PAPER

Benchmarking Modern Edge Devices for AI Applications*

Pilsung KANG†a, Member and Jongmin JO†b, Nonmember

SUMMARY  AI (artificial intelligence) has grown at an overwhelming speed for the last decade, to the extent that it has become one of the mainstream tools that drive the advancements in science and technology. Meanwhile, the paradigm of edge computing has emerged as one of the foremost areas in which applications using the AI technology are being most actively researched, due to its potential benefits and impact on today’s widespread networked computing environments. In this paper, we evaluate two major entry-level offerings in the state-of-the-art edge device technology, which highlight increased computing power and specialized hardware support for AI applications. We perform a set of deep learning benchmarks on the devices to measure their performance. By comparing the performance with other GPU (graphics processing unit) accelerated systems in different platforms, we assess the computational capability of the modern edge devices featuring a significant amount of hardware parallelism.

key words: edge computing, deep learning, performance benchmark, GPU (graphics processing unit), TPU (tensor processing unit)

1. Introduction

The development of AI and machine learning has been substantial over the past decade, so that it is having a transformative impact nowadays across most industry sectors, spanning education, energy, finance and, most recently, medicine[1]. For instance, it was reported that out of the 9,262 U.S. patents received by IBM inventors in 2019, more than 1,800 patents (nearly 20 percent) were AI-related[2].

In the meantime, the ever-increasing growth of IoT (Internet of Things) devices is generating a tremendous amount of diverse types of real-time data every day. Due to the exponential speed of the increasing data, it is estimated that 79.4 zettabytes of data will be created by 41.6 billion networked devices by 2025, which amounts to every person generating 20 megabytes every minute[3]. The “big data” on networked IoT devices are typically transferred to central servers located on the cloud computing platforms for large-scale data mining and pattern analysis applications. However, it is expected that cloud infrastructure will not be able to accommodate the enormous data produced by IoT devices, mainly due to the network bandwidth bottleneck that cannot adequately process the sheer amount of data. In addition, network latency can also be a significant issue for time-critical applications where processed results are imperatively needed but getting back the results from the cloud server may become too late due to latency[4].

The central idea of the edge computing paradigm is aimed at addressing these issues[5]. By performing intended operations with the data right at the “edge” of the network, instead of simply sending the data to the center of the cloud, the response time can be improved without having to listen to the cloud server. In this work, we examine major entry-level edge devices of the latest generation to evaluate the state of the art of the edge computing technology for AI applications. Using Google Coral Dev Board[6] and NVidia Jetson Nano Developer Kit[7] (Jetson Nano for short) as our target edge devices, we perform a series of benchmarks based on deep neural networks to evaluate their performance and capabilities. Although the devices are limited in terms of computing power and hardware resources, both of them are powered by accelerators to enhance their performance behavior at the edge computing environments. In particular, Coral Dev Board is equipped with the Edge TPU which is specially designed for deep learning applications. In contrast, Jetson Nano is accelerated by the GPU coprocessor which is traditionally more suitable for scientific applications.

There is a sizable amount of benchmark reports on the edge devices[8]–[11]. However, some reports are already a bit outdated and others miss benchmarks on the latest offerings from the major edge device vendors. Moreover, most reports examine only a particular subset of modern edge devices, thus lacking a comprehensive comparison between these devices with respect to the AI domain applications. Extending our previous work[12] that focused on the evaluation of Jetson Nano only, we provide an extensive comparison between Coral Dev Board and Jetson Nano by evaluating their performance behavior using a diverse set of CNN (convolutional neural network) benchmarks.

In this paper, we make the following contributions:

- We present a comparative analysis on recent accelerator-based edge devices specialized for the AI application domain. We choose Google Coral Dev Board and NVidia Jetson Nano as our targets and perform an assorted CNN benchmarks to assess their performance behavior and capabilities for AI applications. Our evaluation perspectives include inference speed, CNN model complexity, and memory utilization.
We evaluate the Jetson Nano device by analyzing its performance behavior against other GPU-accelerated systems over different platforms. These systems include an NVIDIA Jetson TX2 embedded system, an RTX 2080 Super desktop system, and a Tesla V100 high-end system. Comparing Jetson Nano’s performance with more powerful GPU-based systems helps to highlight the strengths and weaknesses of the edge device. This contribution is based on our previous report [12], significantly extended with more thorough experiments and in-depth analysis of the performance benchmark results from the GPU-based systems.

