Evaluation of Edge AI Co-Processing Methods for Space Applications

Master of Science in Technology Thesis
University of Turku
Department of Computing
Robotics and Autonomous Systems
2023
Luca Vicenzi

Supervisors:
Prof Tomi Westerlund
Ir. Egor Tamarin

UNIVERSITY OF TURKU
Department of Computing

Luca Vicenzi: Evaluation of Edge AI Co-Processing Methods for Space Applications

Master of Science in Technology Thesis, 85 p.
Robotics and Autonomous Systems
June 2023

The recent years spread of SmallSats offers several new services and opens to the implementation of new technologies to improve the existent ones. However, the communication link to Earth in order to process data often is a bottleneck, due to the amount of collected data and the limited bandwidth.
A way to face this challenge is edge computing, which supposedly discards useless data and fasten up the transmission, and therefore the research has moved towards the study of COTS architectures to be used in space, often organized in co-processing setups.
This thesis considers AI as application use case and two devices in a controller-accelerator configuration. It proposes to investigate the performances of co-processing methods such as simple parallel, horizontal partitioning and vertical partitioning, for a set of different tasks and taking advantage of different pre-trained models.
The actual experiments regard only simple parallel and horizontal partitioning mode, and they compare latency and accuracy results with single processing runs on both devices.
Evaluating the results task-by-task, image classification has the best performance improvement taking advantage of horizontal partitioning, with a clear accuracy improvement, as well as semantic segmentation, which shows almost stable accuracy and potentially higher throughput with smaller models input sizes. On the other hand, object detection shows a drop in performances, especially accuracy, which could maybe be improved with more specifically developed models for the chosen hardware.
The project clearly shows how co-processing methods are worth of being investigated and can improve system outcomes for some of the analyzed tasks, making future work about it interesting.

Keywords: Artificial Intelligence, Co-Processing Techniques, Edge AI, Edge Computing, Horizontal Partitioning, SmallSats, Space Applications, Vertical Partitioning.

Contents

List of Figures

List of acronyms

AI Artificial Intelligence

ASIP Application Specific Instruction-Set Processors

ASPP Atrous Spatial Pyramid Pooling

BEE Basic Electronic Element

CAN Controller Area Network

CERN European Council for Nuclear Research

CIF Camera Interface

CMS Configuration Memory Scrubbing

CNN Convolutional Neural Network

COCO Common Objects in Context

COTS Commercial Off-The Shelf

CRC Cyclic Redundancy Check

DL Deep Learning

DMR Data Memory Recovery

DNN Deep Neural Networj

DPR Dynamic Partial Reconfiguration

DSP Digital Signal Processing

eDDNN Enabling Distributed DNN

EO Earth Observation

EOT Eyes of Things

ESA European Space Agency

FPGA Field Programmable Gate Arrays

FPS Frame per Second

FPS Frame-Per-Second

GIL Global Interpreter Lock

ILSVRC12 ImageNet Large Scale Visual Recognition Challenge 2012

IMR Instruction Memory Recovery

ISS International Space Station

LCD Liquid Crystal Display

LPM Longest Prefix Matching

ML Machine Learning

MLP Multi-Layer Perceptron

NLR Royal Netherlands Aerospace Center

NMR N Modular Redundancy

NN Neural Network

OBDH On-Board Data Handler

PTQ Post-Training Quantization

PTQ Post-training Quantization

RHBD Rad-hard by design

RNN Recurrent Neural Network

SC-LEARN SpaceCub Low-power Edge Artificial Intelligence Resilient Node

SHAVE Streaming Hybrid Architecture Vector Engine

SmallSats Small Satellites

SoC System-On-a-Chip

SSD Single Shot MultiBox Detector

SSD Single-Shot Detection

TFLite Tensorflow Lite

TMR Triple Modular Redundancy

TPU Tensor Processing Unit

VBN Vision Based Navigation

VPU Vision Processing Unit

1 Introduction

This chapter provides a comprehensive overview of the broader context in which this thesis is situated. Thus, it also introduces the research questions that the project aims to address.

Artificial Intelligence (AI) and Machine Learning (ML)/Deep Learning (DL) functionalities are revolutionizing terrestrial applications in every field, creating an opportunity for innovative and disruptive solutions to be implemented to improve daily life, and not only that. Therefore, the space sector also can take advantage of these techniques, especially because, due to technological progress related to the miniaturization of processing platforms, SmallSats (below 1200Kg) are now affordable for private companies, who have interests in enabling Earth observation (EO) and other applications.

However, one of the main limiting factors, especially for SmallSats, regards the amount of data that needs to be transmitted to Earth for elaboration. Up/downlink bandwidth, on-board and computing capabilities and always increasing data resolution are what limits the performances of those systems while using a classical approach with space ready processors [1].

Thus, the industry and research are going toward the study of hardware accelerators such as Field Programmable Gate Arrays (FPGAs) and Application Specific Instruction-Set Processors (ASIPs), in order to implement intelligence on the edge and consume data processing at the source, increasing data quality and reducing the

required downlink bandwidth [2]. These different modules can be implemented in the system architecture as solo modules, but also can be exploited through the use of modern System-On-a-Chip (SoC), which are designed to have different "blocks" suited for different functions, e.g., general-purpose computing or parallel deployment of algorithms [3]. The most common ASIPs that are evaluated for space mission applications are Tensor Processing Units (TPUs) and Vision Processing Units (VPUs).

1.1 Research Questions

In recent years, combined setups of the above mentioned devices involving a controller and an hardware accelerator (or more) are becoming a standard in research and space missions, These integrated configurations are specifically designed to deploy intelligent functions at the edge of space operations. This transition towards combined architectures is swiftly becoming the standard, driven by the desire to achieve superior performance levels through the utilization of cutting-edge edge computing accelerators. At the same time, these setups maintain space-resistant functionality by leveraging space-grade processors as controllers, ensuring the robustness required for space missions.

This thesis aims to investigate the co-processing performances of a combined setup composed by two different hardware accelerators, both suited for edge computing and able to deploy AI inferencing. In particular, this research project will explore different co-processing techniques and parallelization methods currently in development within research and industry, aimed to increase the performances of the system. The benefits and the drawbacks with respect to a single processor deployment are then compared in terms of accuracy and latency. Power consumption is an important factor as well, but for an initial investigation it is not essential, since AI consumption is well-known. Thus, the power optimization consideration could be part of future work, to further improve the benefits of co-processing.

The focus is on investigating the performances of techniques such as vertical and horizontal partitioning, which allow efficient parallelization of AI models, and that are used in other terrestrial domains like Internet of Things (IoT), to name one. According to the current state of the art, these above mentioned methods should exploit the hardware architecture, and therefore stand as a possible solution for the SmallSats communication bottleneck.

In short, the aim is to implement these techniques on a prototype of technology readiness level 5, and to determine their effectiveness in optimizing the performances of a hardware architecture, in terms of latency and accuracy, or if it is still more convenient to delegate the processing only to one of the devices. Important to mention is also the scope of this system, which is meant to be versatile and open to the final client's use case implementation. Hence, it is not essential to stick to space related experiments when testing the performances, but just taking advantage of heavy tasks to test the computation capabilities.

2 Theoretical Background

In this chapter, the current state of the art is investigated, both on hardware and software level, giving a solid motivation to the methodologies chosen, and described later in chapter 3.

2.1 Hardware Architecture

The hardware architectures that have been studied in this field can be divided between single processor and multi-processor ones. The first category usually takes into consideration SoC, which allows the utilization of different modules for different tasks in order to implement algorithms efficiently. Co-Processing architectures consist of combined setup of both radiation-hardened components and Commercial Off-The Shelf (COTS) instead, allowing several different techniques to divide the computational load and optimize the execution.

In general, all the architectures proposed in the industry and research see a device used as a supervisor, or controller, which manages the data I/O and turns on/off the hardware accelerators when needed [4], [5], [6], [7]. Usually, since the aim is to ensure reliable hardware platforms, the device chosen as a framework processor is a radiation tolerance chip, which assures functioning even in presence of heavy radiation. Another important characteristic to be considered when choosing which device to be used as supervisor regards the available communication ports and protocols, to handle sensors and secondary devices, and that is why FPGAs'

flexibility is highly appreciated in this context [5].

In figure 2.1, a general co-processing architecture considering i_max sensors and j_max hardware accelerators are represented.

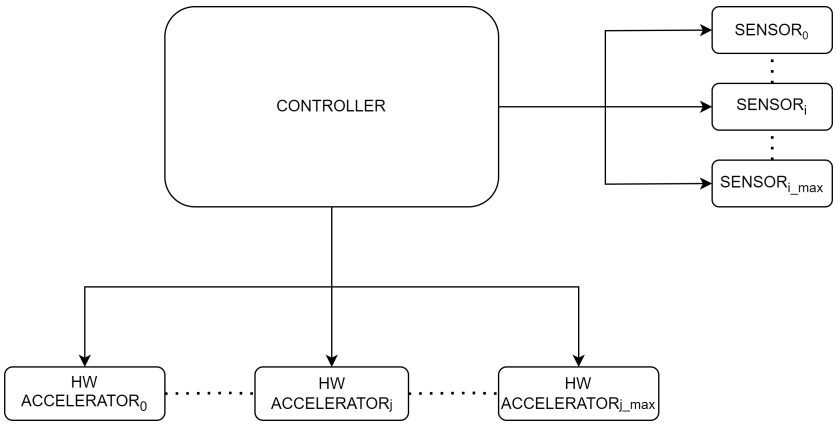

Figure 2.1: General co-processing architecture

Another characteristic that comes up often in the current research papers is the fault tolerance of the systems, which should be able to properly function even in difficult circumstances. Considering application-level fault tolerance, one of the typical approaches is the so-called Triple-Modular-Redundancy (TMR), which allows fault masking. Considering the case scenario, it reflects into using three hardware accelerators, or more in general three devices with inference capability. In this way, an error propagates only if two out of the three modules fail [4]. Also withing any single module, it is possible to assign a modular redundancy assigning the same tasks to different cores of the chip, for example with different Streaming Hybrid Architecture Vector Engine (SHAVE) cores on the Intel Movidius Myriad2 [8]. Another approach to implement the feature is based on using only two devices and slightly differs in cold standby, warm standby and hot standby systems. Any of them has different complexity and consequent fault tolerance time while facing real-time errors, but it is proven that not only fault tolerance is ensured, but also performances can be

increased [9].

On a lower level, existing techniques to improve fault tolerance are aimed for example to ensure memory correction or the communication between chips does not carry error through a Cyclic Redundancy Check (CRC) protocol [8]. Different techniques are aimed for different devices, for example considering FPGAs it is possible to implement Configuration Memory Scrubbing (CMS) and Dynamic Partial Reconfiguration (DPR), while for Intel Movidius Myriad2 there are Instruction Memory Recovery (IMR) and Data Memory Recovery (DMR) [8].

In figure 2.2, it is shown the implementation of N Modular Redundancy (NMR) for Intel Movidius Myriad2 VPU, which is a generalization of the common TMR. In fact, the redundant modules are no longer fixed to 3, but depend on the hardware resources and the design choice. With Myriad2 for example, there are 12 SHAVEs accelerators, and they can be grouped with N=3 (TMR). In this case, there would be 4 groups of cores working in parallel. However, N can be set to a higher or lower value, varying how much the workload is parallelized and the efficiency of the fault tolerance technique.

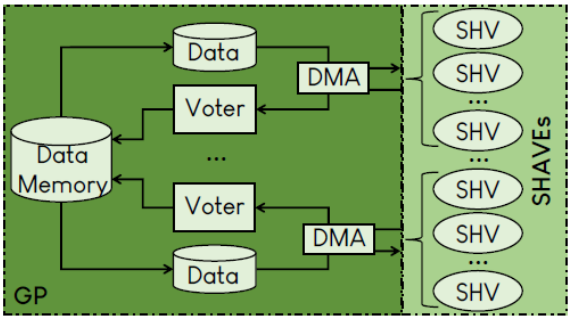

Figure 2.2: N Modular Redundancy (NMR) (© 2022 IEEE [8])

2.1.1 State of the Art

In this section, as the diagram in figure 2.3 shows, the outcome of the state of the art investigation regarding hardware architectures currently used for Edge AI appli-

cations in space is presented. The characterization mainly depends on the quantity of devices used (from one to multiple), and on the features that the corresponding configurations offer. Following the diagram, research examples belonging to the current state of the art are explained more in details.

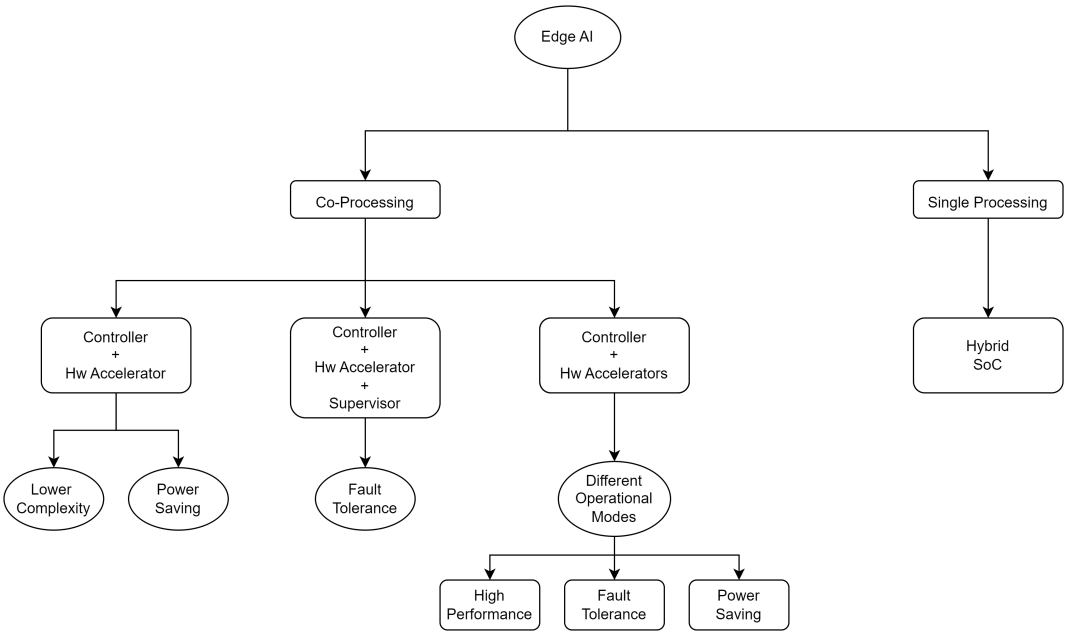

Figure 2.3: Hardware Architectures for Edge AI in Space

FPGAs are becoming a popular choice because of their flexibility in handling different types of communication from different instruments/sensors, complying with different protocol specifications at the same time and allowing less circuit design to implement the system communication.

One of them is the Xilinx XCKU060, used as a frame processor that interfaces with multiple instruments and can occasionally host high-performance heritage functions, e.g., data compression. Other than that, the Intel Movidius Myriad2 VPU has been chosen for the AI acceleration, and the modules implemented in the system are three to ensure fault tolerance with TMR technique. In this case for example, data

arrives to the FPGA via SpaceWire and after transcoding processing it is forwarded to the VPU through the Camera Interface (CIF) , while the VPUs are used for Digital Signal Processing (DSP)/AI acceleration, and consequently the output data is returned to the FPGA through its Liquid Crystal Display (LCD). This unusual way of communication takes advantage fo the LCD controller available on the VPU, which sends the data on the LCD interface, which is also connected to the FPGA, able to read data from it, generating a sort of indirect form of communication. A radiation-tolerant microcontroller (Cobham Gaisler GR716) is implemented as a supervisor to increase reliability, to ensure the correct functioning of the systems with error detection [5]. The system performs well obtaining between 6-20 Frame per Second (FPS) for different kernels such as binning, rendering and different size convolutions, which are typically used in Vision Based Navigation (VBN) pipelines, while more than 1 FPS for deep learning image classification (specifically Convolutional Neural Network (CNN) ship detection with 1MPixel images). These results are achieved with a power consumption in a range between 800-1000mW.

In figure 2.4, the above described architecture with Xilinx XCKU060 FPGA and Intel Movidius Myriad2 VPU is represented. Within the research, the testbed was completed by a host PC, used to give input data and retrieve the output results.

Referring to the summary diagram, this architecture is an example of system formed by a controller, an accelerator and a rad-hard processor as supervisor, which allows fault tolerance implementation.

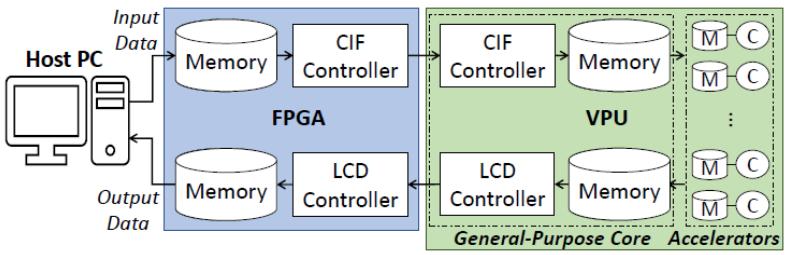

Figure 2.4: FPGA & VPU co-processing architecture (© 2021 IEEE [5])

Another architecture of significant importance in the field is the SpaceCub Low-power Edge Artificial Intelligence Resilient Node (SC-LEARN), which is a CubeSat-sized Edge TPU-based co-processor, as well as the supporting test card designed for rapid prototyping and evaluation. The architecture is a combination of three Google Edge TPUs together with a high-performance processor, responsible for powering on/off the individual Edge TPU module through some switches and able of answering to different scenarios such as switching from disaster detection mode (e.g., earthquakes, tsunamis, floods and fires), to highly accurate targeting modes (e.g., specific object-targeting data). Even if multiple choices are possible, the device taken into consideration is the SpaceCubev3.0Mini, which features the Xilinx Kintex UltraScale KU060 FPGA [4]. SC-LEARN has three operational modes, developed in order to answer to different situations. The first one is the high-performance mode, where the three TPUs are used to exploit parallelization and improve the performances (inference is executable in a parallel platform). The second mode is the fault-tolerance mode, which implements TMR and allows also to switch off one of the modules in case of high-current event or malfunctioning. The last mode is the power-saving one, which is designed for conserving on-board power by depowering two of the three cards. During this mode, any of the three could be the one working, and this could help in case of malfunctioning. The AI models chosen to assess the performances of the system were two, based on their compatibility with the Edge TPU's set of supported operations and their reported high accuracy on publicly available hyperspectral datasets. The first one is a 1D Multi-Layer Perceptron (MLP) documented and tested in [10], while the second network is a spectral-spatial CNN (SS-CNN) for hyperspectral image classification described and tested in [11].

Referring to the initial scheme, the architecture developed by Nasa is an example of a system with a controller and multiple accelerators, which allow the deployment of several different operational modes.

