BENCHCOUNCIL™ AIOTBENCH

STANDARD SPECIFICATION

DRAFT 1.1

SEPTEMBER, 2020

International Open Benchmark Council (BenchCouncil)
http://www.benchcouncil.org
benchcouncil@gmail.com

http://www.benchcouncil.org/AIoTBench/index.html

Acknowledgements

The BenchCouncil acknowledges the substantial contribution of AIoTBench proposal principal submitters: Institute of Computing Technology (ICT), Chinese Academy of Sciences(CAS), and other organizations, and the proposal reviewer BenchCouncil AI committee. The BenchCouncil also acknowledges the work and contributions of the BenchCouncil AIBench subcommittee member companies or institutions in developing the AIoTBench specification.

The BenchCouncil AIoTBench subcommittee would like to acknowledge the contributions made by the many members during the development of the benchmark specification. The list of significant contributors to this version includes Chunjie Luo (ICT, CAS), Xiwen He (ICT, CAS), Lei Wang (ICT, CAS), Fan Zhang (ICT, CAS), Wanling Gao (ICT, CAS), Cheng Huang (ICT, CAS), Xingwang Xiong (ICT, CAS), Jianan Chen (ICT, CAS), Hainan Ye (BAFST), Tong Wu (CNIM), Runsong Zhou (CSTC), and Jianfeng Zhan (ICT, CAS).

AIoTBench Proposal Principal Submitter: Institute of Computing Technology, Chinese Academy of Sciences

AIoTBench Proposal Reviewer: BenchCouncil AI Committee

Contact: benchcouncil@gmail.com (BenchCouncil); luochunjie@ict.ac.cn (ICT, CAS)

BenchCouncil Membership

(as of September 2020)

Full Members

Associate Members

Document Revision History

Date	Version	Description
16 November 2019	Draft 1.0	Mail ballot version (proposed standard)
01 September 2020	Draft 1.1	Standard specification released to the public

Contents

1 Introduction

Due to increasing amounts of data and compute resources, the deep learning achieves many successes in various domains. Recently, researchers and engineers make effort to apply the intelligent algorithms to the mobile or embedded devices, e.g. smart phone, self-driving cars, smart home. On one hand, the neural networks are made more light-weight to adapt the mobile or embedded devices by using simpler architecture, or by quantizing, pruning and compressing the networks. On the other hand, the mobile and embedded devices provide additional hardware acceleration using GPUs or NPUs to support the AI applications. Since AI applications on mobile and embedded devices get more and more attention, benchmarking and ranking of the AI ability of those devices becomes an urgent problem to be solved.

AIoTBench aim to provide a executable mechanism to rank the inference abilities of the mobile and enbeded devices. It covers six model architectures and three frameworks. Specifically, AIoTBench covers three typical heavy-weight networks: ResNet50 [1], InceptionV3 [2], DenseNet121 [3], as well as three light-weight networks: SqueezeNet [4], MobileNetV2 [5], MnasNet [6]. Each model is implemented by three frameworks: Tensorflow Lite, Caffe2, Pytorch Mobile. To compare and rank the AI capabilities of the devices, AIoTBench uses two unified metrics as the AI scores: Valid Images Per Second (VIPS) and Valid FLOPs Per Second (VOPS). They reflect the trade-off between quality and performance of the AI system.

AIoTBench can be used for:

- Comparison of different AI models. Users can make a tradeoff between the accuracy, model complexity, model size and speed depending on the application requirement.
- Comparison of different AI frameworks on mobile environment. Depending on the selected model, users can make comparisons between the AI frameworks on mobile devices.
- Benchmarking and ranking the AI abilities of different mobile devices. With diverse and representative models and frameworks, the mobile devices can get a comprehensive benchmarking and evaluation.

2 Terminology and Background

2.1 Terminology

The terminologies used in this specification are as follows.

