Technical University of Munich
TUM School of Natural Sciences
Physics Department

Master's Thesis for the M.Sc. Degree in Physics (Nuclear, Particle and Astrophysics)

Characterization and Evaluation of Hardware Accelerators for the On-board Data Processing of the AFIS Satellite Mission

Vernando Fransiscus Limodya

January 31, 2024

Characterization and Evaluation of Hardware Accelerators for the On-board Data Processing of the AFIS Satellite Mission

Charakterisierung und Evaluation von Hardwarebeschleunigern für die On-board Datenverarbeitung der AFIS Mission

Topic Supervisor: Prof. Dr. S. Paul
Direct Advisor: Peter Hinderberger, M.Sc.

Acknowledgement

This thesis is not one of the best, but it is a witness for the struggle through blood and tears that I have gone through this past one year to finish it.
Firstly, I would like to thank Peter Hinderberger, my direct supervisor, without whom I would not be able to finish this work, providing support in critical times and seeing the potential in me. I would like to thank also Martin Losekamm for his feedback on my work and on my mid-term presentation.
I would like to thank my parents, who raised me to be a tough and patient person that never gives up. Also for the emotional support they sent through the video calls miles away from my home country, Indonesia. I would like to thank my sister, Vanessa Limodya, and my brother, Vincentcius Limodya, who are supporting me through prayers and just being there for me.
I cannot describe the amount of thanks that I would like to give to my main support team, who includes Abhishek Kapoor, Omar Adawi, Otman Saladi and Edwin Adisoemarta, whose never ending emotional support made me cannot imagine how to have survived the critical times alone. I dedicate my thesis to them.
I would like to thank Xavier Simó, who has made my days in E18 not very lonely, for the lunch breaks in the mensa.
I would like to thank Jorge Marin Ruiz, Devendra Vyas and Azka Ali, who has helped me gain a way better understanding of computer architectures and being the best tutors.
I would like to thank my sister from another mother, Anna-Maria Semeraro, and Süheyla Şeker and Xenia Ritz, who also supported me in every way possible.
I would like to thank Denisa Dosenovicova, my beloved work colleague, to whom I can rant about my thesis too much in the lunch breaks, for being a good listener.
I would like to thank also Mohammed Thabit, Daniela Lama and Luiza Gancz, who occasionally checked on me during my days writing in the Rechnerhalle of the MI building.
Last but not least, most importantly, I would like to thank Jesus for His support, and reminding me to not give up any other day through the verse :

"I can do all this through Him who gives me strength." - Philippians 4 : 13

Abstract

The Antiproton Flux in Space (AFIS) Mission aims to measure the flux of geomagnetically trapped low-energetic antiprotons in the South Atlantic Anomaly (SAA) region. The high event rate of at least in the order of 10^4 events/s and the low bandwidth for data downlink aboard small satellites requires on-board data processing in the form of event triggers and lossless compression by an FPGA. A separate module containing an additional specialized hardware accelerator e.g. for matrix multiplications and convolutions can be employed to support data processing. In this thesis, in order to evaluate advantages and disadvantages of such accelerators, I investigate several feasible candidates based on throughput, power consumption, integrability to the FPGA and suitability of operations. The results based on literature research and the neural network benchmarking show that the Tensor Processing Unit (TPU) is more suitable for smaller networks and the Vision Processing Unit (VPU) for larger networks. Also from these results, optimization methods that improve each accelerator's throughput can be deduced. A small optimization step is done for the PID model by introducing a 1×1 convolutional layer as a means of dimensionality reduction so that the network hyperparameters are optimized for the TPU and the VPU. This results in significant improvement in throughput for the TPU and a slight improvement for the VPU. It can be also observed that magnitude-based pruning does not improve the throughput of both the TPU and VPU.

Zusammenfassung

Die Antiproton Flux in Space (AFIS) Mission hat das Ziel, den Flus geomagnetisch gefangener niederenergetischer Antiprotonen in der South Atlantic Anomaly (SAA)-Region zu messen. Die hohe Eventrate von mindestens in der Größenordnung 10^4 Events pro Sekunde und die geringe Bandbreite zur Grunddatentransmission an Bord kleiner Satelliten erfordern die On-boarddatenverarbeitung in Form von Eventtriggers und verlustfreier Kompression durch einen FPGA. Ein separates Modul kann mit einem zusätzlichen spezialisierten Hardwarebeschleuniger geeignet für Matrixmultiplikationen und Faltungen zur Unterstüzung der Datenverarbeitung eingesetzt werden. In dieser Arbeit, um die Vor- und Nachteile von solchen Beschleunigern auszuwerten, untersuche ich mehrere Kandidaten nach dem Eventdurchsatz, dem Leistungsverbrauch, der Integrationsmöglichkeit zum FPGA und den unterstützten Operation. Die Ergebnisse basierend auf Literaturrecherche und Benchmarking von neuronalen Netzwerken zeigen, dass der TPU für kleine Netzwerke und der VPU für größere Netzwerke besser geeignet ist. Auch von diesen Resultaten lassen sich Optimierungsmethoden neuronaler Netzwerke ableiten, die den Eventdurchsatz jedes Beschleunigers steigern. Ein kleiner Optimierungsschritt wird für das PID-Modell durchgeführt, indem eine 1×1 Faltungsschicht zur Dimensionsreduktion eingeführt wird, so dass die Netzwerkhyperparameter für den TPU und den VPU optimiert sind. Dies führt zu einer signifikanten Verbesserung des Eventdurchsatzes für den TPU und und einer leichten Verbesserung für den VPU. Es lässt sich auch feststellen, dass Größe basiertes Pruning nicht den Eventdurchsatz von dem TPU und VPU verbessert.

Contents

List of Figures

List of Tables

List of Abbreviations

AFIS	Antiproton Flux in Space
AMM	Analog Matrix Multiplier
AMP	Analog Matrix Processor
BRU	Branch and Repeat Unit
CMX	Connection Matrix
CPU	Central Processing Unit
FPGA	Field Programmable Gate Array
GPU	Graphical Processing Unit
HDL	Hardware Description Language
IDE	Integrated Development Environment
LEO	Low Earth Orbit
MXU	Matrix Multiplication Unit
PDP	Payload Data Processor
PE	Processing Element
PEU	Predicated Execution Unit
PMIC	Power Management Integrated Circuit
RISC	Reduced Instruction Set Computing
SAA	South Atlantic Anomaly
SCM	Scientific Computation Module
SEFI	Single Event Function Interrrupts
SHAVE	Streaming Hybrid Architecture Vector Engine
TDP	Thermal Design Power
TID	Total Ionizing Dose
TPU	Tensor Processing Unit
UB	Unified Buffer
VLIW	Very Long Instruction Word
VPU	Vision Processing Unit

Introduction

In his attempt to incorporate special relativity into quantum mechanics, Paul Dirac formulated the relativistic Dirac equation in 1928,[1] whose solution predicts the existence of antiparticles. This prediction is justified by the discovery of the first antiparticle, the positron, by Carl D. Anderson a few years later.[2]

Afterwards, the discovery of more antiparticles followed. In 1955, the antiproton was discovered by Emilio Gino Segrè and Owen Chamberlain using the Bevatron particle accelerator[3] for which they were awarded the Nobel Prize in Physics in 1959. Furthermore, a small flux of antiprotons were found for the first time in cosmic rays in 1979,[4] where the measured flux ratio between antiprotons to protons is consistent with the theory, that antiprotons are mainly produced as secondary particles resulting from the interaction of cosmic ray particles with interstellar matter. Experiments have also found evidence of cosmic-ray antiparticles in the form of positrons.[5]

There are attempts to test exotic theories beyond the Standard Model on the mechanism of antiparticle production in the universe, such as through the annihilation of dark matter. One of the ways to verify these theories is to observe the deviations of the measured antiparticle flux spectra from the expected spectra based on pure secondary production of antiparticles. Therefore, a precise flux measurement of the cosmic ray antiparticles is required.

There are two main types of experiments to detect cosmic rays. The first type of experiments involve ground-based detection, which is based on the detection of extensive cascades of secondary particles formed by the interaction of high-energy cosmic rays of energies above 10^{13} eV with the Earth's atmosphere. The second method is through direct detection using particle detectors situated on satellites or balloons. The energy range for this method is in the magnitude of 10^3 to 10^{15} eV. However, such experiments come with constraints imposed on the weight and volume of the detectors.[6]

In 2006, a satellite-based experiment for direct cosmic-ray detection, PAMELA (a Payload for Antimatter Matter Exploration and Light-nuclei Astrophysics), was launched. The collaboration published an article in 2011 [7] on the discovery of a significant flux of geomagnetically confined antiprotons in the range of 60-750 MeV in the South Atlantic Anomaly (SAA), a region of the inner van Allen radiation belt where it dips to an altitude of 200 km due to the non-concentricity of the Earth's geographic and magnetic poles. Fuki et. al. and Selesnick et. al. also predicted a significant flux of geomagnetically trapped antiprotons in the SAA [8, 9], which is based on the

theory that the main contribution to geomagnetically trapped antiproton flux is from the decay of antineutrons produced by interaction of high energy cosmic rays with the upper atmosphere. However, the predicted values for the antiproton flux are off by two to three orders of magnitude to the measured flux.

For further adjustments from the theoretical and experimental standpoints, data from lower energies is needed to complement the measurements of PAMELA. The Antiproton Flux in Space (AFIS) mission plans to measure the flux of low-energetic geomagnetically trapped antiprotons in the SAA. In contrast to PAMELA that utilizes magnetic spectroscopy, AFIS aims to identify particles based on Bragg spectroscopy inside a cube-shaped detector and, for antiprotons, combined with the analysis of event topologies caused by their annihilation with a nucleon of the detector material. Annihilation is the interaction of a particle or one of its constituents with its antiparticle, resulting them to vanish and release energy in the form of photons and secondary particles.

PAMELA measured the antiproton-to-proton flux ratio in the order of 10^{-6} to 10^{-8} in the SAA.[7] This implies that a very low signal-to-background ratio for AFIS events can be expected. In order to suppress the number of background events, an online event trigger concept is needed and an on-board data processor with a field-programmable gate array (FPGA) architecture is designed to fulfill this task. The integration of an additional specialized component called a hardware accelerator to perform these computations is considered. The capabilities and limitations of such hardware accelerators are needed to be done.

The goal of this thesis is to compare and evaluate several candidates of hardware accelerators and investigate which optimization methods of neural network model algorithms improve the performance of each hardware accelerator. In the first chapter, the production of antiprotons in the inner van Allen radiation belt, particle detection aboard AFIS and its on-board data processing are briefly discussed. Additionally, basic concepts of machine learning and neural networks are explained to provide context for the following chapters. In the second chapter, the results of datasheets survey of feasible hardware accelerators are introduced, considering factors such as theoretical maximum number of operations per second performed, power consumption, supported model specifications, computing architecture, etc. In the third chapter, the measurement results of throughput and power consumption for available USB modules of accelerator chips for several custom-made benchmarks and public open-source benchmarks from other publications are presented. These benchmarking results verify and give us information on the computing architectures of each accelerator and hence also on how to optimize neural network models specifically for each accelerator.

Chapter 1

Background

In this chapter, I will discuss the theoretical foundations of my work, which consist of antiproton production in the inner van Allen belt, and antiproton detection and on-board processing aboard the AFIS mission. Furthermore, I will also introduce several relevant basic concepts of machine learning and neural network architectures, their functionalities and model compression.

1.1 The AFIS Mission

1.1.1 Antiproton Production and Goal of AFIS

Due to the magnetic field of the Earth, charged particles accumulate in toroidal shaped radiation belts called the van Allen radiation belts.[10] There are two such belts, the outer belt, which mainly consists of trapped high-energy electrons between 0.1 MeV and 10 MeV and the inner belt, which consists of mainly low-energetic protons and electrons of less than 5 MeV.[11]

The trapped protons in the inner van Allen belt cannot be of cosmic ray origin because such protons would have had energies in the order of 1 to 10 GeV. The trapped protons have energies lower than this value and are produced due to the interactions of high-energetic cosmic rays with nucleons of the upper atmosphere.[11] Antiprotons trapped in the inner van Allen belt are predicted to be also produced through such interactions.[8] The simplest mechanism is through direct proton-antiproton production from the collision of a high-energetic proton with another proton or a helium nucleus

$$\mathrm{p} + \mathrm{A} \rightarrow \mathrm{p} + \mathrm{p} + \overline{\mathrm{p}} + \mathrm{X} \tag{1.1}$$

A proton-antiproton pair must be produced, rather than only an antiproton, due to baryon-number conservation. Another mechanism is the production of another baryon-antibaryon pair, such as a neutron-antineutron pair.

$$\mathrm{p} + \mathrm{A} \rightarrow \mathrm{p} + \mathrm{n} + \overline{\mathrm{n}} + \mathrm{X} \tag{1.2}$$

Subsequently, the antineutron will decay into an antiproton

$$\overline{n} \rightarrow \overline{p} + e^{+} + \nu_e \tag{1.3}$$

This process of antiproton production from the decay of so-called albedo $\overline{n}$ is called CRANbarD (Cosmic Ray Antineutron Decay) [7] and this gives the largest fraction of trapped antiprotons in the radiation belt. [9]

In 2011, the PAMELA (Payload for Antimatter Matter Exploration and Light-Nuclei Astrophysics) collaboration [7] published the measurements of the flux of antiprotons in the inner Van Allen radiation belt. The measurement was done in the South Atlantic Anomaly (SAA), the region where the Earth's magnetic field is weaker due to the non-concentricity of the Earth's geographic and magnetic poles. Here, the magnetic repulsion of the Earth against the charged particles of the inner van Allen belt is weakened and hence the inner belt dips down to an altitude of 200 km.[12]

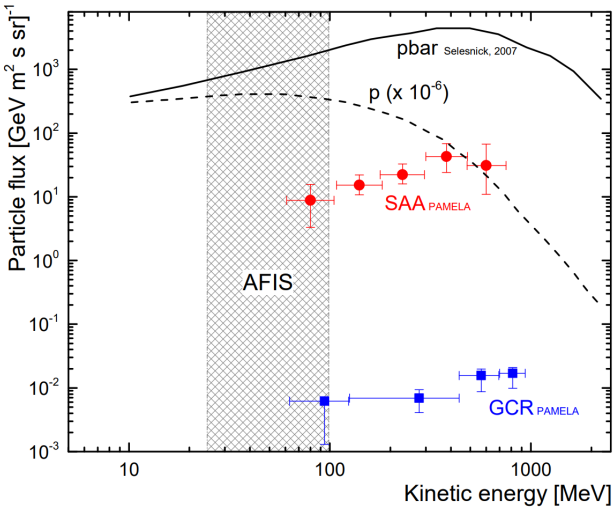


Figure 1.1: Results of trapped antiproton flux measurement by PAMELA in the South Atlantic Anomaly (SAA) are shown in red. Also, the measured cosmic ray antiproton flux is shown in blue [7] and the antiproton flux predicted by the CRANbarD model by Selesnick et. al. is shown in black [9]. The proton flux is shown in a black dashed line as reference. The AFIS experiment measures the trapped antiproton flux in the region marked by the grey strip. Source of figure: [11]

In Figure 1.1 the results of the measurement for trapped antiprotons by PAMELA is plotted. This flux is higher by more than three orders of magnitude than the measured cosmic ray antiproton flux outside the SAA and off by one to three orders of magnitude compared to the predicted trapped antiproton flux based on the CRANbarD model by Selesnick et. al.[9] This discrepancy demands further investigations from theoretical and experimental standpoints. More data at lower energies is required to support future adjustments [11] and the main goal of the AFIS experiment is to measure the trapped

antiproton spectrum for lower energies (25 - 100 MeV) for these adjustments with a detector that has a large geometrical acceptance, which is explained in Subsection 1.1.2.

1.1.2 Antiproton Detection

Plastic Scintillating Fibers as Detector Material

The AFIS experiment employs organic plastic scintillating fibers as active target detector material. Organic plastic scintillating fibers have already been used in the past for particle tracking and calorimetry of hadrons, due to their high signal speed and their capability to handle 100 MHz event rates.[13] The plastic scintillating fibers used for AFIS are from Kuraray and these fibers are wrapped in aluminium foil and aluminized at one end in order to increase the light yield and prevent optical crosstalk.[14]

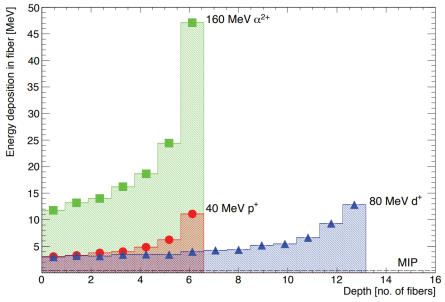


Figure 1.2: Energy deposited to scintillating fibers is plotted against the depth (in terms of a number of fibers) of the track, shown for three particles of different species and energies that are perpendicularly incident on the detector. The shape of this curve is called a Bragg curve.[15]

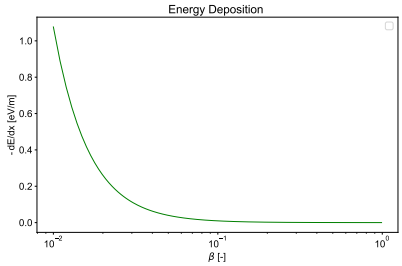


Figure 1.3: The energy deposition per unit length $-\frac{dE}{dx}$ is plotted against the velocity $\beta = \frac{v}{c}$ for the case of a penetrating proton and a polyethylene detector material. With decreasing velocity, the energy deposition of the proton increases, which explains the Bragg curves in Figure 1.2 (plotted by myself)

When an ionizing particle traverses through an array of plastic scintillating fibers, it ionizes the atoms of a material and hence loses energy. The ionization can excite molecules that can de-excite afterwards and emit luminescence photons. These photons are reflected inside the plastic fibres and guided to the photo multipliers to be converted to electric currents. As the penetration depth increases, the particle velocity decreases due to the deposition of its energy into the detector material and subsequently increases the energy deposition per unit length, as given by the Bethe-Bloch formula for charged particles

$$-\left\langle \frac{dE}{dx} \right\rangle = \frac{4\pi}{m_e c^2} \frac{nZ^2}{\beta^2} \left(\frac{e^2}{4\pi\epsilon_0} \right)^2 \left[\ln\left(\frac{2m_e c^2 \beta^2}{I(1-\beta^2)} \right) - \beta^2 \right] \tag{1.4}$$

where $\left\langle \frac{dE}{dx} \right\rangle$ is the mean energy deposition per unit length, m_e is the mass of an electron, Z is the charge of the penetrating particle, $\beta = v/c$ is the velocity of the penetrating particle, I is the mean excitation energy of the medium, n is the electron density of the medium and ϵ_0 is the vacuum permittivitiy. This results in energy deposition curves shown in Figure 1.2, called *Bragg curves*, and the analysis of such curves is called *Bragg spectroscopy*, which is already a widely used method in nuclear and medical physics.[16] The downside of Bragg spectroscopy is that both particle and its corresponding antiparticle will result in the same Bragg curve.