During space missions different case scenarios can occur, therefore a space system architecture needs to have not only advantages in terms of performance, power, programmability, dependability, but also flexibility and adaptability to eventual sudden changes and problems. For instance, an architecture could serve as a centralized processor, it can be placed inside the instrument control unit or payload data handling unit, or be used as an On-Board Computer co-processor, depending on the mission specifics. To face this challenge, another existing architecture is made by an FPGA (Xilinx XC6SLX150 chip on the Trenz Electronic TE0600 Spartan-6 board), which has the function of framing processor and it is very useful for its capability of interfacing different instruments and sensors with different I/O protocols, and furthermore efficiently apply and implement different forms of data reduction, voting systems, hardware watchdogs and timers [7]. Intel Movidius Myriad2 takes care of the DSP&AI acceleration, isolating it from the rest of the system and opening to a fault-tolerance solution that can de-risk less mature and non-space components. To increase the fault mitigation in the architecture even more, redundancy of Myriad2 chips, triple modular redundancy protection of the FPGA circuits or even including a Rad-Hard by design (RHBD) microcontroller as supervisor are options to be considered. The communication between the two devices is handled through an SpW switch, since as most of the COTS HW accelerators, also Myriad2 misses the SpW/SpFi ports to interface with traditional space-grade instruments. The FPGA takes the data coming from its SpW interface, stores it in its DDR memory, and then transcodes it and forward it as Camera Interface (CIF) format, which is supported by the Myriad2 SoC (the paper describes the logical elements of the SpW transcoder implemented in the FPGA). Implementing a satellite pose tracking algorithm developed for an ENVISAT at distances of 20m to 50m, which is representative of an average computationally intensive computer vision pipeline in space applications for this type of systems. Using different optimizations in scheduling the algorithm

tasks, system level results are 2.6 to 4.9 FPS of frame rate for pose tracking on 1Mpixel images, while the power consumption values are from 0.8 to 1.1W.

The above paper study shows an other example of architecture with controller and supervisor, which also clearly mentions the possibility of implementing other accelerators or a rad-hard microcontroller to increase the realiability of the system.

Another research to be considered is the ϕ-Sat-1 mission, which is the first experimental demonstration of Artificial Intelligence (AI) reliability and accuracy in performing on-board cloud detection in a hyperspectral imaging mission by the European Space Agency (ESA). The AI capabilities of the architecture are enhanced by the use of the so called Eyes of Things (EOT), a board developed by ESA's Technology and Quality department that features the Intel Movidius Myriad2 processing unit, which has been already described above [6]. Something to be added respect to what has already been said is the fact that ESA managed to assess a successful radiation characterization of the device, in collaboration with European Council for Nuclear Research (CERN) [1]. The architecture is made up of the Basic Electronic Element (BEE), which contains the On-Board Data Handler (OBDH) and an FPGA. The BEE is the electrical interface to the spacecraft, and it is latch-up protected. It works as a framing processor/controller, taking the input data that is provided by the FPGA, used for transcoding, and communicating with the EOT module to execute the process. The system has been tested implementing a cloud scout segmentation neural network (NN) to identify, classify and eventually discard on-board the cloudy images, while assessing the Hyperscout-2 hyperspectral sensor. Because of the new sensor, the dataset used to train the NN model has been synthesized from a previous existing dataset, and it has been proved that the results are reliable. Thus, this means that it is possible to directly exploit new sensors on board. Tests have been made both in development phase using the synthesized dataset and with real data during the ϕ-Sat-1 mission and the confusion matrix

confirms that false positives respect the requirement of being <1% in both cases. Since the acquired data with the Hyperscout-2 was not meant to challenge the NN, but to simulate the real application of the system, the accuracy of its test is slightly higher than the synthesized dataset one, which instead considers every case scenario randomly (e.g., clouds on snow or on Salt Lake). Accuracies are respectively 96% versus 95.1%.

In figure 2.5, the ϕ-Sat-1 architecture, which has been described above, is shown. It is an example of a system with controller, accelerator and supervisor, according to the summary diagram exposed at the beginning of the section.

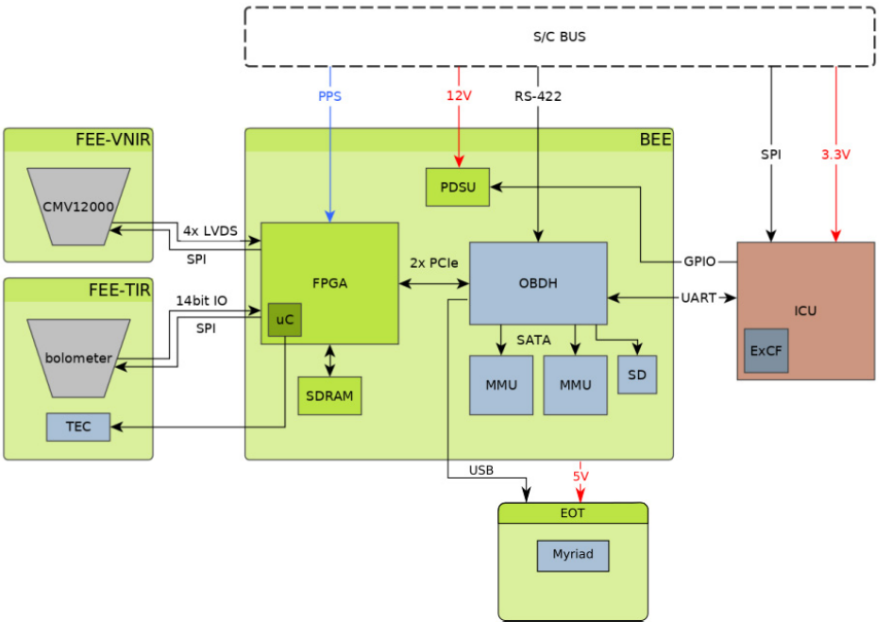

Figure 2.5: ϕ-Sat-1 Mission architecture [6]

ϕ-Sat-1 Mission follow-up by the European Space Agency uses using Intel Myriad X instead of Myriad2. In this case the processor is the HPE's Spaceborne Computer-2 (SBC-2) together with two Qualcomm Snapdragon 855 HDK's, mounted on the International Space Station (ISS) [12]. The system has been evaluated applying different machine learning models such as Mars HiRISE Classifier to classify Mars

surface using orbital imagery, and two Mars MSL Classifiers to respectively classify Mars rovers' images and rovers' parts from the orbit for spacecraft health analysis. These three models worked with an inference time of around 16ms per image (> 60 fps). Other models implemented are UAVSAR Flood Mapping for alert generation, with 166 fps rate performance, and ship detection in satellite imagery, with 10.2 fps. Moreover, a DDR test has also been run to calculate memory bit errors, but none has occurred.

It is also important to consider a broader range of architectures, and not only the ones specifically designed for space applications since the beginning. For example, an existing architecture aimed to get fast IP processing, with special focus on Longest Prefix Matching (LPM) algorithm, one of the heaviest functions implemented in internet routers, is made up by three components: a control processor, a co-processor and a needed customized co-processor interface [13]. The control processor (CPU) executes the algorithm, except for the LPM, that is accelerated thorough HW-SW co-processing on the co-processor. It is a XuantieC910, a high-performance open-source RISC-V processor core developed by T-head. Because of the accelerations, results show an overall 111% performance improvement for every searching iteration of the algorithm, and also power consumptions are lowered by a significant factor.

Again, the mentioned research shows the utilization and evaluation of a controller and accelerator architecture.

2.2 Software Architecture

When targeting high performance along with lower power consumption, COTS-based design with ASIPs (for example Google TPUs and Intel VPUs) and SoCs are the most suited, and they also offer practical advantages in terms of ease of programmability. In fact, for using these heterogeneous architectures and optimizing their processing there is no need time-consuming design of digital circuits, like it

is needed for FPGAs. For example, Intel Movidius Myriad2 is programmed via an LLVM-based vectorizing C/C++ compiler, which lowers the programming complexity of the software architecture [7].

Given the hardware architecture, the real challenge consists in the efficient utilization of the chips' resources through parallelization and by algorithm optimization through the application layer of the software.

While considering FPGAs, the challenge consists in designing the hardware components needed to accelerate the algorithm that needs to be implemented to reach a required performance and also the communication interfaces that are needed to communicate with sensors and devices. Facing the challenge of COTS devices instead, the options are of using already existing frameworks of libraries developed by the chips' producers or to develop custom ones, or even use them both in order to take advantages of the hardware resources.

An example of a methodology and a support framework to handle the programming complexity of parallel algorithms on Intel Movidius Myriad2 together with a critical avionics' architecture is presented in a research proposed by University of Athens, in collaboration with ESA and Intel [7]. Since the hardware resources are fixed and the basic functionalities are already implemented, the complexity is already lighter than other systems that use FPGAs, but still needs to be faced in some way. In the specific case, the algorithm chosen to evaluate the system is a vision-based Pose Estimation/Tracking algorithm, which allows straightforward comparison to competitive implementations already existing on other embedded devices. The tasks are assigned to different components of the SoC, parallelizing and optimizing their running, in order to maximize the use of the hardware and consequently the performance/power rate, using techniques described in the following section 2.2.1.

A custom support framework has been implemented to facilitate programming

complex parallelization over MDK, which is the development tool of Myriad2, that offers only basic functionalities and demands low-level coding when targeting efficient custom implementations. The idea was to create an abstraction layer to handle system resources for I/O, memory and task management, and IPC by developing a set of lightweight, stand-alone, and transparent C/C++ libraries.

The are different modules that have been implemented: SHAVE Inter Process Communication (IPC), Memory Management, I/O Management and SHAVE Acceleration Management (SHAVE stands for Streaming Hybrid Architecture Vector Engines, and they are the key processing units in the device). These add new mechanisms, improve already existing ones' performances, and provide automatic/-transparent device configuration.

Considering the SHAVE IPC module, it is essential because most of the parallel algorithms require inter process communication. Since Myriad2 provides limited hardware for this purpose, e.g., a FIFO structure that transmits at most 128 bytes. In a few words, instead of the classical shared memory programming model proposed by Intel taking advantage of the so-called Connection Matrix (CMX, which is used as a scratchpad memory), the custom module implements a barebone IPC that resembles the MPI standard, so a message-passing protocol, with increased sizes respect to the original model.

Considering the memory management instead, by default when multiple individual kernels want to access the same CMX memory resource, MDK uses a library to load data and code in the CMX at runtime, with some time penalties. The custom module implemented preloads all the kernels' code in DDR memory, without the CMX being allocated yet. Each SHAVE has access to its CMX slice, organized like a memory heap, using a custom "malloc" function. "Free" function is not implemented to simplify the complexity, but to avoid congestions and race conditions there's a dedicated management instance running on each SHAVE.

For the I/O management, the implementation of a software module running on LEON RT (LRT, the second core of Myriad2) wraps the low-level drivers and interrupts of the SoC peripherals and provide an higher level API to accelerate the development of I/O tasks. Of great importance is the SHAVE acceleration management module, within different aspects have been faced. First, high-level abstraction from the hardware, implementing an API to manage low-level operations while accessing SHAVEs.

Since every task may have a different execution time, another important aspect is the amount of idle time that every SHAVE module has after completing a task. Thus, the custom framework implements dynamic task assignment at runtime through a FIFO structure managed by LRT. In this way, the idle time of every SHAVE should be the minimum possible, because as soon as they are done, they get a new available task. Non-parallelizable tasks with bigger cache and/or enabling any required Real-Time Executive for Multiprocessor Systems (RTEMS) feature are handled by LOS, which supports real time elements.

Last but not least, the architecture has different physical implementations for the cache of LEONs and SHAVEs, therefore there is the need of a cache coherence mechanism. This software component has been developed as well as part of the custom framework.

In general, the framework is ideated in order to avoid the introduction of complex schedulers as well as higher-lever automation of cache coherency to optimize the algorithm. Moreover, the setup allows to implement important space requirements such as mitigation, through SW redundancy, cache disabling, error detection, as well as power/thermal control, through core deactivation, and also fault tolerance (the SHAVE acceleration manager can replicate N times any function and do it concurrently in 12%N cores).

The above exposed is an example of custom software framework to improve the

performances of a program to run on an hardware accelerator, in this case Myriad Movidius2. However, the same idea could be applied to other accelerators.

2.2.1 Task scheduling

To get optimization, task scheduling is crucial and must be improved as much as possible. It is important to take into consideration every hardware resource available and the required performances in terms of latencies and power consumption.

There are different approaches on how to handle this challenge, and for example an idea could be to implement an intelligent algorithm to dispatch the task in the most efficient possible way. Reinforcement learning and deep learning (or also the combination of the two, which is a Deep Q-Network) are methods that could actually handle well this challenge, and some research projects are actually relying on those [14], [15]. Although these methods deploy run-time dynamic resources allocation and allow dynamic hardware availability, it needs to be reminded that the quoted case scenarios are quite different from the one considered in this research, and that they are extremely heavy in terms of computations. In fact, for the quoted research the topic regards clusters of edge devices for IoT applications, which have way more computational power on the core cloud processor, and therefore can apply AI. For what regards the architecture this research investigates, it makes not much sense to waste so much computational power just for scheduling, while the aim is to get the maximum performance gain on the actual AI models to be run.

Another approach to save computational resources while defining the dispatching algorithm is fixed scheduling, and the criteria to follow are several. An example can be made taking into consideration the Intel Movidius Myriad2 architecture, which offers the SHAVE subsystem, LEON4 or hardware filters [7]. The first criteria to take into consideration performance versus power gains, e.g., on Intel Movidius Myriad2, hardware filters should be avoided because they are low power, but their

performances are lower than the SHAVEs modules. Another aspect is the parallelization potentiality of a function: sometimes algorithms require to be completely executed sequentially and it is more convenient to run them on LEON4 instead of SHAVEs, while other times the acceleration modules give a huge speedup. Memory access patterns are heavily relevant when considering the effectiveness of SHAVEs, the more they are predictable according to locality principle the better performances SHAVEs have. On the other hand, if they are remote and random, it is more efficient to rely on traditional caching techniques, available on LEON4. Last but not least, library dependencies and heritage issues are software elements that need to be considered when scheduling functions. For example, when the task includes huge software libraries with deep call graphs, it is more practical to map it to LEON4 because of its better support and higher library availability.

Considering these criteria, that slightly change according to the considered architecture, it is possible to define a static schedule to get a significantly high system performance speedup. The above example regards Movidius2, but it could be applied to other accelerators as well.

2.3 Co-Processing AI Acceleration

Different acceleration techniques exist for different algorithms depending on their main used operations and how they are implemented considering the system hardware resources. Since the aim of this thesis is to evaluate the performances of AI models (ML/DL) run on the edge in a space scenario, which involves having more constraints on the needed functioning in terms of speed, for example because of high-rate input data, it is therefore essential to consider ways of accelerating the considered AI models' execution on the edge. Given that the hardware architecture consists of two devices, each one with several cores or modules and capable of running neural network inferences, methods to take advantage of all the computational

capabilities are of important consideration.

In figure 2.6, a diagram summarizing the main AI acceleration techniques able to improve performance for edge AI inferencing in the above discussed architectures is shown. They will be investigated in the subsections below.

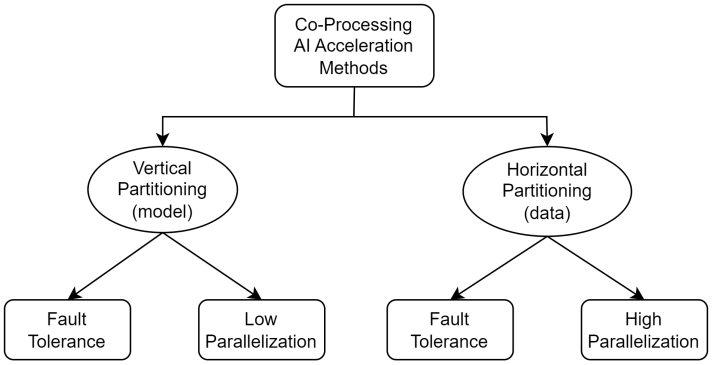

Figure 2.6: AI Acceleration Methods

2.3.1 Vertical Partitioning

A method already implemented and that significantly improves the performances is the so called ***vertical partitioning***, which consists of identifying the independent tasks in the neural network (AI model) and assign them to different processor units, allowing inference parallelization and also fault tolerance if the system implements software redundancy [14]. Usually, AI models are split whenever a single device has not enough resources to fully load it and run it, therefore there is the need of loading only a first part, and then use another device for the remaining layers. Withing the mentioned research instead, the new method called Enabling Distributed DNN (eDDNN) , and briefly described above, is proposed. This method can be implemented for example using a TensorFlow environment and working on the reducing the dependencies of the neural network layers, as it is showed in [16].

In figure 2.7, the difference between the two vertical partitioning approaches are

shown. The main potentiality of eDDNN consists in the fact that in case *(a)* parallel execution is not possible, while in case *(b)* it is.

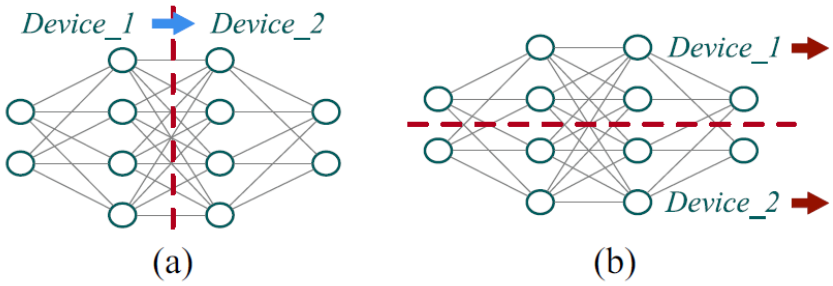

Figure 2.7: (a) Traditional Partitioning (b) eDDNN Partitioning (© 2022 IEEE [14])

2.3.2 Horizontal Partitioning

Although vertical partitioning improves the performances, neural networks will always have a sequential execution because of their layers' dependencies, so even if different layers are assigned to different units, it is difficult to obtain full parallelization. Therefore, another idea is to implement the so called ***horizontal partitioning***, which consists of implementing the same neural network model on all the units with a smaller data size, and apply the same inference to different portions of the input data [14]. The result is obtained by merging the partial results coming from the smaller networks, which have all the network's layers. In this way, ideally full parallelization can be obtained. Also, considering a setup that comprehends several devices able to perform an inference, it also possible to implement a different functioning mode that implements fault tolerance. This could be achieved by using deploying different AI models on the different devices and compare the results, for example following the N Modular Redundancy (NMR) technique. However, in this case while gaining in fault tolerance, the acceleration in performances get lost.

Another consideration that must be made though is that in [14] the considered

devices are stand-alone Internet Of Things (IoT) ones capable of running an AI inference and connected via wireless, so ideally the more devices are added, the more performances should improve. While considering heterogeneous architectures as the ones mentioned above instead, and the number of devices restricted to just two, the maximum number of sub-models that can be implemented depends on the hardware resources available and therefore if the data size is still big, the performances might not improve significantly.

2.4 Custom AI Models

When considering acceleration for deep learning models on constrained devices, research has developed and keeps studying efficient architectures. Existing and widely used AI models are several, also within embedded systems. Neural networks of every sort and shape can be valid for this architecture, from Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), to Deep Neural Networks (DNNs) and Multilayer Perception Networks (MLPs). The first approach is to take into consideration already existing optimized networks for embedded devices, such as MobileNet, ResNet, AlexNet, YOLO, Inception, etc., which can run different tasks like Image Classification, Object Detection and Tracking, to name some [1], [17]. A characteristic that is important to consider while running them is the weight quantization, which may significantly change the performances in terms of latency, power consumption and accuracy (sometimes also if a model can be deployed on the device or not).

