JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 17, No. 2, June 2023, pp. 51-59

Edge Devices Inference Performance Comparison

Rafal Tobiasz*, Grzegorz Wilczynski*, Piotr Graszka, Nikodem Czechowski, and Sebastian Luczak
Bulletprove Spółka Z Ograniczoną Odpowiedzialnością, Puławy, Poland
rafal.tobiasz@bulletprove.com, grzegorz.wilczynski@bulletprove.com, piotr.graszka@bulletprove.com, nikodem.czechowski@bulletprove.com, sebastian.luczak@bulletprove.com

Abstract
In this study, we investigated the inference time of the MobileNet family, EfficientNet V1 and V2 family, VGG models, ResNet family, and InceptionV3 on four edge platforms: Specifically, NVIDIA Jetson Nano, Intel Neural Stick, Google Coral USB Dongle, and Google Coral PCIe. Our main contribution is a thorough analysis of the afore mentioned models in multiple settings, especially as a function of input size, the presence of the classification head, its size and the scale of the model. Since throughout the industry, these architectures are mainly used as feature extractors we majorly analyzed them as such. We show that Google platforms offer the fastest average inference time, especially for newer models like MobileNet or EfficientNet family, while Intel Neural Stick is the most universal accelerator allowing it to run most architectures. These results will provide guidance to engineers in the early stages of AI Edge systems development. The results are accessible at https://bulletprove.com/research/edge_inference_results.csv.

Category: Computer Graphics / Image Processing

Keywords: Edge device; Deep learning; Computer vision

I. INTRODUCTION

A variety of applications exploit machine learning models, be it on a cloud [1-3], personal computers, mobile devices [4, 5] or edge devices [6]. Especially, for the latter, we can observe intensified development of devices and suitable algorithms, since progressively more IoT applications use artificial intelligence (AI) solutions.

A significant fraction of all those applications are in the computer vision domain, such as classification [7, 8], object detection [9, 10], and image segmentation [11]. Those algorithms require a vast amount of computational power to perform inference since the input data is high resolution.

In addition, there are other reasons that encourage the development of edge devices:

- Network load: sending high-resolution data from a vast amount of Internet of Things (IoT) devices to the computational unit may result in unwanted and unpredicted time delays [12, 13];
- Computational unit load: analyzing high-resolution data using current state-of-the-art models may result in a cost-inefficient system [14];
- Safety: sending raw data to the cloud may increase the risk of hacking [15-17] or could lower the trust of a user who, e.g., for a face detection system, does not want his photos on an undisclosed server. Instead, it is better to use a feature extractor on an edge device and send only those anonymous features to the cloud.

To address those challenges, many companies are targeting specialized inference chips, which result in a vast amount of different edge accelerators. Moreover,

Open Access http://dx.doi.org/10.5626/JCSE.2023.17.2.51 http://jcse.kiise.org

Received 02 March 2023; **Accepted** 02 June 2023
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

currently, various architectures are well-scalable and can extract features correctly. However, it is challenging to pick "the best" algorithm or platform. It also varies across applications. Therefore, when choosing a platform and a model to start with, an engineer faces the time-consuming task of testing different variants.

Only a few papers have addressed model profiling on different AI accelerators, among them is work done by Reza et al. [18], and only a few architectures (often with default parameters) have been analyzed. Excellent research in this field has been done by Reddi et al. [19] which defines the proper way of different profiling methods. The benchmarking method that was introduced in their study allows users to perform their tests and share results on a moderated platform.

However, none of those papers present any data regarding feature extractors. Authors analyze models only as classifiers, whereas those pretrained models are mostly used as feature extractors. Therefore, analyzing them as classifiers may be slightly confusing in this setting. Profiling those algorithms only as feature extractors provides clear information on how long the first component of the final algorithm will last.

In this work, we present a comparison of the most popular models available in TensorFlow [20] Keras applications—MobileNet family [21-23], EfficientNet (V1 and V2) family [24, 25], ResNet (V1 and V2) family [26, 27], VGG family [28], and InceptionV3 [29]—on multiple platforms (NVIDIA Jetson Nano, Google Coral USB, Google Coral PCI, and Intel Neural Stick). Those algorithms were also analyzed based on different factors including input sizes and scaling parameters. Despite focusing on profiling models as feature extractors, we also examined them with a classic ImageNet head (1,000 classes) and a more real-life scenario (five output neurons); those results are available in the downloadable CSV file.

This work aimed to do an in-depth comparison of the performance of different models on multiple edge devices to make machine-learning engineers' working process more time- and cost-efficient.