AI: Artifictial Intelligence

AIoT: Artifictial Intelligence on Things. Things refer to end devices which have limited computing and memory resources comparing to datacenter servers, e.g mobile and embeded devices.

Benchmark: A set of programs, datasets and metrics to evaluate the relative performance of an object.

AIoTBench: The benchmark released by BenchCouncil to evalute the performance of AIoT.

ResNet50: A heavy-weight architecture of deep neural network.

InceptionV3: A heavy-weight architecture of deep neural network.

DenseNet121: A heavy-weight architecture of deep neural network.

SqueezeNet: A light-weight architecture of deep neural network.

MobileNetV2: A light-weight architecture of deep neural network.

ShuffleNetV2: A light-weight architecture of deep neural network.

Bottleneck: A modular structure used in deep neural networks.

Block: A modular structure used in deep neural networks

FLOPs: The number of multiply-accumulates needed by a neural network to compute an inference on a single image.

VIPS: Valid Images Per Second.
VOPS: Valid FLOPs Per Second.

2.2 Background

AI achieves many successes in various domains, e.g. image recognization, natural language processing, and speech recognization. Even for a specific domain task, there are various models proposed. Moreover, there are lots of popular deep learning frameworks on which developer can implement their AI applications. For some equipment suppliers, their productions are designed for common purpose. For example, the mobilephone could run different AI models implemented by different frameworks. So, their produtions need to support diverse workloads well. Other equipment suppliers pay attention to a single purpose. . Because the workload is already determined, they could optimize their design according to the target workload. Specialization philosophy simplifies the design of the AI system, and it becomes more and more popular, especially in the embedded field. AIoTBench considers both generalization and specialization scenes through reporting the average performance and the best performance among the workloads.

3 Benchmarking Methodology

AIoTBench follows the PRDAERS benchmarking rules and methodology [7] presented by BenchCouncil. The PRDAERS is the abbreviation of paper-and-pencil, relevant, diversity, abstractions, evaluation metrics and methodology, repeatable, and scaleable.

Paper-and-pencil Approach: AIoTBench can be specified only algorithmically in a paper-andpencil approach. This benchmark specification is proposed firstly and reasonably divorced from individual implementations. In general, AIoTBench defines a problem domain in a high-level language.

Relevant: AIoTBench is simplified and distillated of the real-world application. It abstracts the typical scenario and application of the AIoT.

Diversity and Representatives: Modern workloads show significant diversity in workload behavior with no single silverbullet application to optimize for [8]. Consequently, diverse workloads should be included to exhibit the range of behavior of the target applications. AIoTBench covers the typical light-weight models and development frameworks used in mobile and embeded enviroments.

Abstractions: AIoTBench abstracts the image classification task. AIoTBench is light-weight thus portable across different mobile and embeded devices.

Evaluation Metrics and Methodology: The performance number of AIoTBench is simple to count

and understand. It reflects both the inference accuracy and the speed of the AI devices.

Repeatable, Reliable, and Reproducible: The testing results of AIoTBench are repeatable, reliable, and reproducible.

Scaleable: Models used in AIoTBench have different scale configuration to cover different parameter size and computing complexity.

4 Design

4.1 Benchmark procedure

A complete AIoTBench testing system, as shown in Fig 1, contains multiple components: data set, workload select module, inference module, and score module. The data set is indexed by ID, and placed at the permanent storage of the device in advance. The data set should be the original format, and can not be compressed or preprocessed. Workload select module allows users to choose which workload to run. Inference module executes the workload. It contains four steps, data reading , proprocessing, predicting, and logging. The time of data reading, preprocessing and predicting is counted in final executing time, since reading and preprocessing also matter the performance of AI system significantly. Logging format should be "image_id, workload_id, real_lable, predict_lable, time". After all testing finished, score module analysizes the log file, and computes the final score.