Another approach consists of taking inspiration from commercially available models such as the ones named above and developing a customized neural network to take advantage as max as possible of the architecture hardware resources. This means studying the network layers' dependencies and try to make it so that they can be parallelized in some way, either with vertical or horizontal partition-

ing [14]. An example for space applications is represented by the ϕ-Sat-1 mission, since the cloud scout segmentation DNN has been developed on purpose for it [6]. In that case for example, the device running the model was Intel Movidius Myriad2 and therefore particular attention had to be devoted to the implementation of the convolutional/deconvolutional layers because of the limited intra-layer memory available. Also, the VPU offers 16-bit floating point arithmetic, so the quantization was set to take advantage of that, and an input size reduction was applied to avoid memory saturation.

2.5 Benchmarks

Benchmarking is not always an easy task, since the aim is to evaluate a system in terms of general performances, and it is easy to get results that are valuable, but only for the specific use-case tested within the experiments.

In this merit, parameters that are to be taken into consideration are time of execution of certain tasks and the related power consumption. To do so, for example a set of custom software benchmarks could be implemented, like it is done in [5] in order to test all the different modules of the architecture. Same thing can be done using a third-party software developed for the purpose, but that might be too general considering the use-case. Since the system will be used for edge AI, it might be more interesting to directly test the performances of time/latency, power consumption and accuracy, during both training and inference for an AI model. However, these results severely change depending on the model chosen and that probably is the interesting part.

In general, when considering co-processing setups, it is also important to consider intercommunication latencies together with the processing times, since at the end the system needs to be evaluated as a single block. Also, these communication latencies need to be low enough to not influence the overall throughput, as it is

shown for example in [5].

3 Methodology

In this chapter, the methodology chosen to address the research questions with practical answers is described. This comprehends hardware architecture, co-processing techniques and approaches.

3.1 Hardware Architecture

Following the state of the art trend, the selected architecture for this research project is a co-processing setup, which gives more flexibility and more operational power in terms of available resources. Despite the first thought of approaching an FPGA, a SoC has been chosen as the controller because of its computational resources and several interfaces available for interconnections, and also because to develop AI on the FPGA there would have been a license, and that was in contrast with the project purpose.

The chosen co-processor as AI accelerator has high power efficiency, and the idea was to implement two modules of it, since they are available in development module that can be plugged into SoC starter kit. Unfortunately, these mentioned modules were out of stock, therefore only one has been used.

An interesting characteristic of this architecture is the AI acceleration potentiality both the devices feature, therefore opening to several different co-processing opportunities to try.

Before choosing the architecture, some radiation testing has been performed in

order to have a de-risking analysis, and the results show how both the processors did not have any significant effect. Unfortunately, even if the presence of two hardware accelerator modules would have opened up to implementing different operational modes such as high performance, fault tolerance and low power, as described more in detailed in the previous subsection, this was not possible.

3.2 Planned experiments

In order to answer the research questions mentioned in section 1, there is the need of defining the right experiments to perform, in order to get valuable and meaningful data. This subsection describes the initial plan of experiments, based on the literature review carried out in section 2, which highlights the current state of the art for the subject.

First of all, another general consideration to be made is about the chosen framework to deploy the chosen models, which are presented further below in the section. Among the available open-source software frameworks, the choice is to go with models developed and optimized with Tensorflow (more specifically Tensorflow Lite), mainly because of the amount of available pre-trained models and because of its spread both in research and industry. A framework overview is in the following section.

3.2.1 Tensorflow and Tensorflow Lite

TensorFlow is a versatile and widely used open-source machine learning framework developed by Google [18]. It provides a full range of tools and frameworks for creating, honing, and deploying machine learning models across numerous platforms and applications. TensorFlow is a well-liked option for researchers, data scientists, and developers working on challenging deep learning projects because of its adaptability

and scalability. It is well suited for common machine/deep learning tasks such as image recognition, natural language processing, and reinforcement learning in a variety of environments. In fact, the framework includes support for cloud servers and high-performance computing clusters, allowing deployment for distributed computing. Moreover, GPU acceleration is another essential characteristic that makes the framework one of the most used tools in the area, also considering the rich ecosystem of pre-trained models available.

A modified and lightweight version of the framework called ***TensorFlow Lite*** was created specifically for the deployment of machine learning models on embedded systems and edge devices, such as microcontrollers and other mobile platforms with limited resources like embedded Linux, Android, and iOS devices [19].

It is crucial for allowing AI and machine learning at the edge, where local data processing is necessary for low latency, privacy, and effective resource use. TensorFlow Lite accomplishes this by providing model optimization methods like quantization, model size reduction, and hardware acceleration support, making sure that models function effectively on devices with constrained memory and computing capacity.

The greatest characteristics is that TensorFlow Lite easily integrates with TensorFlow, enabling developers to train advanced models within the TensorFlow ecosystem before converting then, and therefore deploying them to edge devices with optimized performances. This closes the gap between AI research and practical implementation on embedded systems, making this framework a valuable tool for a variety of applications, from real-time object detection and image classification in smart cameras and drones, to voice recognition in smart speakers and sensor data analysis in Internet of Things devices.

The conversion flow from Tensorflow to TFLite is shown below in figure 3.1.

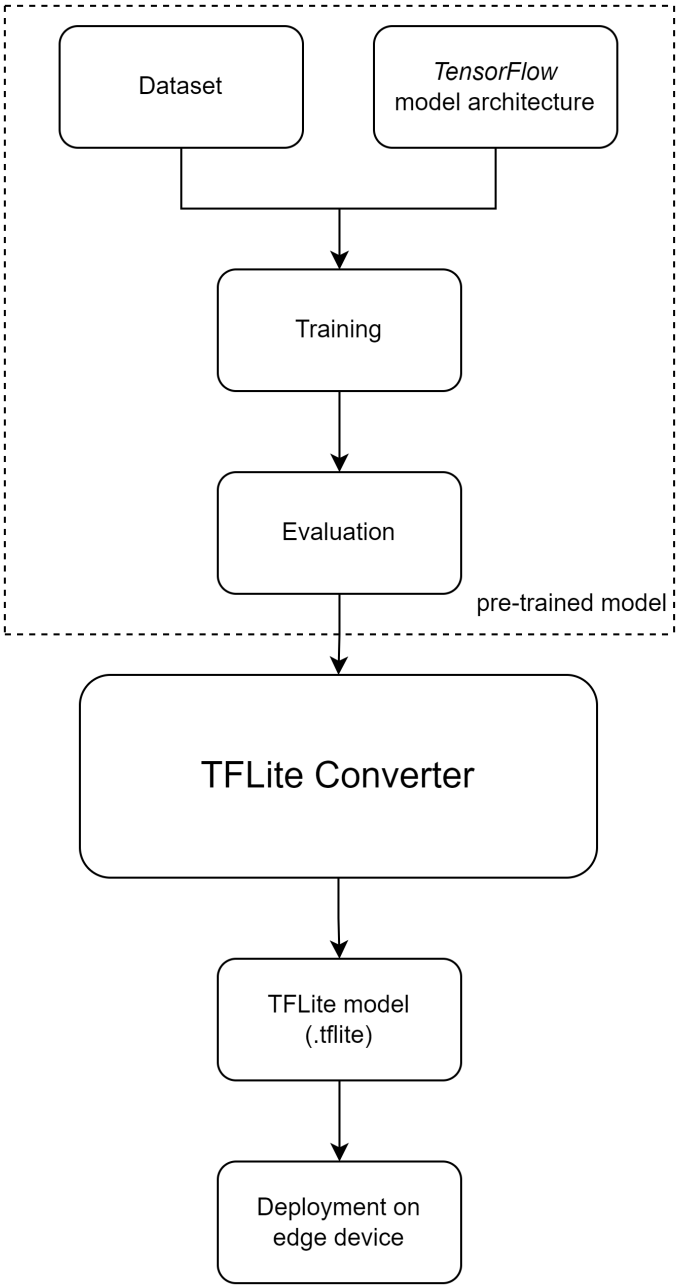

Figure 3.1: TFLite Conversion Flow

Furthermore, its open-source design, multi-language support *(Java, Swift, Objective-C, C++, Python)*, and vibrant community support make it a dependable option for creating edge AI applications that bring innovation in a variety of sectors, including

healthcare, automotive, robotics, and more.

3.3 AI Models

Since the project aim is to study an architecture which is meant to be general and open to every user's use case, and not just related to a specific application, the used AI models themselves are not too important while evaluating the overall performance. What matters is the representation of a typical heavy task that includes the computational complexity of machine/deep learning inferencing. Therefore, the choice of the AI model can be totally arbitrary.

This thesis project has been carried out together with the Royal Netherlands Aerospace Center (NLR) , which provided an AI model they had already tested on the main device.

Therefore, the first initial idea was to take advantage of the available model, that they had already trained and optimized to run on the device, through the manufacturer's software framework. Moreover, this choice would have allowed the implementation of space significant tasks as case scenarios for performance evaluation, which would have been a nice addition to keep it more significant.

However, from that choice some problems follow up. Since the focus of the thesis is not on creating, training and tuning deep learning models, but only to use them to inference, it is preferable to opt for already available models, which do not need to be trained for the specific application. Unfortunately, there are not so many ship detection or cloud detection/segmentation models that are already trained and available to use, therefore the choice of which task to pursue had to change.

The final considered approach took into consideration the list of available models for demo purposes of both the devices, which implement common task such as object detection, image classification, semantic segmentation, and others. These three mentioned applications are the ones that have been chosen as case scenarios,

and for what regards the models themselves, some more considerations have to be made. The hardware accelerator is the device with lower computational capabilities, therefore what can run well on it can also run with high performances on the main device. Following this reasoning, two models per task have been chosen from the list of available optimized ones to run inference on the accelerator device. The idea is to compile them and optimize them also for the main device, which should run them even faster.

Both the devices' producers suggest not to train the AI models directly on the devices, but rather train them on a more powerful hardware and then load the pre-trained model, after the right optimization. Therefore, there was an initial idea to try edge inferencing with different quantization formats for every model, but that is not possible for one of the two boards, and also not really relevant in the context of the project.

Another initial idea was to develop a custom neural network from scratch, according to the hardware characteristics of the architecture, in order to obtain maximum optimization, but again it goes out of the scope for what regards this thesis project.

Considering *object detection*, both the models chosen belong to the MobileNetV2 architecture family, and both of them have been optimized with Tensorflow Post-training Quantization (PTQ) and implement the Single-Shot Detection (SSD) . The input size is 300x300 for both of them.

For *image classification* instead, the first model belongs to the EfficientNet family, and it has been optimized with Tensorflow PTQ as well. The other model is based on MobileNetV3. The input sizes respectively are 300x300 and 224x224, while the dataset for both the models is ILSVRC2012.

Finally, talking about *semantic segmentation*, both the models belong to the DeepLabv3 family, have been optimized with PTQ technique and they have been trained with Pascal VOC2012.

To give an overview, these are the chosen models for the boards, whose details will be investigated more in the following section.

- **Object Detection**
  - SSD MobileNetV2 COCO17 Quant PostProcess 300x300
  - SSD MobileNetV2 COCO17 PTQ 300x300
- **Image Classification**
  - EfficientNet S Quant ILSVRC2012 224x224
  - MobileNetV3 1.0 PTQ ILSVRC2012 224x224
- **Semantic Segmentation**
  - DeepLabv3 MNV2 DM05 PASCAL-VOC2012 Quant 513x513
  - DeepLabv3 MNV2 PASCAL-VOC2012 Quant 513x513

3.3.1 MobileNet

MobileNet is a specialized deep learning architecture tailored for embedded deployment, ideal for devices like smartphones, IoT devices, and edge computing systems. By using *depthwise separable convolutions*, which substantially reduce the number of parameters and processing requirements, it achieves an amazing balance between accuracy and computational cost [20]. Depthwise separable convolution essentially is a convolution split in two layers, which are depthwise and pointwise convolutions. Splitting a regular convolution operation in these two is a method that allows to require way fewer computational resources respect to standard filters, and therefore create light weight models.

Consequently, real-time inference on resource-constrained devices is made possible by this compact architecture, making it an essential tool for a range of edge

applications, from autonomous drones to intelligent cameras. The rapid development and adaptability of MobileNet have established its status in the design of AI solutions for embedded systems.

3.3.2 EfficientNet

As cutting-edge deep learning architecture specifically designed for embedded use, EfficientNet meets the demands of devices such as smartphones, IoT sensors, and edge computing systems. Its unique model *scaling technique*, which uniformly adjusts network depth, width, and resolution using a *compound coefficient*, optimizes both performance and efficiency, as it demonstrated scaling up neural networks based on MobileNet or ResNet [21].

Moreover, through neural architecture search, a new baseline network gets defined and scaled up to the so called EfficientNet family of models. EfficientNet stands out for its excellent ability to balance computing demands and accuracy across a broad range of applications, from object detection to image classification. It emerges as a game-changing element in embedded AI due to its adaptability and scalability, enabling real-time inference on resource-constrained devices and rolling into a new era of intelligent edge applications.

3.3.3 DeepLab

DeepLab is a state of the art deep learning architecture renowned for its exceptional performance in semantic image segmentation tasks, which derive from its unique and peculiar characteristics, that make it a powerful instrument in the field of computer vision [22]. By adopting *dilated convolutions*, DeepLab is able to efficiently gather multi-scale contextual data, which is essential for precise object segmentation. It also uses *Atrous Spatial Pyramid Pooling (ASPP)*, which allows context aggregation at multiple levels and improves the ability to distinguish things from their back-

grounds. Additionally, DeepLab has a decoder module that combines high-level and low-level features to improve segmentation results, producing masks that are more accurate and comprehensive. It is ideally suited for embedded systems and real-time applications due to its adaptability to different backbone architectures and resource-efficient design, and therefore it is a very common NN family for computer vision research and applications.

3.3.4 Single Shot MultiBox Detector (SSD)

Single Shot MultiBox Detector (SSD) is an object detection technique that has become commonly used because of its interesting characteristics [23]. SSD stands out in several ways, that are briefly introduced below in this section.

First and foremost, SSD works in a *single pass over the input image* to carry out the task of real-time object detection. Unlike two-stage detectors such as Faster R-CNN, which include processes like region proposal and object classification. The single-pass method makes SSD much quicker and better suited for low latency applications like robotics and autonomous vehicles, with deployment on embedded systems.

The *usage of "default boxes" or "anchor boxes"* is another distinguishing aspect of SSD. It is able to directly forecast object locations and sizes thanks to these preconfigured boxes that are positioned at multiple sizes and locations inside the input image. This solution is more effective and efficient since it *simplifies the architecture and eliminates the requirement of a separate region proposal stage.*

Additionally, by predicting object bounding boxes and class probabilities at various feature map levels, SSD makes use of a *multiscale technique.* Since the resolution varies across levels, *SSD can efficiently detect objects of various dimensions.* In comparison to some two-stage detectors, SSD may not attain the maximum precision possible, but it strikes a compromise between accuracy and speed, making it suitable

for a variety of real-time object recognition demands.

The object detection and bounding box regression components of SSD are trained concurrently with shared convolutional layers as part of its end-to-end training methodology. In comparison to some other approaches, this makes the *training process simpler*.

In summary, SSD excels in speed and efficiency due to its single-pass design, default box mechanism, and multiscale predictions, making it a desirable option for situations where real-time processing is essential. The decision between SSD and other techniques will rely on the particular needs of the application, including the appropriate compromise between speed and accuracy. However, its performance may not approach the highest precision reached by more complex two-stage detectors.

3.3.5 Post-Training Quantization

Post-Training Quantization (PTQ) is a technique employed in deep learning to optimize trained neural network models for deployment on resource-constrained hardware [24].

Quantization is used to reduce the memory footprint and computational requirements of a model after it has been trained using high-precision floating-point numbers, such as 32-bit or 16-bit. The weights and activations of the model must be transformed into lower-precision fixed-point numbers, most frequently 8-bit integers or 16-bit fixed-point values. Weight quantization rounds the continuous weight values to discrete values within the chosen precision, while activation quantization similarly quantizes the model's activations.

Once it has been quantized, reduced memory consumption, quicker inference times, and increased energy efficiency are all advantages, therefore the model can be used for inference on platforms like smartphones, IoT devices, or in general edge computing devices.

However, quantization may also introduce a loss in the model accuracy, which can sometimes be mitigated through calibration and fine-tuning, in order to find the preferred balance between model size, inference speed and obtained accuracy.

3.4 Datasets

Even though the chosen models are pre-trained ones, it is still relevant to present the datasets that have been used for that purpouse, also because they are used to test the performances as well. Therefore, in this section the datasets related to the chosen models are briefly described.

3.4.1 COCO17

The Common Objects in Context (COCO) dataset is a widely recognized and extensively used benchmark dataset in the field of computer vision. It is well known for its wide range of complexity and rich diversity, which makes it a priceless tool for training and evaluating a variety of computer vision tasks, including object detection, instance segmentation, and image captioning [25]. More than 200,000 photos from COCO represent a wide range of contexts, objects, and situations. These photos have comprehensive, pixel-level annotations that identify the categories, occurrences, and keypoints of various objects.

In addition, COCO offers a wide range of 91 object categories, from simple ones like cars and animals to more complex ones like sporting goods and food. COCO is a crucial dataset to support the development of cutting-edge computer vision algorithms because of its complexity and scale, which has led to advancements in picture understanding, object recognition, and scene analysis.

Within this project, COCO had been used for the object detection models training, and therefore also the test images belong to it.

3.4.2 ILSVRC2012

In the field of computer vision, the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC12) represents a significant turning point. This major contest, organized by the ImageNet project, played a key role in promoting deep learning and object identification [26].

A large data set with millions of tagged photos covering more than a thousand object categories was presented at ILSVRC12. The difficulty was in designing algorithms that could precisely localize items in pictures while also identifying them. It paved the way for ground-breaking deep learning models like AlexNet, which showed off what deep neural networks were capable of when it comes to image classification.

In addition to inspiring extensive research and innovation, the ILSVRC12 competition was an important driver in igniting the deep learning revolution. Since then, the methods and models created for this competition have evolved into the fundamental building blocks for a variety of computer vision applications, ranging from self-driving cars to facial recognition.

Within this project, ILSVRC2012 had been used for the image classification models training, and therefore also the test images belong to it.

3.4.3 PASCAL VOC2012

An important resource in the field of computer vision is the Pascal VOC 2012 dataset, also known as just "VOC2012." This dataset includes 20 object categories and is designed for object recognition, detection, and semantic segmentation [27]. These categories range from ordinary objects like bicycles and birds to animals, cars, and indoor items. The meticulous annotations in VOC2012, which include object category names, exact bounding box coordinates for object localisation, and pixel-level segmentation masks for each item instance within the images, are what make it stand out from other datasets. With the help of these detailed annotations, a

variety of computer vision tasks are made possible, and VOC2012 has become an established standard for analyzing and improving object detection and segmentation algorithms.