II. EDGE AI ACCELERATORS

AI on Edge is focusing on running artificial intelligence models on Edge devices [30]. It allows low power consumption and small physical size at the cost of performance. An Edge AI accelerator is a hardware specialized in processing AI workloads at the edge. Computation is local, close to data collection, which can be beneficial in preserving data privacy or in offline scenarios. Moreover, it reduces latency and communication costs when compared to cloud AI.

A. Google Coral Accelerator

Google Coral Accelerator expands the user's system with an application-specific integrated circuit (ASIC) called Edge TPU, designed to deploy high-quality AI at the edge. Coral can perform 4 trillion operations per second using 2 W of power. Edge TPU supports only 8-bit quantized TensorFlow Lite models compiled using a dedicated tool [31]. Among others, potential use includes predictive maintenance, voice recognition, anomaly detection, machine vision, and robotics [32]. In this paper, we tested two IO interface versions of Google Coral, PCIe and USB.

B. NVIDIA Jetson Nano

NVIDIA Jetson Nano is a small computer equipped with 128-core NVIDIA Maxwell GPU, quad-core ARM Cortex-A57 MPCore processor, and 4 GB of 64-bit LPDDR4 of memory. It is the only standalone Edge device used in this comparison. The platform has an AI performance of 472 giga-floating point operations per second (GFLOPS) and uses 5–10 W of power [33]. Jetson utilizes NVIDIA JetPack SDK, which provides useful features like CUDA, cuDNN, and TensorRT. TensorRT optimizes inference of deep learning frameworks. For example, it allows for the use of FP16 precision for inference [34]. Among others, potential use includes predictive maintenance, voice recognition, anomaly detection, machine vision, robotics, patient monitoring, and traffic management [35].

C. Intel Neural Compute Stick 2

Neural Compute Stick 2 (NCS2) is a plug-and-play AI accelerator. It contains Intel Movidius Myriad X Vision Processing Unit (VPU), which includes 16 SHAVE cores and a dedicated deep neural networks (DNN) hardware accelerator. Intel distribution of the OpenVINO toolkit is used to convert and optimize models for this platform. Among others, potential use includes anomaly detection and machine vision [36].

III. METHODOLOGY

A. Compared Models

TensorFlow is one of the most popular deep learning frameworks. Alongside many tools for creating, training, and profiling neural networks, it also provides a set of already trained popular image classification networks. Those architectures are widely used in the industry. We picked multiple algorithm families based on different motives.

1) MobileNet

This family of models consists of MobileNetV1, MobileNetV2, MobileNetV3-Small, and MobileNetV3-

Large. They were particularly designed to run efficiently on mobile devices, hence the name.

MobileNetV1 [21] is based on a streamlined architecture that uses depth-wise separable convolutions. The authors introduced two hyperparameters that efficiently tradeoff between latency and accuracy:

- Width multiplier α: number of layer's input channels M (where M is the default number of channels) becomes $\alpha * M$, and the number of output channels N becomes $\alpha * N$. It takes place for every layer.
- Resolution multiplier ρ is implicitly set by setting the input resolution.

Its fast inference is an effect of putting nearly all of the computation into dense 1×1 convolutions. It is crucial because they are implemented with highly optimized general matrix multiply (GEMM) functions and do not need any initial reordering in memory.

MobileNetV2 [22] is based on an inverted residual structure where the shortcut connections are between the thin bottleneck layers. Lightweight depth-wise convolutions are the source of nonlinearity since they were removed in the narrow layers. The two hyperparameters that were introduced earlier remain the same. This network improved the state-of-the-art for a wide range of performance points on ImageNet [37] classification benchmark.

MobileNetV3 [23] models leverage complementary search techniques complemented by the NetAdapt algorithm. The authors also introduced the swish activation function and rebuilt MobileNetV2's bottleneck by adding squeeze and excitation in the residual layer. MobileNetV3 family outperformed, at that time, the current state-of-the-art models on ImageNet taking into account top-1 accuracy and latency.

2) EfficientNet (V1 and V2)

In their work [24], the authors improved model scaling and identified that balancing network depth, width, and resolution leads to better performance. They focused on optimizing FLOPS rather than latency since they did not target specific hardware. By using the neural architecture search (NAS) they came up with a new scalable baseline network and called this family of models EfficientNets (now EfficientNets V1). There are eight models available in TensorFlow applications, from B0 to B7.

In the following paper, the authors once again used a combination of NAS and scaling to optimize the training speed and parameter efficiency. It resulted in the EfficientNetV2 [25] family. It achieved the state-of-the-art top-1 accuracy on ImageNet, outperforming even the famous ViT [38]. This family consists of seven models, from B0 to B3 and S, M, and L.

