RESEARCH ARTICLE | MAY 08 2023

Comparison of tensorflow and tensorflow lite for object detection on Raspberry Pi 4

Muhammad Syihabuddin Az Zuhair; Andi Widiyanto ✉; Setiya Nugroho

AIP Conference Proceedings 2706, 020129 (2023)
https://doi.org/10.1063/5.0120245

Comparison of Tensorflow and Tensorflow Lite for Object Detection on Raspberry Pi 4

Muhammad Syihabuddin Az Zuhair[1,a)], Andi Widiyanto[1,b)] and Setiya Nugroho[1,c)]

[1]*Department of Informatics Engineering, Universitas Muhammadiyah Magelang, Magelang, Indonesia*

[a)] *muhammadazzuhair@gmail.com*
[b)] *Corresponding author: andi.widiyanto@ummgl.ac.id.*
[c)] *setiya@ummgl.ac.id*

Abstract. Object detection is a field in machine learning that requires heavy computations. Some object detection applications such as vehicles or traffic detection require fast inference due to the nature of their input data. This requirement for heavy computing is not suitable for edge computers like Raspberry Pi that only have limited computing resources. One of the common frameworks used for machine learning, Tensorflow provides a specific package dedicated to being used in edge computing called Tensorflow Lite. This paper aims to present a performance comparison of these two frameworks on a Raspberry 4 Pi model B board. This paper focuses on the latency and memory usage differences between the two frameworks. The result of the comparison process shows an average of 90 percent increase in inference speed on Tensorflow Lite with MobileNet models compared to the Tensorflow framework. The initial inference on Tensorflow Lite is also significantly faster on both MobileNetV1, MobileNetV2, InceptionV2, and Resnet 50 models with an average of 124 percent increase. Moreover, the memory usage for all tested models is reduced by half on average when using Tensorflow Lite compared to the original Tensorflow model. This performance increase can be taken into consideration to increase the performance of object detection in edge computers.

INTRODUCTION

Nowadays, traffic monitoring has become a challenge due to the increasing number of vehicles and transportation infrastructure. The increase in vehicle numbers requires an automated solution for traffic surveillance. Automatic vehicle detection is part of computer vision that has become a popular research field. Vehicle detection systems can help reduce the manual task for monitoring the traffic. However, processing a computer vision algorithm usually needs a machine with high computational capabilities, thus limiting the application of the system on stationary computers such as server workstations or personal computers. The limitations of such computers are that they are not mobile. On top of that, processing computer vision algorithms on a server is not an option in an area with poor internet coverage.

Even though edge devices like Raspberry Pi tend to have lower performance compared to MacBook Pro, FogNode, Jetson TX2 and Nexus 6P, they are more cost efficient [1]. Edge devices are also part of modern life indicated by at least 90 percent mobile phone penetration in the market [2]. It is also indicated by the increasing trends of Internet of Things (IoT), popularized alongside Artificial Intelligence and Cloud Computing as Industrial Revolution 4.0.

In recent years, edge devices such as Raspberry Pi have seen a rise in computer vision applications. Lots of research have been done to optimize the implementation of computer vision algorithms on edge devices. Edge device utilization can reduce processing latency because it does not require external data transmission. Aside from that, utilizing edge devices can reduce power consumption [3] when doing computation tasks as shown in Figure 1.

An increase in demand for computer vision implementation in the edge led to more efficient, lightweight, and simpler neural network architectures. To accommodate such trends, machine learning frameworks like Tensorflow release frameworks that are dedicated to edge inferencing called Tensorflow Lite. TensorFlow itself already supports a variety of devices including multicore CPUs, general-purpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs) [4]. The release of Tensorflow Lite helps edge devices to achieve similar task, even with

3rd Borobudur International Symposium on Science and Technology 2021
AIP Conf. Proc. 2706, 020129-1–020129-7; https://doi.org/10.1063/5.0120245
Published by AIP Publishing. 978-0-7354-4447-8/$30.00

some tradeoffs. Tensorflow Lite uses an optimized version of inference models. It is designed to be lightweight, fast, and supports multiple edge platforms (mobile and embedded devices).

Even though numerous researches have already utilized Tensorflow Lite in embedded computer vision applications, only a few researches has measured its effectiveness and efficiency compared to the original Tensorflow framework. Research conducted by Chunjie Luo et al. in 2020 compares the performance of different mobile machine learning frameworks and models on several smartphones [5]. Another experiment performed by Xingzhou Zhang et al. in 2018 compares several machine learning frameworks' performances on various edge device types [1]. The result of their research can show the performance metric comparison between the tested frameworks. Unfortunately, the comparison between the original frameworks and their mobile version counterparts is still not identified in detail.