Researchers and professionals have used its features all over the world to create and compare cutting-edge models, pushing the boundaries of object recognition and semantic segmentation. Despite its age, VOC2012 is still regarded as an institution of computer vision research, having made a lasting impact on the field, and providing the basis for further datasets and benchmark assessments.

Within this project, VOC2012 had been used for the semantic segmentation models training, and therefore also the test images belong to it.

3.5 Co-Processing Methods

According to the research questions, and therefore the will of evaluating the performances of the system, the co-processing methods to be implemented are extremely important, and answer directly to the sub-question presented in section 1.1 .

First of all, in order to have a comparison to *single processing*, it is useful to have experiments also with only one of the devices alone at a time. Given that the co-processing setup should improve the performances in any case, this could be used as a comparison ground to analyze the results and see how much the speedup and the accuracy increase/decrease are.

Considering the current state of the art investigated in section 2.6, some co-processing methods have been chosen to evaluate the double processors setup. Starting from the basics, *simple parallel case* with an AI model running on both devices in parallel has to be considered to see what the improvement on the throughput is and/or if there is any difference in accuracy.

The next step is what should be the most interesting section of this thesis: the implementation of co-processing techniques such as *horizontal and vertical partition-*

ing.

Regarding the former, the idea is to run the model, with lower input size, in parallel on both the devices, and ideally this could be exploited to even more devices if they were available. The main focus in this case is on how to split the input data, which essentially is, for what regards this project, a set of test images. Depending on how the image gets cut into tiles, which could have different shapes and sizes, some tasks may react in different ways, achieving better or worse results. For example, it would probably be expectable to see semantic segmentation not affected at all by the split, because of its nature. In fact, segmentation models work more on a pixel level, and divide the image into regions based on the features detected, which we would expect to be independent (up to a certain extent) from the integrity of the full picture. Taking into consideration object detection and image classification instead, splitting the original image into tiles may prevent to identificate objects because of them being cut in smaller pieces, and that is why another characteristic of really important consideration is the overlap factor. Ideally, it should be high enough so that the objects which need to be detected fit inside it. Only in this way you can be sure no detections are lost while applying horizontal partitioning, but it is not always easy to obtain. For instance, if the object is exactly at the center of the input image and its dimensions are quite large, then it might be difficult to get a precise detection, since every tile is going to have only a slice of the object. Therefore, the overlap factor is an important parameter that will be investigated in section 5.

In case of the latter, the problem gets shifted from the input data format to the models themselves, which increases even more the problem complexity. In fact, instead of having a single challenge to face, when considering vertical partitioning a unique solution that optimizes the run of every model does not exist by definition, even for models sharing the same task. Finding a solution to be applied to all of

them would not give optimal results and would be less meaningful, therefore, due to strict development time constraints, the idea is to implement this technique only within a single chosen model (so only one task). The initial idea is to check the architectures of all the models, and see which one offers a less difficult architecture to split, while which task it refers to is not relevant.

In fact, the challenging part is studying its layers' dependencies, and therefore if there are some that can be deployed in parallel. Something that needs to be considered as well is the communication overhead introduced by interrupting a neural network in the middle of its workflow. In fact, while processing the input image data, a deep learning model computes an high numbers of parameters in order to detect features, and therefore there is the need to transmit also all these mentioned values that the single layers output. This huge amount of data to be transferred from one device to another could result in a communication bottleneck, which is something to avoid.

Consequently, a first approach thought with the consultancy of NLR is to split the neural network right after a compression layer, or in some point of the processing in which the data to be transferred is reduced to the minimum possible, so that the communication latency is as low as possible.

Moreover, the strong dependencies and sequentiality some layers have still remains the hardest challenge in order to successfully apply the vertical partitioning technique.

3.6 Benchmarks

Of important consideration while evaluating the performances of a system is the benchmarks choice in order to achieve the goal. The challenge consists in finding meaningful ones, which are general enough to be used for comparison with other systems which execute similar tasks and work in a similar context.

In this project, the use case is embedded AI deployment, and therefore the benchmarks refer to deep learning models. For every task there are different techniques to record the efficiency.

Furthermore, latencies are always interesting measurements to record, because they give an idea of the actual throughput of the system.

Even though power consumption would be a meaningful metric to be considered, especially because of development for a space application, for what regards this thesis it is not of interest. It is given as granted that AI applications demand quite a high amount of energy, and the focus of this project is more about the computational capability and reachable accuracy by the system, rather than optimizing its consumption.

3.6.1 Latency

In this merit, the idea is to be able to have performances at system level, which essentially means computing the throughput capability of the system in full regime. This can be achieved by recording the inference times, to get an average value which can be transformed into a rate.

In reality, there are other times which need to be considered, such as pre-processing, post-processing and inter-communication among the devices. Considering the communication times as negligible because quite small in comparison with the other values, the average total time for a single image can be computed like shown below. Considering three vectors of N images, containing pre-processing, inference and post-processing times:

$$t_{pre_i}, t_{inf_i}, t_{post_i}$$

with *i=0...N-1* for all of the elements. By summing up all the pre-processing, inference, and post-processing times among them, and then all together, a vector

containing the total processing time can be obtained:

$$t_{tot} = \sum_{i=0}^{N-1} t_{pre_i} + \sum_{i=0}^{N-1} t_{inf_i} + \sum_{i=0}^{N-1} t_{post_i}$$

Dividing by N allows to compute the average time to deploy the deep learning model for a single image, while the frame rate is just the inverted value. Note that times are stored in *ms*, therefore to get Frame-Per-Second (FPS):

$$t_{avg} = \frac{t_{tot}}{N}, rate = \frac{1000}{t_{avg}}$$

It is also interesting to compute the rate in terms of Megabyte-Per-Second (MB/s), which is easily computed considering a vector containing all the input images sizes in kilobytes (KB), called *size*:

$$rate_{MB/s} = \frac{\frac{\sum_{i=0}^{N-1} size_i}{1000}}{\frac{t_{tot}}{1000}} = \frac{\sum_{i=0}^{N-1} size_i}{t_{tot}}$$

Note: dividing by 1000 in both cases is necessary to convert KB in MB, considering the file sizes, and ms to s, considering the times.

Regarding techniques such as horizontal partitioning, in which there is the need of additional processing to merge the different image tiles results. For these latencies calculations, these formulas are still valid considering the mentioned additional times included in the post-processing times.

More in general, these measurements allow to identificate the slowest parts within the entire inference process, and therefore take action in trying to optimize them.

3.6.2 Accuracy

In the context of accuracy metrics for deep learning models, there are different ones for every chosen task.

Regarding **object detection**, the most common metric is the mean Average Precision (mAP), which provides a comprehensive assessment of a model's accuracy. mAP is based on two other concepts: precision and recall.

Precision is a metric that shows how well the model avoids false positives. It is calculated as the ratio of true positives (correctly identified objects) to the total number of positive detections, which includes both true positives and false positives. The precision formula is shown below:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

In other words, precision assesses the proportion of predicted object instances that are actually correct.

Recall, on the other hand, investigates the model's ability to find all relevant objects in the image. It is computed as the ratio of true positive detections to the total number of actual objects in the ground-truth data, which includes both true positives and false negatives. The recall formula is represented as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Shortly, recall measures the percentage of actual objects that were successfully detected by the model.

mAP combines precision and recall providing a comprehensive evaluation of object detection performance across various object categories. It involves calculating the *Average Precision (AP)* for each object category, which is essentially the area under the *Precision-Recall curve* for that category [28].

The Precision-Recall curve is obtained by varying the confidence threshold for object detections and recording precision and recall values at each threshold.

The mAP is then computed as the mean of the AP values across all object categories. Considering N as the total number of object categories, the formula:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i$$

mAP takes into account both the precision, which measures accuracy, and recall, which measures completeness, to provide a sort of summary metric for object detection.

Regarding **image classification**, several metrics are commonly used to assess the performance of machine learning models. These metrics are essential for quantifying how effectively a model can correctly classify objects within images [29].

Top-1 Accuracy is a fundamental one, measuring the proportion of images for which the model's highest-confidence prediction matches the actual class label [30]. *Top-5 Accuracy* instead, considers whether the correct class label is among the top five model predictions, making it more flexible when objects may have multiple plausible labels. In general, accuracy can be computed as shown:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Other significant metrics are *precision*, *recall* and *F1-score*, which have already been described above and can be computed with the same formulas.

These image classification accuracy metrics collectively offer a comprehensive assessment of a model's performance. While top-1 and top-5 accuracy provide insights into the model's ability to make accurate predictions, precision, recall, and the F1-score offer a more nuanced evaluation, considering both correct and incorrect predictions, and how well the model captures specific classes within the dataset.

In the domain of **semantic segmentation** instead, where the goal is to classify each image pixel into a specific class label, various metrics are commonly used.

The first one to be mentioned is *pixel accuracy*, which measures the overall correctness of the segmentation model pixel-by-pixel. It calculates the percentage of correctly classified ones in the entire image. While useful for assessing global ac-

curacy, it may not capture the implications for class-specific performances [31]. To get the single-class behaviour, the pixel-wise approach has to be exploited considering one single class each time. The formula for computing pixel accuracy is shown below:

$$\text{Pixel Accuracy} = \frac{\text{Number of Correctly Predicted Pixels}}{\text{Total Number of Pixels}}$$

Another important metric is the *Intersection Over Union (IoU)*, also known as Jaccard Index. It quantifies the overlap between predicted and ground truth segments, providing segmentation quality insights by assessing how well the model captures the object boundaries. The scale goes from zero to one, with higher IoU values indicating better boundary alignment (ideally, value one shows perfect match with the ground truth). It can be computed with the formula shown below, where both the prediction and ground truth are image masks related to a single class:

$$\text{Intersection over Union (IoU)} = \frac{\text{Prediction} \cap \text{Ground Truth}}{\text{Prediction} \cup \text{Ground Truth}}$$

These IoU values offer an analysis of how the model works per single class, and they can be averaged to extrapolate a more general model's behaviour evaluation.

Considering all of these metrics, a comprehensive understanding of a model's strengths and weaknesses in pixel-wise classification and objects boundaries delineation is provided.

4 Experiments

With respect to the planned experiments described in detail in section 3, some technical issues have been encountered, and therefore, due to the limited available time to pursue the thesis, the actual implementation differs from the original plan.

In this section, the actual setup and experiments that have been run are described in detail.

4.1 System Architecture

When considering the setup for the experiments, the primary factor to address is the system architecture. As initially planned, the comparison revolves around two devices operating in single processing mode and exploring various co-processing techniques.

Originally, the plan was to implement three modes of operation: simple parallel processing, horizontal partitioning, and vertical partitioning. Unfortunately, due to time constraints and technical issues related to models optimization (which are detailed in subsection 4.3), the third mode could not be implemented, contrary to the program. Some of the reasons to avoid deploying vertical partitioning for this thesis project are the method being not a really common solution and the significant additional effort required. Consequently, the decision was to exclude vertical partitioning from this project, choice which allowed a more focused approach to the other techniques and enhanced system stability.

4.1.1 Single Processing

The single processing setup is the simplest one, as it is the simplest and more direct implementation. The chosen device must have the trained model already loaded in its memory, together with the input data (in this case, a set of 200 images to be tested for inference).

The first step is the setup of the model TFLite interpreter, which can take the optimizations according to which devices is running onto, and then there is a loop in which the input images are read one-by-one, and the pre-process, inference and post-process chain is applied. Afterwards, the results are saved locally in files.

To summarize, the entire workflow is shown below in figure 4.1.2.

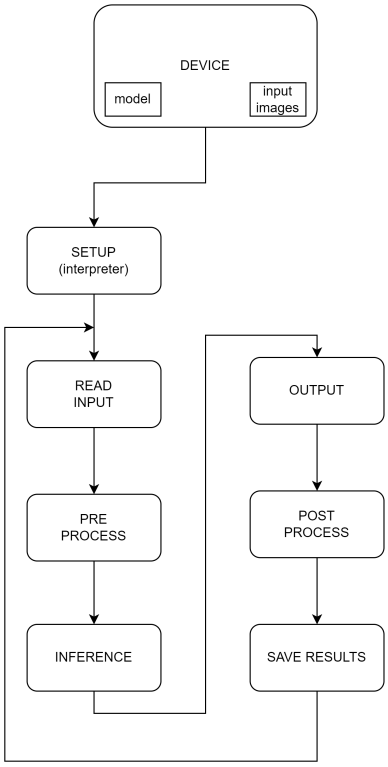

Figure 4.1: Sigle Processing System Architecture Overview

4.1.2 Simple Parallel

The architecture of the simple parallel co-processing method gets a bit more complicated, but it is still straight forward. With respect to single processing, both the devices are running concurrently and have the model already loaded in their memories, while the images are locally stored just in the controller memory.

The accelerator workflow consists in setting up the interpreter and the connection with the controller, and then it enters a loop in which it waits for input images from the controller. Every image gets processed with the chosen model's pre-process and inference, and then the results are sent back to the controller. In this case, the post-process is split among the two devices: for example, considering the object detection task, the accelerator takes care just of extracting the detections and their info from the raw model output, but it is the controller's duty to reconstruct the original image with bounding boxes after receiving the results.

Concurrently to the accelerator workflow, the controller starts with setting up as well, and then it executes the same workflow explained for the single processing mode in section 4.1.1.

The images are fetched from a local folder, stored in the controller memory, and multiprocessing is implemented in order to make the two processes read the first available image without repetitions, until there are available ones. Detailed explanations about the concurrency and the communication implementation are below, in section 4.2. This system ensures the implementation of the most optimal workload subdivision among the two devices, since they both fetch an image as soon as they are done with the previous one, avoiding idle state conditions.

To summarize the workflow of the entire system and make it more clear to the reader, a diagram is shown in figure 4.2.

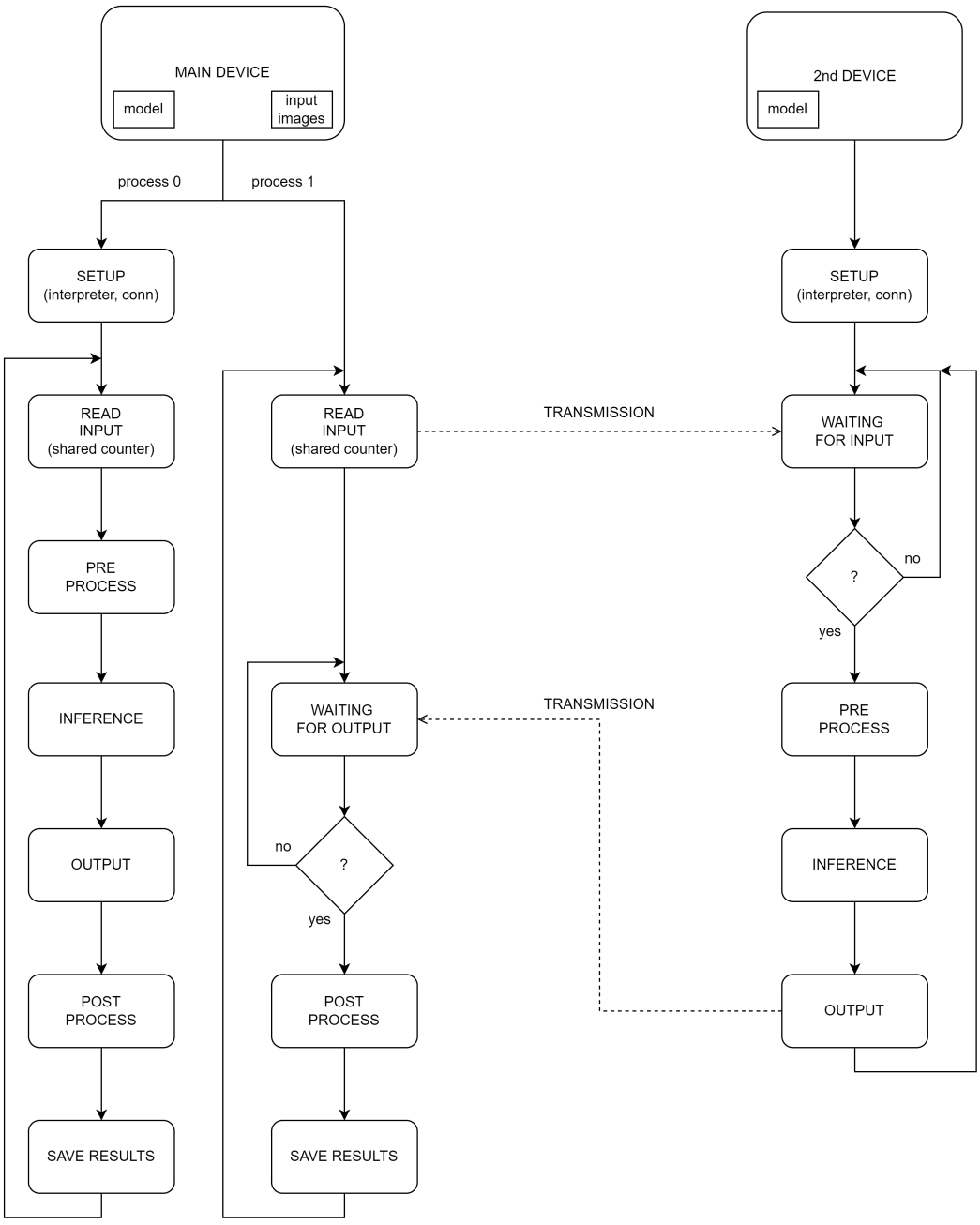

Figure 4.2: Simple Parallel System Architecture Overview

4.1.3 Horizontal Partitioning

The architecture used to implement the horizontal partitioning co-processing method closely resembles the simple processing setup, but it presents some additional steps.

First of all, the controller takes all the images from the folder and it splits them into tiles, with details explained in the following subsection 4.1.3. Therefore, the two processes fetch the single tiles instead of fetching the entire images.

Differently from single parallel mode, with this technique the results of every inference are just part of the results for a single image, therefore all the results are stored according to the original image index. Consequently, once all the tiles have been elaborated the controller has the duty to take into account all the results and for every image it has to merge the single tiles results.

This operation differs from task to task, and it may be extremely computationally heavy for some cases, as described in section 4.1.3.

Image Split

An image can be split into tiles in several different ways, which can affect the predictions of deep learning models, especially when considering tasks whether objects need to be classified or detected. In fact, partitioning the image in tiles might also split the objects that need to be recognized in the picture, making it more challenging for the model. This is why the approach used for splitting the image is relevant in the consideration of the accuracy results.

Considering some limitations regarding chosen AI models input sizes, within this project only one configuration for splitting the images has been used (more details about the reasons in section 4.3), and it consists in splitting the images in four parts.

Before getting split, every image is made squared without modifying the aspect ratio, but adding overlaps to the lowest dimension, and consequently resized to the expected size (if needed). Then, four squared tiles are generated: for instance, most

of the images for object detection are 640x640, therefore the tiles would be 320x320. Additionally though, a key element for the success of the horizontal partitioning technique is the overlap between the tiles. It allows to avoid cutting in too small part objects that need to be recognized and improves the results, even though it introduces some redundancy in the elaboration, increasing the total elaboration time.