3) InceptionV3

As the Inception family of models had a crucial role in the development of convolutional neural networks for vision tasks, we decided to profile InceptionV3 [29]. Here authors aimed to utilize added computation more efficiently by factorizing convolutions and adding aggressive regularization. InceptionV3 achieved, at the time, the state-of-the- art top-1 error on ImageNet.

4) VGG

As with the former model, we decided to profile this [28] family of networks based on historical reasons. The authors' main contribution was to increase the depth of the convolutional network using small 3×3 kernels. This allowed them to build 16–19 (VGG16, VGG19) layered architectures that earned first and second place at the ImageNet Challenge 2014 in the localization and classification tasks, respectively.

5) ResNet (V1 and V2)

In their work [26], the authors introduced residual connections that allowed the training of substantially deeper networks than their predecessors. On ImageNet they evaluated ResNet-152—networks 8× deeper than VGGs and with lower complexity—achieving state-of-the-art. We profiled ResNet-50, ResNet-101, and ResNet-152 (we refer to this family as ResNetV1).

In the following work [27], the authors took on the propagation formulations behind the residual building blocks, trying to make the forward and backward signals flow easier. They rethought the residual blocks, moreover removed ReLU from the "easiest" path after the addition operation. For the ResNetV2 family, we profiled analogous models to its predecessor. We profiled ResNet families because of the same reasons as InceptionV3 and VGGs.

B. Model Parameters

The only measurement that was taken in this work is the inference time. Since we wanted to profile the influence of different model hyperparameters, we created extensive sets of architectures for every model family. Each neural network is built with different parameters.

1) Input size

Nowadays, AI applications need to analyze images of various sizes, ranging from 224^2 to 1024^2, or larger. This trend is also noticeable on edge devices. Taking into account restricted computational resources an engineer has to pick a suitable architecture that will allow efficient use of the model, e.g., in real-time.

2) Preprocessing

Every analyzed model has some specific input preprocessing, e.g., reducing the mean and dividing by standard deviation. It is important to know how such simple data alteration will affect the model's inference time.

3) Classification heads

In this work, our key focus is on analyzing models as feature extractors (no classification heads). Currently, those architectures are used as feature extractors in the industry, hence, it is crucial to know how computing features will affect the inference time. Furthermore, we profiled architectures with classic ImageNet head (1,000 classes) to make our results comparable to others. Last but not least, we analyzed smaller classification heads (five classes) which represent more "real-life" case scenarios for image classification.

4) Width multiplier α

The afore-mentioned parameter is specific only to the MobileNet family which allows efficient scaling of the models.

5) Precision

Only applicable to models on Jetson Nano, possible types are FLOAT32 and FLOAT16 (this board does not support INT8).

C. Benchmark Prerequisites

Each platform imposes different model preparation techniques as well as environmental requirements.

1) Google Coral

Each model has to be quantized, converted to a TFLite format, and compiled with the EdgeTPU compiler. The Coral USB Dongle was tested on Ubuntu 20.04.5 LTS with Intel Core i5-1135G7 2.4 GHz, 32 GB of RAM, a USB 3.0 port, and Edge TPU runtime for Linux that operates at the maximum clock frequency [31]. The Coral PCIe was tested on Radxa CM3 IO Board with Ubuntu 20.04.4 LTS, 4 GB of RAM, and analogous to PC, Edge TPU runtime. The coral benchmarking script is based on [39], which is also the reason for using timeit [40] package.

2) Inter Neural Stick

Models were not quantized, as Neural Stick does not support quantization, only converted to ONNX format with tf2onnx package [41]. The Intel Neural Stick was tested on Ubuntu 20.04.4 LTS with AMD Ryzen 5 1600, 128 GB of RAM, and a USB 3.0 port. Moreover, it needed OpenVINO-dev with additional libraries (like TensorFlow and ONNX) [42], an OpenVINO toolkit, [43] and set up variables with setupvars.sh from the prior downloaded toolkit. The Neural Stick benchmarking script is based on the hello-dassification.py script from the OpenVINO toolkit [43]. Additionally, models were compiled with the latency hint.

3) NVIDIA Jetson Nano

Each model is saved to Protocol Buffers format, and then, it is converted to TensorRT [44] architecture using TensorFlow's experimental converter. To make experiments quicker we compiled TensorRT models on RTX3090 and then built them on the target device.

