Inference Performance Comparison of Convolutional Neural Networks on Edge Devices

Sheikh Rufsan Reza(✉), Yuzhong Yan, Xishuang Dong, and Lijun Qian

Center of Excellence in Research and Education for Big Military Data Intelligence (CREDIT Center), Prairie View A&M University, Texas A&M University System, Prairie View, TX 77446, USA
sreza@student.pvamu.edu, {xidong,liqian}@pvamu.edu

Abstract. With the proliferation of Internet of Things (IoT), large amount of data are generated at edge devices with an unprecedented speed. In order to protect the privacy and security of big edge data, as well as reduce the communications cost, it is desirable to process the data locally at the edge devices. In this study, the inference performance of several popular pre-trained convolutional neural networks on three edge computing devices are evaluated. Specifically, MobileNetV1 & V2 and InceptionV3 models have been tested on NVIDIA Jetson TX2, Jetson Nano, and Google Edge TPU for image classification. Furthermore, various compression techniques including pruning, quantization, binarized neural network, and tensor decomposition are applied to reduce the model complexity. The results will provide a guidance for practitioners when deploying deep learning models on resource constrained edge devices for near real-time and on-site learning.

Keywords: Model compression · Edge device · Deep learning · Internet of Things

1 Introduction

The Internet of Things (IoT), with pervasive interconnected smart objects operating together, has become particularly popular with the rapid development of small low-cost sensors, wireless communication technologies, and new internet techniques [1]. It is anticipated that the number of connected IoT devices will reach nearly 50 billion in 2020. As a result, huge amount of data has been generated that needs to be processed. In this context, machine learning techniques can be applied to build models to process data and a popular architecture is shown in Fig. 1. In this case, the cloud will collect the data from different IoT sensors and devices and process it by machine learning algorithms. Typically, these models are deep neural networks (DNN) that require high computational power to train.

S. Paiva et al. (Eds.): SmartCity360° 2020, LNICST 372, pp. 323–335, 2021.
https://doi.org/10.1007/978-3-030-76063-2_23

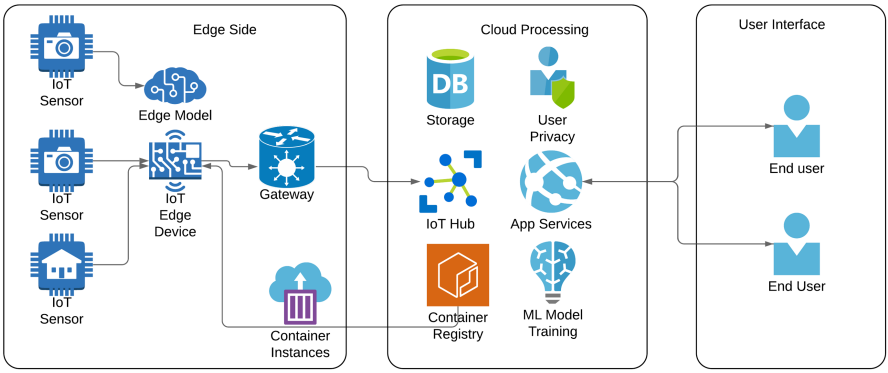

Fig. 1. Processing IoT data in cloud

Recently, the proliferation of IoT and the recent breakthroughs in deep learning have fueled a growing demand for intelligent edge devices featuring near real-time and on-site learning. Although centralized big data processing is mature, it is either not feasible to transmit all the data to a central location, or the delay incurred may exceed the tolerance of the time-sensitive applications [2], because of the densely deployed IoT devices and limited wireless communications bandwidth. In addition, there are security and privacy concerns when transmitting raw data that may contain private information [3]. Hence, mobile edge computing has been proposed to bring computing closer to the data [2]. Wireless edge networks provide highly sophisticated network and client nodes, equipped with rich sensing, computation, and storage resources. In this scenario, there is an opportunity to *leverage mobile edge devices as learning engines* which can use the locally collected data sets at edge nodes such as audio and video to derive local learning models (edge model, see Fig. 1) without sending the raw data to a central location. Compared to the centralized learning solutions, this approach reduces communication costs, improves latency, and preserves data privacy.

