



UDC 004.93'1, 519.684

OPTIMIZATIONS OF DEEP LEARNING OBJECTS DETECTION MODELS FOR INFERENCE ACCELERATION ON GENERAL-PURPOSE AND HARDWARE-ACCELERATED SINGLE-BOARD PLATFORMS

Dmytro Myroniuk^{ID}, **Bohdan Blagitko**^{ID}
Ivan Franko National University of Lviv,
107 Tarnavsky St., UA–79017 Lviv, Ukraine

Myroniuk, D. M., Blagitko, B. Ya. (2025). Deep Learning objects detection through model optimization to accelerate the process on general-purpose single-board platforms. *Electronics and Information Technologies*, 29, 57-68. <https://doi.org/10.30970/eli.29.6>

ABSTRACT

Background. Description and preparation of modern approaches for deep learning object detection models are provided. Deep learning frameworks for model training and inference, such as TensorFlow and TensorFlow Lite, are used as bases. The concepts of deep learning model optimization are analyzed.

Materials and Methods. The quantized int8 models are used as a baseline for optimization effectiveness estimation. The delegation approach includes software or hardware-optimized variants of neural operations. It prepared to speed up the inference process on target devices. The device with reduced performance resources or microcontroller without floating-point blocks uses a case of base-optimization model with int8 weights. The TensorFlow Lite framework has various quantization types outlined in a detailed explanation. Benchmarks for modern single-board devices are ready, and the correlation between using different optimization approaches, types of single-board platforms, and model inference speed analyses.

Results and Discussion. All tested models are pretrained using the MS COCO dataset (80 classes). All models were prepared for the experiment with 8-bit full integer quantization and output-TF Lite model generation using TensorFlow Object Detection API Docker images and Python 3.11. The testing data samples are obtained from the MS COCO validation dataset archive. The size of the image input is 640x640 RGB. The comparison of image recognition time to 640x640 RGB was conducted on Raspberry Pi 5, Raspberry Pi 4, and Jetson Nano 2GB. Only the Raspberry Pi 5 target device achieved real-time execution (100 ms at most or one fps) as it has more CPU performance than other devices.

Conclusion. Confirmation of the real-time execution approach was achieved by using reference models with reduced image sizes (320x320 RGB). TensorFlow standard model Zoo models, compiled with the TensorRT compiler, were used for the Jetson Nano target as an NPU-optimized case. Real-time execution (100 ms at most or one fps) is reaching for most models and target devices. Such an approach is suitable for less powerful devices with ARM Cortex-A processors.

Keywords: single-board computers, modeling, benchmarking, neural networks, object detection, optimization.

INTRODUCTION

The deep learning model inference of low-power devices has become main stream in several years. That kind of device class combines optimal performance. It is suitable for



© 2025 Dmytro Myroniuk & Bohdan Blagitko. Published by the Ivan Franko National University of Lviv on behalf of Електроніка та інформаційні технології / Electronics and information technologies. This is an Open Access article distributed under the terms of the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/) which permits unrestricted reuse, distribution, and reproduction in any medium, provided the original work is properly cited.

running deep-learning models with low power consumption. The model's optimization is one of the most dominant parts of modern deep learning topics. The basic trend of last year is using optimized models with low-cost and low-power MCUs and microcomputers. The based on such platforms solutions are more autonomous than systems based on the cloud solutions desktop. It has enough performance to make inferences in real time.

The single-board computer solutions analyzed for use with object detection deep learning algorithms in this article. These models have more computational complexity than image classification or localization tasks. That kind of model needs more powerful devices. The benchmarks for different object detection models are efficient. The correlations between general-purpose and hardware-accelerated platforms analyses, also. All types of platforms used: general-purpose (Raspberry Pi 4 and Raspberry Pi 5 [2]), platforms with internal accelerator (NVIDIA Jetson Nano 2 GB [3] with internal CUDA-based GPU with 128 cores), general-purpose platforms with external HW accelerator (Raspberry Pi 4, 5 with Google Coral Accelerator [4] as TPU) as target platforms.

LITERATURE REVIEW

Model benchmarking.

Model benchmarking is the most essential task for model testing on target devices. Several organizations are preparing benchmarks for the target device's fundamental performance estimation. The most authoritative organization for such a task is MLPerf [5], which has ready base submission standards for HW vendors. In addition, deep learning software vendors prepare base benchmarks for different devices, for example, Tensorflow Object Detection API models archive [6] and Ultralytics YOLOv5-v8 [7]. However, such vendors do not cover all devices, especially modern single-board target devices with external accelerations.

Deep Learning models for object detection task deployment in real-time systems.