D. Benchmarking Process

Inferences in our tests are synchronous and blocking, similar to the single-stream scenario described by Reddi et al. [19], i.e., the batch size is equal to 1, but our metric is the whole latency time. The benchmarking script is written in Python programming language, and inference time is measured with the timeit package. Models are analyzed in a grid search fashion over all possible parameters: input size, preprocessing, classification-head, and precision. The process runs as follows:

- model creation from TensorFlow's application module,
- outcome compilation to meet platform-specific requirements,
- an input image creation of size [*input size*, *input_size*, 3] with platform-specific data type,
- warm-up inferences with a compiled model on a selected platform (10 for every run), and
- proper inferences along with time measurement.

At the end of every grid search run, the script creates a CSV file with

- model parameters,
- minimum inference time,
- maximum inference time,
- mean inference time,
- the standard deviation of inference time, and
- median inference time for Jetson Nano (skewness of inference time distribution).

The default number of proper inferences is 1024. In special cases, it might vary, e.g., for the MobileNetV3 family on the Neural Stick it was 128 caused by the duration of the inference.

IV. RESULTS

Our experiment covered 3,095 unique test cases, focusing predominantly on MobileNet and EfficientNet model families. For visualizations and Table 1 results, we used models which included input data preprocessing, and without classification heads. Those are the results we wanted to focus on, since, in this work, we first analyzed architectures for being time-efficient feature extractors. Those picked results provide good insight into the rest of the collected data.

All the test results are available in a CSV format at https://bulletprove.com/research/edge_inference_results.csv.

Table 1. Inference time comparison for every model family representative per platform and input size

Platform	Model name	Input size (pixel)	Frames per second	Time (ms)			
				Mean	SD	Min	Max
Coral USB	**MobileNetV2**	**224**	**365.82**	**2.73**	**0.11**	**2.41**	**3.02**
Coral PCIe	MobileNetV2	224	289.01	3.46	0.24	2.55	4.69
Coral USB	**MobileNetV2**	**512**	**96.51**	**10.36**	**0.08**	**10.05**	**10.57**
Coral PCIe	MobileNetV2	512	85.33	11.72	0.41	10.56	24.30
Jetson FP16	MobileNetV2	224	82.75	12.08	1.91	5.20	17.58
Coral USB	**EfficientNetV2B0**	**224**	**75.91**	**13.17**	**0.41**	**11.77**	**14.09**
Coral PCIe	EfficientNetV2B0	224	68.39	14.62	0.15	12.88	15.03
Jetson FP16	EfficientNetV2B0	224	50.92	19.64	1.11	11.54	24.46
Neural Stick	MobileNetV2	224	40.31	24.81	0.31	24.06	25.44
Jetson FP16	**InceptionV3**	**224**	**36.35**	**27.51**	**4.49**	**6.75**	**47.51**
Coral USB	**VGG16**	**224**	**35.73**	**27.99**	**0.13**	**27.41**	**28.30**
Jetson FP16	**ResNet-50**	**224**	**35.13**	**28.47**	**4.98**	**4.47**	**46.09**
Neural Stick	InceptionV3	224	34.76	28.77	0.28	27.70	29.55
Coral PCIe	VGG16	224	31.31	31.94	0.16	31.04	34.92
Coral USB	InceptionV3	224	27.78	35.99	0.23	35.09	36.45
Coral USB	ResNet-50	224	26.96	37.09	0.18	36.07	37.48
Neural Stick	ResNet-50	224	26.03	38.42	0.27	37.59	39.14
Coral PCIe	InceptionV3	224	23.23	43.05	0.14	41.92	43.99
Coral PCIe	ResNet-50	224	22.71	44.04	0.14	43.09	44.68
Jetson FP16	MobileNetV2	512	21.91	45.64	8.35	6.82	84.63
Neural Stick	EfficientNetV2B0	224	20.20	49.50	0.38	48.18	50.26
Coral PCIe	**EfficientNetV2B0**	**512**	**16.26**	**61.50**	**0.21**	**58.74**	**62.63**
Coral PCIe	**InceptionV3**	**512**	**15.44**	**64.75**	**0.14**	**63.61**	**65.83**
Coral USB	InceptionV3	512	15.35	65.15	1.54	62.04	69.33
Jetson FP16	EfficientNetV2B0	512	14.43	69.30	5.50	41.00	95.82
Coral USB	**ResNet-50**	**512**	**14.13**	**70.75**	**0.22**	**69.59**	**71.09**
Coral USB	EfficientNetV2B0	512	13.56	73.76	2.53	70.05	83.77
Coral PCIe	ResNet-50	512	12.24	81.71	0.19	79.10	83.16
Neural Stick	VGG16	224	9.99	100.10	0.27	99.34	100.98
Neural Stick	MobileNetV2	512	9.35	106.94	0.48	105.60	108.00
Neural Stick	ResNet-50	512	5.82	171.78	0.66	169.68	175.44
Neural Stick	InceptionV3	512	5.68	176.08	0.62	173.58	177.87
Neural Stick	EfficientNetV2B0	512	5.62	178.09	0.46	176.53	179.37
Jetson FP16	VGG16	224	4.92	203.45	42.62	5.64	332.42
Neural Stick	**VGG16**	**512**	**1.87**	**533.86**	**0.71**	**531.54**	**536.14**
Coral PCIe	VGG16	512	-	-	-	-	-
Coral USB	VGG16	512	-	-	-	-	-
Jetson FP16	VGG16	512	-	-	-	-	-
Jetson FP16	InceptionV3	512	-	-	-	-	-
Jetson FP16	ResNet-50	512	-	-	-	-	-