We provide a parallel performance evaluation of the GPU-accelerated systems in terms of the CNN benchmarks. GPUs are widely used for accelerating AI applications in different platforms from embedded to high-performance computing. By varying the batch size – one of the key parameters in neural network processing – of the CNN benchmarks, we measure the throughput performance of diverse GPU-based systems as well as the Jetson Nano device and evaluate the performance behavior from parallelizing neural networks.

The remainder of this paper is organized as follows. Section 2 compares architectural differences between Google Coral Dev Board and NVIDIA Jetson Nano devices, and provides the hardware and software setup information for the benchmarks. Section 3 describes performance criteria for evaluating the devices. Section 4 analyzes benchmarks results and compares the performance between Google Coral Dev Board and NVIDIA Jetson Nano. Section 5 evaluates NVIDIA Jetson Nano in comparison with other GPU-accelerated systems. Section 6 contrasts our work with related research projects. Finally, we summarize our work and make conclusions in Sect. 7.

2. Setup: Hardware and Software for Benchmarks

In this section, we compare Google Coral Dev Board and NVIDIA Jetson Nano architectures, present hardware and software environments where we perform the benchmarks, and also briefly describe the CNNs used in the benchmarks and their characteristics.

2.1 Target Edge Devices: Jetson Nano and Coral Dev Board

2.1.1 NVIDIA Jetson Nano Developer Kit

The Jetson Nano Developer Kit is one of the latest offerings from NVIDIA for edge computing. Jetson Nano’s GPU is based on the Maxwell microarchitecture (GM20B). It comes with one streaming multiprocessor (SM) with 128 cores, which allows to run multiple neural networks in parallel. Jetson Nano has 4GB of LPDDR4 DRAM which is integrated on the board to be shared by the CPU (Quad-core ARM Cortex-A57) and the GPU. To cut the cost, Jetson Nano is designed to include only the Gigabit Ethernet without additional network connectivity such as Wi-Fi and Bluetooth. NVIDIA CUDA [13] is the primary programming model for describing parallel operations using the cores in the GPU. Like other NVIDIA GPU products, a set of sophisticated programming libraries is also available for Jetson Nano, which includes cuDNN and cuBLAS.

2.1.2 Google Coral Dev Board

Coral is a platform by Google for building AI applications on an edge device [14], and Google Coral Dev Board is one of the latest offerings which features the “edge” version of the TPU (tensor processing unit) [15], an application-specific integrated circuit (ASIC) designed for accelerating neural network machine learning, particularly using Google’s own TensorFlow framework [16]. The Edge TPU operates as a coprocessor in a edge device, which is the same for the GPU in Jetson Nano. The systolic array architecture adopted for the design of the Edge TPU allows for an efficient aggregation of tens of thousands of ALUs (arithmetic logic units) and faster data transfer rate between the TPU and the memory. However, unlike Jetson Nano’s GPU module that can perform a fairly wide range of computations in different applications (hence referred to as the general-purpose GPU), the Edge TPU is fine-tuned for matrix operations which are very frequent in neural network machine learning.

Table 1 compares the system configuration between Google Coral Dev Board and NVIDIA Jetson Nano. While they are both equipped with quad-core ARM processors, they adopt different types of accelerators: Coral Dev Board features the Edge TPU to accelerate deep learning applications and Jetson Nano is fueled by the 128-core GPU coprocessor to be flexibly used for general purpose applications. Jetson Nano has an advantage in that it provides 4x larger memory than Coral Dev Board. On the software side, Coral Dev Board comes with Mendel Linux 4 (a Debian Stretch derivative) as the OS and supports only TensorFlow Lite, where it is required that CNN models

| Processor | Google Coral Dev Board | NVIDIA Jetson Nano Dev. Kit |
|-----------|------------------------|-----------------------------|
| SoC       | XNPU MX230 SoC         | Quad Cortex-A57             |
| CPU       | Edge TPU               | (Quad Cortex-A57 + Cortex-M4F) |
| Memory    | 1GB LPDDR4             | 4GB LPDDR4                  |
| Flash Memory | 8GB eMMC               | 16GB eMMC                   |
| LAN       | Gigabit Ethernet      | Gigabit Ethernet            |
| Wi-Fi     | 802.11a/b/g/n/ac 2.4GHz | No                         |
| Bluetooth | 4.2 with BLE support  | No                          |
| OS        | Mendel Linux 4.0      | Linux for Tags 32.2.2       |
| Supported Frameworks | TensorFlow Lite | Major Frameworks (TensorFlow, PyTorch, and Caffe) |
| TDP       | 12.5 ~ 13W             | 10W                         |
| Cost      | $129.99                | $99                         |
are trained using quantization aware training [17] and the parameters are represented as int8 or uint8. By contrast, Jetson Nano uses Linux for Tegra v32.2.2 (a Ubuntu 18.04 derivative) as the OS with other programming libraries such as CUDA, cuDNN, and TensorRT, which support a variety of different deep learning frameworks including PyTorch, Caffe, and Keras as well as TensorFlow.