For this thesis, the overlap factor is an interesting parameter which might change the final results in terms of accuracy. The experiments have been run with overlap that goes from 10% up to 30% of the initial squared image pixels. In particular, for object detection images are 640x640, hence these sizes are 32, 64, and 96 pixels, while for the other two tasks images are 500x500, therefore they are 25, 50 and 75 pixels.

The way of splitting the images in four tiles is represented down here in figure 4.3, where the overlapped parts are represented by the four strips and the central square. For every strip, every pixel is shared by two tiles, while the overlapped square is shared by all the tiles.

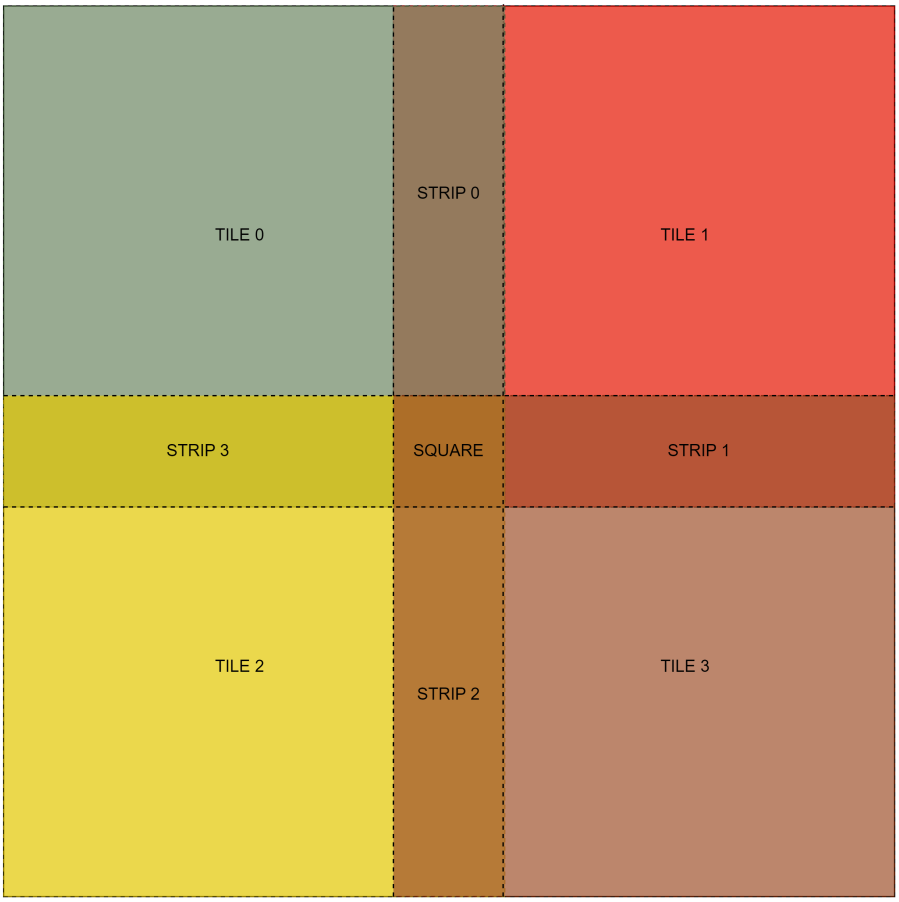

Figure 4.3: Image Split into Tiles

As an example of how the partitioning algorithm works, here below an image belonging to the COCO test set used for the experiments is shown in figure 4.4.

Figure 4.4: Original COCO Image

And consequently, the tiles generated from the above image are shown in figure 4.5. The visible black bands on top and at the bottom are present because of the squaring function, that is described earlier.

Figure 4.5: COCO Image Tiles

Results Merge Operation

Within the horizontal partitioning techniques, the collected results from the devices are related to single tiles of the images. Therefore, there is the need to merge them to get the full pictures ones, to evaluate them. This operation varies depending on the task.

For object detection, first there is the need of repositioning the detected objects bounding boxes, and then to keep only the meaningful ones. In fact, an object could be recognized from multiple tiles resulting in multiple overlapping bounding boxes, but in the end, it is only one that needs to be shown. Furthermore, some objects might be recognized as different items in the different tiles, so the lowest score prediction has to be discarded.

For image classification, the merge operation is simpler, since there is no need of localizing the classified objects, but they just need to be ordered by scores.

For semantic segmentation, the output of the model is a mask of the image at pixel level, meaning that to merge the results a single image mask is created by unifying the four different tiles. In this process, the four stripes and the square

are crucial parts of the process since they require an algorithm to assign the right prediction to each pixel. The chosen approach for this research is the simplest possible: the pixel gets the label of the prediction that appears more than once within the tiles predictions. This means that for the strips both the tiles agree, while for the square that at least two of the tiles agree (50%). In case this does not happen, the solution thought for this project is a random choice among the predictions, but some other approaches would surely work better. For instance, locality principle could be added to introduce some spatial memory, but considering the purpose of this thesis and the time constraints, a random choice works fine.

As mentioned already in the previous section, merging the results of tasks like semantic segmentation could be computationally really heavy, since the logic is a loop over every element of a 2D array, which could have quite high dimensions. In this thesis, it has been optimized through the use of the Numpy library, but it is something that could still be improved very much (details in section 6.4).

Examples of horizontal partitioning merged results are presented below in figure 4.6. There is one example image for each task, showing positive results cases.

Figure 4.6: Horizontal Partitioning Merge Results Examples

It is also meaningful to show cases where the inference did not give great results, and an example of this for object detection is shown in figure 4.7 below.

Figure 4.7: Horizontal Partitioning Object Detection Bad Merge Results Example

An example of not great results for semantic segmentation is shown below in figure 4.8, obtained because of the random choice of labels while considering the overlapped sections in the results merge function. As it is clear from the picture, the overlapped areas present labels with 'noise' of black pixels, set by the random choice of the algorithm.

Figure 4.8: Horizontal Partitioning Semantic Segmentation Bad Merge Results Example

4.2 Inter-Device Communication

The initial idea of using two AI accelerators modules in the architecture was not possible due to the lack of available products in the producers' shop. Therefore, only a single accelerator has been used, with its starter kit board. Hence, the fastest setup to have for communication was through a server-client system over TCP protocol, via LAN network.

Of course, having a more reliable and stable communication method such as Controller Area Network (CAN), I2C or SPI would increase the system performances, since they would not be affected by latency or other effects such as interference and signal attenuation, which are already not causing major problems because of the wired connection.

The transmission times are recorded and taken out of the throughput computations, addressing this communication stability improvement as possible future work, as described in section 6.4.

Considering instead the concurrency of device inter-communication and inference process on the controller, it has been achieved through the use of multiprocessing. In

fact, the framework device has two general processors that can run the two processes in parallel, while using multithreading is not as effective in Python because of the Global Interpreter Lock (GIL). Hence, the main process runs the local inference, and shares a counter variable with the second process, which handles the communication with the accelerator: transmission of the input image and reception of the results. To ensure a safe run and avoid concurrent accesses to the shared variable, a simple semaphore is implemented (mutex). This way, deadlock cannot happen.

4.3 AI Models

With respect to the planned set of models to be used for the experiments, there have been some changes due to time constraints. The idea of selecting models pre-trained to run on the accelerator with optimization and re-optimize their root to efficiently run on the controller as well is still a solid approach to face this type of challenges, but it could not be performed for this thesis. The main issue was within the controller software framework developed by the manufacturer to compile deep learning models and create delegates, which essentially schedule the operations to the hardware accelerators available in the board, significantly increasing the speed.

Unfortunately, there were problems with the setup of this framework and with running the script to optimize the models. Furthermore, the functioning of the optimized models also was not improving in some cases, and due to lack of time there has been the need to change models, in order to concentrate on the real focus of this project. Hence, the final approach has been to choose pre-trained controller models belonging to the same family of NN architectures (when possible) or running the same models of the accelerator but without optimization.

Another important consideration regards the models' input sizes. In fact, to achieve better time results within the deployment of the horizontal partitioning technique, the models' input sizes should be decreased to the tiles dimensions, in

order to make the process faster. However, since the focus of this project is not on developing deep learning models and just takes advantage of pre-trained ones to test co-processing methods, this change has not been possible. Therefore, it has to be considered during the timing analysis in section 5.

To summarize, these are the chosen models for the boards:

- Object Detection
  - SSD MobileNetV1 COCO17 MLPerf 300x300 ***(Controller)***
  - SSD MobileNetV2 COCO17 MLPerf COCO17 300x300 ***(Controller)***
  - SSD MobileNetV2 COCO17 Quant PostProcess 300x300 ***(Accelerator)***
  - SSD MobileNetV2 COCO17 PTQ 300x300 ***(Accelerator)***
- Image Classification
  - EfficientNet S Quant ILSVRC2012 224x224 ***(Both devices)***
  - MobileNetV2 1.0 PTQ ILSVRC2012 224x224 ***(both devices)***
- Semantic Segmentation
  - DeepLabv3 MNV2 DM05 PASCAL-VOC2012 Quant ***(Both devices)***
  - DeepLabv3 MNV2 PASCAL-VOC2012 Quant ***(Both devices)***

For simplicity, in section 5 the models will be referred with indexes such as 1 and 2, 3 and 4, and 5 and 6 for the corresponding two configurations of each task, in order object detection, image classification and semantic segmentation.

4.3.1 Datasets

With respect to the initial plan of using the official test set of each dataset, it has been difficult to retrieve the correspondent annotations to evaluate the predictions

against the ground truth. Therefore, the choice has been to create a test subset from the validation set, since the annotations are easily accessible.

The image selection has been made randomly, in order to avoid a significant imbalance in the distribution of different class labels within the experiments input.

4.4 Benchmarks

Regarding benchmarks, several different metrics were available to evaluate the models performances, as explained in section 3.6. In this section, the selected metrics for every task are listed.

4.4.1 Latency

Considering latency, all the metrics considered in section 3.6.1 have been implemented in order to evaluate the overall performances of the system, but also to consider the weights that every part of the process has in the entire chain. This was done for all the three tasks investigated.

Regarding the specific benchmarks' metrics chosen, they all are explained in detail in section 3.6.2.

4.4.2 Accuracy

Considering **object detection**, the initial idea was to use the pycocotools python API, which is an official library from COCO dataset to evaluate the mAP of a model, given the predictions and the annotations [25]. Unfortunately, there were some issues with the functioning of this library, and therefore the choice was to compute metrics manually. Hence, for object detection three benchmarks' metrics have been implemented: *precision*, *recall* and IoU.

For the other two tasks, the approach has been to manually implement the chosen relevant metrics, in the same way that has been done for **object detection**.

Considering **image classification**, the computed metrics are: *top-1 accuracy*, *top-5 accuracy*, *precision*, *recall* and *f1-score*.

Considering **semantic segmentation** instead, the calculated metrics are: *pixel accuracy*, *precision*, *recall*, *f1-score* and IoU.

5 Results and Analysis

In this section, the achieved results of this project are presented and discussed. The approach of analysis is task-by-task, since it is not relevant to compute performance metrics among different tasks.

5.1 Object Detection

In this section, the object detection results are presented.

5.1.1 Latency

Considering timings, first results to analyze are regarding the pre-process, inference and post-process times. For both the models, similar considerations can be exposed.

In every configuration, it is clear that if the model is optimized on both the devices, then the controller is way faster in every step of the inference process, and that is constant for every configuration. Through the different setups, consistency can be noted, apart from the post-process times for the controller in horizontal partitioning configuration, which has a very high standard deviation. This means the times have a high degree of variability, probably due to the different amount of detected objects per tile.

The above mentioned pre-process, inference and post-process times for every configuration are shown in figure 5.1.

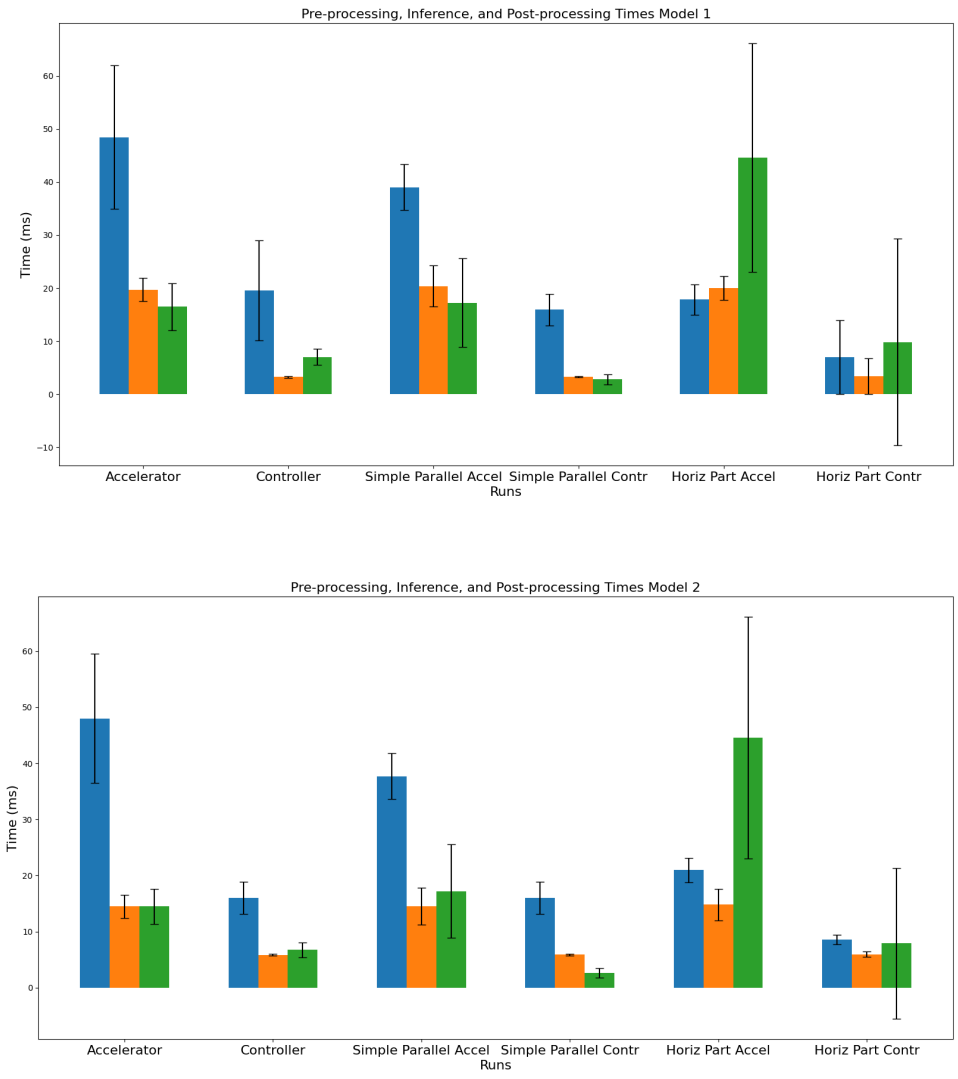

Figure 5.1: Object Detection Pre-Process, Inference and Post-Process times

Focusing on the horizontal partitioning technique, meaningful to observe are the merge times with different percentages of overlap. As expected, the time increases with the spatial area of overlap of the tiles, as shown in figure 5.2

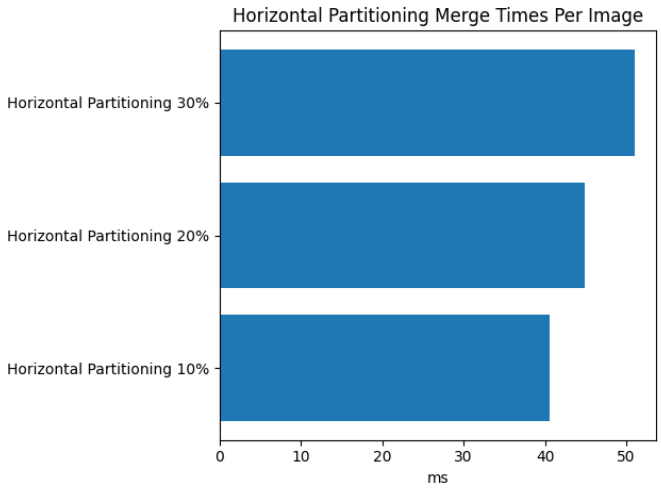

Figure 5.2: Object Detection Horizontal Partitioning Merge Times

It is also worth mentioning the transmission (TX) times for the two configurations which need inter-device communication. Nevertheless, they are not important for the following considerations, since they can be drastically diminished switching to wired communication, or directly placing both the devices on a single board.

As it is shown in figure 5.3, TX times for the accelerators are around 10ms more (on average), and that is because to stabilize the socket-server system some manual delays had to be implemented in the loops.

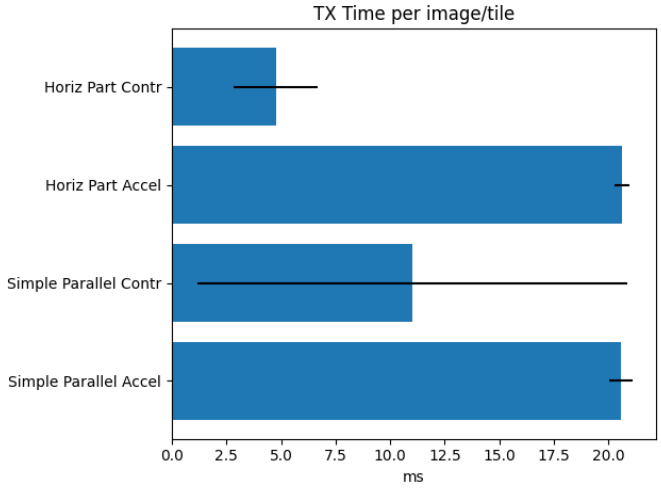

Figure 5.3: Object Detection TX Times

More interesting for the evaluation of the co-processing techniques are the average times that the system takes to elaborate a single image, which is also the inverse of the overall throughput. Again, the two models have a similar behaviour, so the same conclusions can be drawn.

As it is clear from figure 5.4, the times taken from the single processing mode with only the accelerator are way higher, while the other configurations are quite similar. The fastest configuration is the simple parallel one, but this is probably related to the fact that the horizontal partitioning technique has been implemented without changing the models' input sizes. Therefore, it is expected to not see a time improvement, since all of the tiles need to be resized to the original image size.

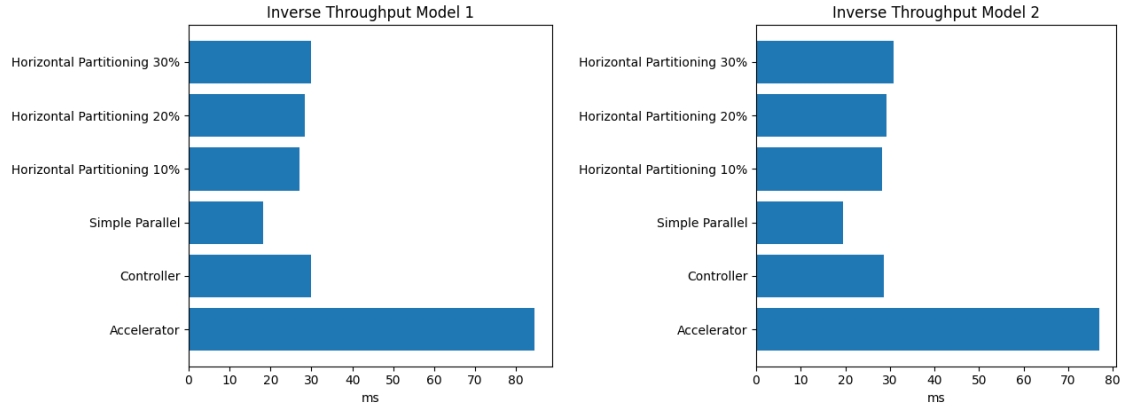

Figure 5.4: Object Detection Time per image

Looking at the throughput comparison shown in figure 5.5, the same trend is even clearer. Among the horizontal partitioning configurations, the smaller overlap comes with the higher rate, as expected.