The Detector for AFIS

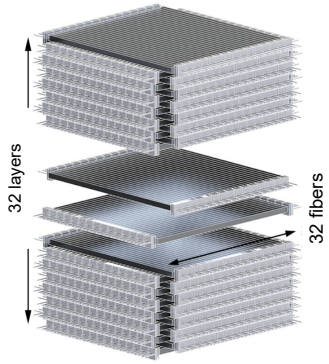


Figure 1.4: The detector for AFIS comprises 32 layers of scintillating plastic fibers, where each layer has 32 fibers and is rotated 90° with respect to its neighbor. Each fiber has a silicon photomultiplier at the end to convert the scintillating light into electrical currents.[15]

The detector structure used for the experiment is shown in Figure 1.4. It comprises of 32 layers of 32 plastic scintillating fibers, where each layer is rotated with 90° with respect to its neighbor. If four fibers must be hit in order to be considered for particle detection, the resulting geometrical acceptance of the detector is 925 $\mathsf{cm}^2\,\mathsf{sr}$, enabling almost omni-directional particle detection.[17] This fiber configuration has approximately the size of a 1U CubeSat and has an antiproton sensitivity range of 25 to 100 MeV. The lower limit of 25 MeV comes from the minimum energy of an antiproton to traverse the detector's insulation layers and to be observed of its stopping inside the detector and its annihilation. The upper limit of 100 MeV comes from the approximate minimum energy for an antiproton to pass through the detector without stopping.

The scintillation light resulting from particle detection are reflected to 1024 silicon

photomultipliers (SiPMs) at the end of each fiber to be converted to electrical currents. 64 of these 16-channel ASICs read out the SiPMs and are responsible for analog-to-digital conversion. The digitized signals from the ASICs are transmitted to the FPGA for further processing such as event triggers and compression.[15]

In Figure 1.5 simulated events from the detector can be observed. One event consists two images, corresponding to the projection with respect to the YX plane (bottom) and to the YZ (top) plane, each with the size of 16×32. The values of the pixels are the deposited energies in MeV.

On the other hand, high-energy proton interactions with detector material can also create secondary particles but boosted in the direction of the primary particle.

Most antiprotons slow down to non-relativistic energies before annihilating, as its annihilation cross-section increases rapidly with decreasing velocity. The annihilation produces secondary particles that are emitted isotropically due to momentum conservation, about four of which are detectable.[11] During the annihilation, twice of the antiprotons rest mass is converted into energy. This can be observed in the leftmost figure. The second type of event is shown in the middle figure, where a particle is completely stopped inside the detector but does not annihilate. This event is unlikely caused by an antiproton, since an annihilation should have been observed in this case. The event on the right occurs when a high-energetic particle outside the sensitivity range traverses through a detector without producing secondary particles. In this case, a technique based on Bragg spectroscopy can be used to identify if the event comes from a proton or antiproton. An event trigger, which recognizes the signatures of the event topologies, is planned in the future.

Event Rate and Signal-to-Background Ratio Estimate

According to the simulations of [15], during the week of maximum solar activity a very high data rate in the order of at least 10^4 events per second can be expected in the SAA, most of which are from low-energetic trapped protons and from the simulations of [11] an antiproton count rate estimate of $7.9 \cdot 10^{-3} = o(10^{-3})$ per second is reported, neglecting the trigger and reconstruction efficiencies. This results in a very low signal-to-background ratio in the order of $o(10^{-3}/10^4) = o(10^{-7})$. The low signal-to-background ratio requires us to design an on-board triggering concept for the events to improve it.

1.1.3 On-board Data Processing

The Necessity of an On-board Data Processor and its Requirements

First of all, on-board processing is needed to read out digital signals from the 64 detector data acquiring ASICs.

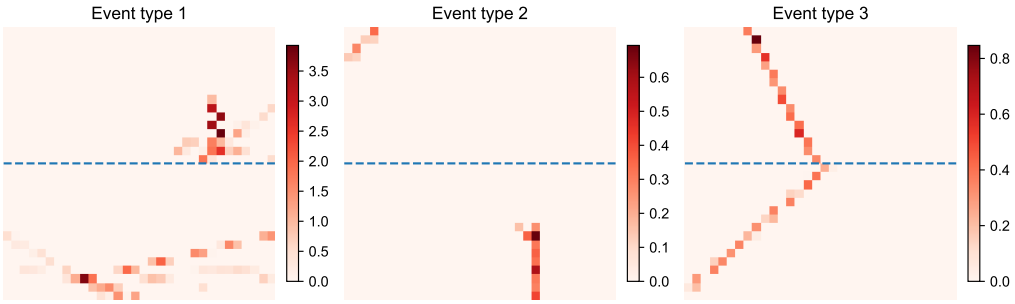


Figure 1.5: Three simulated events of AFIS. An event consists of one-channel images with 16×32 pixels each for the projection w.r.t. the YX and YZ planes. The first event originates from a antiproton inside the energy range, which decays inside the detector material. The second event is caused by a proton, since it is stopped but does not annihilate inside the detector. The third event can originate either from a high energetic proton or antiproton.

Next, the very high data-rate of at least 10^5 events/s is expected. For a Lower Earth Orbit (LEO) mission, an orbit lasts 90 to 120 minutes.[18] Since the SAA spans from $-50°$ to $0°$ geographic latitude and from $-90°$ to $+40°$ longitude,[19] the satellite will be inside the SAA for between $\frac{0-(-50)}{360} \cdot 90 = 12.5$ and $\frac{40-(-90)}{360} \cdot 120 = 43.3$ minutes and this corresponds to available measurement data of size $1024 \cdot 12$ bits/event $\cdot 10^4$ events/s $\cdot (12.5 \cdot 60)$ s $= 92.2$ Mbit to $1024 \cdot 12$ bits/event $\cdot 10^4$ events/s $\cdot (32.5 \cdot 60)$ s $= 319.6$ Mbit. However, the use of small satellites imposes limitations to the buffering capabilities of the hardware and data transmission bandwidth, prompting us to reduce the data size. Therefore, an on-board data processor is required to implement a triggering system and to compress the data that pass through the triggers. This simultaneously increases the signal-to-background ratio from $o(10^{-7})$.

However, this processing module must meet certain requirements. Due to the restrictions in mass and volume for small satellites, the physical size of components such as solar panels and antennae is limited. This sets a tight upper limit on power consumption and the data transmission bandwidth.[20] Therefore the on-board data processor must also consume very little power for its operation in order to fit into the power budget and simultaneously the selection of the relevant events through event triggering must have sufficient computation speed in order to be completed by the time of downlink and with negigible data losses due to the already very low signal-to-background ratio.

Computing Architecture of the On-Board Data Processor

There are several options for the computing architecture for the on-board data processor, which we alternatively call the Payload Data Processor (PDP). Since digitized signals have to be read from 64 ASICs, sequential computing such as microcontrollers and CPUs are not suitable, as the ASICs will be read in sequence, resulting in slow readout speeds and hence introducing a high latency. A parallel computing architecture is therefore implemented, so that the digitized signals can be read simultaneously.

Even though capable of performing parallelized operations, the use of a graphics processing unit (GPU) is rejected since it has high power consumption (a minimum of 5W)[21] and often requires cooling. ASICs can be designed and tailored specifically for a mission, can also process data in parallel and can have low power consumption. However, they have no configurability after their production, have long development times and high costs.[22]

Therefore, we have decided to use a field programmable gate array (FPGA). Unlike ASICS, FPGAs are reconfigurable, allowing flexibility to adapt to the changing needs of the mission. However, FPGAs require a high programming/configuration effort, because they require Hardware Description Languages (HDL).

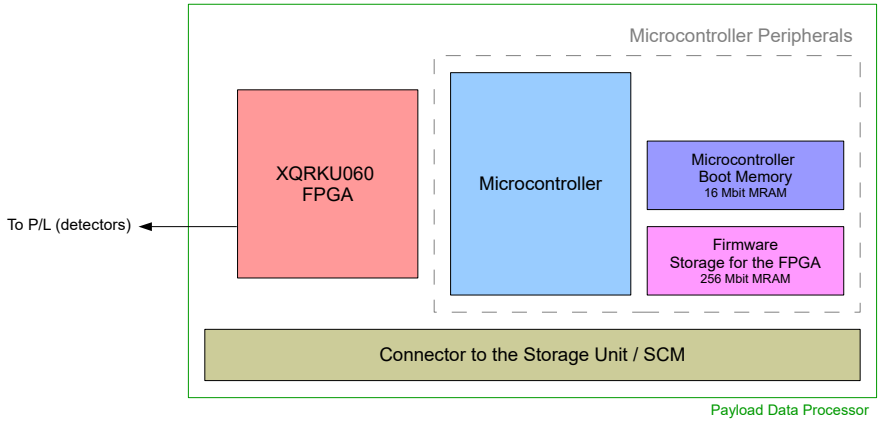


Figure 1.6: The planned architecture of the Payload Data Processor (PDP) baseboard. Components in the dashed box are peripherals of the microcontroller, which serves as the external controller and programmer of the FPGA. The Storage/Compute Unit Connector enables the connection to an external board containing storage-related components and/or a computational module (such as the Scientific Computation Module). [23] Abbreviations: P/L – Payload

The planned system architecture of the PDP baseboard is given in Figure 1.6. The heart of the PDP is a Xilinx XQRKU060 FPGA, which operates the interfaces of

the AFIS detector in parallel. This particular FPGA is chosen because it is radiation-hardened, making it suitable for space-missions, and have enough computational resources to be the sole payload data processor of most missions.

A microcontroller is added in the PDP for the programming of the FPGA with different firmware versions, handling errors, e.g. due to single event upsets (SEUs), which are bit-flips caused by ionizing particles, usually in memory components.[24] The microcontroller also takes over telemetry-gathering tasks and power supply control to the FPGA, so that the FPGA's purpose is only to read-out, process and transmit the instrument data. The boot memory for the microcontroller and the firmware storage for the FPGA are stored in different MRAMs, but both are peripherals of the microcontroller.

Other components that may support the operation of the FPGA should be integrated in the form of separate boards in order to implement modularity, which provides flexibility to the PDP design for the needs of future missions, allows independent space-grade testing and maintenance. Examples of designated modules include a storage-unit board for memory-related components, such as MRAM or radiation-hardened flash, or an additional module that contains hardware accelerators and its supporting components to assist and accelerate the FPGA computations, called the *Scientific Computation Module*.[23]

The Concept for Data Acquisition and Data Processing Chain

Initially the digital input from the ASICs are compiled into a single 1024×12-bit vector and this serves as an input for a low-level particle trigger, i.e. simple algorithms that determine whether the event is relevant. These include cuts e.g. based on the minimum fibers fired and the total particle energy, and Kalman filters. Higher trigger levels are planned for the future and a separate module called the Scientific Computation Module (SCM) (explained in the following subsection) can support the processing for these triggers. The relevant events that pass through the triggers will be losslessly compressed to be transmitted to the ground station, such as through Huffman encoding or run-length encoding.

The Motivation of the SCM and Hardware Accelerators

The design of the on-board payload data processing uses a modular approach, which is realized by using a PDP baseboard, as described in Figure 1.6, that will be used for multiple planned future missions. Additional mission-specific components are integrated into the PDP from a different module. This strategy can be advantageous, because e.g. the baseboard and the mission-specific modules can be separately evaluated and optimized in terms of performance, power consumption and radiation-tolerance.

The FPGA of the PDP baseboard has limited computational resources and buffering capabilites. For example, a neural-network based trigger concept for AFIS will involve matrix multiplications, which can require addition and multiplication blocks with the complexity $O(n^3)$, where n is the number of input bits involved.[25] For instance, if the resources in the FPGA are not sufficient for a certain set of computations or a computational bottleneck is produced, a plausible solution is to integrate additional hardware into the PDP, where such computations can be delegated into. In this work, we will narrow our scope to matrix multiplications and convolution operations for deep neural-network based applications.

For matrix multiplications, a CPU or a microcontroller is not suitable because of their sequential computing architecture, which means that every elementary operation is performed in sequence. With the complexity of $O(n^3)$, the computation time for large matrices increases rapidly. The GPU is again not chosen due to the power constraints, regardless of its ability to perform parallelized operations. In fact, the FPGA is more performant in matrix multiplications than GPUs, even though GPUs are more cost-effective.[26]

I can take a look into hardware accelerators, which are computing architectures specialized for matrix multiplications and convolutions. In this thesis,

1. I evaluate the advantages and disadvantages of hardware accelerators for the SCM based on throughput, power consumption, integrability to the FPGA and suitable algorithms

2. I investigate methods to optimize throughput for each hardware accelerator

Hence, in the following, several hardware accelerators are compared systematically based on a set of criteria through literature research in Chapter 2 and through running benchmark algorithms in Chapter 3 to characterize the computing architecture and evaluate the throughput and power consumption of available hardware accelerators. However, these hardware accelerators are not assimilated directly into the PDP baseboard to ensure modularity, especially if they are not radiation tested. These are integrated into a separate module called the Scientific Computation Module (SCM). The SCM will consist of the hardware accelerator and its supporting components, such as volatile or flash memory, power management ICs, etc., and will be interfaced with the FPGA from the baseboard through a high-speed data and control interface.

1.2 Machine Learning and Neural Networks

1.2.1 Machine Learning

The conventional way to automatize the processing of data is to explicitly design algorithms for it, meaning the duty of writing the explicit algorithms is given to the

user. In contrast to this, *machine learning* attempts to automatize the finding of these algorithms. The computer will try to find out the meaningful relationships and patterns from examples and observations [27], inspired by how humans learn to find connections between causes and outcomes. To do this, a large amounts of problem-specific data are fed into a certain algorithm or *model* and *learns* the patterns and features from this data set. After that, it can be applied to other problem-specific data.

According to the given problem and available data, there are three types of Machine Learning cases:[27]

- *Supervised learning* – the training data set consists of input features x and target variables y. Once the model is trained, the model hopefully can predict the target variables y of other input data not from the training data set. Depending on the data type of y, we can further distinguish, between *regression*, when y is a scalar value, and *classification* models, when y is a class.

- *Unsupervised learning* – the training data set only consists of input features x, *without* the target variables y. The goal of the learning process is to find groups that share certain common properties (*clustering*) or data representations that are projected from a higher-dimensional space into a lower-dimensional space (*dimensionality reduction*, e.g. Principal Component Analysis (PCA), Singular Value Decomposition).

- *Reinforcement learning* – there is no training data set, but we describe the state of the environment, specify a goal, provide a list of allowable actions and environmental restrictions. There are agents in this environment, which must perform actions and get rewards or punishments depending on their actions. The agents will learn which decisions to take based on the feedback they obtain. Mainly, this way of training has great success in closed world environments, e.g. autonomous driving and games.

In all machine learning models, the raw data are transformed into numerical features that are compatible for machine learning algorithms and therefore the dimensionality of the data is reduced. This step is called *feature extraction* and is usually done by data scientists. For instance, there is a model to predict the age of snakes based on their length. Feature extraction is the step of determining the lengths of snakes from raw data in the form of snake images.

In conventional explicit programming, both the feature extraction of the inputs and the building of the model are all done manually. In contrast, the building of machine learning models are performed through automated iterative learning, but in general not necessarily the feature extraction step.

In *deep learning*, a subset of machine learning, the feature extraction is automated, meaning the trained models can receive the raw data directly as an input. It is called

"deep" because of the use neural networks, which consist of multiple layers, which will be discussed in Subsection 1.2.2. By the time of the creation of this work, the use of neural networks is almost synonymous with the term deep learning.

Table 1.1: The differences between machine learning categories based on which steps of the model building process is automated. In general, in machine learning feature extraction is not extracted. However, in deep learning, the feature extraction is automated.

	Explicit Programming	**Conventional Machine Learning**	**Deep Learning**
Feature extraction	manually	manually	automated
Model building	manually	automated	automated

The concept types of models for machine learning algorithms are various, as shown in Figure 1.7. Conventional machine learning algorithms, such as support vector machines and decision trees, don't have a functionality to automatically discover representative features needed for the corresponding learning task. Meanwhile, deep neural networks with more complex neuron architectures and consisting of more than one hidden layer, can provide it.

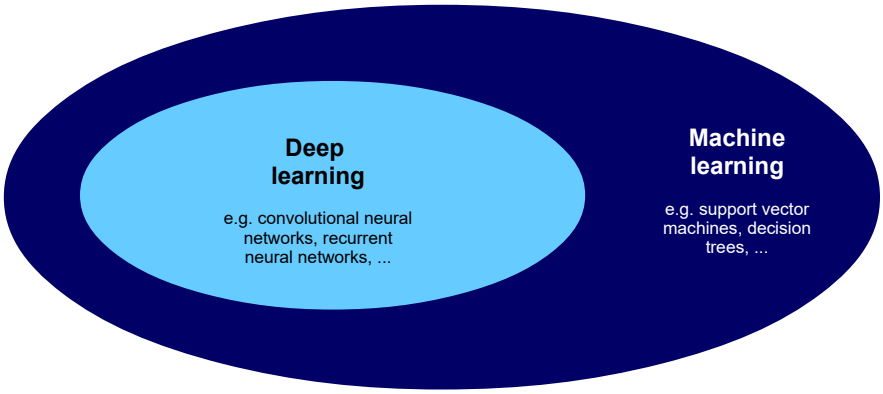


Figure 1.7: Machine learning concepts and examples based on their category. Neural networks types such as convolutional and recurrent neural networks, are main topics in deep learning.

Currently there are a number of open-source programming frameworks that are developed in particular for Machine Learning, such as TensorFlow, PyTorch, ONNX, etc. However, I will stick to the use of TensorFlow throughout the article, since it is the only framework that is supported by all the accelerator chips of interest that will be listed in Chapter 2 and we will limit ourselves with the use of neural networks.

1.2.2 Neural Networks

Neural networks are a collection of connected nodes, called *neurons*.[28] This nomenclature is inspired by the biological neuron, where each neuron receives an input in form of electrical impulses and transfers this impulse into other neurons as an output. A neuron can have several input and several output connections. A group of neurons that have the same hierarchy form a *layer* and each neuron is connected to all neurons in the previous layer. A deep neural network consists of at least one *hidden layer*, which are between the input and output layers. The connections between each neuron are not equivalent in general, and they are called *weights*. While training, both the weights and the so-called *hyperparameters*, which are the parameters corresponding to the architecture of the neural network, are fine-tuned in order to reach a sufficient prediction accuracy. If done correctly, the weights and the hyperparameters will represent the important features of the raw data.