Bolded rows show the best results for a certain model and input size, whereas those without measurements indicate an inability of running that setting.

A. Performance

Fig. 1 shows the platform's performance on MobileNetV2. Coral devices achieve the highest frames-per-second performance. NCS2 was the only device to successfully run models with all input sizes, which makes this platform the most versatile.

As shown in Fig. 2 the Coral is generally faster than NCS2 and Jetson Nano in this model group, especially in the case of smaller input size. Apart from NCS2, platforms had difficulties running models with larger input sizes.

Fig. 3 reflects the inference results on larger models. For models with input size 224 and outside the VGG family, the NCS2 is the fastest. Moreover, the device covers the broadest range of model configurations. With increasing model input size Coral slightly outperforms other platforms. Due to memory errors, only four inference test cases were completed successfully for Jetson Nano.

In addition, Figs. 1–3 shows the difference in the slope of curves, which indicates that inference time scales differently with the number of parameters across the compared platforms.

As shown in Table 1, for almost every unique architecture and input size setting Coral (either USB Dongle or PCIe) is the best platform. The only exception occurs for the InceptionV3 and ResNet-50 models; however, it is crucial to mention that these architectures do not fit entirely on the Google's platform. Nevertheless, the performance gap between the Google's platforms and the others deepens with an increase in the input size, which indicates better computation optimization for the prior platforms.

Results for Jetson Nano have, on average, 8.77 times bigger standard deviation of inference time than for other platforms. This value is based on data from Table 1 and only for MobileNetV2 and EfficientNetV2B0; however, this trend is correct for all measurements. It is a valuable insight for, e.g., designing a system with a strict maximum inference time constraint.

For MobileNetV2, architecture designed especially for mobile devices, we can observe a significant difference in FPS between Corals and other platforms. For an input size of 224 Coral USB was 4.42 times faster than Jetson and 9.08 than Neural Stick. In addition, Corals performed better for an input size of 512 than other devices for the

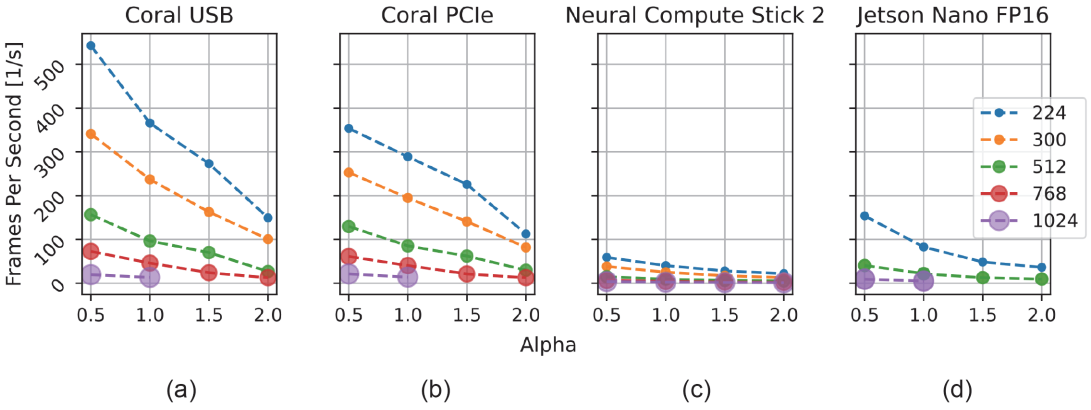


Fig. 1. Scaling up MobileNetV2 alpha parameter on four platforms: (a) Coral USB, (b) Coral PCIe, (c) Neural Compute Stick 2, and (d) Jetson Nano FP16. Missing data points are the result of compilation errors.

