	INT8/FP16/FP32	INT8	FP32
TOPS	4	22.6/11.3/1.3	15	11.3
Memory	1 GB 32-bit LPDDR4	16 GB 256-bit LPDDR4X	2 GB 128-bit DDR4	11 GB 352-bit GDDR5X
Power (watt)	–	10/15/30	20	250
Process (nm)	–	12	28	16

Note that TOPS stands for Tera Operations per Second. We found or calculated from official specifications according to the corresponding precision. The power and process of Edge TPU are not reported by Google. We approximate the power as 2.5 W by our experimental devices.

Xavier: Our experiments deploy NVIDIA's deepstream reference applications [4], which contain YOLO-based applications implemented by TensorRT, on Xavier platforms. TensorRT 5.0.3 is installed in our experimental Xavier device. TensorRT is able to parse Deep Learning models and deploy optimized models on Xavier. TensorRT provides a plugin layer for developers to implement customized operations that are not supported by TensorRT (e.g., leaky ReLU). Using the plugin layer, we are able to deploy the original YOLO-based application on Xavier without modifying and retraining the model. We evaluate Xavier in both 15-W and 30-W modes. We set the INT8 data type in the deepstream application to compare with other hardware devices.

NovuTensor: NovuMind's NovuTensor is a special-purpose processor for AI inference applications. Designed for convolutional neural networks, it achieves high throughput and low latency for convolution computations. NovuTensor convolves the feature map with 3×3 kernels using the native tensor processors.

In our experiments, we deploy Tiny-YOLO and YOLOv2 applications on NovuTensor using NovuSDK. Hardware-friendly YOLOv2 and Tiny-YOLO models are deployed on NovuTensor by replacing the activation functions by ReLU and removing reorganization and route layers. NovuSDK provides APIs for developers to control the hardware.

5.3 Results

Figure 3 illustrates the mAPs of Tiny-YOLO and YOLOv2 applications running on different edge AI processors with 416 × 416 resolution input images. After quantization, the accuracy of YOLOv2 drops 2% to 3% and the accuracy of Tiny-YOLO drops 1% to 2% on all edge AI processors compared to the accuracy on GPU. Low precision arithmetic like INT8 reduces accuracy but accelerates the execution time [22]. The differences of accuracy degradation among different edge AI processors are varied, which is because these edge AI processors implement the quantization in different ways.

Fig. 3. Accuracy of YOLOv2 and Tiny-YOLO with 416 × 416 image size

Figure 4 shows the latency of processing a batch of input images. Edge TPU is slower than other edge AI processors, which are 28.74 ms and 94.37 ms for Tiny-YOLO and YOLOv2 applications, respectively. The architecture of YOLOv2 is more complex than Tiny-YOLO. The latency of Edge TPU increases more than other edge AI processors, when the neural network's architecture becomes complex. NovuTensor can achieve 14.08 ms and 24.78 ms for one batch of input images, which is faster than Xavier in the 15-W mode (i.e., 16.68 ms and 38.39 ms) and similar with (i.e., 14.16 ms for Tiny-YOLO) or slower than (i.e., 17.85 ms for YOLOv2) Xavier in the 30-W mode, respectively.

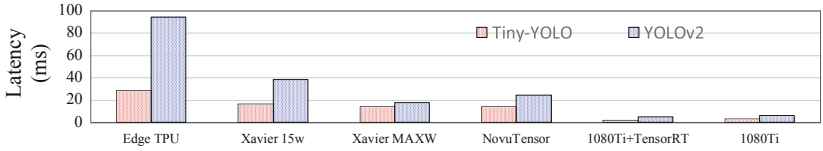


Fig. 4. Latency of YOLOv2 and Tiny-YOLO with 416 × 416 image size

Figure 5 illustrates the energy efficiency by taking the energy consumption into account. Edge TPU achieves relatively higher energy efficiency considering its very low power usage and GTX 1080 Ti seems not very energy-efficient since its large power consumption. NovuTensor has better energy-efficiency than Xavier in both 15-W and 30-W modes for the YOLOv2 application. NovuTensor also shows better energy-efficiency than Xavier in the 30-W (max) mode and comparable energy-efficiency as Xavier in the 15-W mode for the Tiny-YOLO application.