Real-time object detection is one of the most popular tasks in the modern object detection sphere. Here are several approaches to real-time object detection with different configurations.

Article [8] presented benchmarking for different types of optimizations for Raspberry Pi devices. Authors have prepared several models with applied optimization techniques. However, this benchmark does not cover the newest models of Raspberry Pi devices (Raspberry Pi 5) with applied hardware accelerations. This research also does not cover modern object detection models, moreover. This is used widely in the target topics.

Real-time applications have become a prevalent task in modern information technologies; the Articles [9] and [11] present applying deep learning object detection models for real-world applied tasks: peach fruit detection and weed control application. In the first case, the authors used a configuration based on a single-board computer Raspberry Pi 4 with 8 GB of RAM and Google Coral accelerator as an external accelerator. In the other case, the authors used the Jetson Nano module and TensorRT inference engine.

The authors described examples of benchmarking YOLOv3 models on edge mobile devices based on Cortex A-series CPUs [10]. The research was conducted on a Samsung Galaxy Note 8. It is also applicable to Cortex A-series processors.

MATERIALS AND METHODS

Used target devices

All types of single-board computers were used in research as target devices. It can be used for practical cases in various categories. It combines good performance with small power consumption of the devices. Configurations of that kind determined some challenges

and limitations in the deep learning inference on target devices. Single-board computers with different hardware configurations are used for deep learning model inference to improve performance in this article:

General purpose devices class – Raspberry Pi 4 and 5 were used for this case; Computers of that kind have become a standard of general-purpose single-board computers in recent years. All devices were configured for mobile deep learning using appropriate frameworks (TensorFlow Lite in our case). In this research, different configurations of Raspberry Pi model B are used:

1) Raspberry Pi 4 model B, 4 GB. Table 1 gives Details of the OS and target configurations.

A general-purpose device with the external accelerator class. Some single-board platforms can accelerate deep learning model inference via internal GPUs and special ARM delegates, with CMSIS-NN library as an SW acceleration. Several external accelerators for general single-board devices can be used to increase overall performance for deep learning model inference. Google Coral Accelerator is used here as a member of such devices. Details about the HW configuration of the Google Coral accelerator are given in Table 3.

2) Raspberry Pi 5 model B 4 GB. Table 2 gives Details of the OS and target configurations.

Table 1. Raspberry Pi 4 model B HW details

Parameter	Value
Model	Raspberry Pi 4 Model B 4 GB
CPU	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
RAM	4GB LPDDR4-3200 SDRAM
OS	Raspberry Pi OS (Bullseye)

Table 2. Raspberry Pi 5 model B HW details

Parameter	Value
Model	Raspberry Pi 5 Model B 4 GB
CPU	Quad-core 64-bit Arm Cortex-A-76 @ 2.4 GHz
RAM	LPDDR4X-4267 SDRAM 4GB
OS	Raspberry Pi OS (Bullseye)

Table 3. Google Coral Accelerator HW details

Parameter	Value
Model	Google Coral Accelerator
Computational performance	int8:4 TOPS
Supported types	Quantized integer (8-bit)
Power consumption	2 W
Used SW	Google Edge TPU library

Specialized single-board computers with internal accelerators. Such a class of devices were designed for accelerated computations, such as deep learning models running, parallel computations, machine learning, etc. Here is an example of such devices:

- Nvidia Jetson Nano/Orin/Xavier;
- Google Coral Dev Board;
- Hailo AI (as an extension of Raspberry Pi 5 via M2 HAT+ interface).

As a main difference between such devices, we can specialize in internal integration with board interfaces and using high-speed internal bus for connection. Also, some devices have extended support of quantized models: float16 or int16x8 quantization types are supported for NVidia devices, so in some particular cases, we can achieve better results for quantized models. NVidia Jetson Nano 2 GB development board is used as a member of such a category. Details of OS and target configurations are given in Table 4.

Table 4. Jetson Nano 2 GB HW details

Parameter	Value
Model	NVidia Jetson Nano 2 GB
CPU	Quad-core Cortex-A57 64-bit SoC @ 1.43GHz
RAM	2GB LPDDR4 1600 MHz SDRAM
GPU	Nvidia Maxwell architecture with 128 NVidia CUDA Cores, 921 MHz
OS	Ubuntu Tegra OS, based on Ubuntu 18.04

Used software

Deep Learning frameworks

TensorFlow and TensorFlow Lite are a base deep learning framework for model training and inference. This framework has vast functionality for model training, inference, and converting to smaller data types. The TensorFlow Lite sub-framework has its model interpreter, which can support various deep learning models. TensorFlow Lite also supports inference using CPU optimizations, such as XNNPACK and ARM-NN delegate.