Let us consider the surveillance camera of an office as an example. If the camera captures videos and transmits them to the cloud, these videos can be attacked by hackers and transmitting these videos will require a large amount of bandwidth. On the contrary, if the camera connecting to an edge computing device and the monitoring software (edge model) can be run on the device without sending the raw videos to the cloud (and only send alarms with associated information when intrusion is detected), it will save a lot of bandwidth and be more secure.

However, the practical realization of such systems remains a challenge due to the limited computing and energy resources available at the edge devices, as well as the growing complexity of the state-of-the-art DNNs. There are some recent studies on the inference performance of popular deep learning models on edge devices. For example, it has been demonstrated that Jetson Nano can run MobileNet at 64 fps for the images with resolution 300×300 on TensorFlow

framework, and be able to process larger images (960×544) at 5 fps by running a Resnet-18 SSD (single shot multibox detector) [4].

This paper presents a comprehensive comparison of inference performance of three convolutional neural networks (CNN) models, namely, MobileNet V1 [5], MobileNet V2 [6], and Inception V3 [7] by running image classification tasks on three edge devices, NVIDIA Jetson TX2, NVIDIA Jetson Nano, and Google Edge TPU. The models needs to be simplified before executing them on resource constraint devices. Now various compression techniques including pruning [8], quantization [9,10], binarized neural network [11,12], and tensor decomposition [13] are tested to reduce the model complexity. The goal is to evaluate the advantages and disadvantages of deep learning models when performing inference on different edge devices, and provide guidance on the practical choices of compressed DNNs vs. edge devices to achieve specifications of (particularly delay-sensitive) applications.

2 Deep Learning Models and Tools

2.1 Convolutional Neural Network Models

Convolutional neural network (CNN) [14] is a category of multilayer neural network including three main components: convolutional layer, pooling and nonlinear layer, and lastly the fully connected layer, where the basic architecture of CNN for image classification is shown in Fig. 2. The convolutional layer is used to extract features by parsing small convolutional filters across the input. Afterwards a pooling layer is added between two convolutional layers to speed up the computation and decrease the size of input. The fully connected layer for the output is a feed forward layer which uses the softmax activation function to get the prediction probabilities of the input belonging to different classes. We can utilize CNN to build different kinds of applications such as image classification [15], object detection [16], object tracking [17] and many more.

This paper uses MobileNet V1 [5], MobileNet V2 [6] and Inception V3 [7] for performance comparison. These models were created to be suitable for edge devices by reducing the model complexity and computational cost. As a result, these models are not too complex yet they have good accuracy while reducing cost. MobileNet [5] is a popular neural network suitable for mobile and embedded based computer vision tasks. The aim was to achieve high accuracy with low computational cost. The main idea is to change the standard convolution into depth-wise convolution and 1×1 point-wise convolution. Width multiplier and resolution multipliers are additional to control the input widths. These approaches can significantly reduce the computational cost of the model. Equation (1) shows regular convolution operation (RCO) and Equation (2) shows depth-wise followed by point-wise convolution operation (PCO) [5].