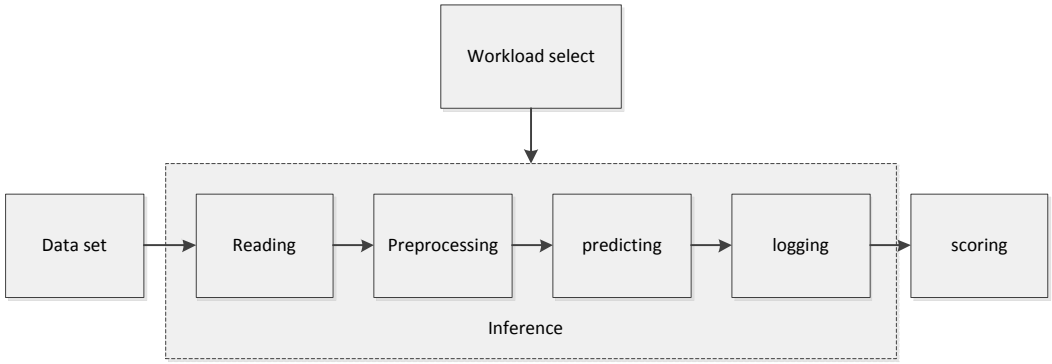


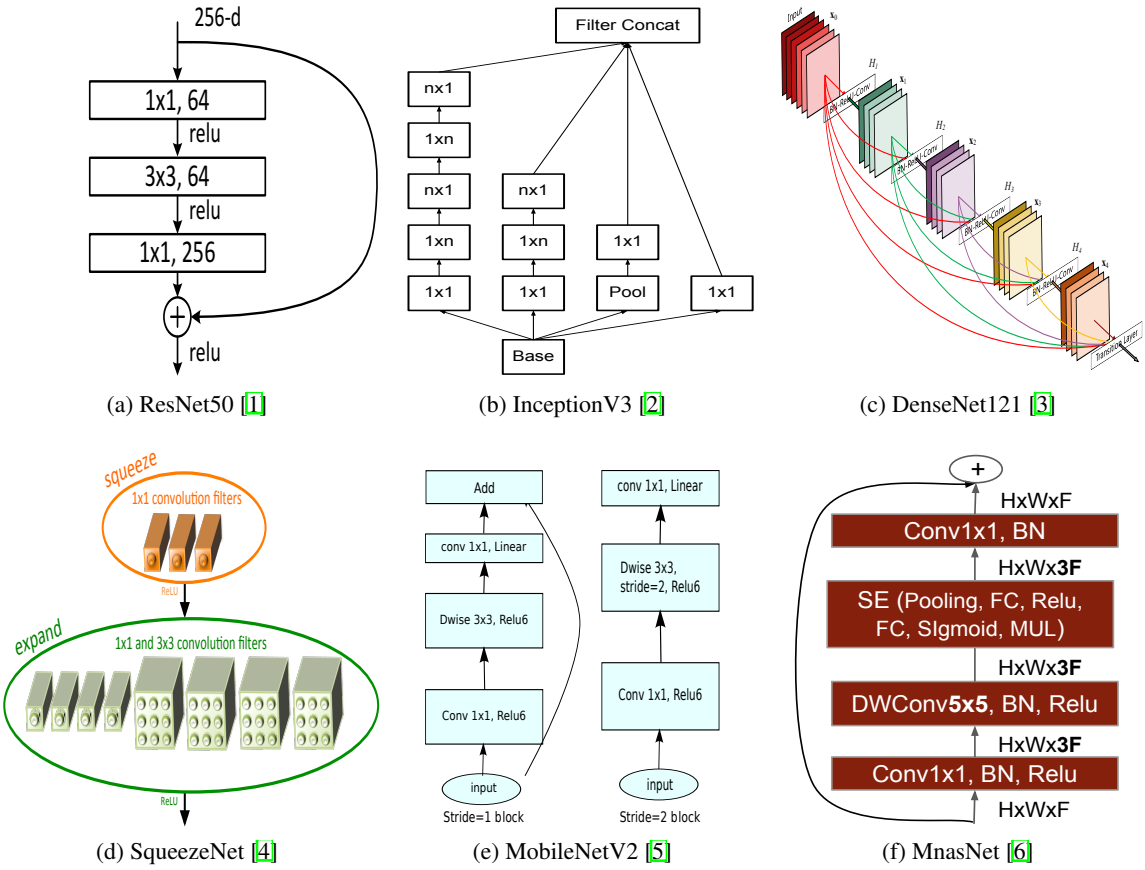
Figure 1: The procedure of AIoTBench.