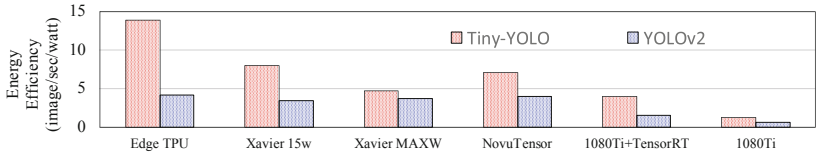


Fig. 5. Energy Efficiency of YOLOv2 and Tiny-YOLO with 416 × 416 image size

We also evaluate the performance of these edge AI processors with larger input images (i.e., 1024 × 1024), as shown in Fig. 6. All the devices can provide similar-accuracy inference results compared with GPU, thus the accuracy results are not shown here. Through our experiments, Xavier and NovuTensor can achieve comparable low latency and high energy efficiency.

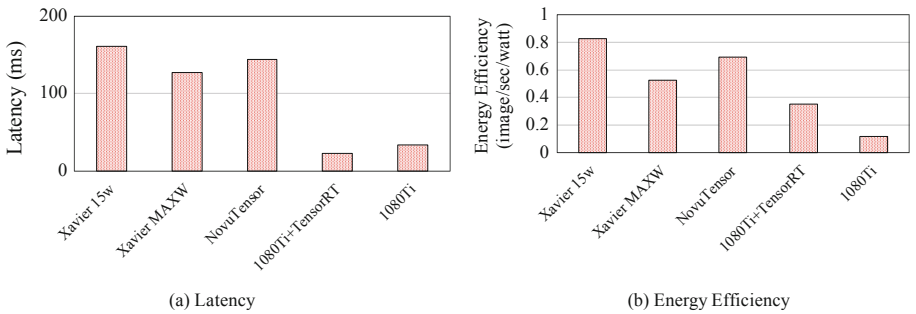


Fig. 6. Performance of YOLOv2 with 1024 × 1024 image size

5.4 Observations and Summary

We combine all the measured results in our experiments and demonstrate them in three-dimensional charts as shown in Fig. 7. In order to compare edge AI processors clearly, we normalize the experimental results into these three dimensions. Performance, one of the three dimensions, is defined as the reciprocal of latency. In this way, if an edge AI processor can achieve a bigger score in the

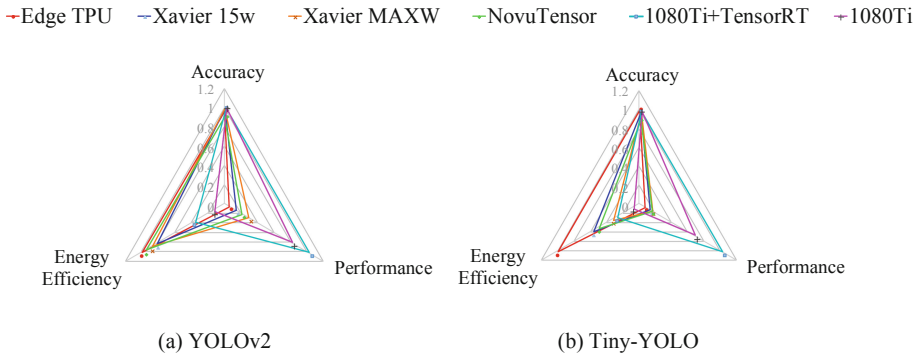
three-dimensional charts, it means the edge AI processor can be a better choice in the corresponding dimension. Through these two radar charts, we obtain some interesting observations as follows:

- All these edge AI processors are able to provide similar-accuracy inference results compared with the GTX 1080 Ti. Quantization from FP32 to INT8 data type during inference causes an accuracy drop around 1% to 3%.
- These edge AI processors are slower than GTX 1080 Ti GPU due to less computing cores and less power consumption. Edge TPU is **9.5X** and **14.79X slower** than GTX 1080 Ti with running Tiny-Yolo and YOLOv2, respectively. Despite Xavier and NovuTensor are slower than the GPU, Xavier is **2X** and **5.28X faster** than Edge TPU in the max power mode, and NovuTensor is **2.04X** and **3.8X faster** than Edge TPU, respectively, when running with Tiny-Yolo and YOLOv2 applications.
- Edge TPU exceeds other edge AI processors in energy efficiency, and it delivers **2.9X** and **1.13X higher** energy efficiency than Xavier as well as **1.96X** and **1.04X higher** energy efficiency than NovuTensor for Tiny-Yolo and YOLOv2, respectively.