$$Cost_{RCO} = D_K * D_F * M * N * D_K * D_F \tag{1}$$

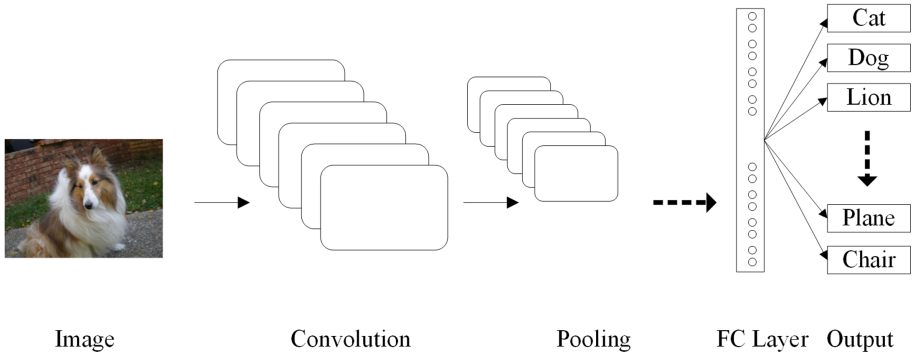

Fig. 2. An example convolutional neural network

$$Cost_{PCO} = D_K * D_F * M * D_K * D_F + M * N * D_F * D_F \quad (2)$$

where D_K and D_K are the height and width of filter while D_F and D_F are the height and width of input feature map size. Moreover, M is the number of input channel and N is the number of output channel. The comparison between these two convolution operations results in $1/N + 1/D_K^2$, which means PCO has less computation than the RCO [5].

MobileNet V2 [6] was introduced as the improved version of mobile models. The difference between MobileNet version 1 and 2 is that version 2 has a new layer which is 1×1 projection layer, which is observed in Fig. 3. It makes the number of channel smaller. It has 17 building blocks followed by 1×1 convolution, global average pooling and classification layer. In addition, when the number of channel though a layer is same, residual connection is introduced to help the flow of gradients through the network. The overall architecture is like the input comes in a low dimensional tensor which is expanded to higher dimension. Then the depth-wise layer is used to filter the data. Afterwards projection layer again compresses the data. The model was trained using 16 GPU with a batch size of 96. The version 2 is more lighter than the version 1 [6].

Inception v3 [7] uses approaches such as factorizing convolution, auxiliary classifier, grid size reduction to build a more robust and lighter model. For example, a large 5×5 filter is replaced by two 3×3 filters. The number of parameters is reduced from 25 to 18. Moreover, a 3×3 filter is replaced by 3×1 and 1×3 filter. The number of parameters is reduced from 9 to 6. Finally, the number of parameters is reduced from 25 to 6.

2.2 Model Compression

Considering the resource limitation of edge devices such as limited memory and low computational resources, regular CNN models have to be simplified before deploying in resource constrained devices [18]. Many simplification techniques are being developed where the tradeoff is between speed and accuracy [19]. The

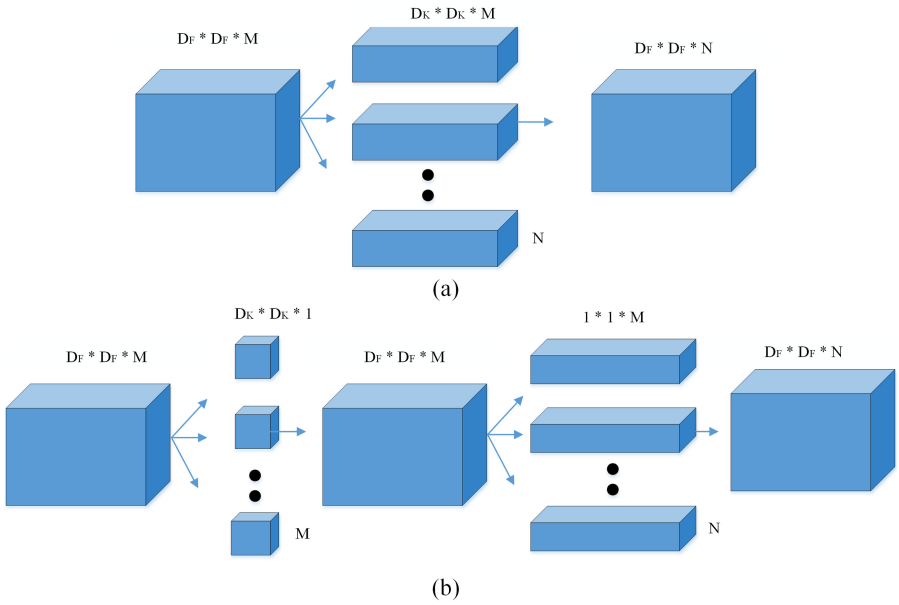

Fig. 3. (a): Regular convolution operation (RCO), (b): Point-wise convolution operation (PCO) [5]

goal is to reduce the memory required to store the weights. Pruning [8], quantization [10], data compression [20] are some of the popular techniques. The idea of pruning is to reduce the number of neurons by finding the less important neurons [8] in a network. Quantization method is used to reduce the number of floating or bit point operation [10], which speeds up the algorithm as lesser weight is transferred between memory and cores. If the floating point operation is reduced, memory consumption is also reduced. For example, binarized neural network is where activations and weights are represented in binary, can be easily implemented in a memory constrained device [11]. Another example is data compression technique. Parameters computed in the training phase are stored in compressed form where it is decompressed at runtime [20].