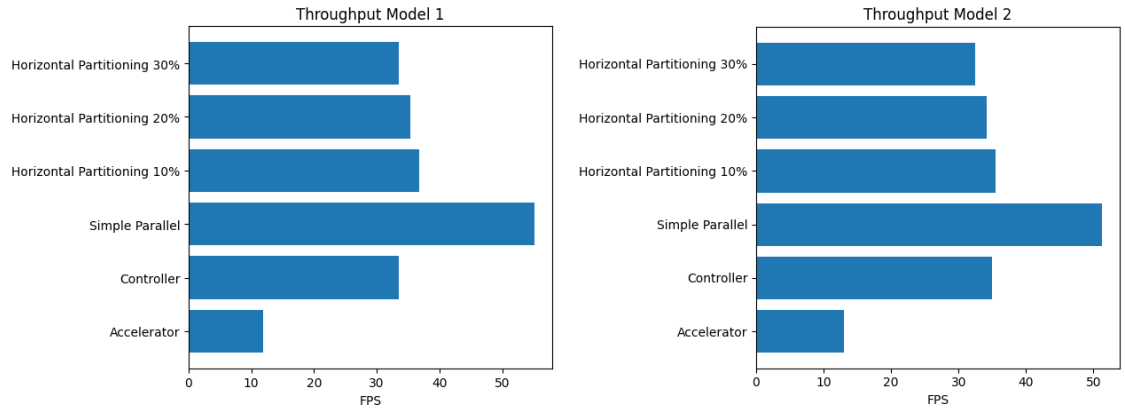

Figure 5.5: Object Detection Throughput

The same trend is visible in figure 5.6, which represents the data throughput for every configuration.

Note: horizontal partitioning values are estimation calculated multiplying by 4 the input images sizes.

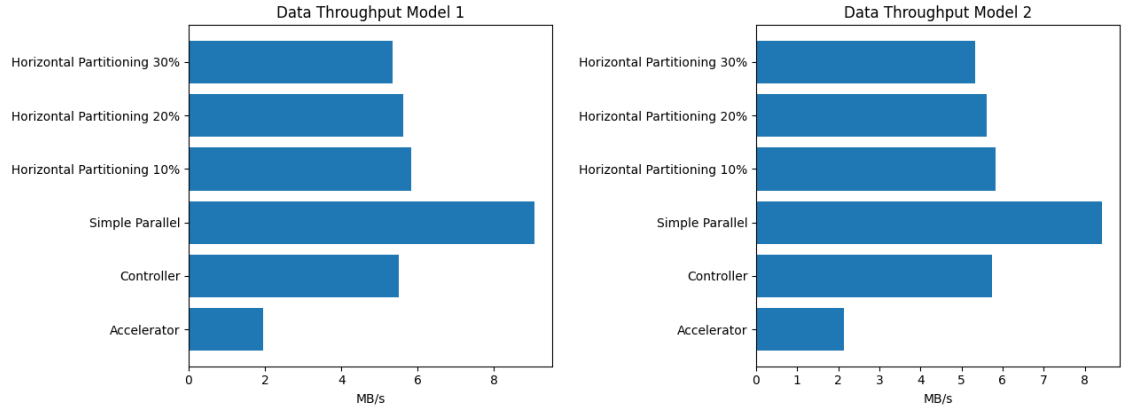

Figure 5.6: Object Detection Data Throughput

5.1.2 Accuracy

Given the consideration about the fixed models' input sizes, which make the latencies less meaningful to analyze, the accuracy results gain even more importance. In fact, an improvement or worsening of the accuracy is crucial when deciding whether to implement the method.

The precision-recall graph shown in figure 5.7 shows clearly how the accuracy drops when implementing horizontal partitioning, at least for object detection tasks. In fact, both recall and especially precision heavily drop when applying the mentioned co-processing technique, especially when considering model 1. In this case, single processing on the controller and simple parallel configuration are similar with good performances, while the single processing on the accelerator is the best configuration, especially for the really high recall.

Considering model 2, the accelerator is still the best configuration, with a way lower recall, but a higher precision.

These results are probably related to the fact that objects get split in smaller pieces and not recognized, considering that the merge results reconstructions maybe lack precision for the bounding box, and therefore evaluation metrics consider it as false positive.

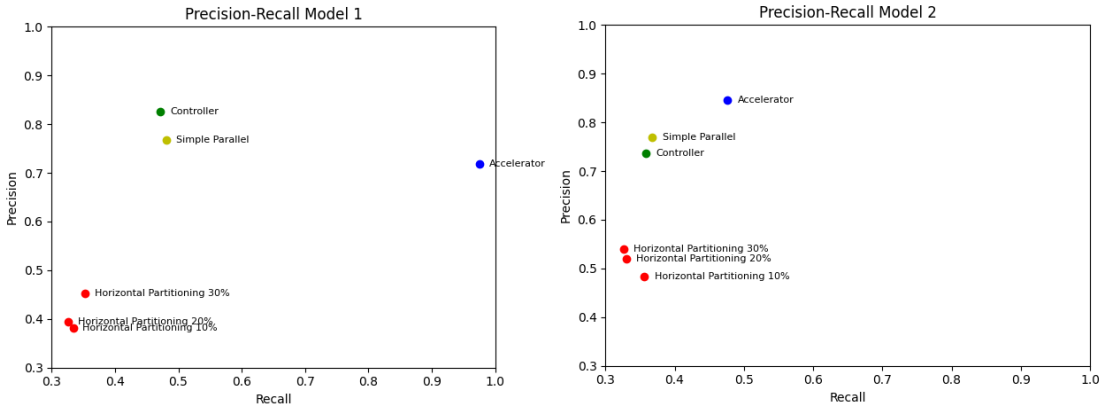

Figure 5.7: Object Detection Precision-Recall

The second implemented metric is the IoU: also for this one, results for model 1 and model 2 differ.

Considering the former, the average IoU value is higher for the single processing mode on the controller, followed by the simple parallel configuration and the single accelerator. Way lower is the IoU achieved with horizontal partitioning.

Considering the latter, the trend is similar, but the single accelerator configuration wins.

This mentioned data is shown in figure 5.8.

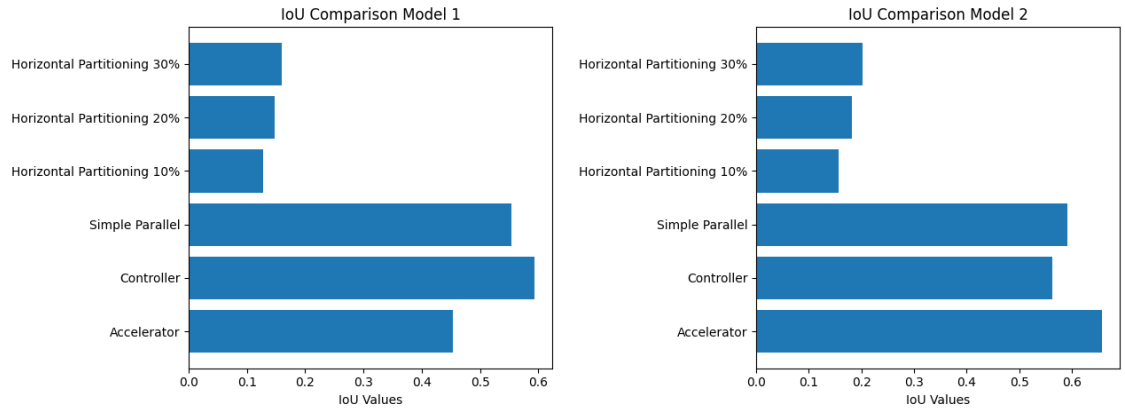

Figure 5.8: Object Detection IoU

5.2 Image Classification

In this section, the image classification results are presented.

5.2.1 Latency

Again, the first results to analyze are regarding the pre-process, inference and post-process times. For both the models, similar considerations can be exposed.

In every configuration, it is clear that since the model is not optimized to run on the controller, the correspondent inference time is extremely high, and that is constant for every configuration. Through the different setups, consistency can be noted, apart from the pre-process times for the accelerator in simple parallel configuration, which has a very high standard deviation. This means the times have an high degree of variability, therefore that data is not particularly meaningful.

The above mentioned pre-process, inference and post-process times for every configuration are shown in figure 5.9.

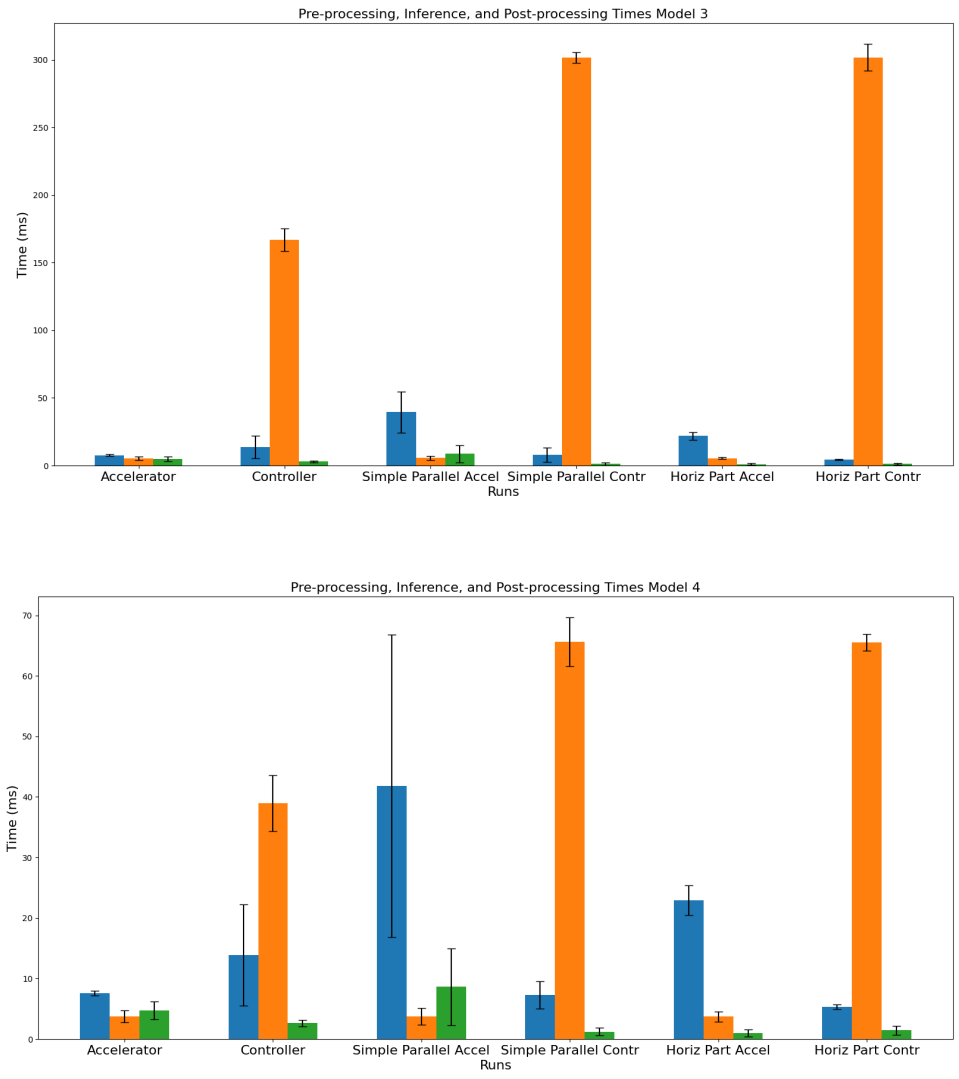

Figure 5.9: Image Classification Pre-Process, Inference and Post-Process times

Regarding the horizontal partitioning technique merge times, the same trend seen in section 5.1.1 is shown here, as it is expected and clear from figure 5.10.

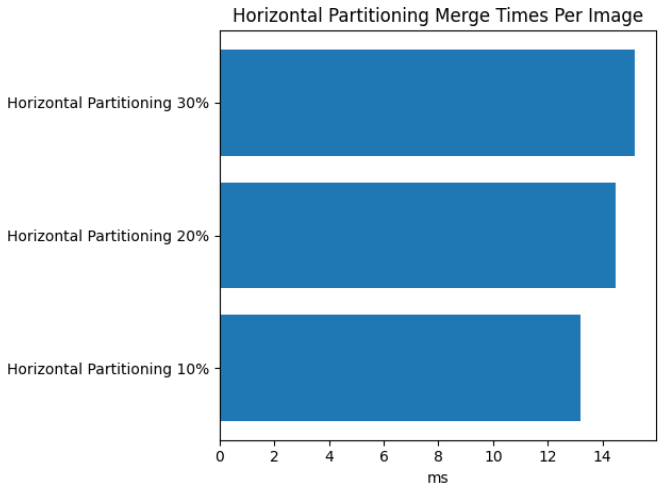

Figure 5.10: Image Classification Horizontal Partitioning Merge Times

Regarding TX times, the same consideration proposed for the object detection task can be done for image classification as well, as visible in figure 5.11.

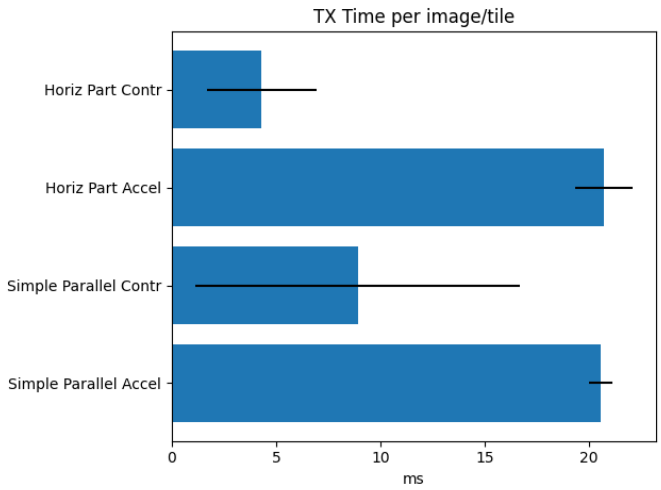

Figure 5.11: Image Classification TX Times

Considering the average times that the system takes to elaborate a single image, which is also the inverse of the overall throughput, it is clear how the controller is way slower than the accelerator. Consequently, also simple parallel and horizontal partitioning techniques result in being slower than the single processing mode with

the accelerator.

An interesting comparison is between the two models and regarding the co-processing techniques: all of them are heavily slower on while running model 4.

The mentioned results are shown in figure 5.12.

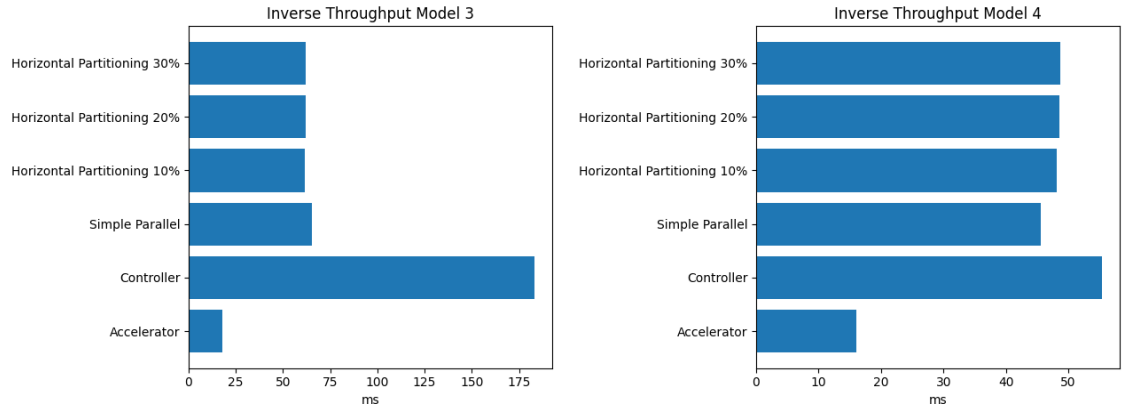

Figure 5.12: Image Classification Time per image

The throughput data, being the inverse of the time discussed above, follows the reversed trend, with the single processing accelerator mode having the highest rate for both the models. The co-processing methods' performances are unfortunately limited by the low rate achieved by the controller.

These results are visible below in figure 5.13.

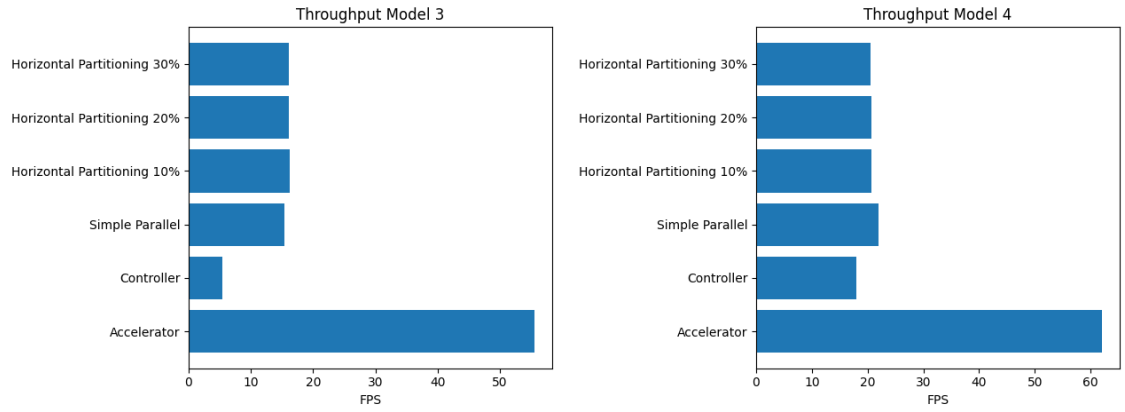

Figure 5.13: Image Classification Throughput

The same trend can be seen in the data throughput graph, which is shown below in figure 5.14.