As specified by our observations, these edge AI processors can perform Tiny-YOLO and YOLOv2 applications within a 3% accuracy drop. NovuTensor and Xavier achieve low latency and relatively high energy efficiency for object detection workloads. Edge TPU has the advantage of energy efficiency.

6 Related Work

This section presents related work about benchmarking AI processors.



(a) YOLOv2 (b) Tiny-YOLO

Fig. 7. Comparison of factors on YOLOv2 and Tiny-YOLO with 416 × 416 image size

6.1 Modern AI Processors

Deep learning especially neural networks have been proven in many state-of-the-art application systems to solve classification and object detection tasks [23,28]. The opportunity to design AI processors with high performance and efficiency attracts more and more researchers and engineers [10]. Multiple methods can execute deep learning workloads on modern AI processors, such as GPUs [14], FPGAs [9], or ASICs [11,20,25]. Some customized AI processors have been effectively studied such as k-NN accelerator on FPGA [39], k-NN classifier of IP cores design [34,38].

6.2 Edge AI Benchmarking

Due to the emergence of different edge AI processors for inference, we do see the effort of benchmarking these devices in the community. AIoT Bench [33] contains benchmarks for image classification, speech recognition, transformer translation, and micro workloads on Android-based systems and Raspberry Pi. The urgent requirements of edge AI benchmarking are also discussed in this paper [33]. The study in [1] runs several Deep Learning models on multiple edge devices and it mainly measures the latency of object detection workloads. A survey on different Deep Learning benchmarks summarizes multiple popular available benchmarks in the community [40]. EdgeAI Bench [21] contains four different benchmarking frameworks. These frameworks aim at benchmarking four specific scenarios using Deep Learning technologies in Edge Computing environments. EdgeBench [15] compares two serverless edge computing services on a standard Raspberry Pi 3B model. Compared to these related studies, this paper focuses on benchmarking three modern edge AI processors (i.e., Edge TPU, NVIDIA Xavier, and NovuTensor) with object detection workloads.

7 Conclusion and Future Work

We propose a benchmarking methodology to systematically evaluate three different kinds of edge AI processors (i.e., Edge TPU, NVIDIA Xavier, and NovuTensor) from the three dimensions of accuracy, latency, and energy efficiency. Based on our experimental results, we observe that NovuTensor and Xavier can provide comparable performance in latency and energy efficiency, which satisfy the major requirements for Deep Learning inference applications. They also have comparable performance in latency compared with GTX 1080 Ti. Edge TPU consumes less energy but is much slower for inference, which may influence the consumers' usage experience. Accuracy is important but seems not a major factor to make a selection on these edge AI processors, since they all can provide similar inference accuracy.

In the future, we will evaluate more combinations of different neural networks and edge AI platforms. We plan to propose an easy-to-use benchmarking toolkit for different edge AI processors.