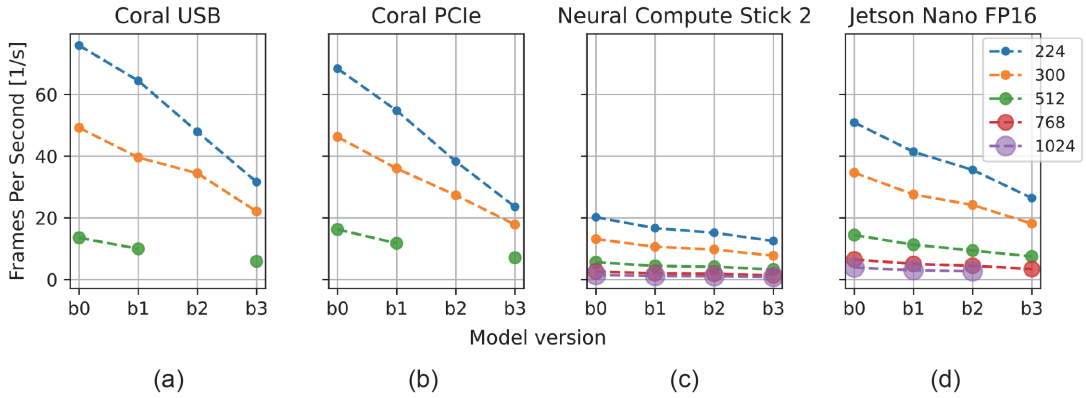


Fig. 2. Scaling EfficientNetV2 on four platforms: (a) Coral USB, (b) Coral PCIe, (c) Neural Compute Stick 2, and (d) Jetson Nano FP16. Missing data points on EfficientNetV2B2 are as the result of compilation errors.

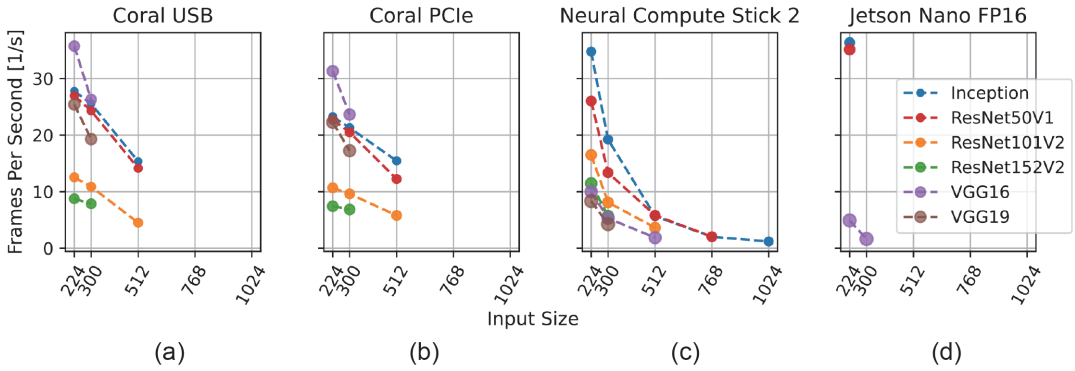


Fig. 3. Model input size versus frames versus platform on larger model families: (a) Coral USB, (b) Coral PCIe, (c) Neural Compute Stick 2, and (d) Jetson Nano FP16. Missing data points for specific networks are mainly the result of exceeding the platform's model size limits.

smaller one—1.16 times faster than Jetson Nano and 2.39 than Neural Stick.

B. Limitations

On Google Coral, when the model size exceeds on-chip memory size limits, the model's data has to be fetched from the external memory, which results in additional latency. Exceeding the unspecified model size limit on Google Coral results in a compilation failure.

Coral does not support the hard-swish activation function, which is required to compile MobileNetV3 directly in a way that allows full TPU computation. Therefore, some operations are executed off-chip, increasing model latency.

For large enough models, Neural Stick does not behave like Google's platform (using on-chip and off-chip memory), i.e., throws NC'-OUT-OFMEMORY error terminating running script. However, it is still able to work with more models than Coral.

In the case of Jetson Nano, it takes significantly longer to prepare an inference model compared to other platforms, hence for prototyping, it might be problematic. It turned out to be a bottleneck of our experiment. Similarly to Neural Stick, exceeding GPU's RAM results in an out-of-memory error. What is more, this was a common problem with NVIDIA's platform, especially for the bigger networks which is evident on all figures, especially Fig. 3.