2.3 Software Tools

Tensorflow was the basic library used for this paper. TensorRT is a software development kit built on CUDA for high performance deep learning inference [21]. It can be used in many aspects as it can provide the maximum throughput as well as reduce the latency. TensorRT is compatible with frameworks such as TensorFlow or Matlab, and it supports model libraries of Keras, PyTorch and Caffe [22], which is illustrated in Fig. 4. TensorRT is capable of providing reduced bit point operations for image classification, natural language processing, object detection and so forth [21,23]. In our experiments, frozen graph was initially

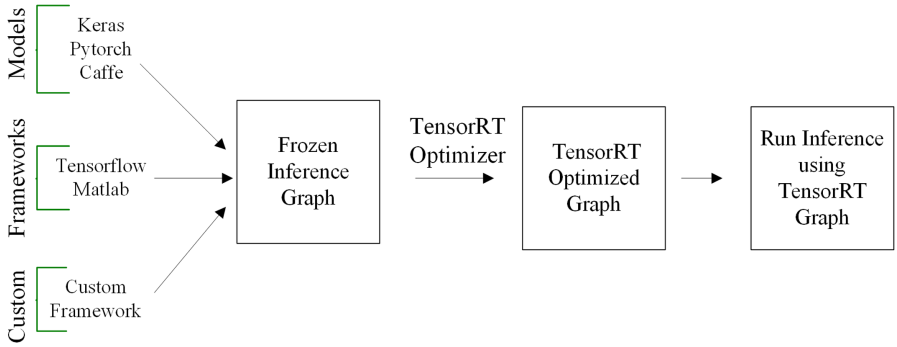

Fig. 4. TensorRT workflow [21]

created from the Keras application which operates on 32 bit operation. Afterwards TensorRT converter is used to create the TensorRT graph which operates on 16 bit operation. Moreover Keras application [24] was used to execute the 32 bit point operations. Lastly the Edge TPU compatible file was derived from tensorflow Lite [25] which is the lighter version of tensorflow.

3 Edge Devices

As shown in Fig. 5, Nvidia Jetson hardwares have been used for this research. This is a good platform for DL/AI tasks. These have a lightweight GPU mounted on top of them making them run deep learning algorithm smoothly. Jetson TX2 and Nano both have a 4-core ARM A57 core where TX2 performs at 2 GHz. Both have 256-core Pascal and 128-core Pascal.

Edge TPU is the third device applied in this paper. Edge TPU devices help to deploy strong AI model on mobile devices, and it can be used in multiple application such as machine learning, robotics, medical, retail and many more [26]. Tensor Processing Units (TPUs) are custom-developed application-specific integrated circuits (ASICs) specially designed for machine learning task by Google Inc. Google has introduced this in both cloud and edge devices. The edge TPU devices are Google Coral Dev Board and USB Accelerator. This paper uses the USB Accelerator and referred as Edge TPU, as shown in Fig. 5. It supports TensorFlow lite and its hardware helps to run machine learning models faster than most other devices. Edge TPU supports only TensorFlow lite models are illustrated in Fig. 6. It uses 8-bit integer operation with TensorFlow Lite quantized models. This on-board coprocessor can perform fast image classification task. The TensorFlow Lite model is built with quantize-aware training. Edge TPU has a matrix processor designed such that it can compute hundreds of thousands of operations in a single clock cycle where a conventional GPU can only do tens of thousands. The hardware is designed in such a way that it was able to do matrix multiplication operations like operating thousands of multipliers and