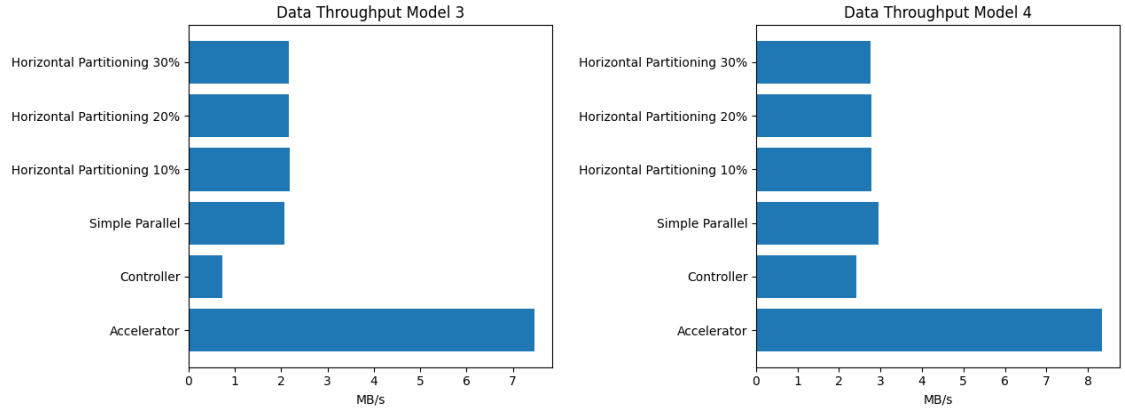

Figure 5.14: Image Classification Data Throughput

5.2.2 Accuracy

The same consideration about the importance of the accuracy results in this project made in section 5.1.2 is valid also here.

Considering the precision-recall graph for image classification, the two models

have quite distinguished outcomes. Starting from model 3 results, they show how the single processing mode with the accelerator has both higher precision and higher recall, while the other co-processing techniques still have decent values of precision and recall, but they are limited by the mediocrity of the model on the controller.

Considering model 4 instead, the controller's performances heavily increase, and therefore the horizontal partitioning techniques gain both in precision and recall, with really interesting performances. All of these results can be seen in figure 5.15.

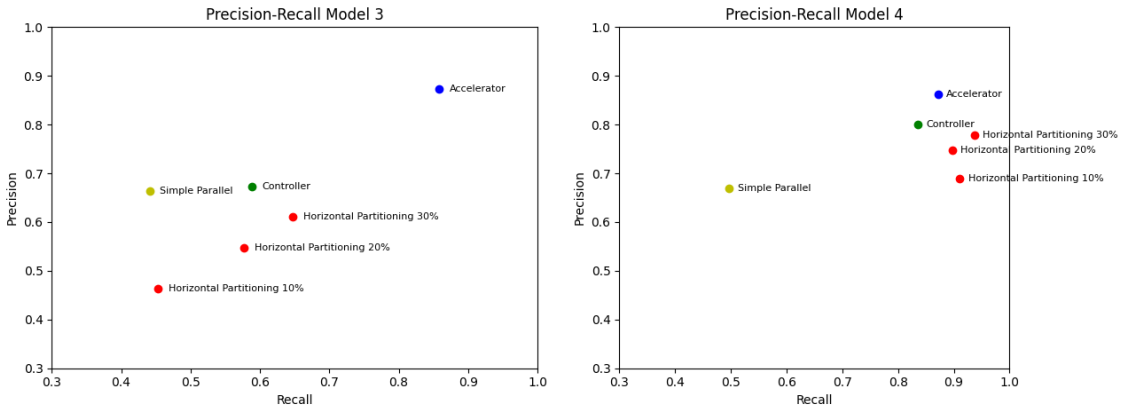

Figure 5.15: Image Classification Precision-Recall

The F1-score, being computed from precision and recall, follows the same trend, as shown in figure 5.16 below.

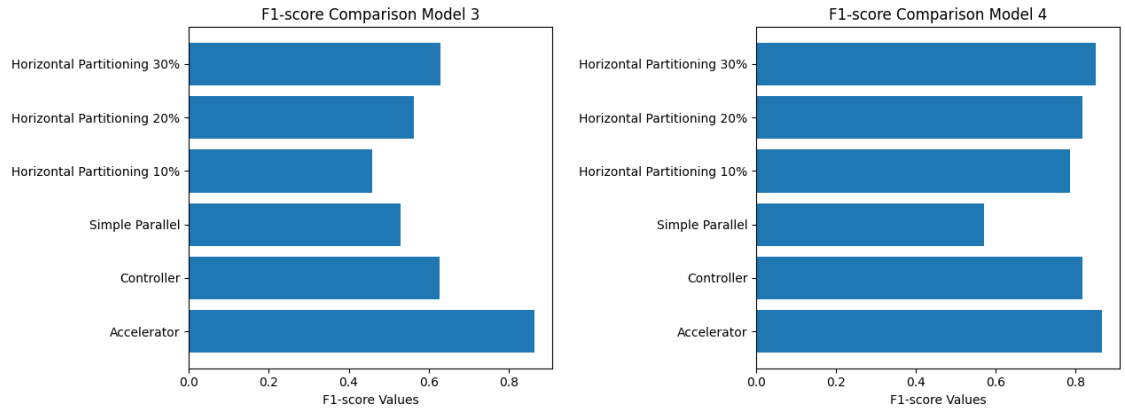

Figure 5.16: Image Classification F1-score

Other important metrics are the top-1 and top-5 accuracy, which actually show how model 4 has interesting performances when using horizontal partitioning co-processing methods, as well as the two devices running in single processing mode.

For model 3 instead, the controller can be considered as the bottleneck, also for the co-processing techniques. These results are visible in figure 5.17, which contains top-1 accuracy values, and in figure 5.18, which shows top-5 accuracy values.

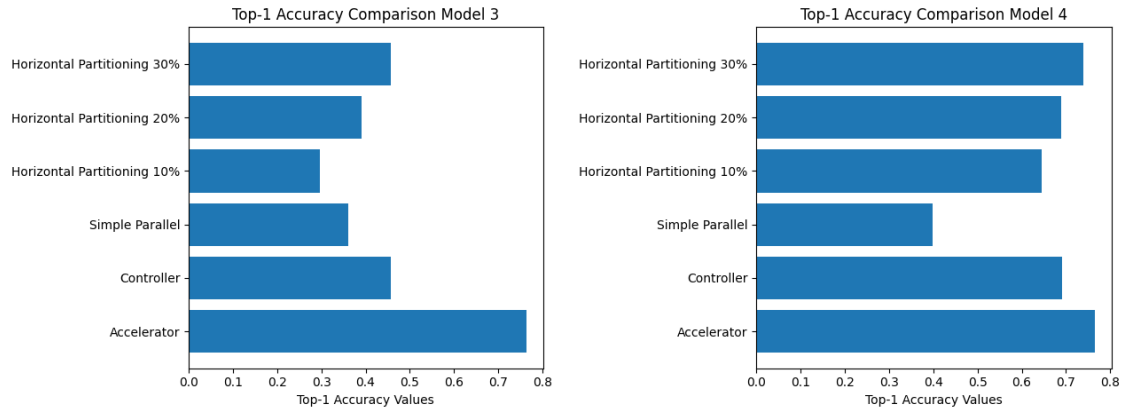

Figure 5.17: Image Classification Top-1 Accuracy

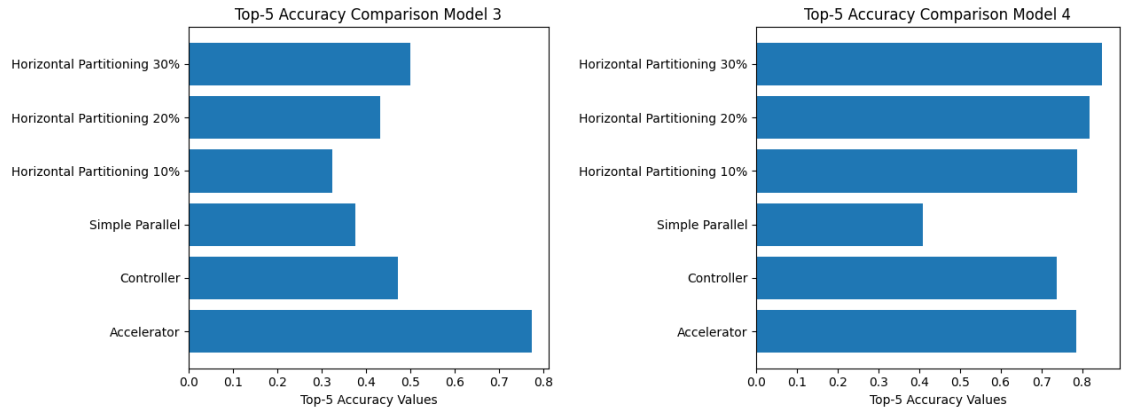

Figure 5.18: Image Classification Top-5 Accuracy

5.3 Semantic Segmentation

5.3.1 Latency

Again, the first results to analyze are regarding the pre-process, inference and post-process times. For both the models, similar considerations can be exposed.

Considering semantic segmentation, both models have not great time performances, even if they had been optimized for the accelerator (the models are large). Therefore, differently from the other tasks there is just a slight difference between the two devices, making the co-processing methods more interesting also time-wise.

Interesting characteristic which is easily noticeable from the graph in figure 5.19 is that post-process times are huge when using horizontal partitioning, and that is because of the slow pixel-wise operation the models' output need to have.

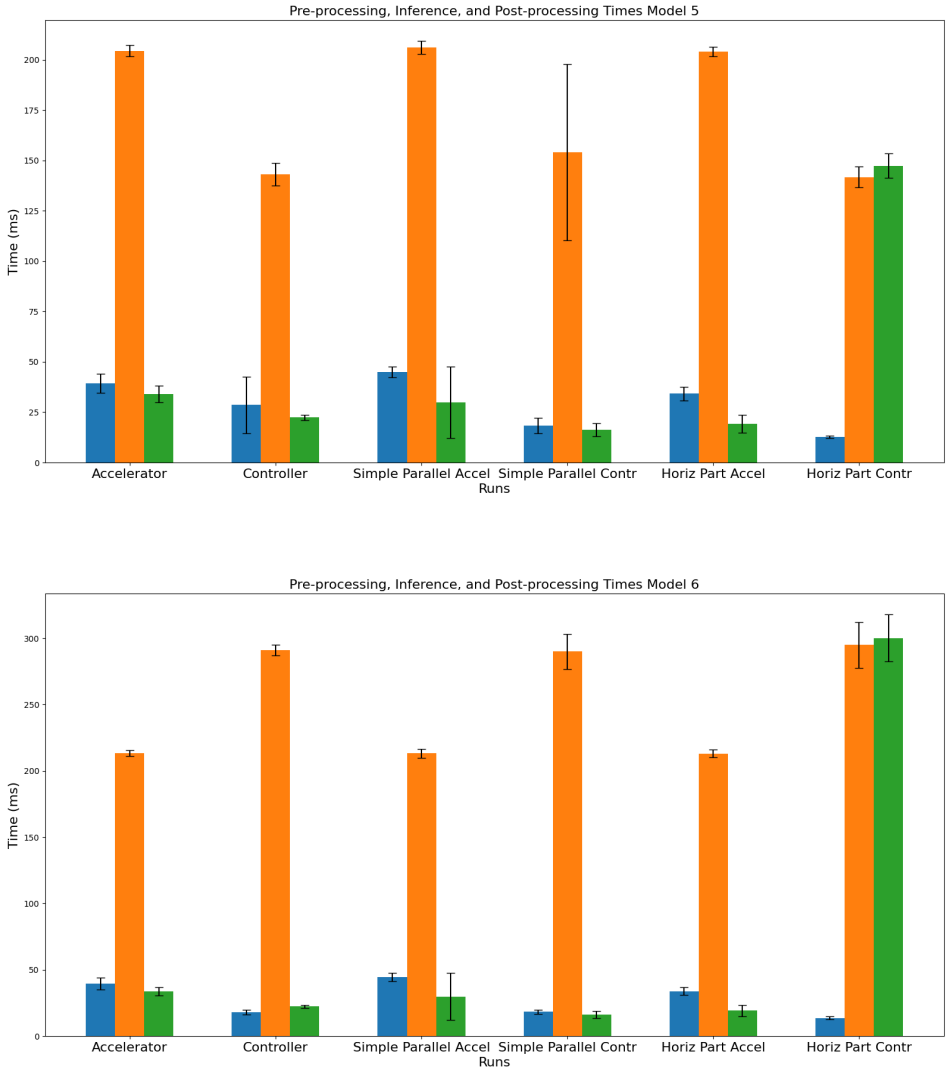

Figure 5.19: Semantic Segmentation Pre-Process, Inference and Post-Process times

Regarding the horizontal partitioning technique merge times, the same trend seen in section 5.1.1 is shown here, as it is expected and clear from figure 5.20. However, for semantic segmentation the operations are extremely computationally heavy, and that is mostly because of the overlapped sections, which need to be examined pixel-wise.

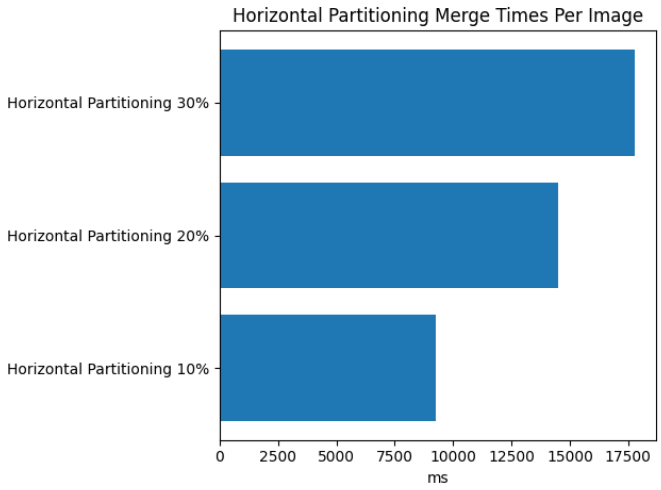

Figure 5.20: Semantic Segmentation Horizontal Partitioning Merge Times

Regarding TX times, shown below in figure 5.21, it is noticeable how another bottleneck is the results transfer from the accelerator to the controller. The reason behind this behaviour is the models' raw outcome nature, which is a 2D array of the image size, with a high data size.

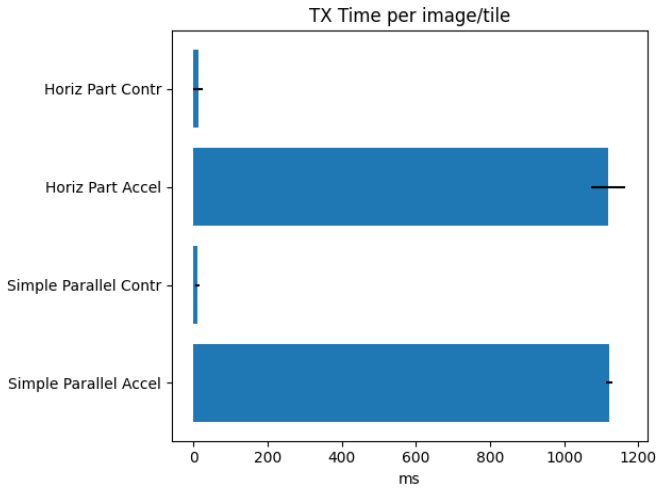

Figure 5.21: Semantic Segmentation TX Times

Considering the inverse of the throughput, it is clear that the horizontal partitioning is way slower than the other methods, and that is mainly because of the

heavy post-process and merge needed. The graph in figure 5.22 shows the results for both the models.

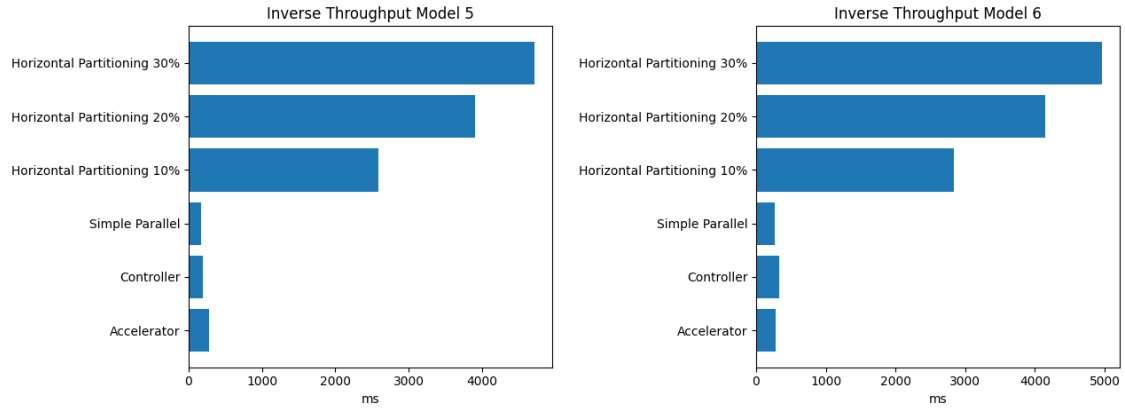

Figure 5.22: Semantic Segmentation Time per image

Looking at figure 5.23 below, the throughput follows the trend extrapolated from the previous observed data. It shows how simple parallel mode brings some small improvements in terms of timings.

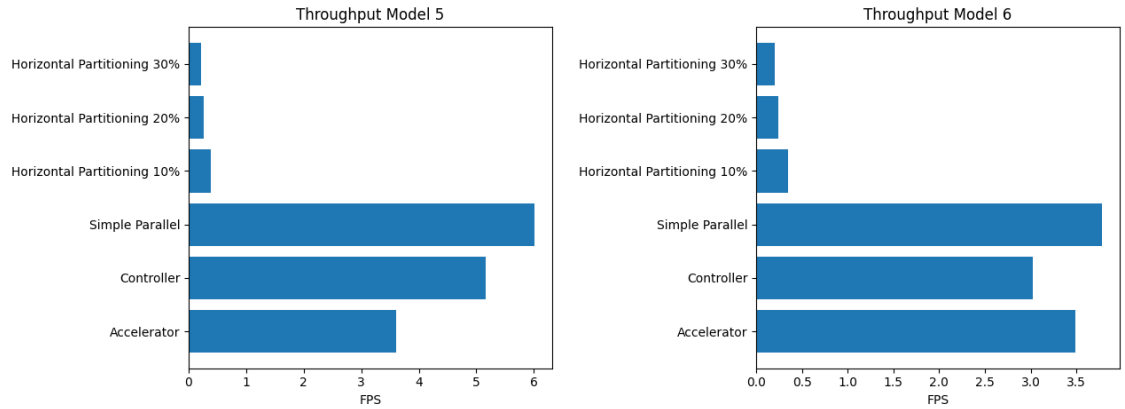

Figure 5.23: Semantic Segmentation Throughput

The data throughput graph shown below in figure 5.24 is consistent with the previous ones.

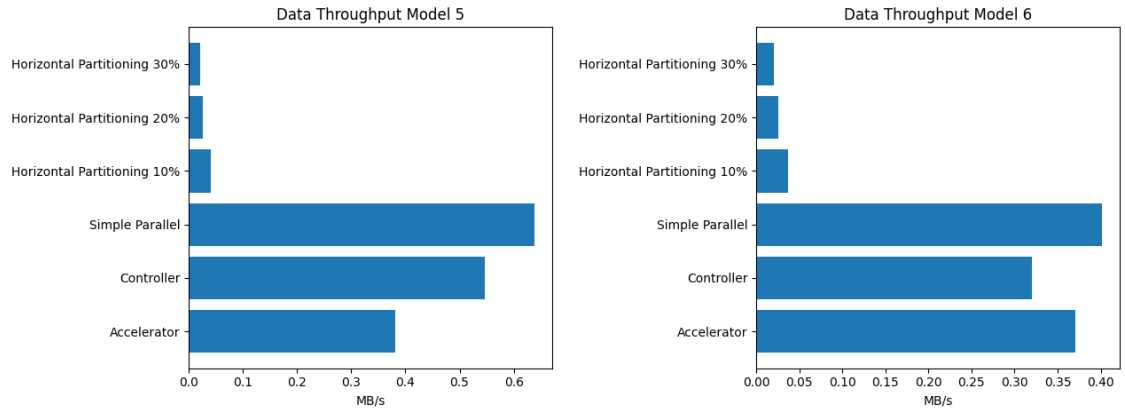

Figure 5.24: Semantic Segmentation Data Throughput

5.3.2 Accuracy

The same consideration about the importance of the accuracy results in this project made in section 5.1.2 is valid also here.