### 2.2 CNN Benchmarks

Figure 1 shows a general description of the workflow of CNN build, deployment, and inference on edge devices, where the steps are different between Coral Dev Board and Jetson Nano. For Jetson Nano, as shown in Fig. 1 (a), the model build stage first parses the CNN architecture specification file given as input and constructs the network, where prototxt (plaintext protocol buffer schema) is typically used for defining model parameters in the specification. Then, NVidia TensorRT compiles and optimizes the pre-trained model for the execution environment, thus completing the model build process.

The inference stage performs classification using the conventional operations in CNN such as convolution and pooling. In general, the model build process consumes much larger amount of time than inference. In this paper, we evaluate only the inference performance in the benchmark since the supported operations are executed on the TPU whereas the others are executed on the CPU. In addition, the Compiler assigns a 64 bit “caching token” to the TensorFlow Lite model, so that it can be compared with the token of the data in the TPU’s SRAM. If the tokens match, the Runtime uses the cached data, or else, the Runtime invalidates the cache and uses the model’s parameter data. We note that the TPU’s SRAM operates like more of a scratch pad allocated by the Compiler rather than the traditional cache hardware.

Table 2 summarizes the CNN benchmarks and their characteristics used in our experiments. Specifically, we use GoogLeNet [18], Inception V2 [19] and V4 [20], MobileNet V1 [21] and V2 [22], and SSD MobileNet V2.

### 3. Evaluation Perspectives

#### 3.1 Inference Speed

Since deep learning applications at the edge are mainly used for object detection and image classification, the application performance based on inference speed is most important. In our evaluation experiments, we report frames per second (FPS) as inference speed of each benchmark. As 30 FPS is commonly considered as “real-time” in most video applications, we place an emphasis on the CNN benchmarks with more than 30 FPS on Google Coral Dev Board and NVidia Jetson Nano. For statistical validity, we report inference speed of each benchmark averaged over 3 runs.

#### 3.2 Model Complexity

Together with the inference speed for a CNN, we consider the model complexity of the CNN benchmark in analyzing the device performance. Although the inference speed
is usually inversely proportional to the model complexity, some other factors, such as architectural characteristics of a device and CNN model structure, can be critical to a degree where they significantly affect the performance. In our evaluation, we count the number of learnable weight parameters in a CNN and report it as the representative complexity of the CNN model. We specifically consider the number of parameters in the convolution and the fully connected layers, ignoring “hyperparameters” in other subsidiary operations such as pooling and padding.

3.3 Memory Usage

We measure the amount of memory used when the benchmark is executed on the devices. As we expect that the memory usage is directly affected by CNN model complexity, quantitative evaluation of the memory usage is critical in evaluating the capacity and capability of the edge device over different CNN benchmarks. Specifically, we measure by sampling the memory usage observed with $\text{top}$, averaging over the sample outputs while the benchmark process is running.

4. Evaluation: Coral Dev Board and Jetson Nano

We compare Google Coral Dev Board and Jetson Nano based on the performance behavior across the CNN benchmarks used for evaluation. For the experiments, we use the reference models provided by Google [23] and NVidia [24], where the ImageNet dataset was used to train CNNs for image classification and the MS COCO dataset was used to train SSD MobileNet V2 for object detection. We set the batch size at inference to 1 since processing multiple images in parallel turns out to be ineffective for the edge devices. For instance, the GPU utilization of Jetson Nano reached 98% even for batch size 1, indicating that increasing the batch size over 1 would not be profitable.

4.1 Inference Speed

Figure 2 compares the CNN benchmark inference speed between Coral Dev Board and Jetson Nano. For Coral Dev Board, MobileNet V1 shows the highest performance of 417 FPS, while Inception V4 shows the worst performance of 10 FPS. Similarly, Jetson Nano shows the best speed for MobileNet V1 with 80 FPS and the worst for Inception V4 with 10 FPS. Overall, the performance behavior between Coral Dev Board and Jetson Nano is almost the same in that the inference speed is directly dependent upon the CNN model complexity and inversely proportional to the number of parameters of CNN. For instance, both the devices show the lowest speed for Inception V4 which is structured with the largest amount of weight parameters.