Our experiments in this paper focus on the performance comparison between the Tensorflow Framework and its Lite. The runtime workloads in our inference experiments cover several different popular object detection models. The experiment result will be able to help understanding the advantages and disadvantages between the two frameworks mentioned before.

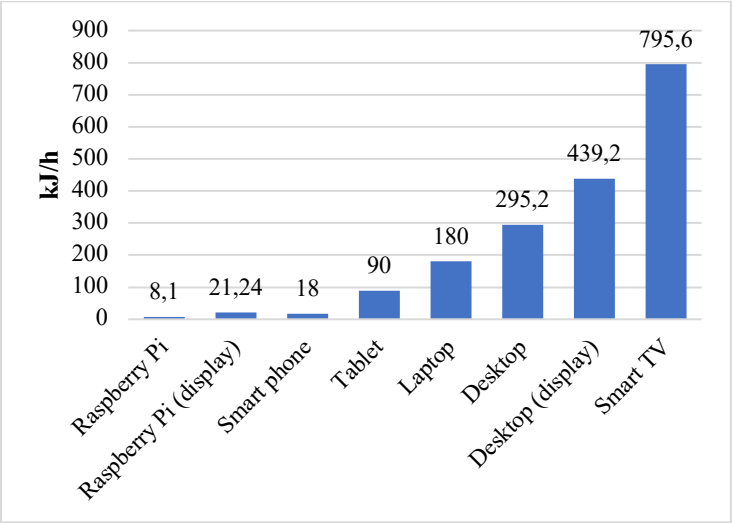

FIGURE 1. Energy consumption of Raspberry Pi compared to other devices (kJ/h)

METHOD

Device Specifications

Our experiments use the Raspberry Pi 4 model B board with 4 GB of RAM capacity. Generally, Raspberry Pi 4 model B uses Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC processor with clockspeed of 1.5GHz for each of its cores [6]. We powered our Raspberry Pi board through the USB-C port. The file storage used is a 16 GB SanDisk Ultra MicroSDHC inserted directly to the MicroSD slot on the Raspberry Pi board. The script used for inference are run through SSH connection to the Raspberry Pi board.

Frameworks

TensorFlow is a popular machine learning framework that provides support for machine learning that is adopted widely. TensorFlow uses dataflow graphs to represent computation, shared state, and the operations that mutate that state. Tensorflow supports parallel computation because it maps the nodes of a dataflow graph across many machines in a cluster. On top of that, it also maps them within a machine across multiple computational devices, including multicore CPUs, general-purpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs) [4]. Models created in Tensorflow can also be executed across different environments. Tensorflow was published and maintained by Google.

Tensorflow Lite is the successor of Tensorflow Mobile. Tensorflow Mobile was initially developed by Google to accommodate running machine learning algorithm on mobile devices. Because machine learning algorithm execution is computationally expensive, Tensorflow Mobile and Tensorflow Lite uses model optimization to improve execution performance.

Tensorflow Lite models are converted from a regular Tensorflow model (SavedModel) or Keras model using the Tensorflow Lite Converter tool provided by Tensorflow package [7]. The conversion result is a model file format based on Flatbuffers that can be used for inferencing on edge devices (Figure 2).