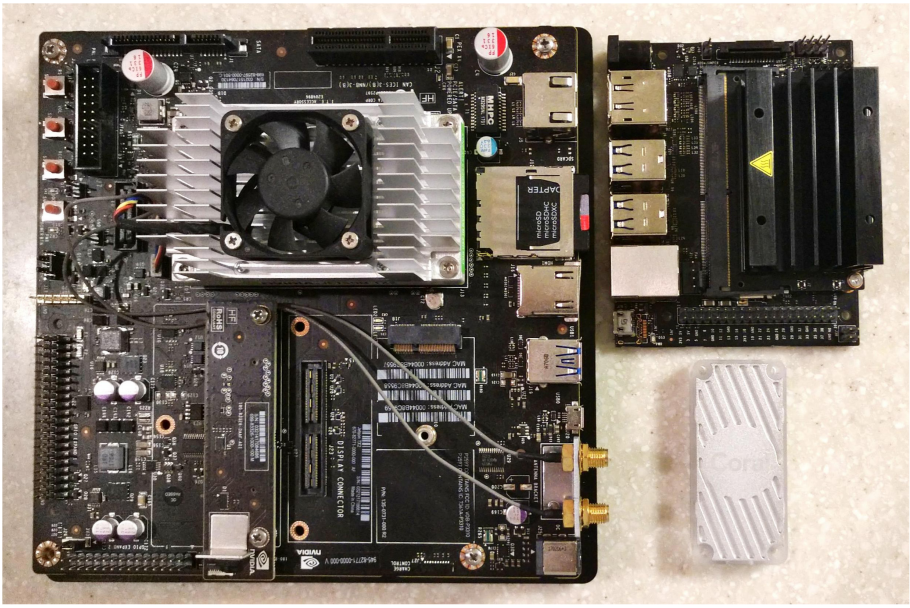

Fig. 5. Left: Jetson TX2, Right (Top): Jetson Nano, Right (Bottom): Google Coral TPU USB Accelerator

Table 1. Edge device comparison

	Jetson TX2	Jetson Nano	Google Coral TPU USB Accelerator
Memory	8 GB	4 GB	NA
Storage	32 GB	16 GB eMMC	NA
Processor	Quad-Core ARM Cortex-A57 MPCore	Quad-core ARM Cortex-A57 MPCore	NA
AI Accelerator	256 Cuda Core (Pascal)	128 Cuda Cores (Maxwell)	Edge TPU

adders instantly from the operation without sending those to the memory [25]. The comparison of the three devices is presented in Table 1.

4 Experiment

4.1 Dataset

We employ ImageNet 2012 validation dataset consisting of 50,000 images and 1,000 classes for testing the accuracy [27]. We run pre-trained models created using TensorFlow framework on different devices for performance comparison.

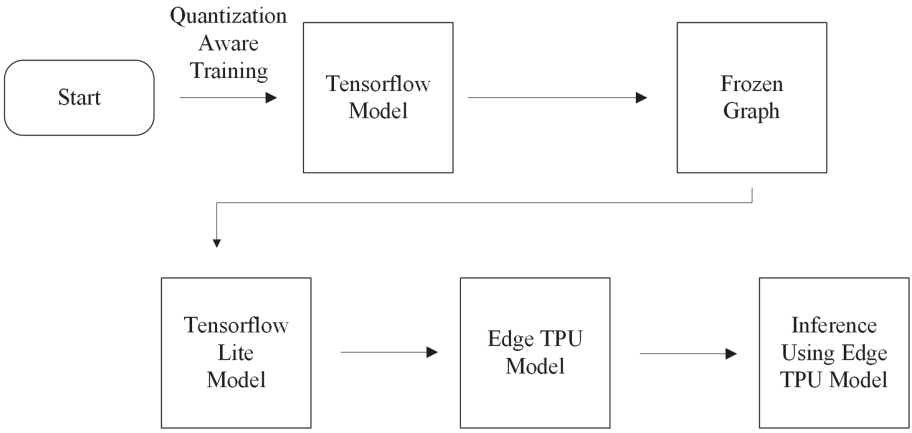

Fig. 6. TPU USB accelerator workflow [25]

4.2 Evaluation Metrics

We utilize different evaluation metrics for performance comparison. They are listed below.