However, it is clear from the results that Coral Dev Board shows dominating performance for relatively simple CNNs compared with Jetson Nano. Specifically, it shows 417 FPS for MobileNet V1 (4.24×10⁶ params) and 385 FPS (3.75×10⁶ params) for MobileNet V2, which is over 5× faster than Jetson Nano. This performance improvement is mainly due to the 8MB of SRAM in the Edge TPU, some portion of which is allocated by the Edge TPU Compiler to cache the model parameters. The smaller the model parameters in a CNN, the better it becomes for the model to fit in to the Edge TPU SRAM cache, thus resulting in substantially higher performance.

Table 3 shows the cache behavior of the Edge TPU for each CNN for classification, excluding SSD MobileNet V2 for object detection. In this table, the model complexity is quantitatively characterized by the parameter data size in megabytes (MB). We specifically show the amount of SRAM available for caching, the amount of actual cached data, and the uncached amount along with model complexity (small/medium/large) and inference speed (FPS). The cached and the uncached amounts are also shown in parenthesis as the relative percentage over the whole parameter.
size. The cache effect in Edge TPU is obvious from the table that the simpler the CNN model, the more completely it fits in the SRAM cache, thereby achieving a significant speedup. For instance, all the 3.75 MB of the parameter data of MobileNet V2 are entirely cached in the Edge TPU SRAM to show 385 FPS. In contrast, as for Inception V4, only 14% of the whole parameters are cached during the benchmark execution to show just 10 FPS. Here, we note that the speedup due to the SRAM cache is super-linear over model simplicity. Between the two CNNs above, for example, the difference in parameter size is about 11×, whereas the difference in inference speed is more than 38×.

By contrast, the speedup for Jetson Nano is not as dramatic as Coral Dev Board. Considering the difference in model complexity between MobileNet V2 and Inception V4 is 11×, Jetson Nano shows just 6× speedup, which is significantly worse than Coral Dev Board. Although this is due to the very small L2 cache of Jetson Nano (0.26 MB), we doubt that simply increasing the cache size will greatly improve the performance, because managing an increased cache may incur serious overhead such as the overhead of cache coherence protocols.

4.2 Memory Usage

Figure 3 plots the amount of memory used during each CNN benchmark execution between Coral Dev Board and Jetson Nano. Coral Dev Board used from 4.2% (MobileNet V1) to 13.1% (Inception V4), which amounts to using from 42 MB to 131 MB considering Coral Dev Board is equipped with 1GB of DRAM. This is an order-of-magnitude improvement in memory usage compared to Jetson Nano’s approximately 30% memory consumption which amounts to using close to 1.2GB of memory over the course of benchmarks.

We estimate that the noteworthy reduction in memory usage by Coral Dev Board is primarily due to the systolic array architecture of the Edge TPU, which allows to pipeline the computation over several multiply-accumulate (MAC) units, through which each multiplication result is passed to next MACs that can perform multiplication and summation at once. Hence, the Edge TPU does not need to access the memory to hold or fetch temporary results is avoided as the GPU does. The use of quantized models in benchmarking Coral Dev Board adds to the reduction in memory usage compared to Jetson Nano. While it is typical to use 32-bit floating points for model parameters, the Edge TPU uses 8-bit integer parameters quantized for proper execution, which results in saved memory. However, using a quantized model is less precise and needs careful examination so as not to significantly hurt the inference accuracy of neural networks depending on applications.

4.3 Power Consumption

Table 4 shows the power consumption measured under the benchmark execution and at idle times between Coral Dev Board and Jetson Nano. We measured the power consumption both at the benchmark execution and at idle times for Coral Dev Board and Jetson Nano. During benchmark execution, Coral Dev Board used 5.5 Watts on average, which is 10% less than Jetson Nano’s. However, Coral Dev Board was measured to use 4.8 Watts at idle times, which amounts to more than 2× power consumed by Jetson Nano. Therefore, our measurement results show that Coral Dev Board’s system-wide power behavior has room for further improvement at idle times. Considering performance-per-watt, we can summarize that Coral Dev Board shows nearly 4.5× better behavior for simple networks, while performing on par with Jetson Nano for larger ones.