As some of the used models for testing use PyTorch – TFLite model conversion as a middle format. The ONNX format is also one of the most used model formats in modern deep learning. It also supports various optimizations, such as delegating using ARM software runtime optimizations (based on CMSIS-NN library) and XNNPACK delegate. Also, the ONNX format supports parallelization between CPU cores to improve inference speed for multi-core processors.

The NVidia computers use TensorRT, an optimized framework and inference engine. The ONNX and a special TensorRT wrapper convert the functionality of the base models to TensorRT format.

Used optimizations

Several types of optimizations are used in this research. The quantized int8 models are used as a baseline for optimization effectiveness estimation. The following model optimizations are namely:

- TensorRT model optimization and inference on its engine;
- ARM NN delegate for CPU operation optimizations;
- PyCoral tools for Google Coral TPU optimization.

Software delegation of several cases used for the base software optimization as default int8x8 models. The delegation approach includes software or hardware-optimized variants of neural operations. It prepared to speed up the inference process on target devices.

All used single-board computers are based on ARM Cortex A-series processors with ARM architectures. It is used for the base software optimization ARM ACL library.

The ARM NN is for Cortex A-series processors by a special inference engine and Mali GPUs maintained by ARM. It provides one of the most optimized software implementations for NN operations. The ARM NN can be included in the TFLite inference engine via a delegation approach for Python or used directly via C++ and CMake.

The second popular software optimization used in this research is XNNPACK. XNNPACK is a highly optimized solution for various CPU architectures, such as x86, ARM, RISC-V, etc. Google developed it. XNNPACK provides low-level software implementations for neural operations. It can be included in the TFLite interpreter as a delegate.

The research uses the PyCoral library as a hardware-optimized delegate. PyCoral is to run the Google Coral accelerator as a tool for optimizing neural operations. It includes several tools for model preparation and a dedicated TFLite delegate. It is used for Google Coral devices as the inference engine.

Used quantization configurations

Models for mobile inference use optimized models in addition to standard models. Most modern acceleration systems for mobile platforms use model quantization, which converts trained float32 models to data types with reduced memory usage representation. Devices with reduced performance resources or microcontrollers without floating-point blocks can use that model. This work uses a case of base-optimization model with int8 weights. Such models will have reduced computational complexity but can also have reduced accuracy parameters. Table 5 describes a detailed explanation of different quantization types for the TensorFlow Lite framework [12].

Table 5. Quantization types benefits

#	Technique	Benefits	Hardware
1	Dynamic range quantization	4x smaller, 2x-3x speedup	CPU
2	Full integer quantization	4x smaller, 3x+ speedup	CPU, Edge TPU, Microcontrollers
3	Float16 quantization	2x smaller, GPU acceleration	CPU, GPU

Used models

The models for this research use popular mobile object detection models. Such models can be deployed to smaller devices as they use lighter architectures than classic object detection models, which are appointed to state GPUs with large numbers of shader cores. So, the most important criterion for such models is less computational complexity. It will make them more suitable for ARM-based processors of target boards.

As the baseline models for this research, models with input size 640x640 RGB are used. Such an input size is the most popular for modern object detection models. It can save most of the valuable features of recognized objects and provide better accuracy than models with smaller sizes.

On the other hand, models with input size 320x320 RGB are used to provide a segmental recognition approach. Such a class of models still can provide suitable accuracy for large objects. It is less accurate for smaller objects as such size cannot provide enough key features to make a confident forecast.

All tested models were pre-trained using the MS COCO dataset [13] (80 classes) and tested using the validation dataset of the MS COCO 2017 competition. Table 6 provides a list of all researched models with appropriate sources.

RESULTS AND DISCUSSION

Baseline estimation

Inference on target devices with full input size models prepared as the baseline experiment. All used models were ready for the experiment with 8-bit full-integer quantization and output .tflite models generation using TensorFlow Object Detection API Docker images and Python 3.11 as the basic programming language on the reference x86 device. Table 7 presents the testing results for this experiment.

Experiment parameters: TF/TFLite, standard TFLite 8-bit full integer quantization, and Python 3.11.9. The testing data samples are obtained from the MS COCO validation dataset archive. The size input image is 640x640 RGB.