- FP denotes floating point operation taken 32 bit and 16 bit for Jetson devices and 8 bit operation for Edge TPU.
- Accuracy refers to the ratio of number of correct predictions to the total number of testing samples.
- Memory denotes how much dynamic memory has been allocated by the python thread in mebibyte or MiB (1 MiB = 1024×1024 bytes).
- Load denotes the pre-trained model loading time in seconds.
- Time denotes the total time python script requires to execute the task.
- Average Inference (Avg Inf) is the average inference time of a single image in seconds.
- FPS (inf) represents the frame per second which is how many images can this model run in one second. Now FPS (inf) shows the rate of processing image while only considering inference whereas the last column FPS shows the rate of processing image while taking model loading, image preprocessing and inference into consideration.

Frame per second (FPS) and Frame per second (inf) are given in equation (3) and (4).

$$FPS = \frac{N}{M + \sum_{n=1}^{50000}(P_n + I_n)}, \tag{3}$$

$$FPS(inf) = \frac{N}{\sum_{n=1}^{50000} I_n}. \tag{4}$$

Here N denotes the 50,000 validation images. M in the denominator is the model loading time, P is the time for preprocessing of images, and I is inference time.

The model loading occurs only once but preprocessing and inference occurs for all 50,000 images.

Table 2. Performance comparison on various edge devices with MobileNet V1.

Device	FP	Accuracy	Memory (MiB)	Load (sec)	Time (sec)	Avg. Inf. (ms)	FPS (inf)	FPS
TX2	32 bit	0.68364	1595.426	30.98	2582.42	40	25	19.36
TX2	16 bit	0.68374	2267.48	363.11	2033.77	20	50	24.58
Nano	32 bit	0.68362	1147.215	20.04	4591.97	70	14.29	10.89
Nano	16 bit	0.68372	2136.59	82.95	2151.96	20	50	23.23
TPU	8 bit	0.68008	108.516	3.06	1235.07	9.43	106.04	40.48

Table 3. Performance comparison on various edge devices with MobileNet V2.

Device	FP	Accuracy	Memory (MiB)	Load (sec)	Time (sec)	Avg Inf (ms)	FPS (inf)	FPS
TX2	32 bit	0.68048	1818.398	53.95	2799.24	40	25	17.86
TX2	16 bit	0.68048	1914.27	187.56	2278.44	20	50	21.94
Nano	32 bit	0.68048	1546.164	28.93	5471.34	90	11.11	9.14
Nano	16 bit	0.68084	2102.309	78.96	2206.76	20	50	22.66
TPU	8 bit	0.69026	103.078	3.07	1315.2	11.28	88.65	38.02

Table 4. Performance comparison on various edge devices with Inception V3.

Device	FP	Accuracy	Memory (MiB)	Load (sec)	Time (sec)	Avg Inf (ms)	FPS (inf)	FPS
TX2	32 bit	0.76276	1674.637	88.99	8860.52	150	6.67	5.64
TX2	16 bit	0.76284	3656.887	1945.94	4302.21	20	50	11.62
Nano	32 bit	0.76276	1044.277	47.38	17752.81	320	3.13	2.82
Nano	16 bit	0.76264	3213.441	469.4	4191.77	50	20	11.93
TPU	8 bit	0.7705	147.883	3.13	25463.41	490	2.04	1.96

4.3 Results and Analysis

We evaluate the performance of MobileNet V1, MobileNet V2 and Inception V3 with 32 bit, 16 bit and 8 bit operation running on Jetson devices and Edge TPU. Jetson devices were used to compute the 32 bit and 16 bit operation while Edge TPU was used to do the 8 bit operation.