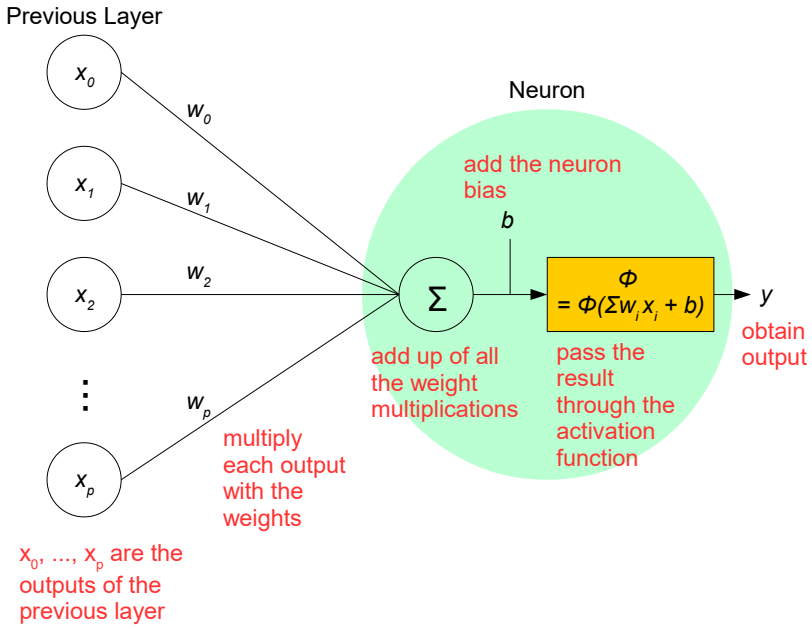
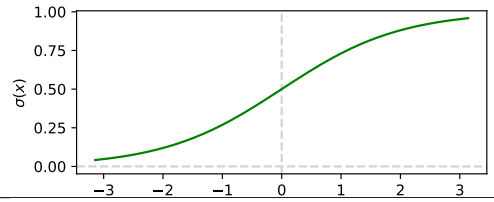
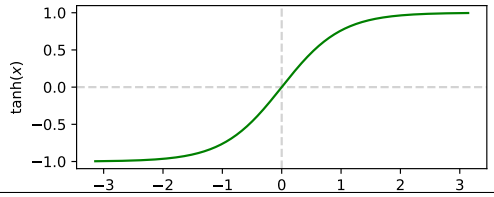
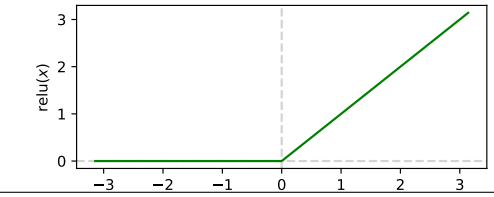
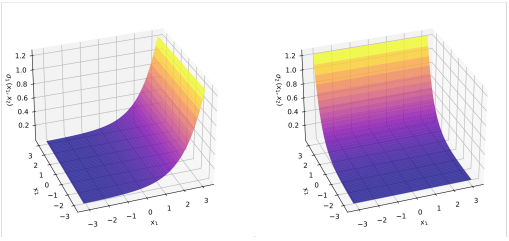


Figure 1.8: The operations performed for a neuron. The sum of input-weight products $w_i x_i$ the together with the bias b are passed through the activation function ϕ to produce the output y.

In most cases, the outputs of each neuron are summed up together with a bias vector b and passed through a function ϕ before serving as an input for the next layer.

ϕ is called an *activation function* and play a role in introducing non-linearities in the model and therefore enabling the learning of non-linear relationships between the input and output data. Some examples of activation functions used in practice are listed in Table 1.2.

Table 1.2: List of most used activation functions

Activation Fct.	*Equation*	*Plot*
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	
tanh	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	
ReLU	$\text{relu}(x) = \max(0, x)$	
Softmax	$\sigma(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}}$ for $k = 1, \ldots, n$ and $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ is the vector representing the layer	

The summary of operations done in evaluating a neuron are visualized in Figure 1.8. Mathematically we could write the output y in a compact way through vector multiplications

$$y = \phi(\boldsymbol{w}^T \boldsymbol{x} + b) \tag{1.5}$$

where $\boldsymbol{x} = (x_1, x_2, ..., x_p)$ is the input vector, $\boldsymbol{w} = (w_1, w_2, ..., w_p)$ is the weights vector, b is the bias value and ϕ is the activation function.

There are some layer architecture types of neurons that are more often than others, such as the ones listed in the following.

Fully-Connected (Dense) Layers

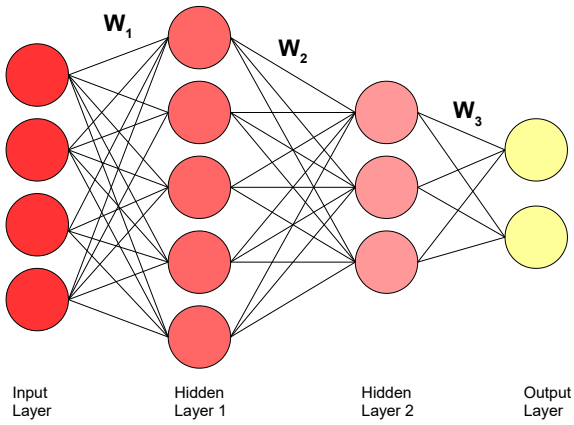


Figure 1.9: Illustration of a simple fully-connected neural network architecture with 2 hidden layers.

Fully-connected or dense layers are the simplest layer architecture for neural networks. Here all the neurons in a certain layer are connected to all the neurons in the previous and the next layers, like the one shown in Figure 1.9.

Based on the Equation (1.5) the output vector of a layer can therefore be written as

$$\boldsymbol{h}(\boldsymbol{x}) = \phi(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}) \tag{1.6}$$

where $\mathbf{W}$ is the *weight matrix* and $\boldsymbol{b}$ is the bias vector. Therefore, e.g. for our particular example in Figure 1.9 the output is

$$\boldsymbol{y}(\boldsymbol{x}) = \phi_3(\mathbf{W}_3\,\phi_2(\mathbf{W}_2\,\phi_1(\mathbf{W}_1\,\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + \boldsymbol{b}_y) \tag{1.7}$$

where $\mathbf{W}_i$ are the weight matrices, ϕ_i activation functions and $\boldsymbol{b}_i$ bias vectors of each corresponding layer.

As we can see, the evaluation of fully-connected dense architectures can be performed by matrix multiplications, vector additions and element-wise evaluation of the

activation function. Therefore, the total amount of arithmetic operations (addition and multiplication) in a dense layer is given by

$$N_{op} = 2 \times N_{\text{MACs},\mathbf{W}x} + N_{\mathbf{b}} = 2 \times (n_{\text{inputs}} \times n_{\text{dense}}) + n_{\text{dense}} \tag{1.8}$$

where N_{op} is the number of operations, $N_{\text{MACs},\mathbf{W}x}$ is the number of Multiply-Accumulates (MACs) from $\mathbf{W}\boldsymbol{x}$, $N_{\mathbf{b}}$ is the number of additions from $\boldsymbol{b}$, n_{inputs} is the number of input nodes and n_{dense} the number of nodes in the dense layer.

Convolutional Layers

For certain problems such as image classification, the correlation between neighbouring pixels is an important feature. The simple fully-connected dense architecture does not facillitate this functionality, and therefore we need *convolutional layers*.

To build convolutional layers, we need *filters* or *kernels* $\{k_i\}$. The values in these filters are the trainable parameter weights and after training, the filters will capture important features from the training input data. To calculate the output of a convolutional layer, we need to slide these filters across the input data and perform an element-wise product of the kernel and the overlapping input region. This is expressed mathematically for the i-th output row, j-th output column and the f-th output channel as [29]

$$o(i,j,f) = \sum_{c=1}^{C} \sum_{l=1}^{L} \sum_{m=1}^{M} x(i+l, j+m, c)\, w_f(l,m,c) \tag{1.9}$$

where x is the corresponding input region or input feature map, L and M are the height and width of the filters respectively and C is the number of input channels. For a better visualization, refer to Figure 1.10.

A parameter in the convolutional layer is the stride S, which is the distance between subsequent positions of the kernel.

Another thing to be decided is what happens at the edges of the input image. Some zero values on the edges might have to be added, and this is called *padding*. There are three padding scenarios

- VALID: No padding is used ($P = 0$). Output size will be reduced.
- SAME: We add $P = \lfloor \frac{K}{2} \rfloor$[1] zeroes on each side, where K is the kernel width/height. Output size will be preserved if $S = 1$.
- FULL: We add $P = K-1$ zeroes on each side, where K is the kernel width/height. Output size will be increased if $S = 1$.

[1] $\lfloor x \rfloor$ is the floor function. This has the input $x \in \mathbb{R}$ and outputs the greatest integer less than or equal to the input.

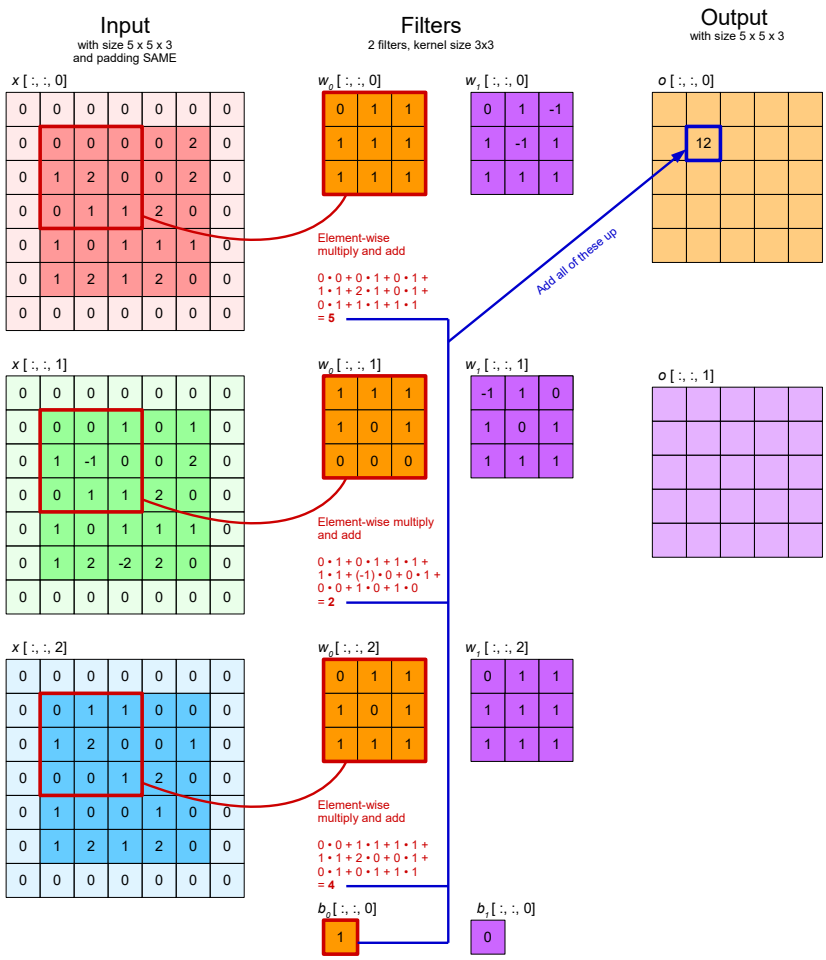


Figure 1.10: The visualization of a convolutional layer, for a three channel input (RGB), two filters and a stride of 1. Each filter also has three channels. The number of output channels are equal to the number of available filters. The values in the output are a result of the dot product and summation of the corresponding input region with the filters. The padding scheme used here is SAME.

In general, the number of arithmetic operations (additions and multiplications) for a convolutional layer is given by

$$N_{op} = 2 \times N_{\text{MACs, conv.}} + N_{\text{b}} \tag{1.10}$$

where N_{b} is the number of additions from the *bias matrix* $\mathbf{b}$ and $N_{\text{MACs, conv.}}$ the number of Multiply-Accumulates (MACs) due to convolution. This can be calculated

by

$$N_{\text{MACs, conv.}} = w_O \times h_O \times c_O \times w_k \times h_k \times c_I \tag{1.11}$$

with w_O the output width, h_O the output height, c_O the number of output channels (which is equal to the number of different kernels), w_k the kernel width, w_h the kernel height and c_I the number of input channels. The output dimensions are given by

$$\begin{aligned} w_O &= \lfloor \frac{w_I + 2P - w_K}{S} \rfloor + 1 \\ h_O &= \lfloor \frac{h_I + 2P - h_K}{S} \rfloor + 1 \end{aligned} \tag{1.12}$$

with w_I input width and h_I input height.

Most computer architectures support direct dot product convolution, but some do not, such as the Coral Edge Tensor Processing Unit (later introduced in Section 2.2). In this case, both the input data feature map (the region of the input image for one convolution) and the kernels have to be reshaped into general matrix multiplications (GEMM), for example through the `im2col` algorithm, as shown in Figure 1.11 .[30] The input data feature map is reshaped into a $h_O w_O \times c_I h_F w_F$ and all the kernels have to be concatenated into a $c_I h_F w_F \times c_O$ matrix. After multiplying these two matrices, the output feature matrix with the shape $h_O w_O \times c_O$ is obtained and this can be reshaped into channels.

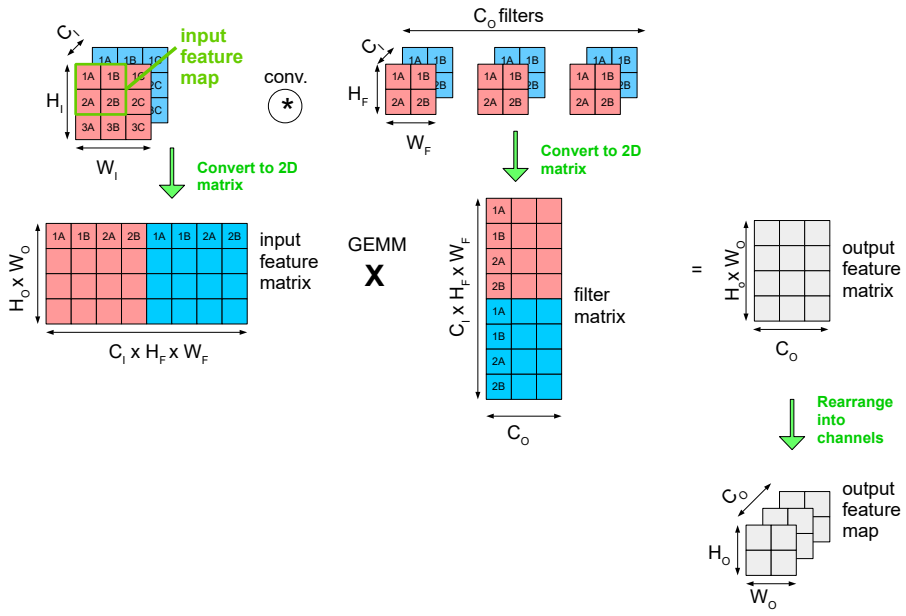


Figure 1.11: im2col algorithm for calculation of convolutions. In this algorithm, convolution operations are transformed into general matrix multiplication (GEMM) operations.

Pooling Layers

In between successive evaluations of convolutional layers, usually *pooling layers* are used, to reduce dimensionality and avoid overfitting. We can set a window with a certain *pool size* where we calculate a certain statistical quantity in this sliding window. The statistical quantities most used are maximum (max pooling), minimum (min pooling) and average (average pooling). The padding schemes are equivalent to convolutional layers. This is visualized in a simplistic way in Figure 1.12.

1.2.3 Model Compression

Several hardware cannot accommodate large models due to limited resources, e.g. memory storage and the limited dimensions of the processing element arrays. If we already have a pre-trained model that has a large size, it can be compressed for deployment in these hardware accelerators. Several examples of model compression techniques are quantization, pruning and knowledge distillation.

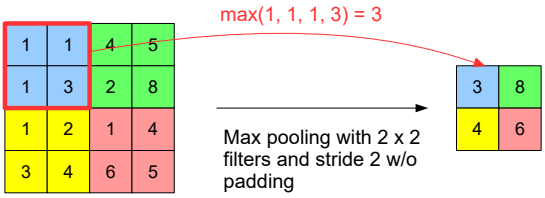


Figure 1.12: Visualization of the max pooling layer with 2x2 pool size and a stride of 2 without padding for one channel.

1.2.3.1 Quantization

Quantization reduces the number of bits to represent each of the model weight in exchange for a smaller and more compact model. Normal TensorFlow models have single precision values (Floating Point 32 or FP32), meaning each of the values in the model are encoded in 32 binary bits. A value with the precision FP32 has three main parts (based on IEEE 754 standard):

- sign bit (1 bit) - the sign of the number (positive or negative)
- exponent bits (8 bits) - the magnitude of the number in powers of 2
- mantissa/fractional bits (23 bits) - consisting of the significant digits

With this encoding we can represent numbers approximately between $1.18 \cdot 10^{-38}$ and $3.40 \cdot 10^{38}$ for both signs.

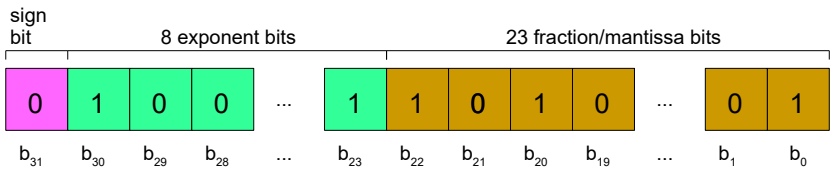


Figure 1.13: The structure of a FP32 number. 1 bit encodes the sign, 8 bits encode the binary exponent, 23 bits encode the mantissa part of the number (the significant digits).

TensorFlow models can be compressed to FP16 precision, meaning the model only need 16 bits to store each number. In FP16, the exponent bits are reduced to five and the mantissa bits are reduced to 10, and therefore reducing the range of which decimal numbers can be represented.

Another type of quantization is INT8 or UINT8 (unsigned INT8), where each of the values is represented by 8 bit integers. For INT8 the range is between -128 to 127 and for UINT8 0 to 255.

The extreme case of quantization is a binary neural network, where the weights are converted to 1-bit weights, i.e. -1 and 1. However, as expected, this binarization causes severe information loss and such models are difficult to optimize.[31]

1.2.3.2 Pruning

In deep neural networks, a lot of the weights do not contribute significantly to training and inference, e.g. due to their small values. Therfore, removing these would not remarkably affect the accuracy.

One can prune neurons or synapses (connection between neurons) and similarly for convolutional layers, one can also prune the convolution filters, whose L_1 or L_2 matrix norm is the lowest. Depending on the important criteria, one can decide which neuron, synapse or filter to prune. The easiest approach is to prune certain weights that do not reach a certain threshold or certain filters that have a matrix norm, e.g. the Frobenius norm

$$\|A\|_F = \left(\sum_i \sum_j |a_{ij}|^2 \right)^{1/2}$$

less than a certain threshold.

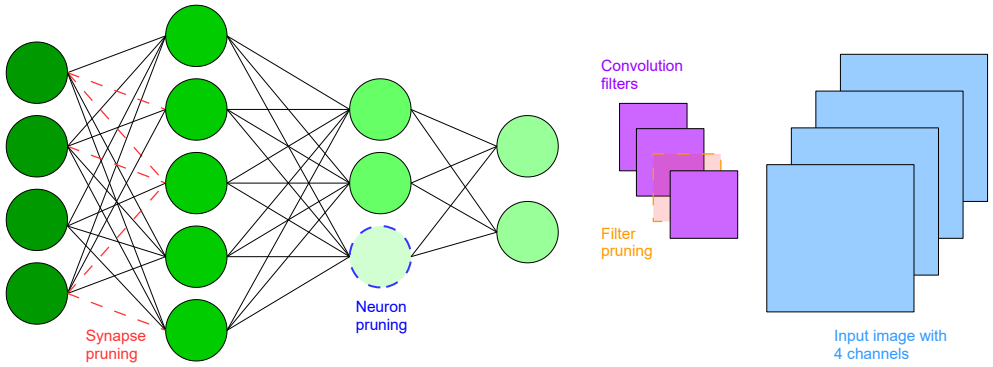


Figure 1.14: Pruning of neural networks. For fully connected layers, one can either prune the connections between the neurons (synapses) or the neuron directly. For convolutional operations, one can prune filters. Which neuron, synapse or filter to prune, depends on which criteria one considers as important.