Table 6. Used models

Architecture	Source	Input size	Parameters number
SSD-MobileNetV1	TF OD API [6]	640x640	31,200,598
SSD-MobileNetV2	TF OD API [6]	640x640	2,870,772
EfficientDet_Lite3x	TF OD API [6]	640x640	9,872,762
YOLOv5s	Ultralytics YOLOv5 [7]	640x640	7,026,307
SSD-MobileNetV1	TF OD API [6]	320x320	6,861,134
SSD-MobileNetV2	TF OD API [6]	320x320	2,741,320
EfficientDet_Lite0	TF OD API [6]	320x320	3,366,672
YOLOv5n	Ultralytics YOLOv5 [7]	320x320	1,767,283

Table 7. The speed estimation inference for models with input size 640x640 RGB on target devices

Model name	Model optimization	\bar{x} , ms	Σ , ms
Raspberry Pi 4 Input size 640x640			
Basic models			
tf2_ssd_mobilenet_v1_fpn_640x640_coco17_ptq [6]	Basic Tflite model	2467.0	109.0
ssd_mobilenet_v2_fpn_lite_640x640_coco17 [6]	Basic Tflite model	244.5	6.0
efficientdet_lite3x_640_ptq [6]	Basic Tflite model	743.5	156.5
yolov5s [7]	Basic Tflite model	376.0	6.0
SW-optimized			
tf2_ssd_mobilenet_v1_fpn_640x640_coco17_ptq [6]	ARM NN Delegate	2117.4	42.6
ssd_mobilenet_v2_fpn_lite_640x640_coco17 [6]	ARM NN Delegate	597.0	10.4
efficientdet_lite3x_640_ptq [6]	ARM NN Delegate	1046.0	28.0
yolov5s [7]	ARM NN Delegate	427.3	11.6
HW accelerated			
tf2_ssd_mobilenet_v1_fpn_640x640_coco17_ptq [6]	Coral	430.94	13.6
ssd_mobilenet_v2_fpn_lite_640x640_coco17 [6]	Coral	372.57	5.93
efficientdet_lite3x_640_ptq [6]	Coral	–	–
yolov5s [7]	Coral	139.87	2.89

Table 7. Continuation.

Raspberry Pi 5 Input size 640x640			
Basic models			
tf2_ssd_mobilenet_v1_fpn_640x640_coco17_ptq [6]	TfLite	285.00	4.92
ssd_mobilenet_v2_fpnlite_640x640_coco17 [6]	TfLite	79.03	3.10
efficientdet_lite3x_640_ptq [6]	TfLite	188.83	5.76
yolov5s [7]	TFLite	67.93	3.75
SW-optimized			
tf2_ssd_mobilenet_v1_fpn_640x640_coco17_ptq [6]	ARM NN Delegate	355.26	12.90
ssd_mobilenet_v2_fpnlite_640x640_coco17 [6]	ARM NN Delegate	193.99	6.34
efficientdet_lite3x_640_ptq [6]	ARM NN Delegate	304.65	13.69
yolov5s [7]	ARM NN Delegate	74.70	3.85
HW accelerated			
tf2_ssd_mobilenet_v1_fpn_640x640_coco17_ptq [6]	Coral	359.94	5.90
ssd_mobilenet_v2_fpnlite_640x640_coco17 [6]	Coral	130.07	2.90
efficientdet_lite3x_640_ptq [6]	Coral	-	-
yolov5s [7]	Coral	139.51	2.10
Nvidia Jetson Nano 2GB Input size 640x640			
Basic models			
tf2_ssd_mobilenet_v1_fpn_640x640_coco17_ptq [6]	TfLite	2133.95	42.14
ssd_mobilenet_v2_fpnlite_640x640_coco17 [6]	TfLite	238.39	4.66
efficientdet_lite3x_640_ptq [6]	TfLite	704.94	8.20
yolov5s [7]	TFLite	343.52	4.94
SW-optimized			
tf2_ssd_mobilenet_v1_fpn_640x640_coco17_ptq [6]	ARM NN Delegate	2107.91	37.87
ssd_mobilenet_v2_fpnlite_640x640_coco17 [6]	ARM NN Delegate	524.02	7.06
efficientdet_lite3x_640_ptq [6]	ARM NN Delegate	1044.74	19.04
yolov5s [7]	ARM NN Delegate	379.24	8.84
HW accelerated			
tf2_ssd_mobilenet_v1_fpn_640x640_coco17_ptq [6]	TensorRT	732.59	50.00
ssd_mobilenet_v2_fpnlite_640x640_coco17 [6]	TensorRT	141.88	16.59
efficientdet_lite3x_640_ptq [6]	TensorRT	-	-
yolov5s [7]	TensorRT	109.23	4.82

A graphic representation of the received results is presented in Fig. 1. As represented in Table 7 and Figures 1(a), (b), (c), and using an academic representation of real-time execution (100 ms at most or one fps), real-time execution is reached only for Raspberry Pi 5 target device, which has more CPU performance in comparison to other devices. Real-time execution is not attained for most models with the baseline list. Namely, such an approach is used for less powerful devices with ARM Cortex-A processors. So, base models of large input sizes are unsuitable for such devices.