4.2 Task

Currently, AIoTBench contains the task of image classification. A classifier network takes an image as input and predicts its class. Image classification is a key task of pattern recognization and artificial Intelligence. It is intensively studied by the academic community, and widely used in commercial applications of AIoT. Image classification is also widely used in other AI benchmarks, and becomes a de facto standard to evaluating AI system. ImageNet classification dataset [9] is used, which has 1280000 training images and 50,000 validation images with 1000 classes. In AIoTBench, all classification models are trained on the ImageNet 2012 training set. And the inference uses the ImageNet 2012 validation set.

4.3 Model

Considering the diversity and popularity, AIoTBench uses three typical heavy-weight networks: ResNet50 [1], InceptionV3 [2], DenseNet121 [3], as well as three light-weight networks: SqueezeNet [4], MobileNetV2 [5], MnasNet [6]. These networks are designed with very different philosophies and have very different architectures. And they are widely used in the academia and industry. The typical modules of



(a) ResNet50 [1] (b) InceptionV3 [2] (c) DenseNet121 [3]

(d) SqueezeNet [4] (e) MobileNetV2 [5] (f) MnasNet [6]

Figure 2: The typical modules of different models

the models used in AIoTBench are shown in Fig. 2. And Table 1 presents the FLOPs [1], parameters, and the original reference accuracy of the selected models.

Table 1: The FLOPs, parameters, and the original reference accuracy of the models used in AIoTBench

Model	**FLOPs** (Millions)	**Parameters** (Millions)	**Accuracy** (%)
ResNet50	3800	25.6	76
InceptionV3	5000	23.2	78.8
DenseNet121	2800	8	74
SqueezeNet	833	1.25	57.5
MobileNetV2	300	3.4	72
MnasNet	315	3.9	75.2

4.4 ResNet

ResNet [1] was proposed by Microsoft Research in 2015. In ILSVRC (ImageNet Large-Scale Visual Recognition Challenge [9]) 2015 and Microsoft COCO (Microsoft Common Objects in Context [10]) 2015 competitions, methods based on ResNet won the 1st places on the tasks of ImageNet classification, ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation. Moreover, it won the best paper award of CVPR 2016 [11]. Many variants of ResNet have proposed later, e.g. PreAct

[1] FLOPs (some literatures use Mult-Adds) refers to the number of multiply-accumulates to compute the model inference on a single image. FLOPs is widely adopted as the metric to reflect the computational complexity of a model.

ResNet [12], ResNeXt[13]. Considering the tradeoff between accuracy and computation cost, ResNet50 is the most popular architecture among the family of ResNet and its variants. Fig. 2a shows the typical module of ResNet50. It can be defined as $y = F(x) + x$, where x and y are the input and output feature maps. The function F consists of 1x1, 3x3 and 1x1 convolutional layers. The key contribution is the identity mapping ($+x$) which is performed by a shortcut connection and element-wise addition. Without extra parameter and computation complexity, identity mapping significantly alleviates the vanishing gradient problem which hinders the training of very deep networks. With 25.6 million parameters and 3800 million FLOPs, ResNet50 achieves 76% validation accuracy on ImageNet.