The first metric to be considered for semantic segmentation evaluation is pixel accuracy. For the results obtained and shown below in figure 5.25, both models follow the same trend.

In general, single processing modes and simple parallel mode offer better pixel-wise performances, even though the difference is not too significant.

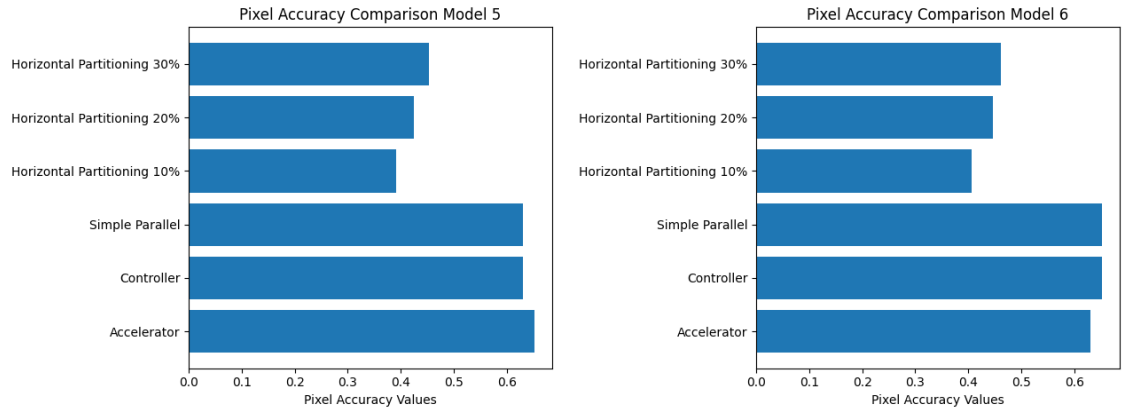

Figure 5.25: Semantic Segmentation Pixel Accuracy

Same behaviour is present in figure 5.26, representing the average IoU for both the models in every configuration.

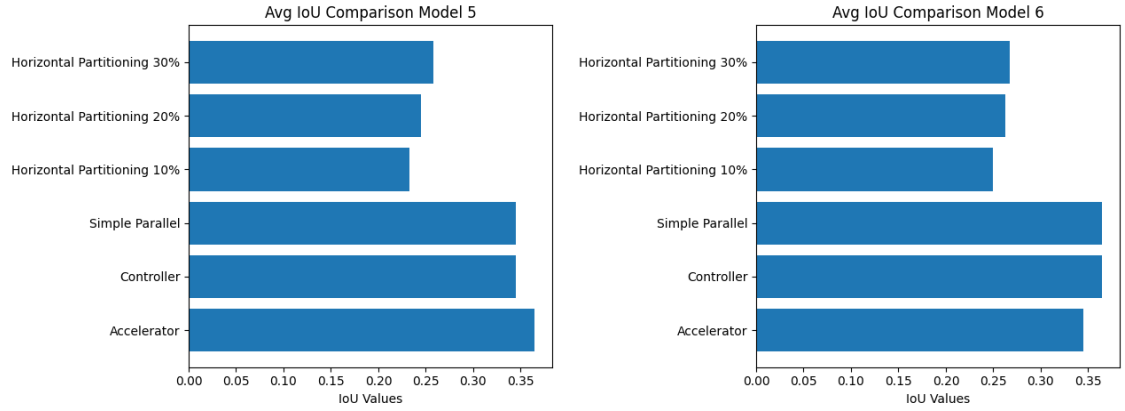

Figure 5.26: Semantic Segmentation IoU

More interesting is the precision-recall graph shown in figure 5.27 below: all the configurations have interesting results for this task, but still simple parallel and single processing mode seem to maintain the lead regarding this metric.

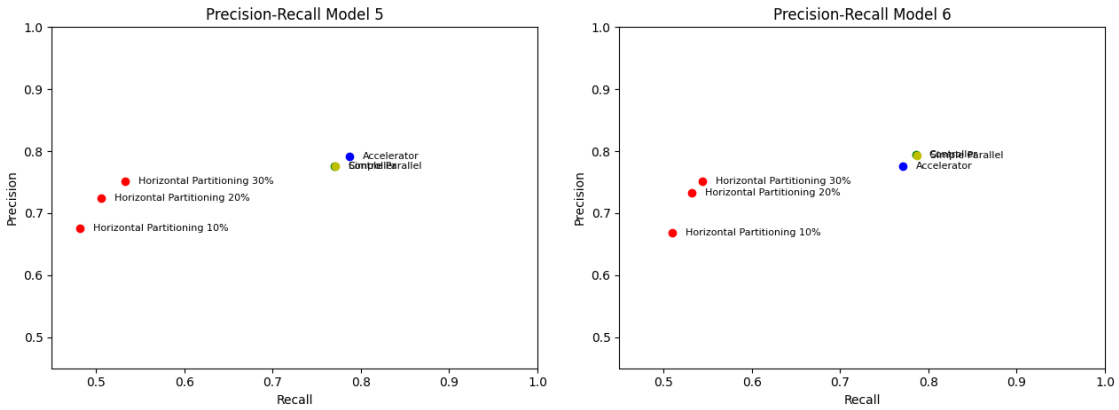

Figure 5.27: Semantic Segmentation Precision-Recall

The F1-score follows the same trend by definition, but showing in a clearer way that also the horizontal partitioning results are worth of interest. The data is shown in figure 5.28 below.

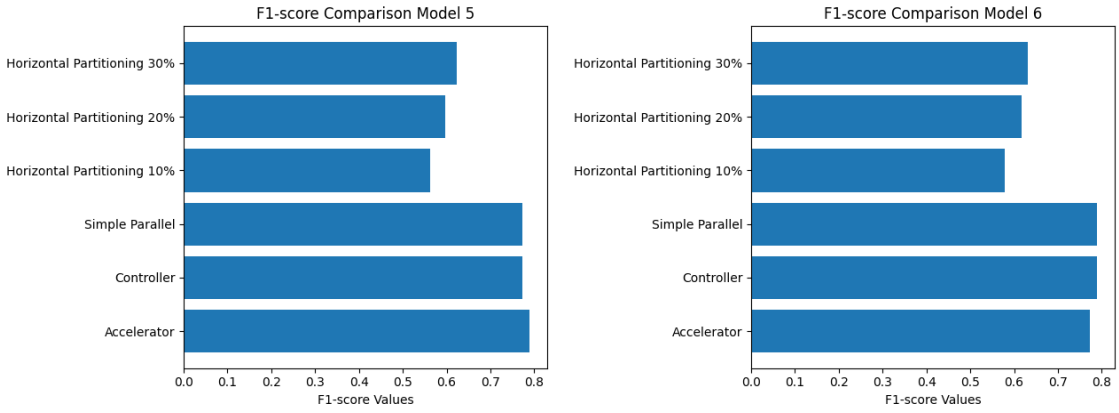

Figure 5.28: Semantic Segmentation F1-score

6 Conclusion

In this section, the conclusions that can be drawn from the experiments results are addressed to answer the research questions exposed in section 1.1. In short, this section contains the assessment on whether the co-processing techniques tried within the experiments can improve the performances of an Edge AI system for space applications, or if they worsen them. The terms of comparison for the evaluation are accuracy and latency for this first project.

Moreover, ideas for future work to improve the existing prototype whereas weak points have been detected, and proceed researching are proposed.

As it has been already done while discussing the results, the most appropriate approach is to evaluate the co-processing methods within every task, since there are different conclusions coming out of them.

To give an initial overview of the conclusions, they differ according to each single deep learning task. Especially image classification, but also semantic segmentation with the tested co-processing method can offer improvements worth of deploying a co-processing setup rather than a single processing one. Fundamental for making these results more meaningful is the implementation of smaller input sizes models, to observe the real potential latency gain.

Nevertheless, not all of the task obtain nice results, and object detection is just an example of those. This is probably due to the nature of the task itself, which proposes to recognize and also precisely locate objects, which for horizontal

partitioning are split in smaller pieces.

More details regarding every task follow below.

6.1 Object Detection

Starting off from object detection, first consideration to explicit is regarding the models chosen. Since they are optimized on both devices, the collected latency values do not show any particular improvement due to the necessity of resizing the tiles to the original image input size. Therefore, the rate is the same, but horizontal partitioning technique overall takes about four times more time. The simple parallel method slightly improves the rate, but not significantly.

Considering accuracy instead, while simple parallel technique gives about the same accuracy result as the single processing modes, horizontal partitioning gets worse evaluation metrics, for whatever considered overlap percentage. Hence, in conclusion it seems like for object detection, using horizontal partitioning is not really convenient with this setup, probably because of the objects which get split into the different tiles.

However, considering more efficient object detection models and different models' input sizes for horizontal partitioning techniques, there might be improvement either on the timing or on the accuracy side (or maybe both).

6.2 Image Classification

Moving on to image classification, the consideration about the chosen models is different: the controller runs a not optimized model, which shows largely slower performances respect to the accelerator. Therefore, from a latency point of view the co-processing techniques seem not to make much sense for this task, since they are slowed down by the controller. It would probably be different if also the controller

model was optimized to run faster.

Regarding accuracy, for model 3 the same above exposed object detection final considerations can be drawn, while for model 4 the story is different.

Addressing the thesis research questions, this last case is very interesting to support the co-processing methods used. In fact, all of the metrics for horizontal partitioning are higher than all the other configurations, and they grow together with the overlap factor. This result is meaningful to prove that in for a task such as image classification, splitting the input data into tiles and spreading them to different devices can improve the predictions.

Looking at the bigger picture, with an optimized model for the controller the horizontal partitioning would have better performances on both the aspects evaluated, assessing the adequacy of the mentioned method over a single processing configuration.

6.3 Semantic Segmentation

Semantic segmentation has different issues instead. In fact, the main problem faced with horizontal partitioning method for this task regards the post-process and in particular the tiles results merge time, which is huge.

The main reason for that is strictly connected to the models' input sizes: since they could not be changed, all the four tiles need to be resized to the original model input size, which is quite big *(513x513)*. Hence, considering that the merge function composes the merged result by pasting the four tiles into a single image, the size of overlapped section to be processed is massive. So, most probably this problem would be improved, or maybe totally fixed, with smaller models' input sized as initially planned.

Accuracy results show how the single processing mode and the simple parallel mode have better performances. However, horizontal partitioning still has decent

metrics values, which could totally be acceptable for some applications such as clouds segmentation. Furthermore, the random choice of pixel labels within the overlapped areas of the tiles needs to be considered when assessing the accuracy results. It is important to mention that a better approach, such as locality principle, would definitely increase both models performances.

Thus, investigating more this task with the right models' input sizes could give interesting enough results to take it into consideration for deployment on a satellite.

6.4 Future Work

Considering the implementation of this project respect to the initial plan exposed in section 3, several aspects can be considered as interesting focus for future work to continue the research.

First of all, one of the main reasons for the experiments to show not useful latency improvements is related to the models' input data sizes. Therefore, it would be interesting to focus more on the models' side and develop custom models with different input sizes for every horizontal partitioning configuration. This way, timing considerations would also become meaningful in the context of evaluating performance improvements, and they could also cause different accuracy results.

Moreover, the implementation of smaller input sizes for the deep learning models would also increase the post-process speed, which is a bottleneck for tasks such as semantic segmentation. The following problem could be addressed also by optimizing the algorithm itself with cross-compilation with C language, implementing multiprocessing and parallelizing the images to process, delegating some operations to the hardware accelerators available in the controller board, involving the other devices to split the workload, or combining some of these mentioned techniques together.

Sticking to the initial plan, investigating vertical partitioning would be key to get

a broader comparison for co-processing methods within this hardware architecture, and therefore a really important direction for the research.

More in general, deploying more models and more tasks could also widen the results spectrum and offer an analysis of the different response of different model families and tasks to the above mentioned co-processing methods. This could include the planned idea of developing a simple custom model to be fully optimized for the hardware platform.

Last but not least, improving the setup stability and communication speed is also key for improving the real performances of the system. In fact, all the discussed results in the previous sections are computed without taking into consideration the communication times with TCP sockets. This means that the actual performances of the prototype are lower, and as explained in section 4.2, these components were omitted from the calculations. Improving the data exchange speed would help improving the overall system performance.

References

[1] G. Furano, G. Meoni, A. Dunne, *et al.*, "Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities", *IEEE Aerospace and Electronic Systems Magazine*, vol. 35, no. 12, pp. 44–56, 2020. DOI: 10.1109/MAES.2020.3008468.

[2] V. Leon, G. Lentaris, D. Soudris, S. Vellas, and M. Bernou, "Towards employing fpga and asip acceleration to enable onboard ai/ml in space applications", in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022, pp. 1–4. DOI: 10.1109/VLSI-SoC54400.2022.9939566.

[3] A. D. George and C. M. Wilson, "Onboard processing with hybrid and reconfigurable computing on small satellites", *Proceedings of the IEEE*, vol. 106, no. 3, pp. 458–470, 2018. DOI: 10.1109/JPROC.2018.2802438.

[4] J. Goodwill, G. Crum, J. MacKinnon, *et al.*, "Nasa spacecube edge tpu smallsat card for autonomous operations and onboard science-data analysis", in *Proceedings of the Small Satellite Conference*, AIAA, 2021.

[5] V. Leon, C. Bezaitis, G. Lentaris, *et al.*, "Fpga and vpu co-processing in space applications: Development and testing with dsp/ai benchmarks", in *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1–5. DOI: 10.1109/ICECS53924.2021.9665462.

[6] G. Giuffrida, L. Fanucci, G. Meoni, *et al.*, "The phi-sat-1 mission: The first on-board deep neural network demonstrator for satellite earth observation", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–14, 2022. DOI: 10.1109/TGRS.2021.3125567.

[7] V. Leon, G. Lentaris, E. Petrongonas, *et al.*, "Improving performance-power-programmability in space avionics with edge devices: Vbn on myriad2 soc", *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 3, pp. 1–23, 2021.

[8] V. Leon, E. A. Papatheofanous, G. Lentaris, *et al.*, "Combining fault tolerance techniques and cots soc accelerators for payload processing in space", in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022, pp. 1–6. DOI: 10.1109/VLSI-SoC54400.2022.9939621.

[9] L. Bo, X. Feng, and Y. Yunhong, "A co-processing method based on warm standby systems", in *2014 IEEE Symposium on Computer Applications and Communications*, 2014, pp. 109–113. DOI: 10.1109/SCAC.2014.30.

[10] N. Audebert, B. Le Saux, and S. Lefevre, "Deep learning for classification of hyperspectral data: A comparative review", *IEEE Geoscience and Remote Sensing Magazine*, vol. 7, no. 2, pp. 159–173, 2019. DOI: 10.1109/MGRS.2019.2912563.

[11] C. Deng, Y. Xue, X. Liu, C. Li, and D. Tao, "Active transfer learning network: A unified deep joint spectral–spatial feature learning model for hyperspectral image classification", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 3, pp. 1741–1754, 2019. DOI: 10.1109/TGRS.2018.2868851.

[12] D. E. Buckley Léonie, "Benchmarking deep learning on a myriad x processor onboard the international space station (iss)", California Institute of Technology, Tech. Rep., 2022.

[13] X. Kong, W. He, and J. Han, "A high-performance risc-v co-processor architecture for fast ip processing", in *2022 IEEE 16th International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2022, pp. 1–3. DOI: 10.1109/ICSICT55466.2022.9963438.

[14] Y. Huang, X. Qiao, W. Lai, S. Dustdar, J. Zhang, and J. Li, "Enabling dnn acceleration with data and model parallelization over ubiquitous end devices", *IEEE Internet of Things Journal*, vol. 9, no. 16, pp. 15 053–15 065, 2022. DOI: 10.1109/JIOT.2021.3112715.

[15] J. Youn and Y.-H. Han, "Intelligent task dispatching and scheduling using a deep q-network in a cluster edge computing system", *Sensors*, vol. 22, no. 11, p. 4098, 2022.

[16] B. Balachandran, K. F. Saad, K. Patel, and N. Mekhiel, "Parallel computer for face recognition using artificial intelligence", in *2019 14th International Conference on Computer Engineering and Systems (ICCES)*, 2019, pp. 158–162. DOI: 10.1109/ICCES48960.2019.9068130.

[17] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing", *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019. DOI: 10.1109/JPROC.2019.2918951.

[18] M. Abadi, P. Barham, J. Chen, *et al.*, *Tensorflow: A system for large-scale machine learning*, 2016. arXiv: 1605.08695 [cs.DC].

[19] *Tensorflow lite*. [Online]. Available: https://www.tensorflow.org/lite/guide.

[20] A. G. Howard, M. Zhu, B. Chen, *et al.*, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017. arXiv: 1704.04861 [cs.CV].

[21] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks", *CoRR*, vol. abs/1905.11946, 2019. arXiv: 1905.11946. [Online]. Available: http://arxiv.org/abs/1905.11946.

[22] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, *Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs*, 2017. arXiv: 1606.00915 [cs.CV].

[23] W. Liu, D. Anguelov, D. Erhan, *et al.*, "SSD: single shot multibox detector", *CoRR*, vol. abs/1512.02325, 2015. arXiv: 1512.02325. [Online]. Available: http://arxiv.org/abs/1512.02325.

[24] *Post-training quantization | tensorflow lite*. [Online]. Available: https://www.tensorflow.org/lite/performance/post_training_quantization.

[25] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, *Microsoft coco: Common objects in context*, 2015. arXiv: 1405.0312 [cs.CV].

[26] O. Russakovsky, J. Deng, H. Su, *et al.*, "Imagenet lsvrc 2012 validation set (object detection)",

[27] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*, http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[28] P. Henderson and V. Ferrari, "End-to-end training of object class detectors for mean average precision", *CoRR*, vol. abs/1607.03476, 2016. arXiv: 1607.03476. [Online]. Available: http://arxiv.org/abs/1607.03476.

[29] Z. Vujovic, "Classification model evaluation metrics", *International Journal of Advanced Computer Science and Applications*, vol. Volume 12, pp. 599–606, Jul. 2021. DOI: 10.14569/IJACSA.2021.0120670.

[30] S. Goyal, *Evaluation metrics for classification models*, Jul. 2021. [Online]. Available: https://medium.com/analytics-vidhya/evaluation-metrics-for-classification-models-e2f0d8009d69.

[31] R. TS and R. Senthilnathan, "Dataset and performance metrics towards semantic segmentation", *Available at SSRN 4352476*, 2023.