Another approach is to use empirical measurements to guide the pruning procedure.[32] A latency and power consumption budget can be set in the beginning of the

process and the algorithm will measure the latency and power consumption of all possible pruning scenarios. The pruned model that results in the best accuracy and power consumption will be the one that is chosen for deployment.

1.2.3.3 Knowledge Distillation

Knowledge distillation involves a teacher model, which is larger in size, and a student model, which is a smaller model. The goal of training is to transfer the knowledge of the teacher model to the student model. There are a lot of algorithms of how the distillation works, which are also parts of active research.

The most used method of knowledge distillation is called offline knowledge distillation. In this case, during the supervised training of the student model, it is presented with both the true labels of the training data, called hard labels, and the output results of the teacher model, called the soft labels, which serve as regularizers that prevent overfitting. In the loss function, the hard and soft labels are then weighted based on their significance. These weights are hyperparameters that have to be fine-tuned during training.[33]

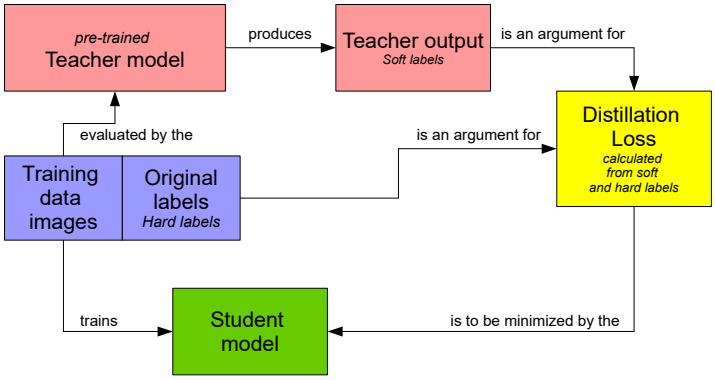


Figure 1.15: Offline knowledge distillation process. The teacher model is pre trained and the output of the teacher model serves as regularizers of the loss function to prevent overfitting. The original labels are also arguments of the loss function. The student model will find the weights that minimizes this loss function for a given training data set.

Knowledge distillation is already being applied in the ATLAS collaboration to compress the model into the FPGA as a trigger algorithm.[34] Here, instead of only taking into output labels of the teacher and student models, the intermediate outputs of different layers of both models are compared during training, because the former option leads to a non-converging loss function.

Chapter 2

Survey of Feasible Accelerator Chips

In this chapter, I will present the analysis of results extracted from the datasheet and publications regarding the candidate hardware accelerators.

2.1 Comparison Parameters of the Hardware Accelerators

Processing Time per Event or Throughput

Due to the expected high data rate in the order of 10^4 events per second, the primary factor in optimizing our accelerator choice is its processing speed. The performance speed of an accelerator can be quantified by the latency, which is the time difference between the time when the first event is entered into the accelerator and the time when the last event is finished being processed, for a series of input events. In general, this is not always the inverse of the throughput, i.e. how many events of input events can be processed per unit time, since for some hardware, the next input event can enter the hardware accelerator even before the current input event is finished being processed - this is called asynchronous inference.

Since not all hardware candidates support asynchronous inference, we will assume our metrics on synchronous inference, meaning an input event must be processed completely before the next input event can start being processed. In this case, the throughput is indeed the inverse of the processing time per event.

In datasheets, manufacturers only provide information on the number of theoretical maximum operations that can be performed per second by the hardware. But this is not meaningful, since the real event throughput depends strongly on the computing architecture, memory bandwidth, and the utilization of the accelerator chip by the algorithm. Therefore, for our purposes, evaluating the throughput should be done by running benchmarks operating on AFIS events, which are based on algorithms that are common for image processing, such as matrix multiplications and convolutions.

Power Consumption

Due to the power limitation on small satellites, all components of the PDP, including the hardware accelerator, must lie under this limit. However, like the throughput, the power consumption depends on the computing architecture and the utilization of the accelerator by the applied algorithm.

Interfacing from the FPGA

Part of the development effort is interfacing the FPGA host with the hardware accelerator for the communication and data flow from and into the SCM and the implementation challenges both on the hardware and software sides need to be considered in choosing the right accelerator. From the hardware point of view, we need to look at the available physical interfaces of the accelerator and which of them can be used for transmitting control commands and for sending and receiving data to be processed. On the software side, it is required to know which control protocols can be sent, whether these are sent as runtime commands during the operation or are transmitted in the beginning so that the accelerator acts as a dataflow machine and whether an operating system is required.

Algorithm Requirements and Optimization

The computing architecture of the hardware accelerator determines the required model size, quantization of models, and which operations and data dimensions are supported. Also, by analyzing the architecture, models can be optimized for maximal throughput and minimum power consumption.

Other factors

Since particle radiation in outer space can cause defects in the accelerators, the radiation tolerance of the accelerators is an important factor. Moreover, the size dimensions are a factor that can determine the form factor of the SCM printed circuit board. Side factors include the price, the availability in the market – whether it is readily purchasable or requires vendor contact, and the availability of information and support for these accelerators. According to its datasheet, the Coral Accelerator Module requires an input voltage of 3.3 V, has a physical size of 15 mm $\times$ 10 mm $\times$ 1.5 mm.[35]

2.2 Google Coral Accelerator Module with Tensor Processing Unit

The Google Coral Accelerator Module is a multi-chip module (MCM), containing the Tensor Processing Unit (TPU), and a power management integrated circuit. Designed by Google, the TPU is an ASIC to accelerate Tensorflow Lite machine learning models with high power efficiency. According to the datasheet [35], it is capable of performing 4 trillion operations per second (TOPS) with only 2 W of power, resulting in a theoretical efficiency of 2 TOPS/W.

Computing Architecture

The first generation of TPUs (TPUv1) was deployed in data centers in 2015 to accelerate only the inference of neural networks. The architecture of the first generation TPU is given in Figure 2.2. While the detailed architecture of the Edge TPU is not disclosed to the public, we assume that it has a similar architecture as its data center counterpart.

The TPU has a systolic computing architecture, which is an architecture whose principle is to maximize the number of computations per memory access to enhance the speed of computations before storing data back to a memory buffer, since buffer accesses introduce memory bottlenecks. The data flow from one processing element to another is managed by the system clock. However, the architecture becomes specialized and the types and order of operations must suit the characteristics and organization of the processing elements.[36]

The center of the processing architecture is the MXU or the Matrix Multiplication Unit. The TPUv1 consists of 256 x 256 MACs (Multiply-Accumulate Units) for 8-bit integer multiply-additions. As shown in Figure 2.1, one 8-bit MAC unit consists of one 8-bit multiplier, one 16-bit adder, and one 16-bit accumulator, which is a register to store intermediate sums.[37] If a is the intermediate sum, b and c are the inputs of

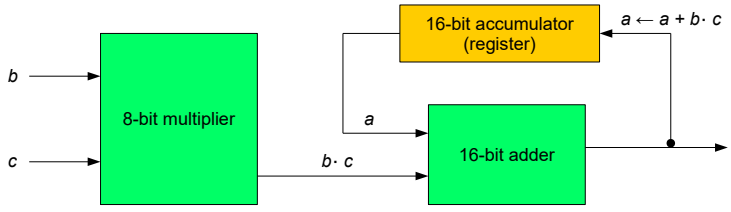


Figure 2.1: The basic block design of a 8-bit MAC unit

the multiplier, one MAC unit does the operation $a < -a + b \cdot c$ in each clock cycle, i.e.

adding the product of the multiplier inputs to the intermediate sum. In the Edge TPU, the dimensions of the MXU are estimated to be 64 x 64 MACs, since together with the maximum Edge TPU clock frequency of 500 MHz, this results in the theoretical maximum throughput of $64 \times 64 \times 2 \times 500 \times 10^6 \approx 4$ TOPS.[38]

The inputs of each layer are stored in the Unified Buffer (UB), which serves as a buffer for intermediate results of each model. meaning it is organized to minimize UB reading and writing interactions by relying on data arriving in each MAC cell in fixed time intervals.

The weights of the model are stored in external DDR DRAM memory chips and weights are stored by command in the Weight FIFOs on the other side of the MXU. In the Edge TPU, the weights are stored in an 8MB on-chip SRAM instead of an external RAM.[39]

After all matrix multiplication/convolution calculations are performed, the results are sent to an activation unit where non-linear activation functions will be evaluated. These activation functions are stored in a ROM.[38] The data from the accumulator can be also sent to a pooling unit where pooling algorithms for dimensionality reduction can be performed. Finally, the results are transmitted back to the Unified Buffer.

Since deep neural networks consist of mainly multiply-accumulate operations and this hardware accelerator consists of an array of processing elements in the form of MAC units, the TPU is suitable and specialized for neural network inference.

Interfaces and Instruction Set

For the TPUv1, both data and command instructions are sent through the PCIe bus via Direct Memory Access (DMA). The Edge TPU extends this with a PCIe Gen2 x1 and also USB 2.0 and USB 3.1 interfaces, but both data and command should be sent from the same interface because when instantiating the Coral TPU Device from a high-level API, only one interface must be specified.[41]

The instruction set for the TPUv1 follows a CISC (Complex Instruction Set Computing) approach, where one instruction needs 10 to 20 clock cycles. Unfortunately, the instruction set for the Edge TPU is not open source and is patented by Google [42] during the creation of this work. However, one may try to reverse engineer these using the partially open-source C++ driver code backend, which would not be an easy task.[41]

Supported Operations

The TPU is optimized for matrix multiplications and convolutions due to the presence of the MXU, which consists of MAC units. The convolution operations are converted into matrix multiplications through the `im2col` algorithm mentioned in Subsection 1.2.2.[30]

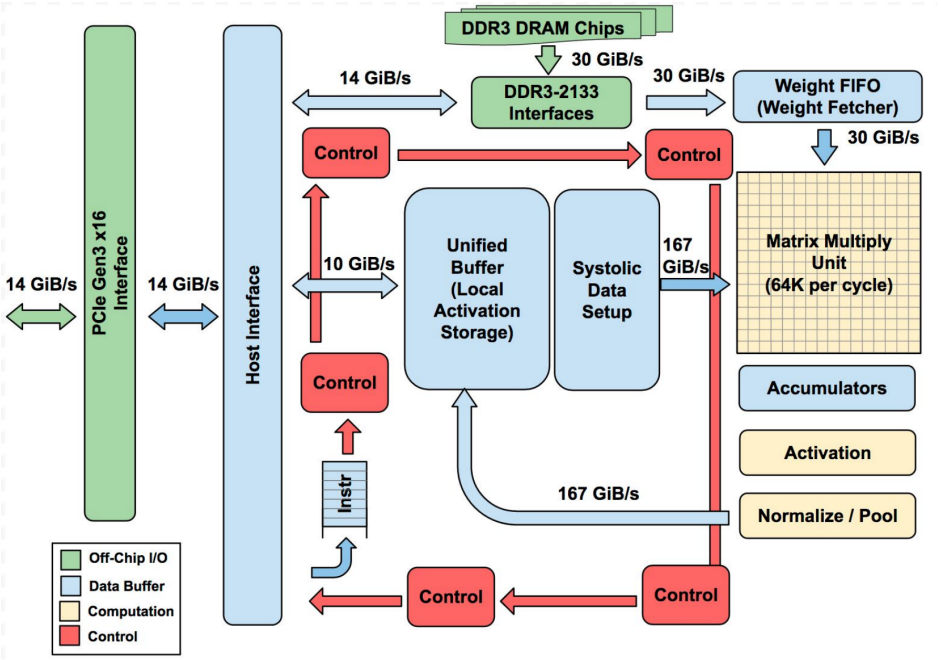


Figure 2.2: The computing architecture of a datacenter TPUv1.[40]

The limited size of the 8MB SRAM imposes a limit on the model size. If the model size is larger than 8 MB, a part of the weights will be stored in an external RAM, slowing the memory access and hence the whole inference process. [39] Because the MAC units perform 8-bit operations, the weights and the inputs must be fully quantized to INT8. The full list of supported operations can be found at [43].

Host System Requirements

For the Coral Accelerator Module, a 64-bit version of Debian 10 or Ubuntu 16.04 (or newer) can be used as a host OS with a host architecture of x86-64 or ARMv8, or a 64-bit version of Windows 10 with x86-64 host architecture.[35] In the host, the library `libedgetpu` has to be built, either locally or cross-compilation by e.g. Docker. Depending on the host architecture, the compilation will result in a `.dll`, `.dylib` or `.so` library file. Also, the Coral PCIe driver `gasket-dkms` must be installed in the host.

Algorithm Development Effort

Any integrated development environment (IDE) with Python or C++/C support can be used to develop and deploy models for the TPU.

To develop models for the TPU, the models have to be built in the Tensorflow machine learning framework and converted to Tensorflow Lite `.tflite` with INT8 quantization. These will be compiled with the Edge TPU Compiler.[39]

For deployment on the TPU, the `libedgetpu` library has to be built in the host, either locally or through cross-compilation with Docker. After that, one just need to use Tensorflow Lite as API for inference by importing this library at the start of the script.

Radiation Tolerance

The TPU has undergone TID (Total Ionizing Dose) irradiation tests, which are irradiation tests based on the material damage caused by ionizing radiation sources, and the results are given in energy per mass for a given material.[24] The TPU can withstand 50 Krad(Si) of radiation, making it suitable for LEO missions at around 1000 km altitudes and a 5-year lifetime with largely safety margins.[44]

NASA has also conducted experiments on the radiation tolerance of the TPU using heavy ions and protons and observed the rate of SEFIs (Single Event Functional Interrupts) per day, which is defined as the condition if a Single Event Upset (or radiation bitflip) leads to a temporary malfunctioning of the hardware component.[24] The results from these experiments, are that the contributions from heavy ions for geosynchronous/interplanetary missions during solar minimum are 0.017 SEFIs/day the contribution from protons is 0.00035 SEFIs/day. [45]

2.3 Intel Movidius Myriad X Vision Processing Unit

The Intel Movidius Myriad X Vision Processing Unit is a System-on-Chip used to accelerate the inference of deep neural networks and image processing with low power efficiency. According to [46] it can reach a theoretical maximum throughput of 1 TOPS on deep neural network inferences with a predicted power consumption of around 1 W.[47] The Myriad X VPU has a base frequency of 700 MHz.

Computing Architecture

In Figure 2.3 the computing architecture of the Myriad X VPU is illustrated. This centralizes around 16 Streaming Hybrid Architecture Vector Engine (SHAVE) processors, which is a processor microarchitecture produced by Movidius for the VPU. It is a hybrid of RISC and Very Long Instruction Word (VLIW) processors.

VLIW architectures have the goal of simplifying the hardware by eliminating the need of the hardware to perform dependency checking between the instructions within an

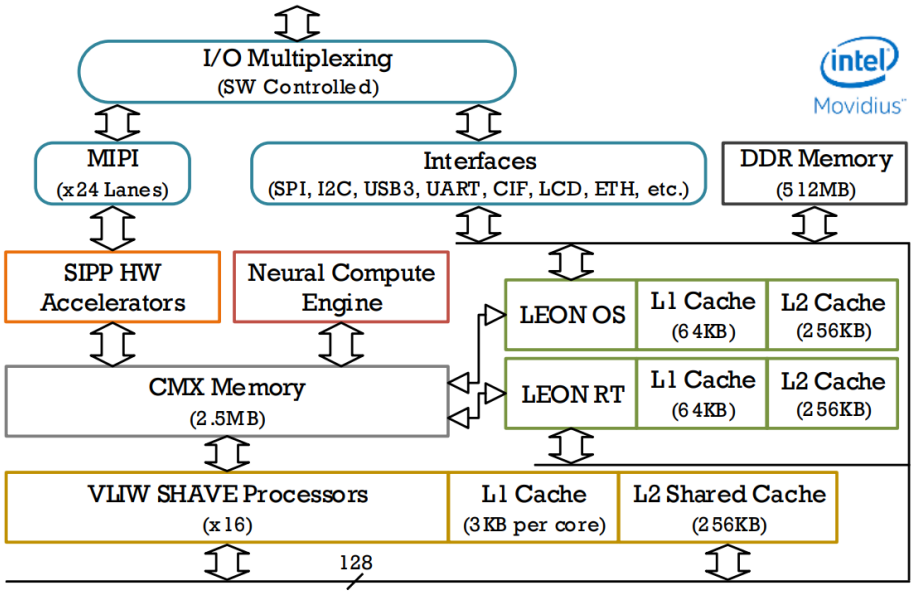


Figure 2.3: The computing architecture of the Intel Movidius Myriad X VPU [48, 49]

instruction word[1]. Therefore, this places the responsibility on the software compiler to ensure the independence of the instructions in the instruction word. The compiler must also contain the knowledge of the processor's execution units and their locations. Independent instructions are grouped in long words (for the case of the VPU, 128-bit long). Each operation can be executed simultaneously in parallel and requires a small and predictable number of cycles to execute.[50]

The microarchitecture of a SHAVE processor can be viewed in Figure 2.4 for the last generation of VPU, the Myriad 2 VPU. Each SHAVE has a $32 \times 128-$bit Vector Register File, a $32 \times 32-$bit general register file called integer register File (IRF), and 8 execution units [51–53]:

- 64-bit load and store units (LSU0 and LSU1) enables data transfer among SHAVE processors through the CMX
- branch and repeat unit (BRU) makes decisions on staying or switching execution pipelines through branch prediction
- predicated execution unit (PEU) calculates boolean values for the BRU
- 32-bit integer arithmetic unit (IAU) is in charge of integer arithmetic
- 32-bit scalar arithmetic unit (SAU) is in charge of scalar floating point arithmetic

[1]A word is a unit of data of a defined bit length.

- 32-bit vector arithmetic unit (VAU) is in charge of vector floating point arithmetic
- 128-bit compare move unit (CMU) coordinates the movement of data in the register files

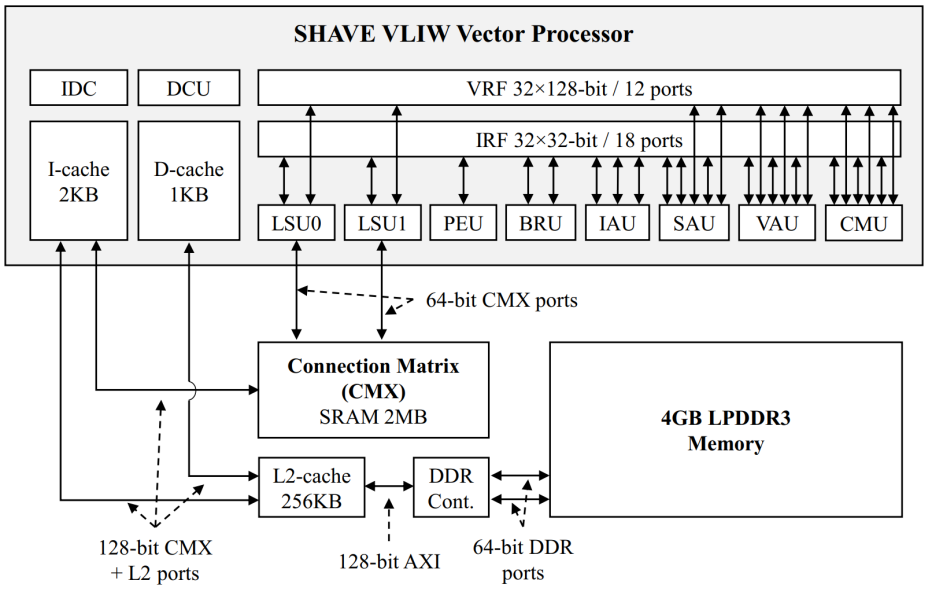


Figure 2.4: The architecture of a SHAVE processor in the Intel Movidius Myriad 2 VPU [51]

The SHAVEs together with the Neural Compute Engine, whose microarchitecture is not disclosed to the public, can achieve the mentioned theoretical maximum throughput for deep neural network inference. The Myriad X VPU is also equipped with Streaming Image Processing Pipeline (SIPP) processors to accelerate the processing of images transmitted through the 24 MIPI lanes.