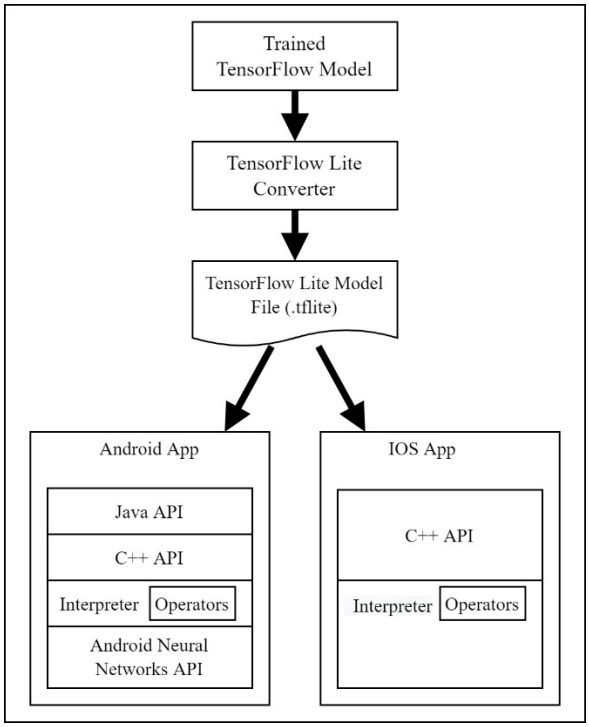

FIGURE 2. Tensorflow Lite architecture

Models

The development of machine learning models has been increasing rapidly in the recent times. Several research has proposed different models each with its own unique approach. The models that are used for performance comparison in this paper are MobileNet V1, MobileNet V2, InceptionV2, and ResNet50 (Figure 3). Tensorflow Lite supports SSD models and CenterNet models, but we only use SSD models in our experiments. All model used in this paper is downloaded from tensorflow model zoo and converted directly to Tensorflow Lite model file (tflite).

MobileNet is a series of models designed and optimized for applications on mobile or embedded devices. It was proposed by Google researchers starting with the first version in 2017 [8], the second in 2018 [9], and the third version in 2019 [10]. MobileNet efficiently trades off between the inference latency and its accuracy. The MobileNet model is based on depthwise separable convolutions. Depthwise separable convolutions is a form of factorized convolutions which factorize a standard convolution into a depthwise convolution and a 1×1 convolution called a pointwise convolution. The pointwise convolution applies a 1x1 convolution to capture the relationship of each feature in every channel. MobileNets structure is mostly built with depthwise separable convolutions except for the first layer which is in the form of a full convolution. We choose this model because of its extensive use in edge inferencing.

Inception was initially called GoogLeNet [11]. It was also a model proposed by Google. GoogLeNet won the task of classification and detection in the ILSVRC (ImageNet Large-Scale Visual Recognition Challenge [12]) 2014. GoogLeNet uses filter sizes of 1x1, 3x3 and 5x5 in different branches to capture multi-scale information of the feature. Its architecture design is mainly considers computational efficiency and practicality. InceptionV2 factorizes the 5x5 convolution to two 3x3 convolution operations to improve computational speed [13]. InceptionV3 that was proposed

in the same paper factorizes 7x7 convolution operations to three 3x3 convolution operations with the same objective. InceptionV4 remove its predecessors complexity by making the model more uniform. The same paper for InceptionV4 also introduces a hybrid inception module that was inspired by the performance of ResNet. In our experiment, we choose InceptionV2 as the model used in performance comparison.

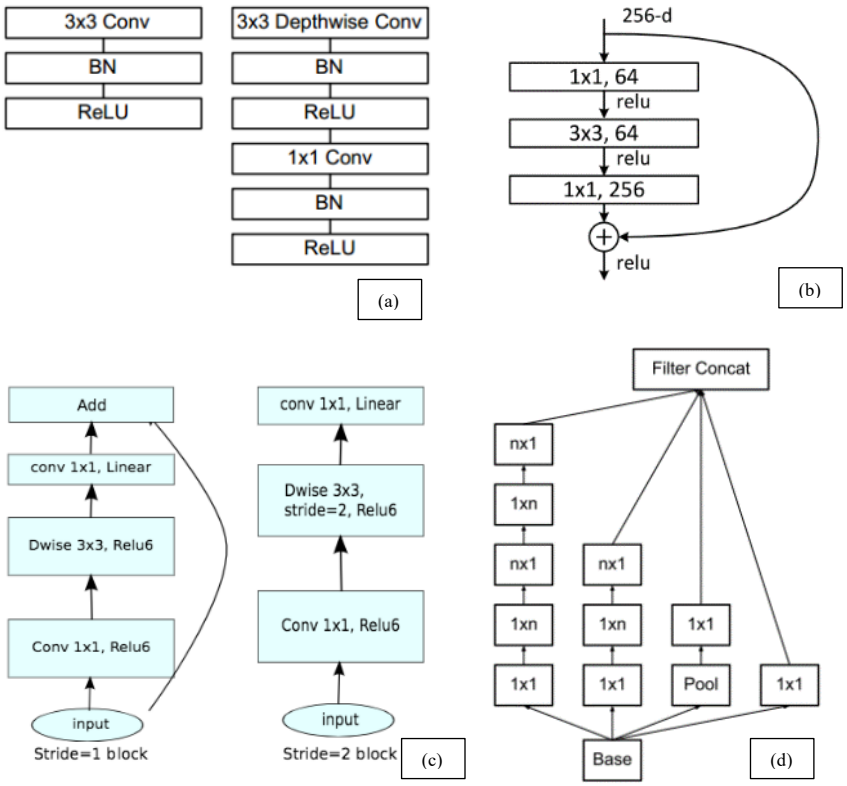

FIGURE 3. (a) Common structure of MobileNetV1, (b) ResNet50, (c) MobileNetV2, and (d) Inception model