4.4 Inference Accuracy

We perform a brief comparison of inference accuracy between Jetson Nano and Coral Dev Board using two classification networks, GoogLeNet and Inception V4, which are trained on the ImageNet dataset with 1000 classes. For measuring inference accuracy, we use 100 images collected randomly on the Internet. Table 5 shows the accuracy measurement results between Jetson Nano and Coral Dev Board. We observe that Jetson Nano is better for GoogLeNet, but, on the contrary, Coral Dev Board is better for Inception V4. The accuracy rate differences are also nearly the same for both the CNN models. The measurement results show that
the 8-bit quantization scheme of Coral’s Edge TPU can provide comparable performance in terms of accuracy without resulting in substantial degradation in inference quality.

## 5. Benchmark Results and Analysis: GPU Systems

In this section, we evaluate Nvidia Jetson Nano by comparing its performance and characteristics with other NVidia GPU-based systems across different platforms.

### 5.1 Hardware and Software for Benchmarks

In addition to NVidia Jetson Nano as our primary edge device, we choose three GPU-accelerated systems for our benchmark: NVidia Jetson TX2, RTX 2080 Super, and Tesla V100 systems. Jetson TX2 is a more advanced embedded device for AI applications. NVidia RTX 2080 Super is a mainstream graphics card in desktop systems, and NVidia Tesla V100 is one of the latest general purpose GPUs for high-performance computing applications. In particular, both the RTX 2080 Super and the Tesla V100 cards include Tensor cores which help to improve the performance of artificial neural network applications. Since these systems are commonly equipped with the NVidia GPU accelerator, we can use a suite of the same CUDA-enabled benchmark implementations so that the benchmark results for Jetson Nano can be more effectively compared and contrasted over a range of different GPU hardware parallelism.

Table 6 shows specifications of the GPUs used in our benchmark. As previously described, the Jetson Nano device is the latest offering from NVidia for edge computing, and its GPU is based on the Maxwell microarchitecture (GM20B). It comes with one streaming multiprocessor (SM) with 128 CUDA cores along with almost 4GB of GPU memory, which enables to run multiple neural networks in parallel. On the other hand, GPUs in other systems allow for a greater amount of hardware parallelism under the CUDA SIMT (single instruction, multiple threads) execution model. Specifically, $2 \times$ number of CUDA cores are available for Jetson TX2, $12 \times$ with RTX 2060 Super, and $40 \times$ with Tesla V100, respectively. Jetson Nano and Jetson TX2 are configured using Jetpack v4.3. The RTX 2080 Super and Tesla V100 systems are setup to closely match the library configurations of Jetson Nano and TX2, so that CUDA (v10.0), cuDNN (v7.6.3), and TensorRT (v6.0.1) have the same versions in all the systems under consideration.

Table 7 shows the CNN benchmarks additionally used for evaluating the GPU systems: VGG 16 and 19 [25], ResNet 50 and 152 [26], and SqueezeNet [27]. VGG is a simple network structure with ease of change. ResNet addresses the “degradation problem” [28] in deep neural networks by using residual blocks and identity blocks. SqueezeNet is a network with reduced number of parameters by using the “Fire” modules.

### 5.2 Inference Speed: Jetson Nano and Other GPU Systems

Figure 4 shows the benchmark results in terms of the frames-per-second (FPS) inference performance measured across the CNNs, where Jetson Nano’s performance is plotted along with other systems for comparison. First of all, we confirm that our Jetson Nano benchmark results in some of the CNNs agree with NVidia’s report available at the developer blog [29]. The CNNs commonly used in both experiments include ResNet 50, Inception V4, and VGG-19. For instance, Jetson Nano’s inference performance with ResNet 50 is measured to be the same 36 FPS in our benchmark as well as NVidia’s.

Despite the understandably worse performance compared against other GPU-based systems, Jetson Nano still
Fig. 4 CNN benchmark performance of Jetson Nano compared with other GPU systems (batch size 1)

shows 30+ FPS on 7 CNN benchmarks, which demonstrates that Nano can be adequate for certain real-time applications in deep learning. These CNNs include ResNet 50 (36 FPS), Inception V2 (54 FPS), and SSD MobileNet V2 (37 FPS). In particular, we observe super real-time FPS performance with GoogLeNet (76 FPS), MobileNet V1 and V2 (80 and 60 FPS), and SqueezeNet (104 FPS). For other CNNs such as VGG 16 and 19, Inception V3 and V4, and ResNet-152, Jetson Nano’s inference speed is quite slow with varied performance from 10 to 22 FPS.