The Myriad X VPU implements memory hierarchy. On top of the L1 caches for each SHAVE, there is also the Connection Matrix (CMX), a shared SRAM that is partitioned into 16 parts for each SHAVE processor, enabling seamless data exchange between them and acting as their collective buffer. The LEON processors are connected to the CMX to control the memory access scheduling of the SHAVEs so that the SHAVE cores are only used for computation. Finally, the lower stages of the memory hierarchy are covered by a shared L2 Cache for the SHAVEs and main memory with a size of 512 MB for the whole SoC.

Interfaces and Instruction Sets

As in Figure 2.3, the Myriad X VPU provides various interfaces, such as SPI, I2C, USB3, UART, CIF, LCD, and Ethernet. Also according to [54], the Myriad X VPU also supports PCIe 3.0. The low-level instruction set architecture (ISA), like the TPU, was not disclosed to the public during the creation of this work.

Supported Operations

The Myriad X VPU supports FP16 operations, meaning that the weights of the model should be quantized to FP16. The matrix multiplication in a VPU can be illustrated as follows. Let A and B be operand matrices of multiplication. Since we can mathematically partition matrix multiplication into smaller block multiplications, we will cache both matrices into the CMX and tile these into blocks. Now, each SHAVE core will perform one block multiplication. For each core, the matrix blocks A' and B' are loaded into the VRF. Due to the VLIW architecture, several instructions can be performed in parallel. The VAU can perform a vector multiplication of the row of A' and the row of $(B')^T$ and the SAU a horizontal sum operation on the products in the same machine cycle. The result of each multiplication block by each core will be stored in the main memory.[53] Finally, the model size cannot exceed 512 MB so that the whole model can fit inside the DDR Memory.

Host System Requirements

Based on the datasheet for the Neural Compute Stick 2, i.e. the USB module of the VPU, the Intel Myriad X VPU supports a 64-bit version of the operating systems Ubuntu 16.04.3, CentOS 7.4 or Windows 10 with the OpenVINO library installed in it and a host architecture of x86-64 or ARM.[55]

Algorithm Development Effort

Any integrated development environment (IDE) can be used to develop and deploy models for both the TPU and the VPU. The host just needs to have Python or C++/C support.

To develop models for the VPU, Tensorflow, PyTorch, ONNX and Caffe can be used as a framework to build the machine learning models. Using the Model Optimizer, these models will be converted into Intermediate Representation (IR) formats, which consist of `.bin` and `.xml` files. For the deployment, the OpenVINO library will be used as an inference API.

Radiation Tolerance

In the same NASA testing documentation as the TPU, it is reported that [45] 0.083 SEFIs/day occur for the Myriad X VPU for heavy ion testing and 0.0035 SEFIs/day occur for protons.

2.4 Mythic M1076 Analog Matrix Processor

The M1076 Analog Matrix Processor (AMP) is a coprocessor designed by Mythic for high-performance AI inference with low power consumption. The specialty of the AMP is instead of storing the weights on a conventional CMOS-based SRAM, the weights are stored in resistive potentiometers, called resistive RAM (RRAM) arrays, which eliminates the von Neumann bottleneck. This is defined as the performance slowdown in conventional computers due to the limited transmission speed of instruction and data from the main memory.[56] It can theoretically deliver 25 TOPS of performance with $3 - 4$ W of power and store 80 M weights for deep neural networks.

The AMP supports several interfaces such as PCIe Gen 2.1 with 4 lanes, GPIO, QSPI, I2C and UART. The instruction set of the AMP is not publicly disclosed. It is not exactly specified which host architectures and operating systems are supported, but some examples such as Intel x86, NXP iMX8, Nvidia Jetson and Qualcomm RB5 are given. [57] Due to the presence of the AMM, the AMP can support matrix multiplications. However, since there is no information regarding the dimensions of the AMM, we cannot find out the maximum number of nodes that can be calculated in parallel. The models are required to be quantized to INT4, INT8 or INT16 and retrained with quantization-aware training.[58]

2.5 Hailo-8 AI Acccelerator

The Hailo-8 AI Accelerator is an accelerator chip designed by Hailo, which has a theoretical maximum throughput of 26 TOPS, which exceeds the throughput of all the previously mentioned chips, with a power consumption of 2.5 W.

There is a lack of information available regarding the computing architecture of the Hailo-8 AI accelerator. The available information about the Hailo-8 AI Accelerator reveal that it consists of partitioned resources which act as control, memory and computational elements. Each layer of a neural network model is broken down into a resource graph, which contains resource elements, by the Hailo Data Flow Compiler. Afterwards, the compiler assigns the resources on the resource graph to the physical resources on the device, so that the data flow is optimized. According to [59], the Hailo-8 AI Accelerator has PCIe and Ethernet interfaces, which allow communication

with the host processor. The Hailo-8 supports x86 and ARM architecture hosts and with Linux and Windows.[59]

Summary of Parameters

In Table 2.1, I have provided a summary on the available information based on the discussed points. As mentioned before, GPUs and CPUs have a minimum power consumption of 5 W, hence they are excluded from the options for hardware accelerators, due to the power limitations set by small satellites. CPUs and microcontrollers are sequential architectures and therefore are not efficient for matrix multiplications and convolutions.

Table 2.1: Summary of important accelerator chip parameters based on literature research

Comparison Parameter	Google Coral Accelerator Module	Intel Movidius Myriad X VPU	Mythic M1076 AMP	Hailo-8 Chip
Max. theoretical throughput	4 TOPS	1 TOPS	25 TOPS	26 TOPS
Interfaces	PCIe Gen2 x1, USB 2.0/3.1	USB 3.1, PCIe Gen 3, Quad SPI, I2C, 16 MIPI lanes	-	PCIe Gen 3, 1-lane
Quantization	INT8	FP16	INT4/8/16	-
On-chip Memory	8 MB	512 MB	-	-

Chapter 3

Performance Evaluation

In the previous chapter, I briefly discussed the computing architecture and several other properties of the candidates of hardware accelerators on a high-level, which can be considered for answering the question of which accelerator to choose.

In the following, several self-built algorithms are tested in the hardware accelerator chips. Since the Mythic M1076 and the Hailo-8 AI Accelerator were not available in the market during the creation of this work, only two chips can be directly tested and verified of their performance – the Google Coral Accelerator Module with TPU and the Movidius Intel Myriad X Vision Processing Unit. For simplicity, I will mention these two chips as TPU and VPU. The running of this benchmarks has two goals:

- To measure the empirical throughput and power consumption of the hardware while processing simulated proton events
- To verify and characterize the computing architecture in order to establish the optimal neural network model specifications for each hardware

3.1 Performance Benchmarking and Power Measurement

3.1.1 Hardware Setup and Methodology

Here I will introduce the methods I used to extract the required comparison parameters of the accelerator chips. To do this I will use the USB modules of each accelerator chip for practical testing, i.e. Coral USB Accelerator with the Tensor Processing Unit and the Neural Compute Stick 2 (NCS2) with the Myriad X Vision Processing Unit, because they are plug-and-play modules and we do not need to disassemble the host computer to integrate these modules.

The Coral USB Accelerator comprises of an Edge TPU with 8 MB SRAM, a ISL91301B power management IC and a STM32L011D3P6 ultra-low power microcontroller [60]. The maximum current consumption in low-power run mode for the microcontroller is 40 µA [61], and with a voltage of 3.3 V this results in a maximum power consumption of 0.2 mW. As it can be seen later in the power measurement

results, this is only a very small portion of the power consumption of the whole USB module, meaning that most of the power consumption originates from the Edge TPU. Unfortunately, there are no sources regarding the constituents of the NCS2 other than the Myriad X VPU.

Throughput Measurement

The hardware setup to measure the throughput performance is shown in Figures 3.1. The testbench host is a Lenovo ThinkPad T440 with an Intel i5-4300U CPU with Haswell architecture, which has 2 cores running with a clock frequency of 1.90 GHz. The USB modules are connected to the host via USB 3.0 type A interface.

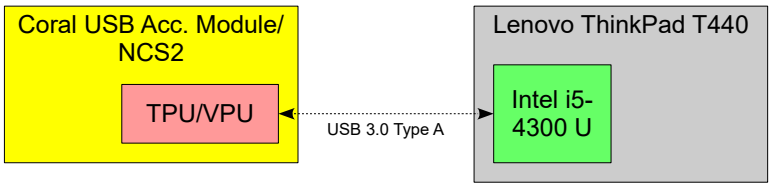


Figure 3.1: The test setup to measure the throughput of the TPU/VPU using the Coral USB Accelerator/Neural Compute Stick 2

The algorithms used for benchmarking must be converted to their respective formats. For designing my own benchmarks I used TensorFlow and exported this to TensorFlow Lite and frozen graph (`.pb` format) formats. For the TPU, I needed to perform full integer quantization of the weights on our TensorFlow Lite model and compile this with the Edge TPU Compiler into a model compatible with the TPU with INT8 quantization. For the VPU, we need to convert our frozen graph models into IR (Intermediate Representation) format using the Model Optimizer by OpenVINO, consisting of a binary (`.bin`) file and an `.xml` file.

To perform inferences on the models I used the frameworks TensorFlow Lite for the TPU and OpenVINO Inference Engine for the VPU. For some comparison, I also performed inferences on the host CPU also using the OpenVINO Inference Engine, because this framework optimizes the models for the CPU.

The only measured quantity in this setup is the inference time. Based on the example code from the Coral Github `classify.py` the method to record the inference time is using the Python `time` package, which we will use for both our measurements with the TPU and the VPU. The pseudo-code for measuring the inference time per event is

1. Make an event array containing the input data with a length equal to the set number of events (stored in host memory)

2. Load model into the accelerator on-chip memory

3. Record the start time using `time.time()`
4. For one input in the event array
   a) Send the input into the chip memory
   b) Send the command to start inference and wait until the inference is completed (synchronous inference)
   c) Get the output from the chip memory
5. Record the stop time using `time.time()`

The inference time per event is given by the difference between the start and stop time divided by the number of events. The throughput is just the direct inverse of the inference time per event. Similar to the approach in [40], I also used the metric Operations per Second (OPS) as a measure of performance, which is the product of the throughput and the number of operations in the model, because this metric takes into account the computational workload for a model.

I also considered whether the USB interface could be a transmission bottleneck, which is the case when the time to transmit the data via the USB interface is longer than the time to do the calculation itself. In order to do this, I measured the average time to set one input into the chip memory (in the pseudo-code, by skipping steps 4b and 4c). The measured transmission times for input data like in Figure 1.5 are in the order of 0.1 to 0.2 ms. As we can see later, the inference times for the benchmarks are mostly longer than 1 ms, and therefore the USB interface still can be used for our purposes.

Power Measurement

To measure the power consumption of the chips, I considered finding electrical pins in the development boards containing the corresponding accelerator chips where I could measure the supplied current to the chip using a laboratory voltmeter. However, this is tricky for the TPU, since having access to its pins on the Coral Dev Board requires us to disassemble the System on Module, wherein the TPU resides, from the baseboard. This may be impractical and forces me to find another approach.

The other approach is to use a USB voltmeter,[62] as similar to the approach in [63]. The downside of this approach is that the power of the whole module is measured, instead of the sole power consumption of the chip. But since most of the power consumption in the USB modules still comes from the accelerator chips themselves, it is still a good approximation. Therefore, the results of the power measurement shall be seen as only a rough approximation of the power consumption of the chip. The measurement setup for the power consumption is shown in the Figures 3.2.

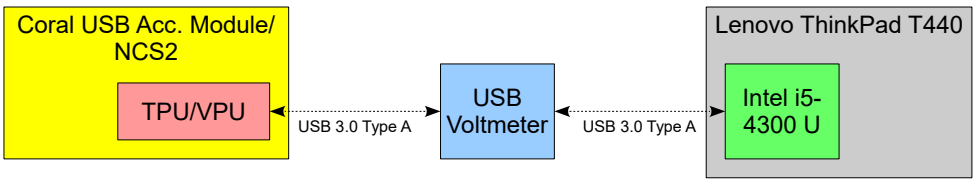


Figure 3.2: The test setup to measure the power consumption of the TPU/VPU using the Coral USB Accelerator/Neural Compute Stick 2

To obtain the approximate power consumption, I recorded a video during the inference and I noted the value for the power in the voltmeter every second for 30 seconds. After that, I took the mean and the standard deviation of the recorded values for each case. Both static power, i.e. the power consumption of the accelerator while not performing computations, and dynamic power, i.e. the power consumption of the accelerator while performing operations, for each algorithm were measured. For every benchmark, I also calculated the number of operations per second per Watt (OPS/W), which serves as a measure of the power efficiency of the hardware accelerator.

3.1.2 Benchmarking Results

In this subsection, I will present the throughput and power consumption results of the measurements originating from the deployment of different neural network architectures, starting from the simplest architectures to the more complex ones.

3.1.2.1 Matrix Multiplication

As the first and simple benchmark, I used dense layers, which consist of matrix multiplications and a bias vector additions, as benchmarks to investigate the maximum number of vectorizable operations that can be performed for each hardware. The architecture of the benchmark is shown in Figure 3.3 and I call this `matmul`. I varied the input vector and the number of nodes in the dense layer in powers of 2. For the TPU, both the inputs and the weights are in INT8 format as well as the VPU, both the inputs and weights are in the FP16. I measured the total inference time for 1000 multiplications for each multiplication model and took the mean times.

In Figure 3.4, I plotted the second base log of the input size and the second base log of the number of the dense layer nodes (which is simultaneously also the output size) against the 10^{th} base logarithm of the processing time and the performance in terms of Giga Operations per Second (GOPS), taking into account the computational workload for a model. It can be observed that for input and output sizes less than 2^{12} the measured processing time per event is constant at around 0.4 ms and increases if

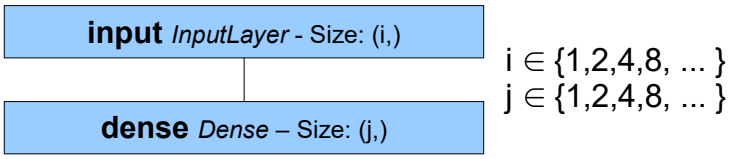


Figure 3.3: The architecture of `matmul` to benchmark matrix multiplications

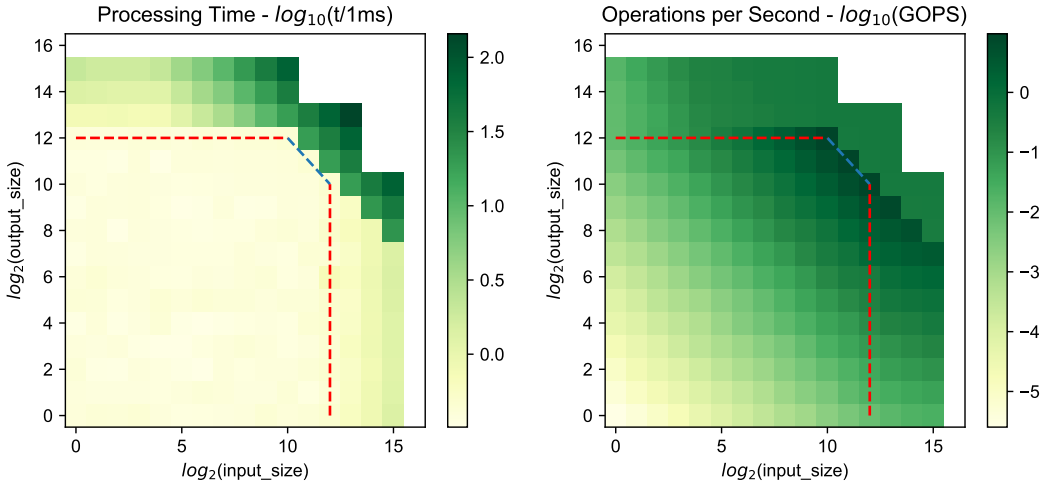


Figure 3.4: The performance evaluation of `matmul` for the Edge TPU

either the input size or the output size exceeds 2^{12}. This implies that the TPU can maximally perform 2^{12} operations in parallel, in accordance to the size of the MXU predicted in Section 2.2, which is $64 \times 64 = 2^{12}$ MAC units. This means that the calculation for one element in the output matrix should not require more than 2^{12} MACs for optimal use, or in other words, if $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times l}$ and we are calculating AB, then the inner dimension n of the factor matrices in the multiplication shall not be larger than 2^{12} for optimal use.

When the product of the input size and the output size is larger than 2^{21}, the compiled `.tflite` model is larger than 8 MB and therefore cannot fit into the SRAM of the TPU. This results in a strong spike in processing time, as shown in Figure 3.4

In Figure 3.5 I plotted the processing time and operations per second for constant input size 2^{13}. For output sizes smaller than 2^{10}, the processing time stays constant regardless of the output size. A significant increase in the processing time can be observed due to the access of data from external memory for output sizes larger than or equal to 2^{10}, because the model size is larger than 8 MB. Therefore, for the TPU it can be concluded that all matrix multiplications (and also convolutions,

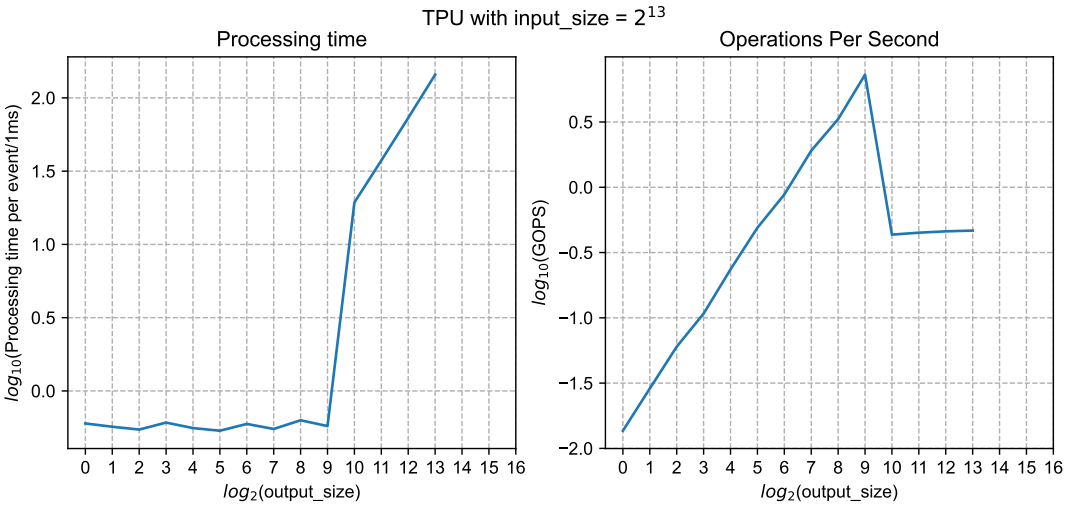


Figure 3.5: The performance evaluation of `matmul` for the Edge TPU for constant input size 2^{13}

since all convolutions are converted into GEMMs in the TPU) are memory bound only. Memory bound is the case when the memory access requires more time than the computations of the hardware, meaning that the resulting throughput is limited by the memory access.