Samples optimization approach for real-time object detection on single-board computers

Partial recognition is an approach that can be applied to solve problems with long execution times for devices with smaller computational resources. The main idea of such a method is to use several models with smaller input sizes to reduce the computational complexity of the overall task.

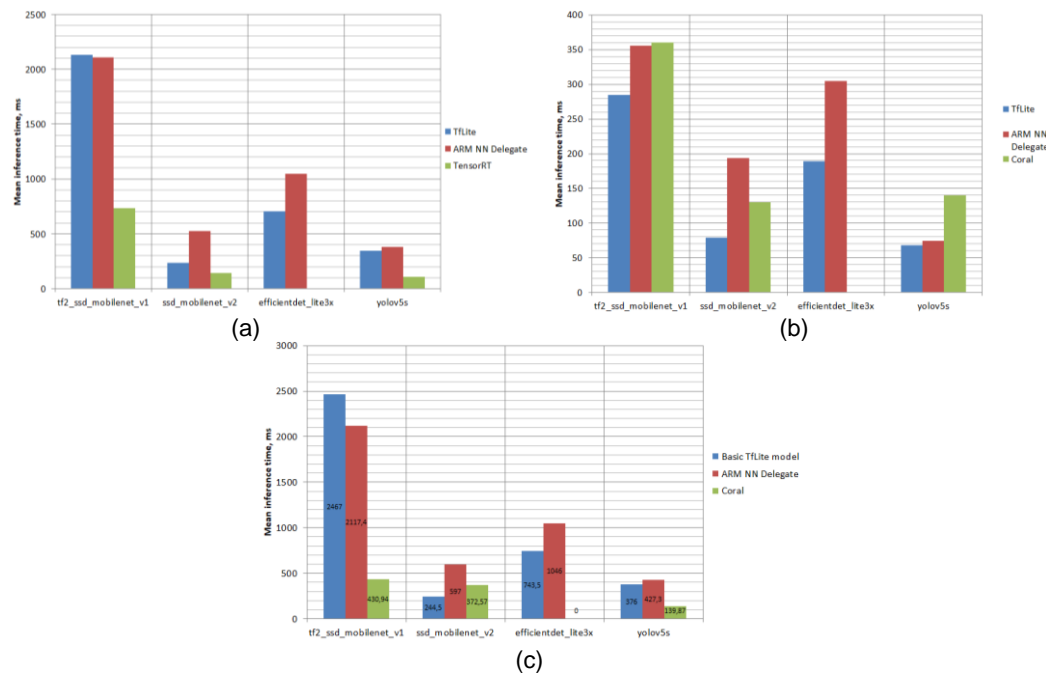


Fig. 1. The time comparison inference for models with input size 640x640 for target devices: Raspberry Pi 5 (a), Raspberry Pi 4 (b), Jetson Nano 2 GB (c).

Reference models with reduced (320x320 RGB images (Fig. 2) are used to confirm the real-time execution approach. The popular object detection model set is used for such an approach. All models were part of the standard TensorFlow Zoo repository and Google Coral model Zoo for external NPU-optimized models. TensorFlow standard model Zoo models, compiled with the TensorRT compiler, are used for the Jetson Nano target as an NPU-optimized case. Table 8 and Figure 3 (a), (b), and (c) present the inference results for 320x320 input size models.

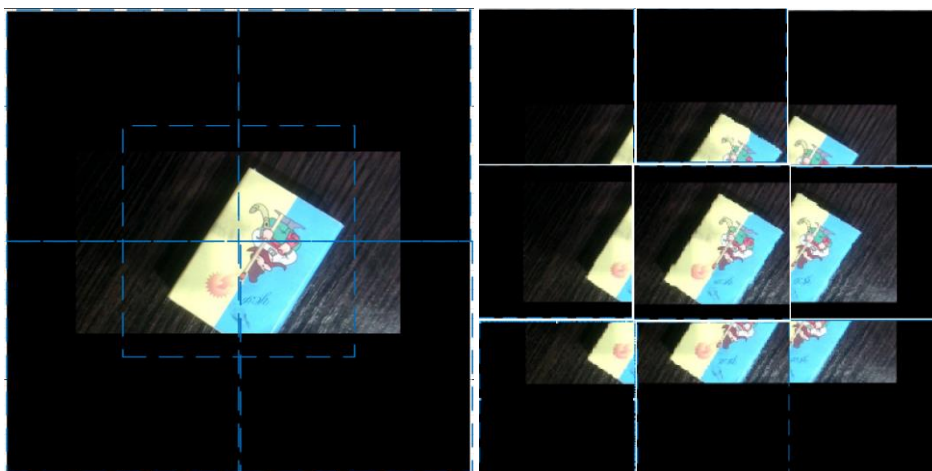


Fig. 2. Visual representation of image portioning.