4.5 Inception

The first version of Inception, also called GoogLeNet [14], was proposed by Google in 2014. GoogLeNet is the winner of the task of classification and detection in the ILSVRC 2014. GoogLeNet increases the depth and width of the network while keeping the computational budget constant. It uses filter sizes of 1x1, 3x3 and 5x5 in different branches to capture multi-scale information of the feature. The computational efficiency and practicality are the main design considerations of Inception architecture. InceptionV2 and InceptionV3 [15] have a similar architecture. As shown in Fig. 2b, InceptionV3 factorizes convolutions with large filter size into smaller convolutions, e.g. using two stacked 3x3 convolutional filters instead of 5x5 filter. Further, it spatially factorizes standard convolutions into asymmetric convolutions, e.g. using 1x3 following by 3x1 instead of 3x3. Comparing to InceptionV2, InceptionV3 uses several training tricks to achieve higher accuracy. InceptionV4 [16] adds residual connection into Inception architecture. AIoTBench chooses InceptionV3 in our benchmark because of its extensive use in the community. InceptionV3 have 23.2 million parameters and 5000 million FLOPs, and it achieves 78.8% accuracy.

4.6 DenseNet

DenseNet [17] was proposed by Cornell University, Tsinghua University and Facebook in 2017. It won the best paper award of CVPR 2017 [18]. Fig. 2c shows the dense block of DenseNet. Different from ResNet which use identity mapping to make shortcut connection, DenseNet connects all layers directly with each other by concatenating them. Each layer concatenates all its preceding layers and passes on itself to all subsequent layers. By direct and dense connection, DenseNet alleviates the vanishing gradient problem and strengthen feature propagation. The densely connected block use very narrow layers, thus reduce the number of parameters. Although each output layer is narrow, the dense block typically has many more inputs since it concatenates all its preceding layers. To reduce the number of input and improve computational efficiency, a 1x1 convolution is used as bottleneck before each 3x3 convolution. DenseNet121 has 8 million parameters, 2800 million FLOPs, and achieve 74% accuracy.

4.7 SqueezeNet

SqueezeNet [4] was proposed by UC Berkeley and Stanford University in 2016. The main purpose of SqueezeNet is to decrease the number of parameters while maintaining competitive accuracy. SqueezeNet employs three main strategies when designing CNN architectures to decrease the parameters: 1) Replace 3x3 filters with 1x1 filters, 2) Decrease the number of input channels to 3x3 filters, 3) Downsample late in the network so that convolution layers have large activation maps. Following these strategies, the Fire module, as shown in Fig. 2d, is designed: 1) a squeeze convolution layer which has only 1x1 filters, 2) followed by an expand layer that has a mix of 1x1 and 3x3 convolution filters. SqueezeNet achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. It has only 1.25 million of total parameters, but achieves 57.5% accuracy. Since SqueezeNet focuses on the compression of model size, it dose not consider the computation complexity when designing, and have 833 million FLOPs.

4.8 MobileNet

MobileNet is another series of models proposed by Google. As a light-weight deep neural network, MobileNet is designed and optimized for mobile and embedded vision applications. The first version of MobileNet [19] is published in 2016, the second version MobileNetV2 [20] is published in 2018, and the third version MobileNetV3 [21] is published in 2019. MobileNet is based on depthwise separable convolutions to reduce the number of parameters and computation FLOPs. Depthwise separable convolutions, initially proposed in [22], factorizes a standard convolution into a depthwise convolution and a pointwise convolution. The depthwise convolution applies a single 3x3 filter to each input channel to capture the spatial relationship of features. The pointwise convolution applies a 1x1 convolution to capture the channel-wise relationship of features. This factorization is widely adopted by the following design of light-weight neural networks, e.g. MobileNetV2 [20], MobileNetV3 [21], ShuffleNet [23], ShuffleNetV2 [24], MnasNet [25]. Beside depthwise separable convolutions, MobileNetV2 introduce two optimized mechanisms: 1) inverted residual structure where the shortcut connections are between the thin layers. 2) linear bottlenecks which removes non-linearities in the narrow layers. MobileNetV3 is searched by hardware-aware network architecture search (NAS) algorithm. Considering the popularity and diversity, AIoTBench chooses the MobileNetV2, and another light-weight architecture searched by NAS, MnasNet, which is described in Section 4.9. MobileNetV2 has 3.4 million parameters, 300 million FLOPs, and achieve 72% accuracy.