5.2.1 Performance Analysis

As expected before, we can see that the SqueezeNet performance is the greatest with its simple network structure, while the VGG 16 and 19 performances are more or less smallest with their complex network architecture. We observe that all the CNN models for which Jetson Nano shows 30+ FPS are relatively simple in terms of complexity with a small parameter data. In contrast, the other CNN models with less than 22 FPS such as VGG 16 and 19 are considered as complex with 42~144 MB of parameter data.

Figure 5 shows performance comparison of each CNN benchmark between the evaluated GPU systems, scaled over Jetson Nano’s inference performance. Here, the batch size used is 1. First of all, we note that the performance values for VGG 16 and 19, which are the most complex CNN models in our experiments, dramatically improve on RTX 2080 Super and Tesla V100. As the hardware parallelism grows only 12× for RTX 2080 and 40× for V100 compared with Jetson Nano, the speedup is 49× for RTX 2080 and 61× for V100, where the geometric mean is used to average the performance between VGG 16 and VGG 19. It is clear that the speedup cannot be regarded as literally “super-linear”, since other factors such as memory size and clock speed need to be considered altogether to examine the performance boost from Jetson Nano. Beside VGG 16 and 19, the CNN models show quite different performance behavior over different GPU systems. For Jetson TX2, it shows close to 2× speedup for all the CNN models (including VGG 16 and 19 as well) compared with Jetson Nano, which is straightforward considering the same 2× more CUDA cores are available on TX2. That is, the CNN benchmark performance increases almost linearly with the number of CUDA cores in this case.

For RTX 2080 Super, the CNN models show the most performance boost, which is about 25× speedup on average over Jetson. This is even greater than Tesla V100 which shows only 18.6× speedup on average even with 40× more parallelism. For instance, Inception V2 shows 19× speedup on RTX 2080 Super, which is in contrast with only 13× performance gain on Tesla V100. We guess that the sequential portion of the CNN computations in Inception V2 becomes relatively substantial and starts to dominate the performance behavior, resulting in a rather degraded speedup as hardware parallelism increases from RTX 2080 to Tesla V100. In contrast, for VGG 16 and 19, we estimate that computations in these models are both sufficiently large enough and highly parallelizable so that their computations become greatly suitable to execute on GPUs with more CUDA cores, thus achieving a seemingly “super-linear” performance gain.

5.3 Effect of Batch Size

To evaluate the performance impact by large batch sizes, we varied the batch size from 8 to 256 in measuring the inference speed of each CNN benchmark. Figure 6 shows
the inference performance of Jetson Nano and Tesla V100 with varied batch sizes for illustration purposes. For Jetson Nano as shown in Fig. 6(a), we observe the performance flattens out early after batch size 8 for all the CNNs, which is expected because Jetson Nano was almost at its peak in GPU utilization even at batch size 1, incapable of processing larger batches efficiently. Further, Jetson Nano fails to process a large amount of data in VGG 16 and 19 with batch size 128 and 256, resulting in a killed execution of the benchmarks. We can judge that using bigger batch sizes is not appropriate for Jetson Nano. Jetson TX2 shows similar performance behavior with large batch sizes, although its inference speed is almost doubled compared to Jetson Nano.

For the RTX 2080 Super system, the performance increases until batch size 16, but the increase quickly drops down after batch size 32. The amount of speed up is relatively large for simpler CNNs and becomes negligible for more complex CNNs. Tesla V100 benefits the most from increasing batch sizes. For simpler CNNs such as MobileNet V1, as shown in Fig. 6(d), the speedup is most dramatic and does not flatten out quickly even at largest batch sizes 128 and 256. For complex CNNs like VGG 16 and 19, Tesla V100 outperforms RTX 2080 Super at large batch sizes, unlike its previous performance behavior with batch size 1 where it performed slower than RTX 2080 Super. This is because Tesla V100 can fully utilize its CUDA cores and Tensor cores to process more and more amount of data in parallel as the batch size becomes larger.

Figure 7 shows inference speed of each GPU system for the CNN models with batch size 64, scaled over Jetson Nano’s performance. In contrast to the previous performance comparison with batch size set to 1 (Fig. 5), Tesla V100 shows a consistently better performance than RTX 2080 Super for all the CNNs, where the speedup over Jetson Nano is highest for VGG 19 (most complex) and lowest for SqueezeNet (simplest).