For the VPU, the results can be viewed in Figure 3.6. Like the TPU, the processing time is approximately constant for input or output sizes below 2^{12} at around 2 ms and there is a flattening of the performance when the input or output size reaches 2^{12}. This can be explained due to the total number of FP16 values that can be stored in all of the Vector Register Files (VRFs) of the SHAVEs. Since one of the VRF can accommodate 32×128 bits of data, this means that 256 FP16 values can be stored in one VRF. Since we have 16 SHAVE cores, this means that we can store $2^4 \cdot 2^8 = 2^{12}$ FP16 values in the VRFs and process them in parallel. All tested `matmul` models fit in the 512 MB main memory of the VPU, and therefore there are no sudden performance drops like the one in the TPU.

I also plotted the processing time and operations per second for constant input size $= 2^{13}$. The logarithm of the processing time increases exponentially after reaching output size 2^7 also where the number of operations per second starts to flatten. This is the region where the VPU is compute bound, which is defined as the case when the computations require more time than the memory access.

For matrix multiplications with dimensions lower than 2^{12}, the TPU can calculate more than four times faster than the VPU. However, the VPU is more performant for large matrices, reaching 0.1 TOPS, due to its larger main memory. Therefore, the TPU can be preferred for small matrix multiplications and the VPU for larger ones.

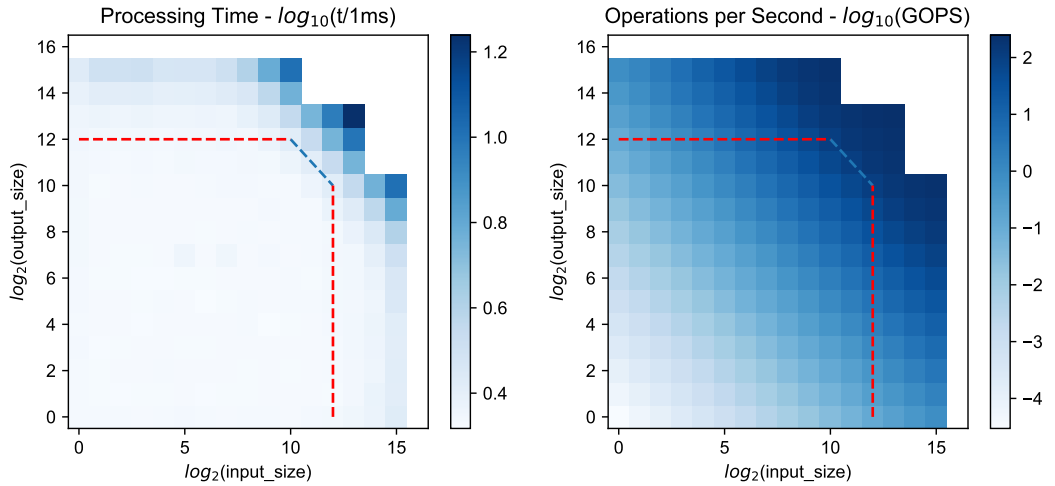


Figure 3.6: The performance evaluation of `matmul` for the Movidius Myriad X VPU

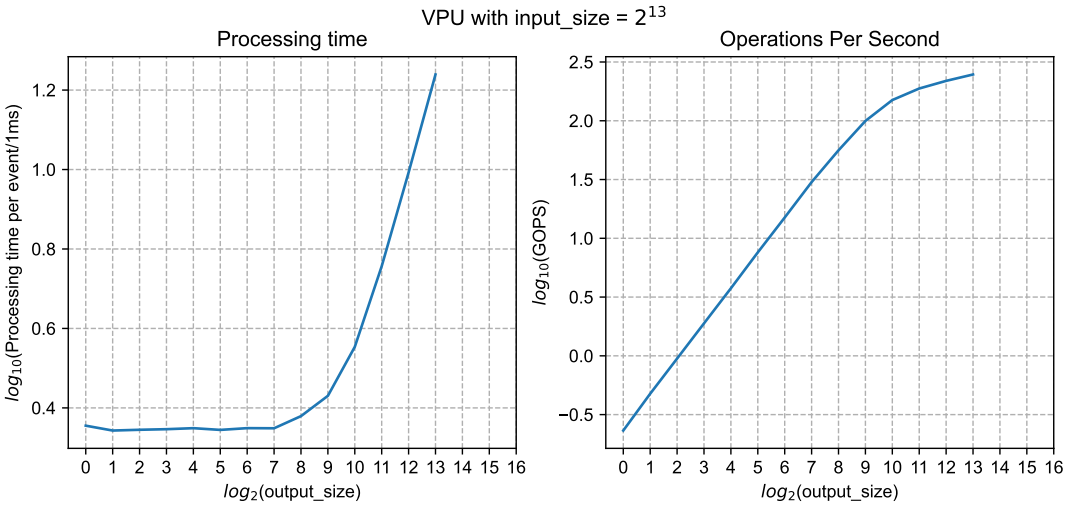


Figure 3.7: The performance evaluation of `matmul` for the Movidius Myriad X VPU for constant input size 2^{13}

3.1.2.2 Convolution Operations

In image classification neural networks, in order for it to recognize the features in the data that is fed into the neural network, correlation between neighbouring pixels are important. Convolutional layers are able to store these features in its filter weights, and that is why they are applied in most image classification networks. Hence, it may be useful to see how the effects of the convolution hyperparameters affects the

performance of the accelerator chips. For this, I made custom CNNs with varying hyperparameters. For all the following models, I used the padding SAME so that all models have the same number of convolution steps.

For each benchmark, I used 10000 sample events of 32×32 sized data, consisting of simulated proton events that are incident from all directions. These events include low-energy protons that are fully stopped in the detector, high-energy protons that pass through the detector completely and high-energy protons that interact with the detector material to create secondary particles. The pixel values are represented by floating point numbers, which show the amount of deposited energy in a scintillating fiber in units of MeV. This dataset is somewhat representative because the majority of events in the AFIS mission expectedly involve protons. However, it is not entirely representative as it lacks antiproton events. Also, in the planned implementation for data acquisition by the on-board data processing, the pixel values are represented by 12-bit numbers instead of 32-bit.

To measure the performance, I measured the processing time and repeated this 10 times. From all the repetitions, I took the mean and the standard deviation, the latter visualized in the following as errorbars. The measurement of the power consumption is described in Subsection 3.1.1.

Effect of Kernel Sizes

In the following the effect of different kernel sizes for convolution operations is observed for all the available hardware accelerators. The kernel sizes determine the size of the filters which store the weights corresponding to the input features. The larger the kernel size, the larger the input features that can be recorded by the convolutional layer. Since the size of interesting features in our events, such as particle showers and particle tracks, vary in size, a mix of convolution layers with different kernel sizes can be most likely used.

For the evaluation on the effect of kernel sizes, I built models with architectures shown as in Figure 3.12, which I denote collectively as `conv_kernel`, where each model consist of only an input layer and a subsequent convolution layer with varying convolution kernel sizes.

I used odd kernel sizes, because they are mostly preferred. [64] Filters with these sizes have a central pixel that symmetrically divides the input feature region, ensuring the capture of information in a balanced fashion in all directions. In contrast, even number kernel sizes can cause asymmetry.

I also did not use kernel size 1, since this is only used for dimensionality reduction (see Subsection 3.1.2.5) and information regarding neighbouring pixels in the input feature map are not captured in the filters.

From now on for all figures representing neural network architectures, let F represent the number of convolution filters, KS the kernel sizes and S the strides.

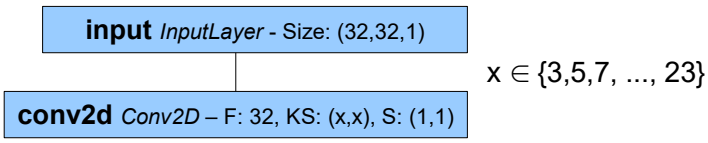


Figure 3.8: The architecture of the models `conv_kernel` to test the effect of kernel sizes in convolutional layers for different hardware accelerators

The throughput measurement results for `conv_kernel` can be viewed in Figure 3.9. The results for the CPU are plotted for comparison. For small kernel sizes, there is less parallelization of operations, that is why the CPU has better performance than both accelerators due to its higher clock frequency and higher power consumption even though it performs the computations sequentially. As the kernel size increases, the computational intensity increases, and the performance of the CPU shows an upward trend until kernel size 19 and reaches a maximum there. At this point, the processes in the CPU transition from being memory-bandwidth bound, i.e. when the cores are not fully utilized for the computation and hence the computations are limited by the speed of the data access, to computationally bound, i.e. when the cores are fully utilized and therefore reach the maximum number of computations that can be executed in parallel. There is an anomaly in the performance for kernel size 7, whose explanation might lie behind the optimization algorithm of the model by OpenVINO for the CPU.

All of the `conv2d` operations in `conv_kernel` translate through the `im2col` algorithm to GEMMs with matrices of size $32 \cdot 32 \times x^2$ and $x^2 \times 32$ where x denote the kernel size. Since $x^2 \leq 23^2 < 2^{12}$, all matrix dimensions are smaller than 2^{12} and hence there are no output matrix elements that require more than 2^{12} to be calculated in parallel. This explains why the TPU's processing time per event stays constant and does not increase for all kernel sizes.

For kernel sizes below 17, the processing time per event stays constant for the VPU but larger than the TPU, because the VPU has lower memory bandwidth, possibly due to the shared use of the CMX between 16 SHAVEs that requires strict scheduling.

Remarkably, there is increase in processing time for kernel size 17 and this increases linearly afterwards, implying that from this point forward, there is a sudden change in how the memory is accessed. To investigate this further, I have increased the input size to 128×128 in order to increase the model size and used randomized FP16 inputs. A sudden drop in performance occurs also for kernel size 17, as shown in Figure 3.10, suggesting that the performance drop is not caused because the model size is too large to fit in the cache, as observed for the case of the TPU for matrix multiplications in Subsection 3.1.2.1. A possible interpretation of this drop in performance has to do with

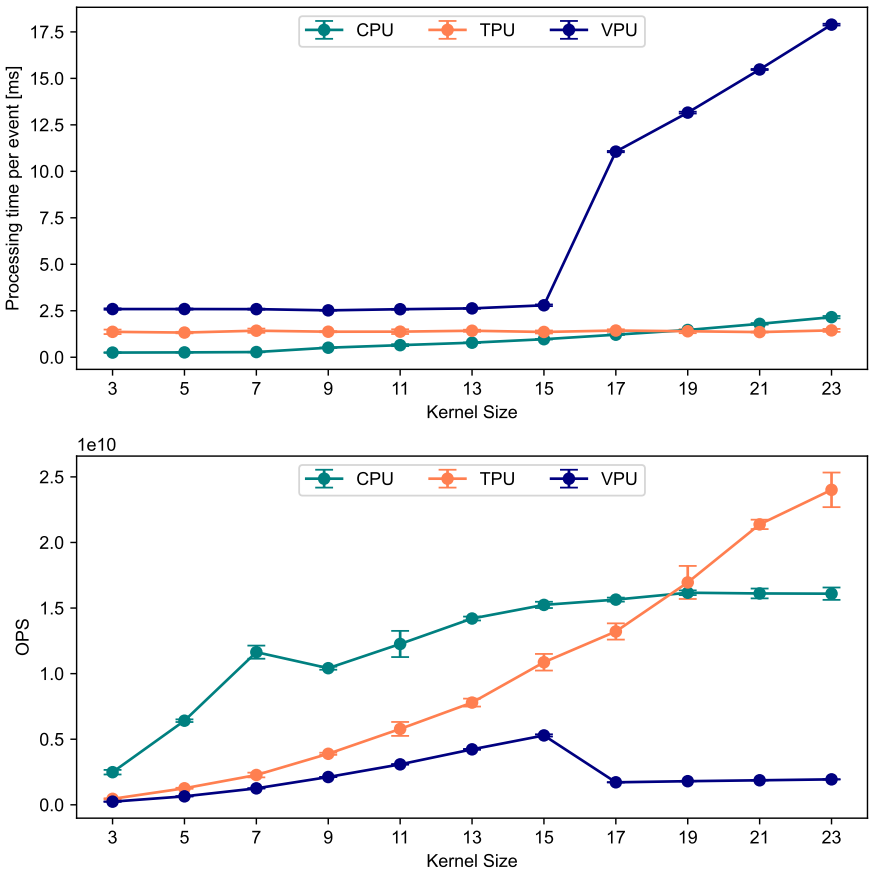


Figure 3.9: The performance evaluation of `conv_kernel` for different kernel sizes

the width of the Vector Register Files (VRF) sizes of the SHAVE processors. Since they are 32×128 bits wide, one VRF can accomodate 256 FP16 values. Therefore, if the kernel size is larger than 16, the input feature map can not fit in the VRFs all at once and therefore might need an extra clock cycle to process the whole input feature map, cumulatively increasing the latency dramatically. Since there are more convolutions that have to be done for 128×128 input size, the rate at which the processing time per event rises for kernel sizes larger than 16 is higher.

In conclusion, the TPU outperforms the VPU across different kernel sizes, due to its higher memory bandwidth.

Afterwards, the dynamic power consumption of both accelerators for `conv_kernel` can be observed in Figure 3.11. For the VPU, the power consumption rises with increasing kernel size, since the computation intensity of the algorithms increases,

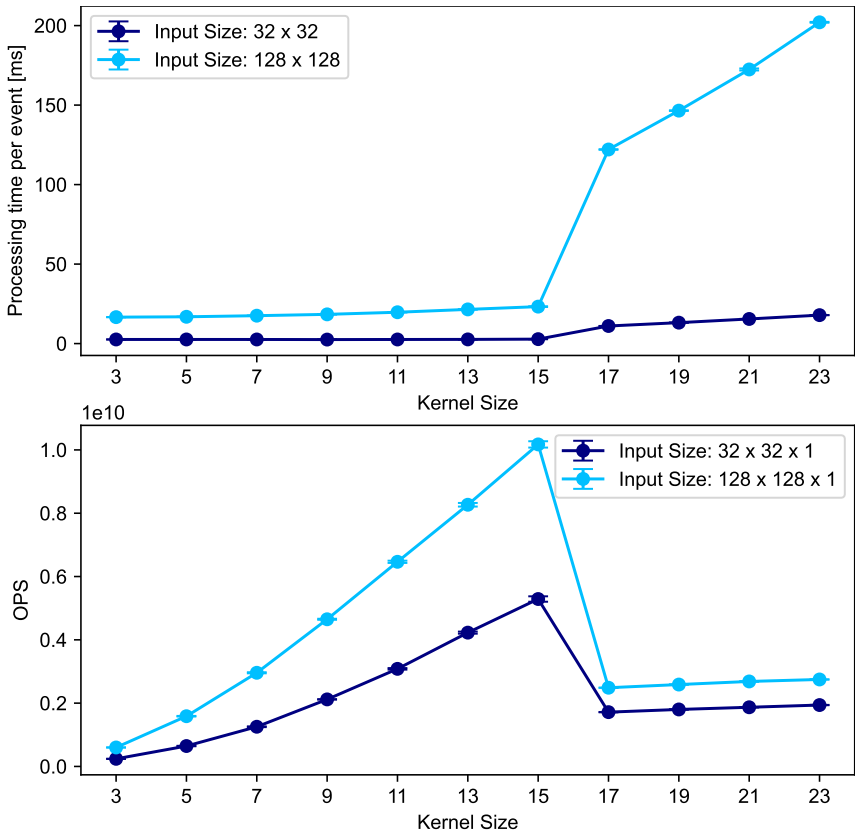


Figure 3.10: The performance of the VPU drops when the kernel size is larger than 16, independent of the input size.

more cache files are accessed and more SHAVE cores are utilized for the computation. The sudden rise in power consumption for kernel size 17 supports the theory with the change of memory access, since memory access from higher hierarchies of memory are usually significantly larger. Meanwhile, the power consumption of the TPU stays fairly constant for increasing kernel size. This is because always all the MAC units of the TPU are utilized no matter the kernel size. Moreover, this also means that the cache file lengths of the TPU are long, so that the number of cache files accessed for different kernel sizes stay the same.

Due to the sudden increase in power consumption and sudden decrease in performance for the VPU at kernel size 17, there is a significant drop in the power efficiency of the VPU, while the power efficiency of the TPU always increases.

All in all, the TPU comes out better than the VPU in terms of operations per second per watt, due to the TPU's lower clock frequency and systolic data flow architecture, which reduces the number of memory accesses needed for the computations. In con-

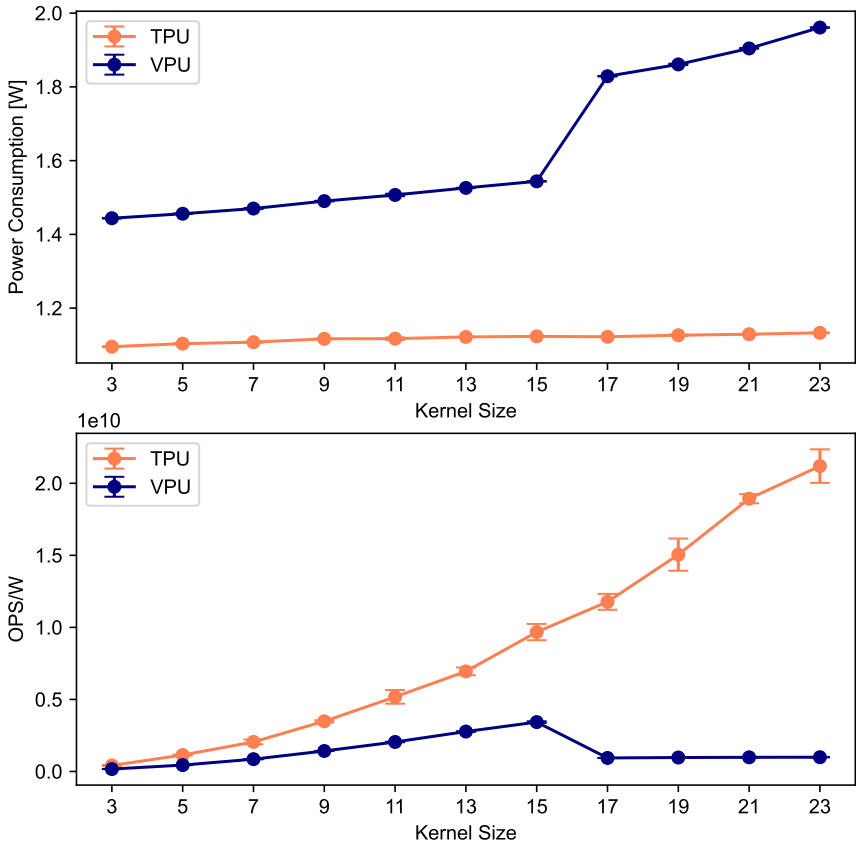


Figure 3.11: The power efficiency evaluation of `conv_kernel` for different kernel sizes

trast, the VPU has a higher clock frequency and since it does not have a systolic data architecture and its multi-layered memory hierarchy, the number of memory accesses is higher and hence increases the power consumption.