V. CONCLUSION

This paper presents an extensive inference time performance comparison on Edge AI devices, specifically: NVIDIA Jetson Nano, Google Coral USB, Google Coral PCI, and Intel Neural Stick. For inference, we use variations of model families: MobileNet, EfficientNet, ResNet, VGG, and InceptionV3. Test configurations included mainly changes in the model's input size, classification head, type, and scale. The experiments' results indicate that Google Coral is the platform that offers the fastest average inference time. On the other hand, Jetson Nano inference tests suggest that the platform is prone to latency spikes, which is undesirable in time-restricted use cases. Moreover, the issues with handling larger models, especially older architectures like VGG and Inception, etc., portray Jetson troublesome in a variety of computer vision use cases where an engineer would like to analyze multiple algorithms. The NCS2 platform is the most universal considering the model type and model size choice in the scope of this experiment. We believe this benchmark will help engineers in developing AI at the edge.

ACKNOWLEDGEMENTS

This research was done as part of the "BulletProve: a system for supporting the training of long-distance shooters using machine learning methods" project and is jointly funded by the National Centre for Research and Development (Szybka Ścieżka 0552/21).

Conflict of Interest(COI)

The authors have declared that no competing interests exist.

REFERENCES

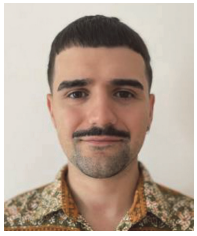
1. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, et al., "Language models are few-shot learners," 2020 [Online]. Available: https://arxiv.org/abs/2005.14165.
2. OpenAI, "Introducing ChatGPT," 2023 [Online]. Available: https://openai.com/blog/chatgpt/.
3. R. Rombach, A., Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022 [Online]. Available: https://arxiv.org/abs/2112.10752.
4. V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran,

and M. Grundmann, "Blazeface: sub-millisecond neural face detection on mobile GPUs," 2019 [Online]. Available: https://arxiv.org/abs/1907.05047.
5. Apple Inc., "An on-device deep neural network for face detection," 2017 [Online]. Available: https://machinelearning.apple.com/research/face-detection.
6. Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: keyword spotting on microcontrollers," 2017 [Online]. Available: https://arxiv.org/abs/1711.07128.
7. Z. Liu, H. Mao, C. Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," 2022 [Online]. Available: https://arxiv.org/abs/2201.03545.
8. Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: hierarchical vision transformer using shifted windows," 2021 [Online]. Available: https://arxiv.org/abs/2103.14030.
9. J. Redmon and A. Farhadi, "Yolov3: an incremental improvement," 2018 [Online]. Available: https://arxiv.org/abs/1804.02767.
10. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "SSD: single shot multibox detector," 2016 [Online]. Available: https://arxiv.org/abs/1512.02325.
11. K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," 2017 [Online]. Available: https://arxiv.org/abs/1703.06870.
12. W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900-6919, 2017.
13. Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: the communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322-2358, 2017.
14. OpenAI, "AI and compute," 2018 [Online]. Available: https://openai.com/research/ai-and-compute.
15. N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT security: an exhaustive survey on IoT vulnerabilities and a first empirical look on internet-scale IoT exploitations," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2702-2733, 2019.
16. L. Franceschi-Bicchierai, "How this Internet of Things stuffed animal can be remotely turned into a spy device," 2017 [Online]. Available: https://www.vice.com/en/article/qkm48b/how-this-internet-of-things-teddy-bear-can-be-remotely-turned-into-a-spy-device.
17. L. Franceschi-Bicchierai, "Internet of Things Teddy bear leaked 2 million parent and kids message recordings," 2017 [Online]. Available: https://www.vice.com/en/article/pgwean/internet-of-things-teddy-bear-leaked-2-million-parent-and-kids-message-recordings.
18. S. R. Reza, Y. Yan, X. Dong, and L. Qian, "Inference performance comparison of convolutional neural networks on edge devices," in *Science and Technologies for Smart Cities*. Cham, Switzerland: Springer International Publishing, 2020, pp. 323-335.
19. V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C. J. Wu, et al., "MLPerf inference benchmark," 2019 [Online]. Available: https://arxiv.org/abs/1911.02549.
20. TensorFlow [Online]. Available: https://www.tensorflow.org/.
21. 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," 2017 [Online]. Available: https://arxiv.org/abs/1704.04861.
22. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetv2: inverted residuals and linear bottlenecks," 2018 [Online]. Available: https://arxiv.org/abs/1801.04381.
23. A. Howard, M. Sandler, G. Chu, L. C. Chen, B. Chen, M. Tan, et al., "Searching for MobileNetv3," 2019 [Online]. Available: https://arxiv.org/abs/1905.02244.
24. M. Tan and Q. Le, "EfficientNet: rethinking model scaling for convolutional neural networks," 2019 [Online]. Available: http://arxiv.org/abs/1905.11946.
25. M. Tan and Q. Le, "EfficientNetv2: smaller models and faster training," 2021 [Online]. Available: https://arxiv.org/abs/2104.00298.
26. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015 [Online]. Available: https://arxiv.org/abs/1512.03385.
27. K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016 [Online]. Available: https://arxiv.org/abs/1603.05027.
28. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014 [Online]. Available: https://arxiv.org/abs/1409.1556.
29. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015 [Online]. Available: https://arxiv.org/abs/1512.00567.
30. S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: the confluence of edge computing and artificial intelligence," 2019 [Online]. Available: https://arxiv.org/abs/1909.00560v1.
31. Google, "Coral AI accelerator datasheet," 2020 [Online]. Available: https://coral.ai/docs.
32. Google, "Edge TPU," 2007 [Online]. Available: https://cloud.google.com/edge-tpu?hl=en.
33. NVIDIA, "Jetson modules," 2023 [Online]. Available: https://developer.nvidia.com/embedded/jetson-modules.
34. NVIDIA, "JetPack SDK," 2023 [Online]. Available: https://developer.nvidia.com/embedded/jetpack.
35. NVIDIA, "Embedded systems for product development," 2023 [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/product-development/.
36. Intel, "Neural Compute Stick 2 product brief," 2019 [Online]. Available: https://cdrdv2-public.intel.com/749742/neural-compute-stick2-product-brief.pdf.
37. J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "ImageNet: a large-scale hierarchical image database," in *Proceedings of 2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, 2009, pp. 248-255.
38. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, et al., "An image is worth 16x16 words: Transformers for image recognition at scale," 2020 [Online]. Available: https://arxiv.org/abs/2010.11929.
39. D. Kovalev, "Pycoral API repository," 2022 [Online]. Available: https://github.com/google-coral/pycoral.
40. Python, "timeit: measure execution time of small code snippets," 2023 [Online]. Available: https://docs.python.org/3/library/timeit.html.
41. J. Zhang, "tf2onnx repository," 2022 [Online]. Available: https://github.com/onnx/tensorflow-onnx.
42. Intel, "OpenVINO toolkit," 2022 [Online]. Available:

https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/download.html.
43. OpenVINO, "OpenVINO toolkit packages," 2022 [Online]. Available: https://storage.openvinotoolkit.org/repositories/openvino/packages/2022.2/linux.
44. NVIDIA, "TensorRT," 2023 [Online]. Available: https://developer.nvidia.com/tensorrt.

Rafał Tobiasz

Rafał Tobiasz received an M.Sc. degree in Automatic Control and Robotics from the AGH University of Science and Technology, Kraków in 2019. He joined BulletProve as AI/ML Engineer in February 2022. His research interests are in machine learning, neural networks, computer vision, and digital signal processing.



Grzegorz Wilczyński

Grzegorz Wilczyński received an M.Sc. degree in Automatics Control and Robotics from the AGH University of Science and Technology, Kraków in 2020. He joined BulletProve as an AI/ML Engineer in February 2022. His research interests are in machine learning, deep networks, generative algorithms, and computer vision.

Piotr Graszka

Piotr Graszka received his Ph.D. degree in computer science from the Warsaw University of Technology, Poland in 2016. In 2017, he joined Mobica as a Computer Vision Engineer working on machine learning related projects for such market leaders as Intel, Facebook and Google. In 2020, he joined Synergy Sports as a Deep Learning Researcher/Engineer working on computer vision and deep learning-based solutions in various sports production and analysis products and projects. Since 2021, Dr. Piotr Graszka also holds a position of the Head of AI Department at BulletProve. His research interests are in object and action detection and recognition in vision systems.

Nikodem Czechowski https://orcid.org/0000-0002-3830-6553

Nikodem Czechowski received Ph.D. in physics from Nicolaus Copernicus University in Torun, Poland in 2015. He joined the BulletProve company in March of 2019, where he started to focus on machine learning and computer vision applications. Currently Nikodem Czechowski is the acting CTO of the BulletProve company, where he supervises a team of AI researchers with a focus on edge AI and military applications.

Sebastian Luczak

Sebastian Luczak received a Bachelor of Science degree in computer science, automation and robotics, electronics and telecommunications from the University of Warsaw University of Technology, Poland in 2005. He finished postgraduate studies in area of IT management form University of Warsaw, Poland in 2010 and in area of CyberSecurity from Naval Academy in Gdynia, Poland 2018. His research interests are in defense system hardware and software. Since 2006 his professional career has been focused on NATO Projects.