Assuming that the system with RTX 2080 Super consumes nearly 550 Watts on average based on the TDP (thermal design power) information of the system configuration, the performance-per-watt behavior between Jetson Nano
and the RTX 2080 Super system is almost identical since we measured Jetson Nano consumes about 5.5 Watts at inference. Compared to the Tesla V100 system which can also be assumed to use 550 Watts on average at execution, we can estimate that Jetson Nano is about 28% less efficient in terms of performance-per-watt. If we use TDP instead of the measured power for Jetson Nano for calculating power efficiency, the performance-per-watt value can get worse.

6. Related Work

For most thorough and recent survey on “edge benchmarking” for identifying the system under test, analyzed benchmark techniques, quality metrics, and benchmark runtime, we refer the reader to the work of Varghese et al. [9]. In addition, we refer the reader to the work by Chen et al. [30] for a comprehensive review on accelerator architectures for deep learning applications. In this work, we narrow the discussion on performance evaluation of latest edge devices based on deep learning benchmarks. Bianco et al. [10] analyze more than 40 deep neural networks on NVidia Jetson TX1 and Titan X, which are both GPU-accelerated. While their report is in-depth with a more thorough analysis using more complete evaluation perspectives such as computational complexity and accuracy rate, Jetson TX1 is almost 5 years old and a bit outdated to apply to the state-of-the-art devices. In contrast, we present benchmark results on the latest edge systems, NVidia Jetson Nano and Google Coral Dev Board, powered by an accelerator.

Kim et al. [11] report deep learning application performance of Raspberry Pi 3 B+ in comparison with a GTX 1080 desktop system. Based on their benchmark results, they report that GPU-based systems can be more adequate for CNN-based applications due to the structural parallelism inherent in CNN models, while RNN-based applications can benefit from different types of accelerators where specialized datapath and control units are capable of managing dependencies in recurrent networks. However, their report does not include hardware-accelerated edge devices, although their findings indicate that CNN inference benchmarks will be greatly benefited from SIMD (single instruction, multiple data) style hardware acceleration due to the parallel nature of the CNN architecture. The work of Antonini et al. [8] is most similar to ours in that both Coral Dev Board and Jetson Nano are covered and analyzed comparatively. Additionally, their work included Intel Neural Compute Stick for performance comparison altogether. By contrast, we present an in-depth examination of GPU accelerators from NVidia across different execution platforms, evaluating Jetson Nano compared with more powerful GPU-based systems.

7. Conclusions

In this paper, we evaluated two representative entry-level edge devices – Google Coral Dev Board and NVidia Jetson Nano. These state-of-the-art devices are powered by accelerators to improve the performance of AI applications which are getting widespread these days. While both devices show similar performance for large networks, Coral Dev Board shows about 5× better performance than Jetson Nano for relatively simple CNNs with a small number of parameters, thanks to its SRAM that can cache parameter data for faster access. Furthermore, Coral Dev Board uses an order-of-magnitude less memory than Jetson Nano throughout the benchmarks, which is attributed to the design of its Edge TPU coprocessor that can pipeline the computation over the systolic array of the MAC units. This is in contrast with the SIMT-based GPU of Jetson Nano. While Jetson Nano can be useful for more general-purpose applications with its GPU, Coral Dev Board presents more favorable performance behavior for deep learning applications.

We also evaluated Jetson Nano’s performance in comparison with other GPU-accelerated systems in different platforms such as desktop and high-end computing environments. Although it is not very comparable to other GPU systems in terms of performance, the CNN benchmark results show that Jetson Nano is capable of efficiently running simple to medium models, showing real-time performance for image classification and object detection applications. Jetson Nano is not very applicable to CNNs with more than 20 MB of parameters. Increasing the batch size at inference is not very effective since the GPU of Jetson Nano becomes almost fully utilized even at batch size 1.

Overall, our results show that both Coral Dev Board and Jetson Nano can be adequately used for not-too-complex deep learning applications in real time. As the technology advances, new devices are becoming readily available for AI applications, which are characterized by different types of hardware parallelism and specialized support for machine learning applications. In that regard, we plan to investigate neuromorphic processors such as Loihi [31] because they pose as an interesting alternative to conventional edge processors for inference workloads with much less power consumption.