4.9 MnasNet

Automated machine learning (AutoML) has emerged as a hot topic. As the main domain of AutoML, neural architecture search (NAS) [26, 27, 28, 29] automatically find the architecture in the pre-defined search space. It can find the optimal combination structure of existing neural unit, but can not invent new techniques. NAS reduces the demand for experienced human experts comparing to hand-drafted design. MnasNet [25] is a neural architecture automated searched for mobile device by using multi-objective optimization and factorized hierarchical search space. It is proposed by Google in 2018. Instead of using FLOPs to approximate inference latency, the real-world latency is directly measured by executing the model on real mobile devices. In fact, the final architecture of MnasNet is very heterogeneous. Fig. 2f shows one of the typical modules of MnasNet. The module is sequence of 1x1 pointwise convolution, 3x3 depthwise convolution, Squeeze-and-Excitation module (Squeeze-and-Excitation is a light-weight attention module originally proposed in [30], which is the winner of the ILSVRC 2017 classification task.), and 1x1 pointwise convolution. MnasNet has 3.9 million parameters, 315 million FLOPs, and achieve 75.2% accuracy.

4.10 Metrics

Distilling the AI capabilities of the system to a unified score enables a direct comparison and ranking of different devices. AIoTBench uses two unified metrics as the AI scores: Valid Images Per Second (VIPS) and Valid FLOPs Per Second (VOPS).

$$VIPS = \sum_{i=1}^{n} accuracy_i * \frac{1}{time_i} \tag{1}$$

$$VOPS = \sum_{i=1}^{n} accuracy_i * FLOPs_i * \frac{1}{time_i} \tag{2}$$

where $accuracy_i$ refers to the validation accuracy of the i_{th} test, $time_i$ refers to the average running time per image of the i_{th} test, and $FLOPs_i$ refer to the FLOPs of the model used in the i_{th} test.

The inverse proportion to the average running time ($\frac{1}{time_i}$) reflects the throughput of the system. Since the trade-off between quality and performance is always the consideration in AI domain, the accuracy is used as the weight coefficient to compute the final AI score. VIPS is a user-level or application-level

metric, since how many images can be processed is the end-user's concern. VOPS is a system-level metric, and it reflects the valid computation that the system can process per second.

The accuracy is an important factor when benchmarking the AI system, since many architectures can trade model quality for lower latency or greater throughput [31] [32]. MLPerf Inference requires that all benchmarking submissions should achieve a target accuracy, then comparing different systems using metrics like latency or throughput. MLPerf Inference does not offer a unified AI score. Different from MLPerf Inference, AI Benchmark offers a unified AI score. It considers the accuracies as part of over 50 different attributes. The final AI score is calculated as a weighted sum of those attributes. The weight coefficients are calibrated based on the results of particular device. AIoTBench also offer the unified AI scores. When computing the final scores, the accuracies are directly used as the weight coefficients of throughput. The metrics of AIoTBench are more intuitive and explicable. They can be explained as the valid throughput of the system, and reflects the trade-off between quality and performance.

4.11 Benchmarking Constraints

AIoTBench should satisfy the following constraints.
1) AIoTBench allows the adjusting of setting of software and hardware on which the the workloads are running.
2) AIoTBench disallows modification of the workloads, including retraining of the model, model pruning, and reducing the input data size.

5 Reference Implementation

5.1 Framework

For the mobile and embeded devices, the framework, with which the models are implemented, is also part of the workload. In AIoTBench, each model is inplemted by three frameworks: Tensorflow lite, Caffe2, Pytorch mobile.