Table 8. Models speed inference estimation with input size 320x320 RGB on target devices.

Model name	Model optimization	\bar{x} , ms	σ , ms
Raspberry Pi 4 Input size 320x320			
Basic models			
mobilenet_v1 [14]	TFLite	70.40	2.85
mobilenet_v2[14]	TfLite	40.69	0.79
efficientdet_lite0 [14]	TfLite	79.82	2.40
yolov5n [7]	TfLite	29.90	1.30
yolov5s [7]	TfLite	90.75	1.54
SW-optimized			
mobilenet_v1 [14]	ARM NN Delegate	68.88	7.19
mobilenet_v2 [14]	ARM NN Delegate	61.10	6.58
efficientdet_lite0 [14]	ARM NN Delegate	147.74	6.92
yolov5n[7]	ARM NN Delegate	47.42	2.77
yolov5s [7]	ARM NN Delegate	109.40	7.19
HW accelerated			
mobilenet_v1 [14]	Coral	10.37	1.88
mobilenet_v2[14]	Coral	15.56	1.94
efficientdet_lite0 [14]	Coral	89.68	4.96
yolov5n [7]	Coral	10.84	0.52
yolov5s [7]	Coral	18.86	1.86
Raspberry Pi 5 Input size 320x320			
Basic models			
mobilenet_v1 [14]	TFLite	10.50	0.31
mobilenet_v2[14]	TfLite	9.29	1.17
efficientdet_lite0 [14]	TfLite	22.76	2.15
yolov5n [7]	TfLite	6.30	0.56
yolov5s [7]	TfLite	14.71	0.83
SW-optimized			
mobilenet_v1 [14]	ARM NN Delegate	11.27	3.45
mobilenet_v2[14]	ARM NN Delegate	13.11	2.75
efficientdet_lite0 [14]	ARM NN Delegate	47.83	4.93
yolov5n [7]	ARM NN Delegate	12.81	1.68
yolov5s [7]	ARM NN Delegate	20.24	2.19
HW accelerated			
mobilenet_v1 [14]	Coral	9.94	2.19
mobilenet_v2[14]	Coral	13.19	2.05
efficientdet_lite0 [14]	Coral	56.81	2.24
yolov5n [7]	Coral	10.81	0.63
yolov5s [7]	Coral	18.89	2.19
Nvidia Jetson Nano 2GB Input size 640x640			
Basic models			
mobilenet_v1 [14]	TFLite	71.14	0.94
mobilenet_v2[14]	TfLite	39.60	0.69
efficientdet_lite0 [14]	TfLite	77.61	2.25
yolov5n [7]	TfLite	31.43	0.38
yolov5s [7]	TfLite	88.35	1.15
SW-optimized			
mobilenet_v1 [14]	ARM NN Delegate	98.43	15.63
mobilenet_v2[14]	ARM NN Delegate	63.00	6.62

Table 8. Continuation.

efficientdet_lite0 [14]	ARM NN Delegate	155.65	5.69
yolov5n [7]	ARM NN Delegate	48.27	3.23
yolov5s [7]	ARM NN Delegate	110.20	11.30
HW accelerated			
mobilenet_v1 [14]	TensorRT	61.24	3.92
mobilenet_v2[14]	TensorRT	41.37	3.92
efficientdet_lite0 [14]	TensorRT	–	–
yolov5n [7]	TensorRT	22.67	5.55
yolov5s [7]	TensorRT	32.59	4.68

A graphic representation of the received results is presented in Figure 3. Real-time execution is reaching for most models and target devices. It is represented in Table 8 and Figures 3(a), 3(b), and 3(c), using an academic representation of real-time execution (100 ms at most or one fps). Such an approach is suitable for less powerful devices with ARM Cortex-A processors.