References

1. Benchmarking Edge Computing. https://medium.com/@aallan/benchmarking-edge-computing-ce3f13942245
2. DarkFlow. https://github.com/thtrieu/darkflow
3. Darknet. https://github.com/pjreddie/darknet
4. Deepstream Reference Applications. https://github.com/NVIDIA-AI-IOT/deepstream_reference_apps
5. Models Built for Edge TPU. https://coral.withgoogle.com/models/
6. MS COCO API. https://github.com/cocodataset/cocoapi
7. NVIDIA Jetson AGX Xavier. https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit
8. Post-Training Integer Quantization. https://medium.com/tensorflow/tensorflow-model-optimization-toolkit-post-training-integer-quantization-b4964a1ea9ba
9. Chakradhar, S., Sankaradas, M., Jakkula, V., Cadambi, S.: A dynamically configurable coprocessor for convolutional neural networks. In: ACM SIGARCH Computer Architecture News, vol. 38, pp. 247–257. ACM (2010)
10. Chen, T., et al.: BenchNN: on the broad potential application scope of hardware neural network accelerators. In: 2012 IEEE International Symposium on Workload Characterization (IISWC), pp. 36–45. IEEE (2012)
11. Chen, Y., Chen, T., Zhiwei, X., Sun, N., Temam, O.: DianNao family: energy-efficient hardware accelerators for machine learning. Communi. ACM **59**(11), 105–112 (2016)
12. Chetlur, S., et al.: cuDNN: efficient primitives for deep learning. arXiv preprint arXiv:1410.0759 (2014)
13. Ciresan, D.C., Meier, U., Masci, J., Gambardella, L.M., Schmidhuber, J.: Flexible, high performance convolutional neural networks for image classification. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)
14. Coates, A., Huval, B., Wang, T., Wu, D., Catanzaro, B., Andrew, N.: Deep learning with COTS HPC systems. In: International Conference on Machine Learning, pp. 1337–1345 (2013)
15. Das, A., Patterson, S., Wittie, M.: Edgebench: benchmarking edge computing platforms. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), pp. 175–180. IEEE (2018)
16. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255. IEEE (2009)
17. Everingham, M., Gool, L.V., KI Williams, C., Winn, J., Zisserman, A.: The pascal visual object classes (VOC) challenge. Int. J. Comput. Vis. **88**(2), 303–338 (2010)
18. Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1440–1448 (2015)
19. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
20. Han, S., et al.. EIE: efficient inference engine on compressed deep neural network. In: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pp. 243–254. IEEE (2016)
21. Hao, T., et al.: EdgeAI bench: towards comprehensive end-to-end edge computing benchmarking. In: 2018 Bench Council International Symposium on Benchmarking, Measuring and Optimizing (Bench 2018) (2018)

22. Hashemi, S., Anthony, N., Tann, H., Bahar, I.R., Reda, S.: Understanding the impact of precision quantization on the accuracy and energy of neural networks. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, pp. 1474–1479. IEEE (2017)
23. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012)
24. Jacob, B., et al.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2704–2713 (2018)
25. Jouppi, N.P., et al.: In-datacenter performance analysis of a tensor processing unit. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pp. 1–12. IEEE (2017)
26. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
27. Wick, C.: Deep learning. Informatik-Spektrum **40**(1), 103–107 (2016). https://doi.org/10.1007/s00287-016-1013-2
28. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
29. Lee, Y.-L., Tsung, P.-K., Wu, M.: Techology trend of edge AI. In: 2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), pp. 1–2. IEEE (2018)
30. Lin, T.-Y., et al.: Microsoft COCO: common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8693, pp. 740–755. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10602-1_48
31. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
32. Lu, C.P., Tang, Y.-S.: Native Tensor Processor, and Partitioning of Tensor Contractions. https://patentscope.wipo.int/search/en/detail.jsf?docId=US225521272&tab=NATIONALBIBLIO
33. Luo, C., et al.: AIoT bench: towards comprehensive benchmarking mobile and embedded device intelligence. In: 2018 Bench Council International Symposium on Benchmarking, Measuring and Optimizing (Bench 2018) (2018)
34. Manolakos, E.S., Stamoulias, I.: IP-Cores design for the kNN classifier. In: Proceedings of 2010 IEEE International Symposium on Circuits and Systems, pp. 4133–4136. IEEE (2010)
35. Nickolls, J., Buck, I., Garland, M.: Scalable parallel programming. In: 2008 IEEE Hot Chips 20 Symposium (HCS), pp. 40–53. IEEE (2008)
36. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788 (2016)
37. Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7263–7271 (2017)
38. Stamoulias, I., Manolakos, E.S.: Parallel architectures for the kNN classifier-design of soft IP cores and FPGA implementations. ACM Trans. Embedded Comput. Syst. (TECS) **13**(2), 22 (2013)

39. Yeh, Y.-J., Li, H.-Y., Hwang, W.-J., Fang, C.-Y.: FPGA implementation of kNN classifier based on wavelet transform and partial distance search. In: Ersbøll, B.K., Pedersen, K.S. (eds.) SCIA 2007. LNCS, vol. 4522, pp. 512–521. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73040-8_52
40. Zhang, Q., et al.: A survey on deep learning benchmarks: do we still need new ones? In: 2018 Bench Council International Symposium on Benchmarking, Measuring and Optimizing (Bench 2018) (2018)
41. Zhao, Z.-Q., Zheng, P., Xu, S.-T., Wu, X.: A review. IEEE Transactions on Neural Networks and Learning Systems, Object Detection with Deep Learning (2019)