5.1.1 Tensorflow Lite

TensorFlow, released by Google, is a free and open-source software library for dataflow and differentiable programming. It is widely used for machine learning applications such as neural networks. Tensorflow uses static computational graphs. Tensorflow lite is released for deploying the models trained by tensorflow on mobile and embeded devices. After the model is trained, it need be converted to a .pb graph, and then executed on mobile or embeded devices using the tensorflow lite interpreter, available on Android as well as iOS platforms.

5.1.2 Caffe2

Caffe is an open-source deep learning framework, originally developed at UC Berkeley. Caffe2, built on the original Caffe and released by Facebook, is a light-weight and modular framework for production-ready training and deployment. Its mobile version supports iOS and Android platforms. Caffe2 uses static computational graphs and nodes representing various operators. The deploy of the model on mobile and embeded devices is similar with tensorflow lite.

5.1.3 Pytorch Mobile

Pytorch, primarily developed by Facebook's AI Research lab (FAIR), is an open-source machine learning library based on the Torch library. It aims to replace for NumPy to use the power of GPUs and provide a deep learning research platform. Pytorch uses dynamic computational graphs. PyTorch mobile supports an end-to-end workflow from Python to deployment on iOS and Android. The deploy of the model on mobile and embeded devices is similar with tensorflow lite.

6 Measuring Prodedure

AIoTBench result submission should contain the following information: the hardware and software configurations, VFPS score of each workload, the average VFPS score, and the maximum VFPS score. All the data are uploaded to BenchHub, a code management system used to host source code and manage projects. The BenchHub's web address is www.benchcouncil.org/benchhub. After the submission, AIoTBench community will check the validity, repeatability, and authenticity of the results. If the results are abidance by the rules, the submission will be reported online.

The tests includes 6 models: ResNet50 (using re for short), InceptionV3 (in), DenseNet121 (de), SqueezeNet (sq), MobileNetV2 (mo), MnasNet (mn). For each model, there are four implementations of Pytorch Mobile (py), Caffe2 (ca), Tensorflow Lite with CPU (tfc), and Tensorflow Lite with NNAPI delegate (tfn). As a result, There are 24 (6 models * 4 implementations) tests for each device. The Garbage Collection of JVM and the heat generation of the mobile device may affect the test. For a fair comparison, the device should be shutdown and waiting at least five minutes of cooling time between each test. The accuracy and the average inference time are logged for counting the final AI score of the measured device.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[4] 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.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[6] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

[7] J. Zhan, L. Wang, W. Gao, and R. Ren, "Benchcouncil's view on benchmarking ai and other emerging workloads," *arXiv preprint arXiv:1912.00572*, 2019.

[8] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 158–169, 2016.

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[10] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, 2014.

[11] "The 29th ieee conference on computer vision and pattern recognition (cvpr 2016)." http://cvpr2016.thecvf.com/.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *ArXiv*, vol. abs/1603.05027, 2016.

[13] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995, 2017.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2015.

[16] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI*, 2016.

[17] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2016.

[18] "The 30th ieee conference on computer vision and pattern recognition (cvpr 2017)." http://cvpr2017.thecvf.com/.

[19] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.

[20] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[21] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1314–1324, 2019.

[22] L. Sifre and S. Mallat, "Rigid-motion scattering for texture classification," *ArXiv*, vol. abs/1403.1687, 2014.

[23] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 2017.

[24] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," *ArXiv*, vol. abs/1807.11164, 2018.

[25] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2815–2823, 2018.

[26] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *ArXiv*, vol. abs/1611.01578, 2016.

[27] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *ArXiv*, vol. abs/1611.02167, 2016.

[28] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," *ArXiv*, vol. abs/1712.00559, 2017.

[29] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *ArXiv*, vol. abs/1802.03268, 2018.

[30] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[31] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, *et al.*, "Mlperf inference benchmark," *arXiv preprint arXiv:1911.02549*, 2019.

[32] A. Ignatov, R. Timofte, A. Kulik, S. soo Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. V. Gool, "Ai benchmark: All about deep learning on smartphones in 2019," *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 3617–3635, 2019.