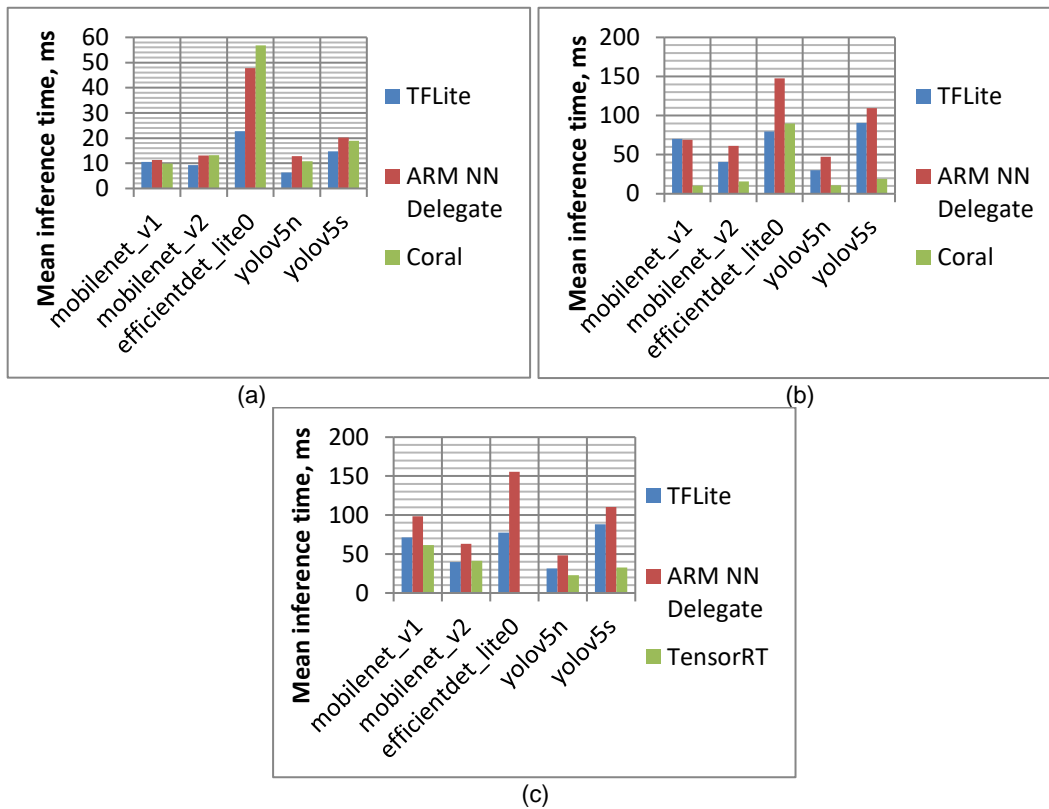


Fig. 3. Inference time comparison for models with input size 320x320 for target devices: Raspberry Pi 4 (a), Raspberry Pi 5 (b), Jetson Nano 2 GB (c).

CONCLUSION

As presented in tables 7 and 8 and diagrams 1 and 2, inferencing large accurate models with good accuracy is not reaching real-time execution on single-board computers based on Cortex A-series processors without optimizations. The popular object detection

models with image sizes 640x640 are mostly not suitable for such target devices but can provide more accurate results on most sizes of objects.

To resolve this problem, partial recognition with pre-processing can be applied, using models with less image size. Such type of models have significantly less computational complexity, but can provide good accuracy for large and medium object sizes. Moreover, preparation algorithms for this approach can be used in other parts of object detection and identification process. It can be helpful in complex systems with multi-stage pipelines for object detection and identification.

Such an approach was tested on partial object detection with an image size 320x320 for an original image size 640x640. Less complex models have provided significant inference time reduction. It can give real-time execution for most researched models and target devices, even by recognizing several interesting zones.

AUTHOR CONTRIBUTIONS

Conceptualization, [D.M.]; methodology, [D.M.]; validation, [B.B.]; formal analysis, [D.M.]; investigation, [D.M.]; resources, [D.M.]; data curation, [D.M.]; writing – original draft preparation, [D.M.]; writing – review and editing, [B.B.]; visualization, [D.M.] supervision, [B.B.]; project administration, [D.M.]; funding acquisition, [D.M.].

All authors have read and agreed to the published version of the manuscript.