ResNets model or Deep Residual Network model has a structure consisting of many stacked "Residual Units"[14]. It was proposed by Microsoft researchers in 2015. It won the ILSVRC [12] 2015 and MS COCO (Microsoft Common Objects in Context [15]) 2015 competitions on the tasks of ImageNet classification, ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation. ResNets model helps the training of a very deep networks where their gradients, in some cases, will become relatively small (vanishing gradient problem). Variants of ResNets model has been released several times later. In our experiment, we chose to use the ResNet50 model because of its popularity. General expression of each residual unit can be seen in Equation 1.

$$y_l = h(x_l) + F(x_l, W_l),$$
$$x_{l+1} = f(y_l) \quad (1)$$

where x_l and x_{l+1} are input and output of the l-th unit, and F is a residual function. Figure 3b shows the common structure of ResNet50 model.

RESULT AND DISCUSSION

Metric

This paper calculated the performance of each framework coupled with several different models during execution time. This paper focuses on the inference latency consisting of initial inference time, average time per inference, and

the total time needed for the inference loop. Aside of that, this paper also compares the memory usage of each framework when running several different models.

The time per inference is calculated by utilizing the "time" function in the python script. The first inference interval took relatively longer due to initialization and thus discarded from the average inference time and only printed to the console interface. The inference loop is run 1000 times to avoid errors. The average time for each iteration is then calculated and printed as the average time per inference.

The memory usage is calculated using the "top" command from the terminal. The shown percentage of usage is then multiplied with the total RAM presents on the Raspberry Pi board. The estimated result is then used to compare the memory footprint of each framework during inference.

Experiment Result

The result of the experiment is shown on Table 1. The presented result shows significant increase of initial inference time on Tensorflow Lite compared to Tensorflow. The result on all models favors Tensorflow Lite over the original Tensorflow framework. The graphical comparison of the result can be seen in Figure 4.

TABLE 1. Initial inference time in miliseconds

Framework	SSD MobileNetV1	SSD MobileNetV2	SSD InceptionV2	SSD ResNet50
Tensorflow	9794.66	18648.57	20919.55	47467.64
Tensorflow Lite	231.28	347.86	1497.60	13413.06

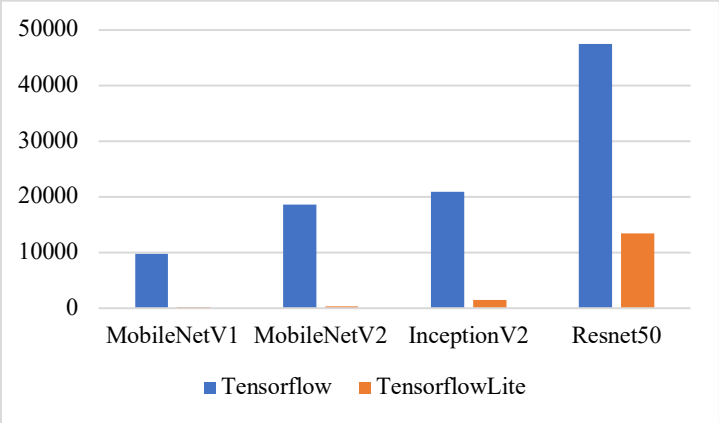

FIGURE 4. Initial inference time with Tensorflow and Tensorflow Lite

Meanwhile, the average inference time after 1000 inference iterations (except for the first iteration) varies between the model used. Using MobileNetV1 model, we can achieve 1.5x faster inference time with Tensorflow Lite compared to the original Tensorflow. Using the MobileNetV2 model with Tensorflow Lite, we can achieve a bigger inference time margin with more than 2x faster inference time compared to using Tensorflow. As for the ResNet50, the inference time is only slightly faster on Tensorflow Lite than Tensorflow. However, when using InceptionV2 model, the inference time is faster on Tensorflow compared to Tensorflow Lite. The average inference time for each framework with several tested pretrained models are shown in Table 2.

TABLE 2. Average inference time in miliseconds

Framework	SSD MobileNet V1	SSD MobileNetV2	SSD InceptionV2	SSD ResNet50
Tensorflow	273.39	551.30	760.51	11080.80
Tensorflow Lite	176.48	243.13	852.53	10888.44

When comparing the memory usage of both frameworks, Tensorflow Lite uses around half the memory used by Tensorflow with the same model (Figure 5). This reduced memory usage is good news for edge devices with relatively smaller memory capacity to run the algorithm.

