IMAGE RECOGNITION IN REAL-TIME FOR GENERAL-PURPOSE SINGLE-BOARD PLATFORMS

Myronyuk Dmytro, Blagitko Bohdan
Department of Radiophysics and Computer Technologies, Ivan Franko Lviv's National University, Lviv, Ukraine
myronyukdmytro@gmail.com, blagitko@gmail.com

I. INTRODUCTION

Modern systems based on deep learning concepts and neural networks of different types can solve a wide range of difficult generalizing tasks, especially in image processing [5]. General-purpose devices are also used for their tasks [3]. They demonstrate excellent results in different class tasks. It's from image classification and object detection to image generation and real-time recognition.

The deep learning model inference of low-power devices has become mainstream over the past several years. That kind of device class combines optimal performance. It is suitable for running deep-learning models with low power consumption. Model 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 solutions based on such platforms are more autonomous than systems based on cloud solutions and desktops. It has enough performance to make inferences in real-time.

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 [1], which has a ready base of 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 [7] and Ultralytics YOLOv5-v8 [6]. However, such vendors do not cover all devices, especially modern single-board target devices with external accelerations.

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 [2] presented benchmarking of optimizations of various types of Raspberry Pi devices. The 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. It is used widely in the target topics.

Real-time applications have become a common task in modern information technology. For example, articles present the application of deep learning object detection models to solve real-world applications: detection of ripe peach fruits [3] and applications for weed control [6]. The authors used a configuration based on a single-board computer for the first case. They have described an example benchmarking YOLOv3 models on edge mobile devices based on Cortex A-series CPUs [7]. The research was conducted for a Samsung Galaxy Note 8. It is also applicable to Cortex A-series processors.

Modern building-optimized network methodologies are analyzed in this article.

A prototype of the object classifier has been realized.

II. MODEL OPTIMIZATIONS

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.

Some of the models used for testing use PyTorch as a base framework to retrain the model, and the ONNX format uses 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 the CMSIS-NN library) and XNNPACK delegate. Also, the ONNX format supports parallelization between CPU cores to improve inference speed for multi-core processors. The computer's NVIDIA uses TensorRT as the optimized framework and inference engine. The ONNX and a special TensorRT wrapper convert the functionality of the base models to the TensorRT format.

Several optimization types are used in this research. The quantized int8 models are used as a baseline for the effectiveness of optimization estimation. The following model optimizations are: TensorRT model optimization and inference on its engine; ARM NN delegate for CPU operation optimizations; and PyCoral tools for Google Coral TPU optimization.

Software delegation of several cases is used for the base software optimization as default int8x8 models. The delegation approach includes software or hardware-optimized variants of neural operations. It is prepared to speed up the inference process on target devices. All used single-board computers are based on ARM Cortex-A A-series processors with ARM architectures. It is used for the ARM ACL base software optimization library. The ARM NN is a special inference engine for Cortex-A A-series processors. The Mali GPUs remained 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 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 as the inference engine for Google Coral devices.

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 a base-optimization model with int8 weights. Such models will have reduced computational complexity but can also have reduced accuracy parameters.

The models for this research use popular mobile object detection models. Such models can be deployed for smaller devices as they use lighter architectures than classic object detection models. They are designed to run on 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 320x320 RGB are used. Such an input size is the most popular for modern object detection models. On the other hand, models with input size 320x320 RGB are used to provide a segmental recognition approach. Such a class of models can still offer suitable accuracy for large objects. It is less accurate for smaller objects as such a size cannot provide enough key features to make a confident forecast.

All tested models were pre-trained using the MS COCO dataset (80 classes) and tested using the validation dataset of the MS COCO 2017 competition.

Table 1 provides a list of all researched models with appropriate sources.

TABLE 1. RESEARCHED MODELS WITH APPROPRIATE SOURCES

#	***Architecture***	***Source***
1	SSD-MobilenetV1	TF OD API [7]
2	SSD-MobileNetV2	TF OD API [7]
3	EfficientDet_Lite3x	TF OD API [7]
4	YOLOv5s	Ultralytics YOLOv5 [6]

III. COMPARISON MODELS TIME INFERENCES

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 for reference on x86 devices. Camera Module PCB dimensions ver.1.3 used as the camera of the prototype.

The model time inference comparison is provided for target devices with input size 320x320: Raspberry Pi 5, Raspberry Pi 4 (b), and Jetson Nano 2 GB. They are involved in Figures 1, 2, and 3.

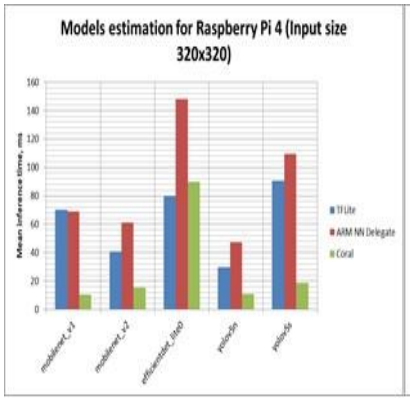

Fig. 1. The inference time comparison. in target device Raspberry Pi 4.

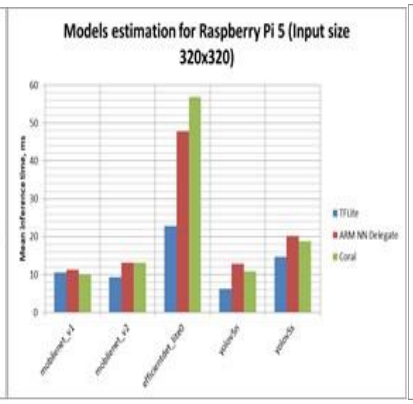

Fig. 2. The inference time comparison. in target device Raspberry Pi 5.

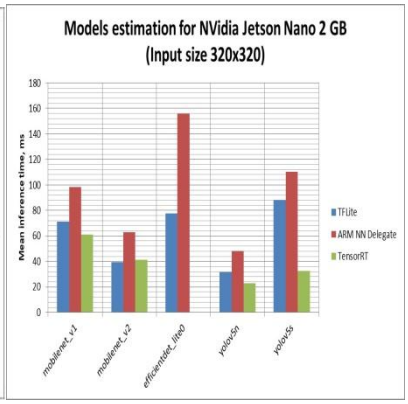

Fig. 3. The inference time comparison in target device Nvidia Jetson Nano 2 GB.

IV. CONCLUSION

The manipulator detects objects of irregular, limited shape with deep learning models using image processing algorithms. It demonstrated an object recognition project example of an object recognition project with high accuracy. When tested, it was possible to correctly recognize and classify objects up to images with a size of 320x320 pixels. The image recognition time comparison was conducted on Raspberry Pi 5, Raspberry Pi 4, and Jetson Nano.

The target device, Raspberry Pi 5, achieved real-time execution (100 ms at most or one fps) only. It has more CPU performance than other devices.

REFERENCES

[1] MLPerf Inference - MLCommons. (2024, November 18). MLCommons.https://mlcommons.org/working-groups/benchmarks/inference/

[2] Ameen, S., Siriwardana, K., & Theodoridis, T. (2023). Optimizing Deep Learning Models For Raspberry Pi. ArXiv (CornellUniversity). https://doi.org/10.48550/arxiv.2304.13039

[3] 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

[4] 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

[5] 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

[6] Jocher, G. (2020, August 21). ultralytics/yolov5. GitHub. https://github.com/ultralytics/yolov5

[7] Tensorflow. (2019, July 15). tensorflow/models. GitHub. https://github.com/tensorflow/models/tree/master/research/object_detection