REFERENCES

- [1] Ltd, R. P. (Trading). (2023). Buy a Raspberry Pi 4 Model B. Raspberry Pi. <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>
- [2] Raspberry Pi Ltd. (n.d.). Buy a Raspberry Pi 5. Raspberry Pi. <https://www.raspberrypi.com/products/raspberry-pi-5/>
- [3] NVIDIA Jetson Nano. (2024). NVIDIA. <https://www.nvidia.com/ru-ru/autonomous-machines/embedded-systems/jetson-nano/>
- [4] Dev Board. (n.d.). Coral. <https://coral.ai/products/dev-board/>
- [5] MLPerf Inference - MLCommons. (2024, November 18). MLCommons. <https://mlcommons.org/working-groups/benchmarks/inference/>
- [6] Tensorflow. (2019, July 15). Tensorflow/models. GitHub. https://github.com/tensorflow/models/tree/master/research/object_detection
- [7] Jocher, G. (2020, August 21). ultralytics/yolov5. GitHub. <https://github.com/ultralytics/yolov5>
- [8] Ameen, S., Siriwardana, K., & Theodoridis, T. (2023). Optimizing Deep Learning Models For Raspberry Pi. ArXiv (Cornell University). <https://doi.org/10.48550/arxiv.2304.13039>
- [9] Assunção, E., Pedro Dinis Gaspar, Khadijeh Alibabaei, Maria Paula Simões, Proença, H., Vasco, & Caldeira, P. (2022). Real-Time Image Detection for Edge Devices: A Peach Fruit Detection Application. Future Internet, 14(11), 323–323. <https://doi.org/10.3390/fi14110323>
- [10] Gabriele Proietti Mattia, & Beraldi, R. (2021). A study on real-time image processing applications with edge computing support for mobile devices. IRIS Research Product Catalog (Sapienza University of Rome). <https://doi.org/10.1109/ds-rt52167.2021.9576139>
- [11] Eduardo Timóteo Assunção, Pedro Dinis Gaspar, Ricardo Alves Mesquita, Maria João Simões, Khadijeh Alibabaei, André Veiros, & Proença, H. (2022). Real-Time Weed Control Application Using a Jetson Nano Edge Device and a Spray Mechanism. 14(17), 4217–4217. <https://doi.org/10.3390/rs14174217>
- [12] Post-training quantization | TensorFlow Lite. (n.d.). TensorFlow. <https://www.tensorflow.org/lite/performance>

- [13] COCO - Common Objects in Context. (n.d.). Cocodataset.org.
<https://cocodataset.org/>
- [14] Models - Object Detection | Coral. (2020). Coral; Coral.
<https://coral.ai/models/object-detection/>

ОПТИМІЗАЦІЇ МОДЕЛЕЙ ДЕТЕКТУВАННЯ ОБ'ЄКТІВ ДЛЯ ПРИСКОРЕННЯ ЇХ ВИКОНАННЯ НА ОДНОПЛАТНИХ ПЛАТФОРМАХ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ ТА ПЛАТФОРМАХ З АПАРАТНИМ ПРИСКОРЕННЯМ

Дмитро Миронюк, Богдан Благітко

*Львівський національний університет імені Івана Франка
 вул. Ген. Тарнавського, 107, Львів, Україна*

АНОТАЦІЯ

Вступ. Надано опис та підготовку сучасних підходів глибинного навчання до моделей виявлення об'єктів. В якості основи використовуються фреймворки глибинного навчання для навчання моделей і висновків, такі як TensorFlow і TensorFlow Lite. Проаналізовано концепції оптимізації моделі глибинного навчання. Квантовані моделі int8 використовуються як базові для оцінки ефективності оптимізації. Підхід делегування включає оптимізовані програмні або апаратні варіанти нейронних операцій. Він підготований для прискорення процесу виявлення об'єктів на цільових пристроях.

Матеріали та методи. Пристрій зі зниженими ресурсами продуктивності або мікроконтролер без блоків з плаваючою комою використовує модель базової оптимізації з вагами int8. Фреймворк TensorFlow Lite має різні типи квантування, викладені в детальному поясненні. Підготовані бенчмарки для сучасних одноплатних пристроїв, а також здійснені кореляції між використанням різних підходів до оптимізації, типів одноплатних платформ і аналіз швидкості виявлення об'єктів. Усі протестовані моделі попередньо навчені з використанням набору даних MS COCO (80 класів). Усі моделі були підготовлені для експерименту з 8-бітним повним цілочисельним квантуванням і генерацією вихідної моделі TFLite з використанням зображень TensorFlow Object Detection API Docker і Python 3.11.

Результати. Зразки даних тестування отримано з архіву набору даних перевірки MS COCO. Розмір вхідного зображення 640x640 RGB. Порівняння часу розпізнавання зображення з 640x640 RGB проводилося на Raspberry Pi 5, Raspberry Pi 4 і Jetson Nano 2GB. Лише цільовий пристрій Raspberry Pi 5 досяг виконання в реальному часі (не більше 100 мс або один кадр/с), оскільки він має більшу продуктивність ЦП, ніж інші пристрої. Підтвердження виконання в режимі реального часу було досягнуто за допомогою еталонних моделей зі зменшеними розмірами зображення (320x320 RGB).

Висновки. Стандартні моделі TensorFlow. Моделі Zoo, скомпільовані за допомогою компілятора TensorRT, використовувалися для Jetson Nano як варіант, оптимізований для NPU. Виконання в режимі реального часу (не більше 100 мс або один кадр в секунду) досягається на більшості моделей і цільових пристроїв. Такий підхід доцільно використовувати на менш потужних пристроях з процесорами ARM Cortex-A.

Ключові слова: одноплатні комп'ютери, моделювання, контрольна таблиця, нейронні мережі, детектування об'єктів, оптимізація.