Effect of Number of Convolution Filters

The number of convolution filters is another hyperparameter of a convolutional layer. To test the effect of this hyperparameter on the hardware performance, I built test neural network models `conv_channels` with an architecture shown in Figure 3.12. The first convolutional layer, `conv2d`, will give an output image that has the number of channels equal to the number of convolution filters. The output image will serve as the input for the second convolutional layer `conv2d_1`. The mostly used number of filters for convolution layers is between 32 to 512,[65] therefore I varied the number

of filters for conv2d from 1 through 512 in powers of 2. I used padding SAME for all convolutional layers.

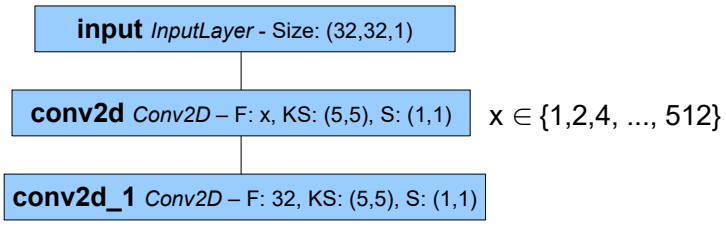


Figure 3.12: The architecture of the models conv_channels to test the effect of the number of filters in convolutional layers for different hardware accelerators

The throughput performance evaluation of conv_channels can be seen in Figure 3.13. It can be observed that the processing time per event for the CPU increases quadratically in general. Like the case for different kernel sizes, there are anomalies for 2 and 4 convolution filters. Maybe these lie in the optimization algorithm from OpenVINO for the CPU.

The convolutions for the TPU are converted in the form of GEMM of size $h_O w_O \times c_I h_F w_F$ and $c_I h_F w_F \times c_o$. For conv2d_1 $h_O = w_O = 32$, $h_F = w_F = 5$, $c_o = 32$ and $c_I = x \in \{1, ..., 512\}$, hence corresponding to factors with size $2^{10} \times 25x$ and $25x \times 2^5$. As we have seen in Subsection 3.1.2.1, the TPU is used effectively if the GEMM dimensions are all less than 2^{12}, i.e. when $x < 163$. Hence, a slight increase in the processing time can be observed for $x = 256$ and $x = 512$. Nevertheless, the number of operations per second performed by the TPU still increases quadratically.

For the VPU, a slight increase in the processing time can also be observed for $x = 128$, $x = 256$ and $x = 512$. Since the 16 SHAVEs can accommodate in total 4096 FP16 values, then if the input feature map of one convolution $5 \cdot 5 \cdot x > 4096$, i.e. when $x > 163$, the calculation will need an extra clock cycle to finish the convolution, increasing the time to complete the computations for one layer.

The TPU is overall superior in terms of throughput to the VPU, like the case for conv_kernel. For both accelerators, the power consumption for increasing x rises. This shows that the number of memory access increases with number of input channels of the second layer conv2d_1.

The TPU is overall superior in terms of operations per second per watt to the VPU as well, like the case for conv_kernel.

Effect of Strides

The last varied hyperparameter for convolution layers is the stride, which is the pixel distance between the subsequent positions of the kernel filters of the convolutions. If

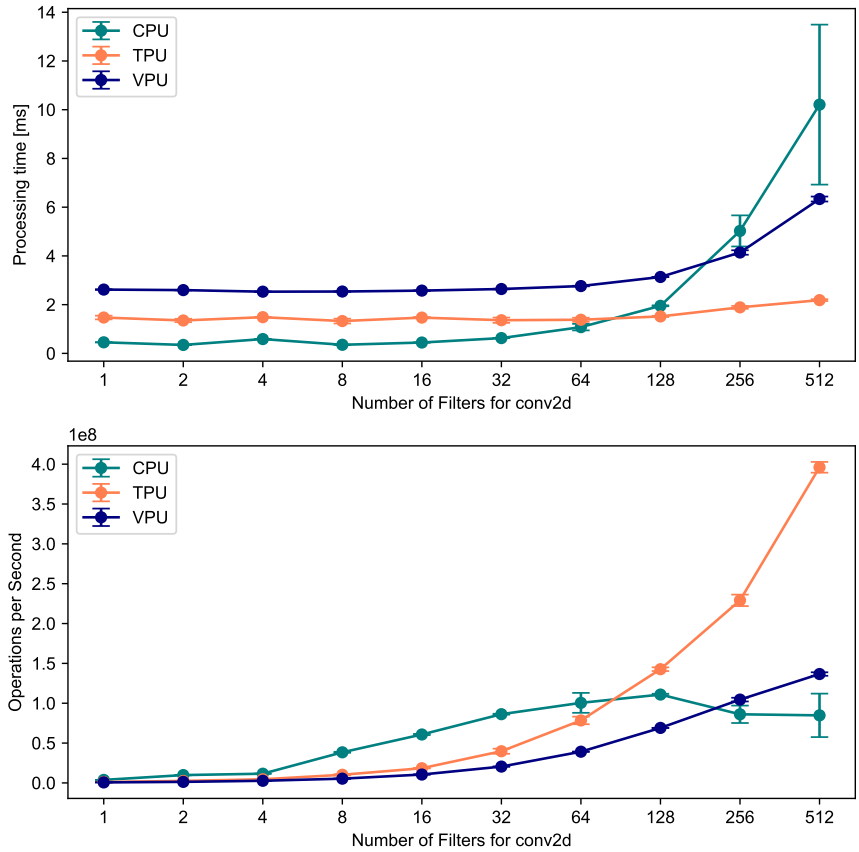


Figure 3.13: The throughput performance evaluation of `conv_channels` for different number of convolution filters of `conv2d`

we set the stride > 1, the convolutional layer can be used as a means of dimensionality reduction like pooling layers. In most cases, the stride is often set to 1 or 2.[66] The reason for testing the effect of the convolution strides is because there is a strong suspicion by [30], that the TPUv2[1] utilizes the channel-first SRAM layout. Here, the data pixels are arranged in the Unified Buffer in such a way that corresponding pixels from the same row and column coordinates but different channels are spatially located next to each other. The memory address generation becomes simple and therefore reduces the memory overhead. Whether this is also the case for the Edge TPU or the VPU, is going to be observed in the following.

To test the effect of convolution strides, I have built models `conv_stride` shown in Figure 3.15, where I have varied the convolution stride of the convolutional layer.

In Figure 3.16, the processing times for each event for the CPU stays constant,

[1] the second generation of the TPU, for datacenter applications

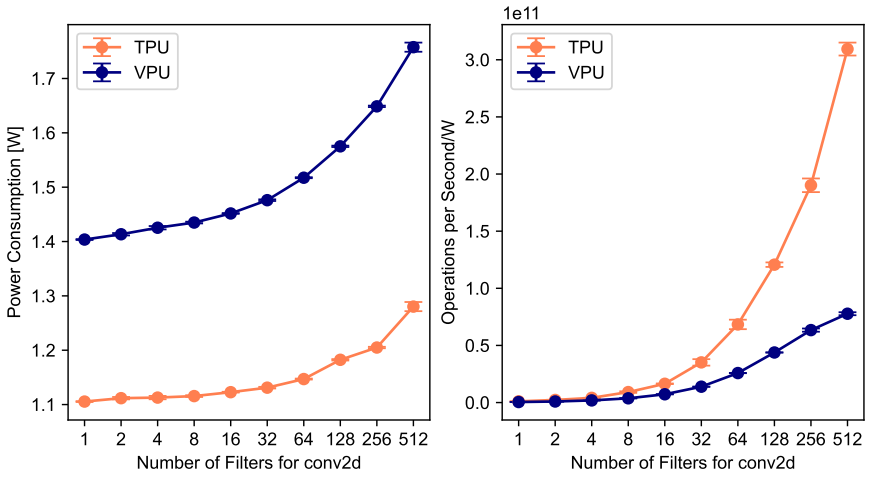


Figure 3.14: The power consumption evaluation of `conv_channels` for different number of convolution filters of `conv2d`

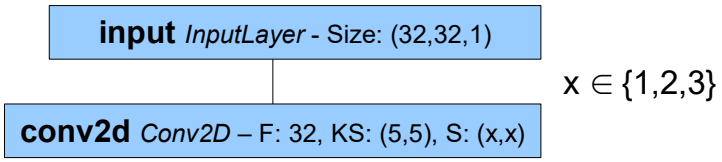


Figure 3.15: The architecture of the models `conv_stride` to test the effect of stride in convolutional layers for different hardware accelerators

even though the number of convolutions is reduced for strides > 1. This means that the throughput decreases with increasing stride, possibly caused by the fact that the address generation takes some time, since they possibly do not use the channel-first memory layout, causing discontinuous access to the SRAM and low bandwidth utilization. In contrast, for both accelerators the processing times decrease accordingly to the decrease of number of convolutions, and therefore the decrease in throughput is not as large as for the CPU. This information might tell us that both the Edge TPU and the VPU uses the channel-first memory layout, resulting in continuous access to the SRAM and utilizing the full bandwidth of the memory access.

There is a slight decrease of operations per second for both accelerators, due to the less number of operations in the models. The power consumption evaluation for `conv_stride` is given in Figure 3.17. For the VPU, the power consumption stays constant with increasing stride while the operation per second per watt decreases.

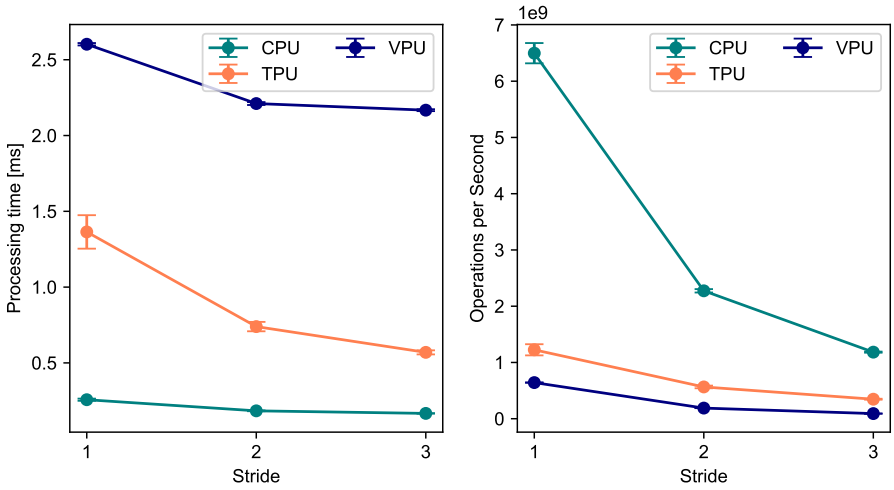


Figure 3.16: The throughput performance evaluation of `conv_stride` for different convolution strides

This is because while the number of memory accesses from the CMX are the same, the number of operations operating on the fetched pixel data decreases.

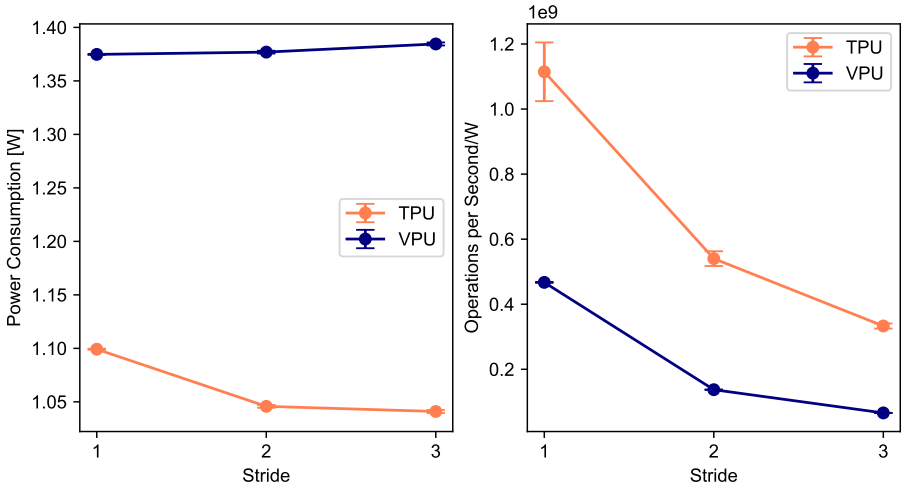


Figure 3.17: The power consumption evaluation of `conv_stride` for different convolution strides

In conclusion, strides > 1 shall be avoided if possible for both accelerators, since the power efficiency will be decreased.

Effect of Activation Functions

In the following, I would like to measure the execution times of the most commonly used activation functions, which are `linear`, `relu`, `sigmoid`, `softmax` and `tanh`. However, these execution times are too fast that the USB interface becomes a data bottleneck. Therefore, I had the idea of attaching the activation layers after a convolutional layer, as in Figure 3.18 and compared the results. I measured the processing time for 10000 events and repeated this 10 times to improve the statistically accuracy.

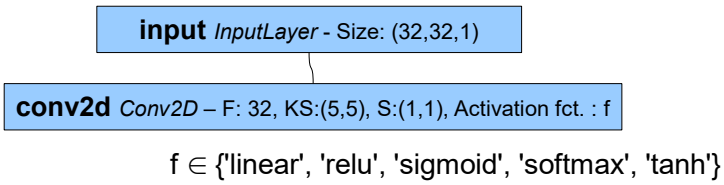


Figure 3.18: The architecture of the model to test the effect of different activation functions for different hardware accelerators

For each corresponding hardware, I took the difference between the processing times per event using the linear activation function (i.e. no activation) with the processing times per event using the remaining activation functions, in order to obtain a fair approximation of the activation layer execution times. The results can be seen in Figure 3.19.

As expected, the `relu` activation introduce little to no additional computing time to both the CPU and the hardware accelerators, because it only involves the evaluation of the logic statement whether the input is positive or negative. Therefore, the computation of `relu` is cheap regardless of the hardware used.

For the CPU, the evaluations of the `sigmoid`, `softmax` and `tanh` functions add small amounts of computation time since these involve the computation of the exponential function, which is computationally expensive.

For the VPU, the executions of the `sigmoid` and `softmax` functions also introduce around 0.4-0.5 ms of additional computing time per frame due to their computational cost. Since the `softmax` requires an output vector as an input, its execution should wait until all the elements of the output vector are evaluated. As the SHAVE vector processors operate independently of each other, the availability of these output vector elements might not be simultaneous, and therefore the execution of the activation function might introduce some overhead.

Remarkably, the evaluation of `tanh` does not introduce significant computational time for both the VPU and the TPU. There is no known reason for this, but I can guess that the x and y values of the `tanh` function are saved in a ROM for both

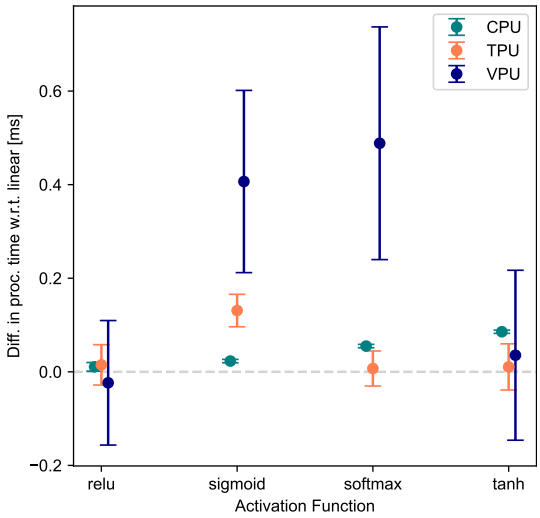


Figure 3.19: The difference of processing times between without activation functions (linear) and with different activation functions. The error bars come from the statistical uncertainty propagations of the difference between processing time per event with and without activation functions.

hardware so that the approximated evaluation of these functions originate from the interpolation of these values. This may be also the case for the evaluation of `softmax` in the TPU and the simultaneous availability of the output vector elements in the TPU is guaranteed since the MXU calculates all operations synchronously in a single clock frequency. Therefore the evaluation of `softmax` does not add a significant amount of time.

Like the VPU, the evaluation of the `sigmoid` function for the TPU also introduces some computing time, due to the complexity of evaluating the exponential function.

As a side note, it is unknown, why the results for the VPU come with high standard errors.

It can be concluded that for the optimized hardware utilization, the use of `relu` and `tanh` are preferred due to their small execution times. For the TPU, the use of `softmax` can also be recommended, since it requires little to no computation time and normalizes the output vector values into probabilities which all add up to 1.

3.1.2.3 Pooling Operations

Pooling layers are often integrated in image classification neural networks as they help reduce the height and width dimensionality of the model and prevents overfitting. Hence, in the following I will evaluate the performance of pooling operations by building benchmark operations with architectures illustrated in 3.20 with varying pool sizes. I used both maximum and average pooling operations.

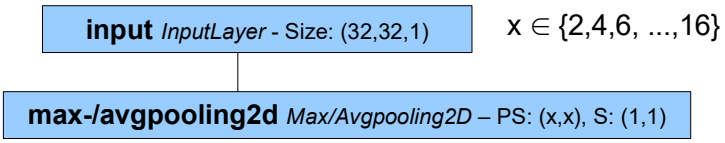


Figure 3.20: The architecture of the model to test the effect of different pool sizes in max / average pooling layers for different hardware accelerators

I measured the processing time of the models for 10000 samples, repeated this for 10 times and took the mean and the standard deviation of these measurements. Like the case for convolutional benchmarks, I used the concatenated 32×32 inputs from Subsection 1.1.2 again to simulate the situation at AFIS and there might be quantization overhead by the CPU since the raw data is in FP32 format.

Table 3.1: Processing Time per Event for Pooling Operations

	CPU	TPU	VPU
Max Pooling	0.16 ± 0.01 ms	0.45 ± 0.01 ms	2.13 ± 0.02 ms
Average Pooling	0.15 ± 0.01 ms	0.46 ± 0.01 ms	2.14 ± 0.02 ms

From the throughput measurements, it can be inferred, that the pool size does not affect the processing time per event and the results of the measurement are compiled in Table 3.1. Moreover, for both accelerators, there is also no difference in performance for max and average pooling operations. Like for the convolution benchmarks, the TPU turns out to be more performant, possibly due to the existence of a dedicated module for pooling operations in the TPU.

Table 3.2: Average Power Consumption for Pooling Operations

	TPU	VPU
Max Pooling	1.05 ± 0.01 W	1.46 ± 0.01 W
Average Pooling	1.04 ± 0.01 W	1.42 ± 0.01 W

Also, from the power measurements, it can be concluded that pool size does not affect the power consumption either and whether max or average pooling is used. The TPU has again less power consumption than the VPU like the case for convolutions.

Therefore, the TPU is the optimal choice due to its performance and power efficiency for pooling operations.

3.1.2.4 Influence of Depth

In Figure 3.21, the influence of adding fully connected dense and convolutional layers can be demonstrated.

First of all, dense layers that are 512, 768 and 1024 neurons wide are applied in series, increasing the model depth. The graphs on the left are processing times per event for these models. It can be observed that for the CPU and VPU this increases linearly with model depth, because every layer added has the same number of operations, therefore the same contribution to the processing time.

For the TPU a linear increase can also be observed for the same reason, but there are two regions with two different slopes. The region with the flatter slope is the case where the model is completely stored on-chip, while the steeper slope is caused because part of the model is stored in an external memory. The transition between these two regions occur at different model depths depending on the dense layer width, because the on-chip memory is filled up quicker with increasing layer width.