Table 2 reflects the result of MobileNet V1, the 16 bit operation case takes more time to load the model but much less time to execute the entire python program than the 32 bit operation. The 32 bit operation and 8 bit operation pre-trained models are from their own application which are more optimized than the 16 bit operation in Keras pre-trained model. This could be the reason for the 16 bit operation to require more memory and more loading time. If we

consider only the average inference time of a single image, then 16 bit operation is faster than the 32 bit operation. Execution time is significantly more in Nano than TX2 with TX2 having more memory consumption.

Table 3 shows that for MobileNet V2 the accuracies for different devices are almost the same. The memory consumption is more for the 16 bit operation probably because loading the library and model is the main source of memory consumption. It is also reflected on the model loading time column. In Jetson TX2, loading the model in 32 bit operation is almost 12 times faster. Also due to memory issue the validation dataset was loaded in SD card for the TX2 in all the experiments. Inference time and fps was better in 16 bit operation in TX2. The Edge TPU outperforms all the other devices by a great margin for two MobileNet models. Accuracy, memory consumption, execution time, fps is better than others.

Table 4 shows the performance of Inception V3 on various edge devices. The network structure of Inception V3 is much more complicated than MobileNet V1 and V2, with more layers and larger input image size. If we compare in terms of model size, MobileNet V2 has 4.5 MB and Inception V3 has 25.1 MB on Edge TPU. This architecture gave Inception V3 higher accuracy but slower average inference time. Similar to previous results, the inference time of 16 bit operation is faster than that of the 32 bit operation. The low FPS of the Edge TPU Coral USB Accelerator was caused by the testing environment, in which we connected TPU Coral USB Accelerator to the host CPU through USB 2.0 ports. If we ran the same model on TPU Coral Dev Board or switch to another host machine with USB 3.0 port connected to TPU Coral USB Accelerator, the average inference time is about 43.6 ms, or 22.94 FPS, nearly two times faster than NVIDIA Jetson TX2 and NVIDIA Jetson Nano. The speed up of running Inception V3 model on Edge TPU is consistent with MobileNet V1 and V2 models running on those edge devices.

5 Related Work

5.1 Model Compression

Model compression has been a popular research topic in recent years since it can reduce the model complexity significantly. For example, the specially designed convolution architecture in MobileNet was able to reduce the number of parameters by 7 times while losing the accuracy only by 1 percent. If this comparison is made with another popular model like VGG 16, then the number of parameters is reduced by almost 33 times [5]. Moreover, model compression can reduce the inference time in addition to the number of parameters. Jiaxiang et al. [9] presented that quantized convolution was able to speed up the inference 4 to 6 times and reduce the number of parameters 15 to 20 times. Han et al. [28] presented a three-stage pipeline containing pruning, quantization and huffman coding which can reduce the Alexnet model size from 240 MB to 6.9 MB.

5.2 Deep Learning Inference on Edge Devices

It is demonstrated in this paper that sophisticated deep learning models can be accommodated on resource constrained devices when applying model compression, which is consistent with studies in the literature. For instance, it has been shown that Jetson Nano runs MobileNet V2 at 64 frames per second, and Google Edge TPU Coral Dev Board runs the same model at 130 frames per second [4]. It takes a single 64-bit Intel(R) Xeon(R) Gold 6154 CPU at 3.00GHz with the coral Edge TPU USB Accelerator 2.4 ms to performs inference using MobileNet V1 or V2. Without TPU, it takes the same desktop CPU about 53 ms and 51 ms to perform the same task [29]. Taylor et al. [30] also presented an adaptive deep learning model selection for edge device which can achieve a 7.52 percent improvement in accuracy and 1.8 times reduction in inference time.