References

[1] “AI is screening billions of molecules for Coronavirus treatments,” https://www.the-scientist.com/news-opinion/ai-is-screening-billions-of-molecules-for-coronavirus-treatments-67520, 2020.
[2] “IBM tops U.S. Patent list for 2019,” 2020. https://newsroom.ibm.com/2020-01-14-IBM-Tops-U-S-Patent-List-For-2019.
[3] “The growth in connected IoT devices is expected to generate 79.4ZB of data in 2025,” https://www.idc.com/getdoc.jsp?containerId=prUS45213219, June 2019.
[4] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D.S. Nikolopoulos, “Challenges and Opportunities in Edge Computing,” IEEE International Conference on Smart Cloud, pp.20–26, Nov. 2016.
[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” IEEE Internet Things J., vol.3, no.5, pp.637–646, Oct. 2016.
[6] “Google Coral Dev Board,” https://coral.ai/products/dev-board, 2019.
[7] “NVIDIA Jetson Nano,” https://www.nvidia.com/en-us/embedded-systems/jetson-nano, 2019.
[8] M. Antonini, T.H. Yu, C. Min, A. Montanari, A. Mathur, and F. Kawsar, “Resource Characterisation of Personal-Scale Sensing Models on Edge Accelerators,” Proc. AIChallengeIoT ’19, pp.49–55, ACM, 2019.

[9] B. Varghese, N. Wang, D. Bervbach, C.H. Hong, E. de Lara, W. Shi, and C. Stewart, “A survey on edge benchmarking,” arXiv preprint arXiv:2004.11725, 2020.

[10] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, “Benchmark Analysis of Representative Deep Neural Network Architectures,” IEEE Access, vol.6, pp.64270–64277, 2018.

[11] J. Kim, C. Han, M. Kee, and G.H. Park, “Performance analysis of deep learning applications for efficient execution on edge devices,” Proc. 2018 KIIEE Conference, pp.1713–1715, 2018.

[12] J. Jo, S. Jeong, and P. Kang, “Benchmarking GPU-Accelerated Edge Devices,” Proc. 2020 IEEE International Conference on Big Data and Smart Computing, pp.117–120, IEEE, 2020.

[13] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable Parallel Programming with CUDA,” Queue, vol.6, no.2, pp.40–53, March 2008.

[14] Google, “An eco system for local AI,” https://coral.ai/, 2019.

[15] N.P. Jouppi, et al., “In-datacenter performance analysis of a tensor processing unit,” arXiv preprint arXiv:1704.04760, 2017.

[16] M. Abadi, et al., “TensorFlow: A system for large-scale machine learning,” Proc. OSDI ’16, USA, pp.265–283, 2016.

[17] B. Jacob, et al., “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” arXiv preprint arXiv:1712.05877, 2017.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp.1–9, 2015.

[19] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” arXiv preprint arXiv:1512.00567, 2015.

[20] C. Szegedy, S. Ioffe, V. Vanhoucke, and A.A. Alemi, “Inception-V4, inception-resnet and the impact of residual connections on learning,” 31st AAAI Conference on Artificial Intelligence, 2017.

[21] A.G. Howard, et al., “MobileNets: Efficient convolutional neural networks for mobile vision applications,” arXiv preprint arXiv:1704.04861, 2017.

[22] M. Sandler, et al., “MobileNetV2: Inverted residuals and linear bottlenecks,” arXiv preprint arXiv:1801.04381, 2018.

[23] Google, “Models built for the edge TPU,” https://coral.ai/models, 2019.

[24] NVidia, “Deep learning inference benchmarking instructions,” https://forums.developer.nvidia.com/t/deep-learning-inference-benchmarking-instructions/73291, 2019.

[25] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.

[26] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp.770–778, 2016.

[27] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size,” arXiv preprint arXiv:1602.07360, 2016.

[28] K. He and J. Sun, “Convolutional Neural Networks at Constrained Time Cost,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.5353–5360, 2015.

[29] NVidia, “Jetson Nano: Deep learning inference benchmarks,” https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks, 2019.

[30] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, “A Survey of Accelerator Architectures for Deep Neural Networks,” Engineering, vol.6, no.3, pp.264–274, 2020.

[31] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S.H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaiakuttty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning,” IEEE Micro, vol.38, no.1, pp.82–99, 2018.

Pilsung Kang is an Assistant Professor in the Division of Computer Science and Engineering at Sun Moon University, South Korea. His research interests include parallel systems, high-performance software, and quantum computing. Kang has a PhD in computer science from Virginia Tech.

Jongmin Jo is a senior in the Division of Computer Science and Engineering at Sun Moon University, South Korea. His research interests include artificial intelligence and edge computing.