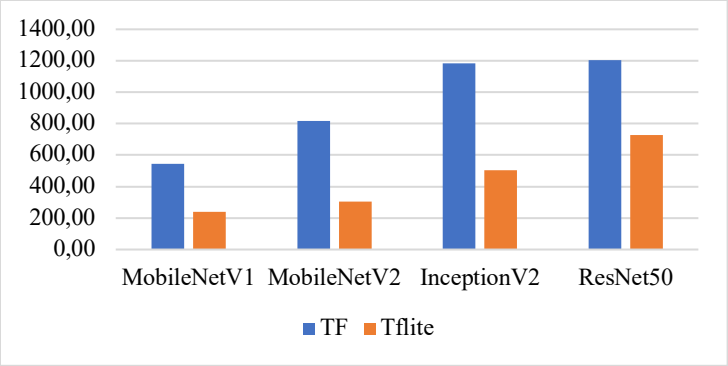

FIGURE 5. Memory usage for when running each model with Tensorflow and Tensorflow Lite

CONCLUSION

The comparison process that consists of latency comparison and memory usage comparison on Raspberry Pi 4 model B shows that Tensorflow Lite improves the performance of inference on the tested board. The low initial inference time needed by Tensorflow Lite can be considered when implementing an object detection that requires quick inference startup. Although average of inference time when using InceptionV2 model and Resnet50 model does not show a great difference, using models dedicated for edge inferencing will improve the performance as we can see with MobileNet models. Moreover, the small memory footprint of Tensorflow Lite can also be considered when running object detection algorithm on devices lower memory. These results should be considered when doing inference on edge devices, such as Raspberry Pi 4.

ACKNOWLEDGMENTS

This research received funding from LP3M Universitas Muhammadiyah Magelang. Our gratitude goes to the laboratory assistants in the Informatics Engineering laboratory of the Universitas Muhammadiyah Magelang.

REFERENCES

1. Zhang, X.; Wang, Y.; Shi, W. PCamp: Performance comparison of machine learning packages on the edges. In Proceedings of the USENIX Workshop on Hot Topics in Edge Computing, HotEdge 2018, co-located with USENIX ATC 2018; 2018.
2. Deloitte Global mobile consumer trends , 2nd edition. *Deloitte* **2017**, 2–20.
3. Akshay, S.; Vishnukumar, B.; Mohan, V.; S. Anand, M. Energy and Performance Analysis of Raspberry Pi with Modern Computing Devices. *International Journal of Engineering & Technology* **2018**, *7*, 777, doi:10.14419/ijet.v7i4.36.24242.
4. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A system for large-scale machine learning. In Proceedings of the Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016; 2016; pp. 265–283.
5. Luo, C.; He, X.; Zhan, J.; Wang, L.; Gao, W.; Dai, J. Comparison and Benchmarking of AI Models and Frameworks on Mobile Devices. *arxiv.org* **2020**.
6. Foundation, T.R.P. Raspberry Pi 4 Model B – Raspberry Pi. *The Raspberry Pi Foundation* **2019**, *4*, 1–15.
7. Google Developers Blog: Announcing TensorFlow Lite.
8. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arxiv.org* **2017**.
9. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition; 2018; pp. 4510–4520.
10. Howard, A.; Sandler, M.; Chen, B.; Wang, W.; Chen, L.C.; Tan, M.; Chu, G.; Vasudevan, V.; Zhu, Y.; Pang, R.; et al. Searching for MobileNetV3. *Proceedings of the IEEE International Conference on Computer Vision* **2019**,

2019-Octob, 1314–1324, doi:10.1109/ICCV.2019.00140.
11. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* **2015**, *07-12-June*, 1–9, doi:10.1109/CVPR.2015.7298594.
12. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* **2015**, *115*, 211–252, doi:10.1007/S11263-015-0816-Y.
13. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In Proceedings of the Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition; 2016; Vol. 2016-Decem, pp. 2818–2826.
14. He, K.; Zhang, X.; Ren, S.; Sun, J. Identity mappings in deep residual networks. In Proceedings of the Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Springer Verlag, 2016; Vol. 9908 LNCS, pp. 630–645.
15. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **2014**, *8693 LNCS*, 740–755, doi:10.1007/978-3-319-10602-1_48.