6 Conclusion

This paper presents a comprehensive comparison of the inference performance of several popular pre-trained convolutional neural networks on three edge computing devices. Specifically, MobileNet V1 & V2 and Inception V3 models have been tested on NVIDIA Jetson TX2, Jetson Nano, and Google Edge TPU for image classification. Furthermore, various compression techniques including pruning, quantization, binarized neural network, and tensor decomposition are applied to reduce the model complexity. Experimental results indicate that Edge TPU outperforms Nvidia Jetson TX2 and Nano in most experiments with faster inference speed and more accurate result. However, Edge TPU cannot work independently because it needs a host computer to complete the task. In addition, TX2 is better than Nano in most cases regarding most of the evaluation results, as expected. It is also observed that MobileNet V1 or V2 are the candidate models for speed while Inception V3 would be more accurate, which is another example of the tradeoff between inference time and prediction accuracy. It is also demonstrate that the choice of the edge computing device and the corresponding deep learning model should match the needs of the specific application. This is our first attempt to measure the inference performance of various DNNs, model compression, and edge device combinations, and we hope the results could serve as a guidance for practitioners when deploying deep learning models on resource constrained edge devices for near real-time and on-site learning.

Acknowledgment. This research work is supported by the U.S. Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)) under agreement number FA8750-15-2-0119. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)) or the U.S. Government.

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
2. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B.: A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutorials **19**(4), 2322–2358 (2017)
3. Neshenko, N., Bou-Harb, E., Crichigno, J., Kaddoum, G., Ghani, N.: Demystifying IoT security: an exhaustive survey on IoT vulnerabilities and a first empirical look on internet-scale IoT exploitations. IEEE Commun. Surv. Tutorials **21**(3), 2702–2733 (2019)
4. Jetson nano: Deep learning inference benchmarks. https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks
5. Howard, A.G., et al.: Mobilenets: efficient convolutional neural networks for mobile vision applications (2017). arXiv preprint arXiv:1704.04861
6. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: Mobilenetv 2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510–4520 (2018)
7. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016)
8. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference (2016). arXiv preprint arXiv:1611.06440
9. Wu, J., Leng, C., Wang, Y., Hu, Q., Cheng, J.: Quantized convolutional neural networks for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4820–4828 (2016)
10. Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental network quantization: towards lossless CNNS with low-precision weights (2017). arXiv preprint arXiv:1702.03044
11. Zhao, R., et al.: Accelerating binarized convolutional neural networks with software-programmable FPGAs. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 15–24 (2017)
12. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1 (2016). arXiv preprint arXiv:1602.02830
13. Cheng, T., et al.: Convolutional neural networks with low-rank regularization (2015). arXiv preprint arXiv:1511.06067
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. **25**, 1097–1105 (2012)
15. Howard, A.G.: Some improvements on deep convolutional neural network based image classification (2013). arXiv preprint arXiv:1312.5402
16. Cai, Z., Fan, Q., Feris, R.S., Vasconcelos, N.: A unified multi-scale deep convolutional neural network for fast object detection. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 354–370. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_22
17. Hong, S., You, T., Kwak, S., Han, B.: Online tracking by learning discriminative saliency map with convolutional neural network. In: International Conference on Machine Learning, pp. 597–606 (2015)

18. Yanai, K., Ryosuke Tanno, and Koichi Okamoto. Efficient mobile implementation of a cnn-based object recognition system. In: Proceedings of the 24th ACM International Conference on Multimedia, pp. 362–366 (2016)
19. Li, X., Zhou, Y., Pan, Z., Feng, J.: Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019
20. Makhzani, A., Frey, B.J.: Winner-take-all autoencoders. In: Advances in Neural Information Processing Systems, pp. 2791–2799 (2015)
21. Deep learning SDK documentation. https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html
22. Vanholder, H.: Efficient inference with tensorRT (2016)
23. Real-time natural language understanding with BERT using tensorRT. https://devblogs.nvidia.com/nlu-with-tensorrt-bert/
24. Géron, A.: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Sebastopol (2019)
25. Tensorflow models on the edge TPU. https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview
26. Internet of things. https://cloud.google.com/edge-tpu
27. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Li, F.-F.: Imagenet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255. IEEE (2009)
28. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding (2015). arXiv preprint arXiv:1510.00149
29. Edge TPU performance benchmarks. https://coral.ai/docs/edgetpu/benchmarks/
30. Taylor, B., Marco, V.S., Wolff, W., Elkhatib, Y., Wang, Z.: Adaptive deep learning model selection on embedded systems. ACM SIGPLAN Notices **53**(6), 31–43 (2018)