Secondly, convolutional layers with 128, 192 and 256 filters each are applied in series and the results are shown on the middle column of the figure. Similar linear behaviour can be observed for all hardware, because each layer added has same number of operations. It can also be observed that the VPU has a remarkably larger slope than the TPU, because as shown in the benchmarks before, the VPU needs in general more time for the evaluation of a convolutional layer than the TPU.

For the TPU, two slope regions can also be recognized, corresponding to the cases when the model can be fully accomodated inside the on-chip memory and when a part of the model must be stored in an external RAM, reducing the data bandwidth significantly.

3.1.2.5 Inception Modules

In 2014, a group of Google scientists proposed a neural network architecture module that has multi-level feature extraction, i.e. a module that integrates several convolutional layers with different kernel sizes, enabling it to capture information at various scales and complexities. On top of that, a pooling layer is also applied in parallel to reduce overfitting. This is named naive Inception module [67], and the architecture is given in Figure 3.22.

However, the number of computations in the naive Inception module might blow up, if the module is applied after a layer with an output with a lot of output channels, such as the case shown in Figure 3.22. The authors of [67] then proposed a workaround using 1×1 convolutions as a means of dimensionality reduction in the channels dimension, therefore reducing the complexity and simultaneously not losing depth in the neural network. This is illustrated in Figure 3.23 and called the Inception module with dimensionality reduction, or just simply the Inception module.[68]

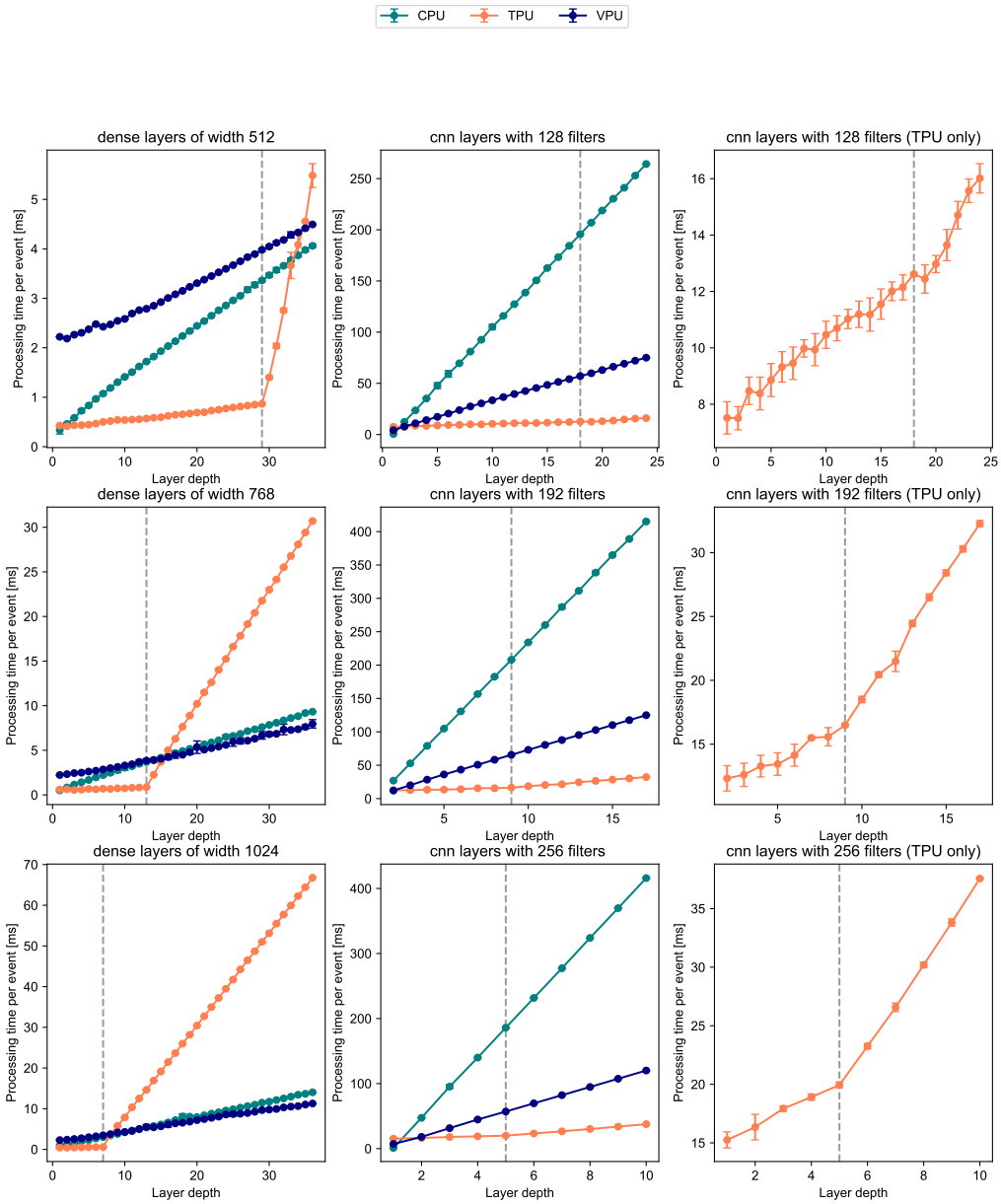


Figure 3.21: The linear processing time change by adding consecutive dense layers with same neuron width (left column), consecutive convolutional layers with same number of filters (middle column). The grey dotted vertical lines are the transitions starting where part of the model is stored off-chip for the TPU. The right column is the middle column but only for the TPU, to clarify this transition.

The results for the throughput measurement for both types of Inception modules as

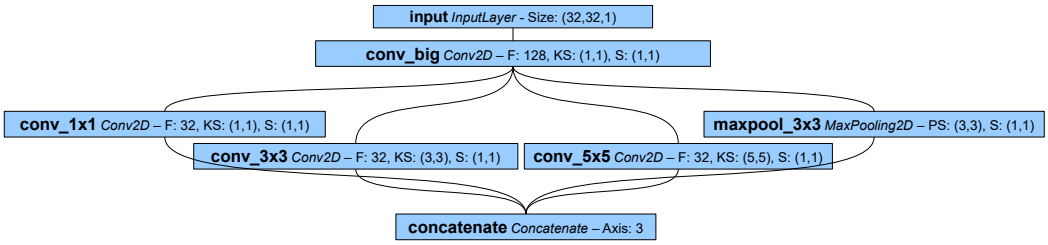


Figure 3.22: The architecture of the naive Inception module

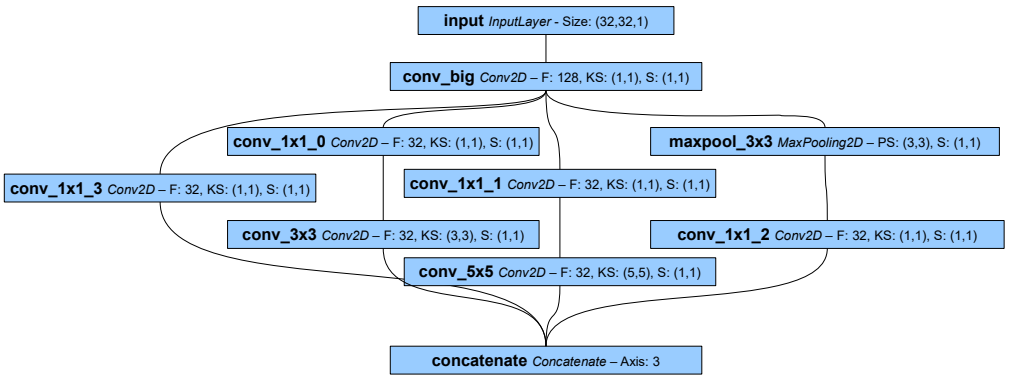


Figure 3.23: The architecture of the Inception module with dimensionality reduction

benchmarks are shown in Figure 3.24. The processing time of the models for 10000 samples was measured, and this was repeated for 10 times and the mean and standard deviation of the measurements were taken.

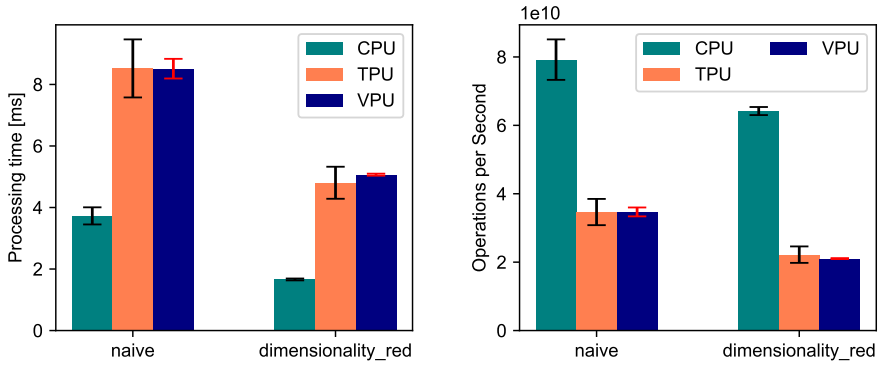


Figure 3.24: The evaluation of the throughput of the Inception module

For the CPU and both hardware accelerators, the naive Inception module requires more processing time per event than the dimensionality reduction Inception module, since the latter has fewer operations. And since small kernel sizes are applied in this module and Subsection 3.1.2.2 it can be observed that the throughput small kernel sizes are similar for both accelerators, it can also be expected that the throughput of both Inception modules for both accelerators are similar.

I also measured the power consumption as described in Subsection 3.1.1.

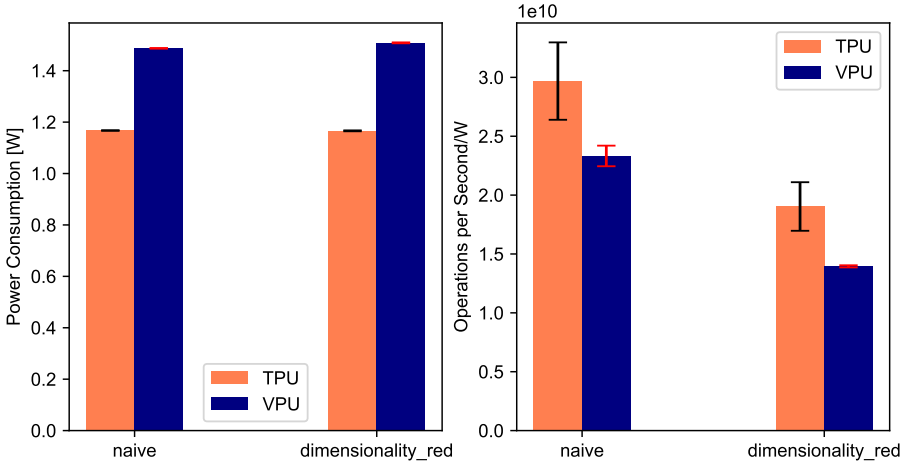


Figure 3.25: The evaluation of the power consumption for the Inception module

For both Inception modules and the case of both accelerators, the values for power consumption are similar. Since there is a smaller throughput for the dimensionality reduction Inception module, this results in lower power efficiency for the dimensionality reduction Inception module.

Even though the throughput and the power efficiency is better for the TPU is better for the case of naive Inception, I would still recommend using the dimensionality reduction Inception module for future applications, merely as it requires less processing time per event.

3.1.2.6 The Angle and PID Models

I use the pre-trained angle and PID models from [28] as benchmarks. The pre-trained neural network architectures are in the appendix. The total number of operations for the angle and PID models are 132.072 and 651.083 million, respectively, calculated by the TensorFlow Lite compiler. The performance results for these benchmarks can be viewed in Figure 3.26.

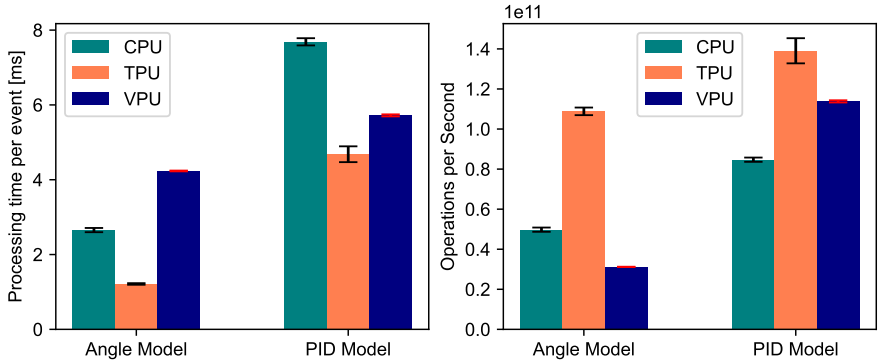


Figure 3.26: The evaluation of the performance of the angle and PID model

To measure the power, I also used the methods described in Subsection 3.1.1 and the results for the power consumption can be viewed in Figure 3.27.

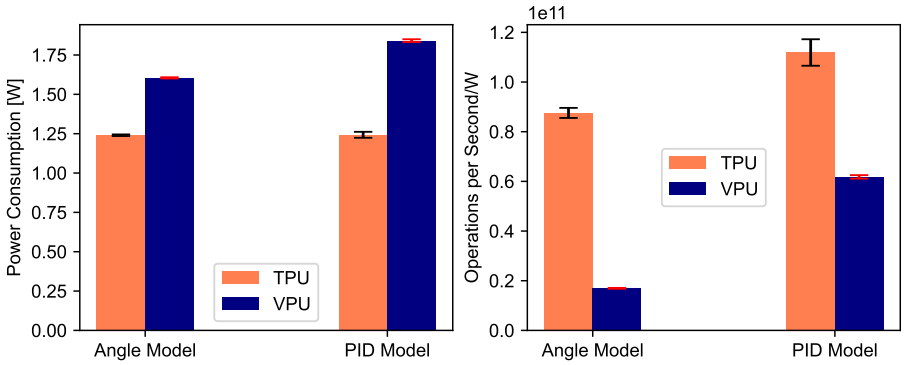


Figure 3.27: The evaluation of the power efficiency on evaluating the angle and PID models

For both benchmarks, the TPU is superior to the VPU in terms of performance and power efficiency because these models are fully accomodated in the on-chip TPU SRAM, which has a higher data bandwidth than the DRAM bandwidth of the VPU.

3.1.2.7 Other Benchmarks

The performance of some benchmarks from other machine learning publications are tested and observed in the following, i.e. MobileNet[69], VGG16[70], VGG19[70] and ResNet50[71], to test performance benchmarks that are larger in data size and do not fit into the TPU's on-chip memory.

Before the input layers of each model, I applied an input layer of shape 32×32 so that the AFIS data could serve as input to the model. The results of the processing time per event are shown in Figure 3.28, where the models are ordered from left to right with increasing model depth. The numbers at the end of the model names indicate the model depth.

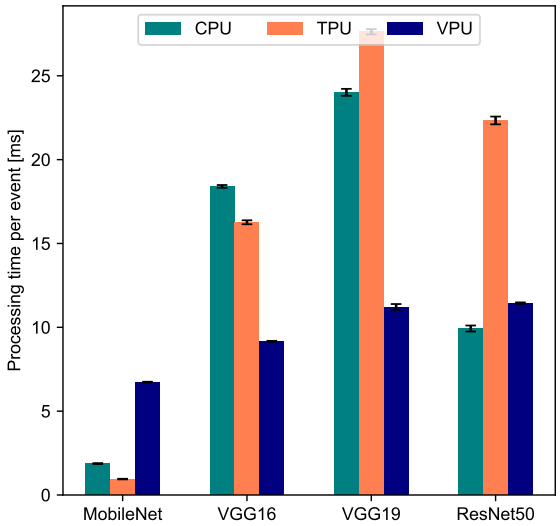


Figure 3.28: The evaluation of pre-trained models from other publications. The model depth increases from the left to the right and the numbers at the end of the model name indicate the depth of the model.

The CPU is faster for ResNet50 due to the lower total number of operations, even though there are more layers present in it. For an input size of 32×32, the VGG19 involves 310.2 million floating point operations [72] and the ResNet50 375.5 million [73].

Moreover, a significant rise in the processing time per event can be observed for the TPU for VGG16, VGG19, and ResNet50, since the model does not fit inside the UB of the TPU. It is interesting to see that the processing time for ResNet50 is shorter than VGG19 even though the model is deeper because the VGG19 has eight layers that have a total input size of 4096, meaning that these layers need extra clock cycles to calculate their outputs since the MXU only has 4096 MACs. Meanwhile, ResNet50 only has two such layers.

For the VPU, all the models fit inside its 512 MB DRAM, and therefore no significant drop in performance is to be expected. The processing time per event for VGG19 and

ResNet50 are similar. This is because even though the input dimensions of eight layers of the VGG19 are larger than 4096, the model is shallower. These effects turn out to compensate each other out.

3.1.2.8 The Effect of Model Pruning

In the following the effect of model compression through pruning is observed. The pruning method I used is based on magnitude, which has the easiest implementation in TensorFlow.

First, a model that already fits in the on-chip memory of the TPU is chosen, i.e. the PID model. The model is pruned to different sparsities, which is the fraction of the weights that have been set to 0 in the models, namely 25 %, 50 %, and 75%.

The measurement shows that the pruning of the PID model does not result in a larger throughput for all hardware. The processing time per event for the CPU stays at 7,55 ms, the TPU at 4,32 ms, and the VPU at 5,74 ms.

After that, a model that does not fit in the on-chip memory of the TPU is chosen, i.e. the VGG16. It is also pruned to sparsities 25 %, 50 %, and 75%. The losslessly compressed 75%-sparse model has the size 5.87 MB and should theoretically fit in the UB of the TPU.

However, similar to the case for the PID model, no improvement in the performance can be observed. The processing time per event for the CPU stays at 18,44 ms, the TPU at 15,92 ms and the VPU at 9,13 ms.

From these results, we can learn that the computer architecture and the compiler of each hardware cannot skip and ignore the pruned operations. Also for the TPU, pruning of the models does not result in the model accommodated within the on-chip memory.

3.2 Discussion

In the following, the results from the previous sections will be used to attempt to answer the main topics of this work.

Evaluation of Hardware Accelerators

In the following, the evaluation of the hardware accelerator candidates are being discussed from different aspects, but mainly focused on the TPU and the VPU.

General Purpose of the TPU and the VPU

The TPU's main purpose is to accelerate neural network models, mainly for object tracking and image classification.[74] Therefore, the hardware is very efficient calculating MACs, but will be inefficient or incapable of performing other types of calculations.

The VPU on the other hand is more versatile, since it is purposed not only to accelerate neural network models, but also for image processing and computer vision tasks.

Throughput Evaluation

The benchmarking results from Section 3.1.2 with AFIS input data for the different accelerator chips are summarized in Table 3.3.

It is shown, that in benchmarks that only consist of one or two different layers, such as matrix multiplication, convolutions, pooling and non-linear activation functions, the TPU delivers better throughput than the VPU. The only case where the VPU matches the throughput performance of the TPU is convolutions with small kernel sizes, such as 1×1, 3×3 and 5×5.

The main reason why the TPU has better throughput is due to its systolic data architecture. Systolic architectures balance the computation with memory access, because for every memory access done in the TPU, multiple calculations are performed on a single data item, including MACs in the MXU, the execution of pooling operations and non-linear activation functions in this order, therefore exploiting the memory access time effectively.[36] In contrast, the RISC approach in the VPU still requires it to perform repetitive read and write tasks to and from the registers, which might not utilize the memory access bandwidth of the registers effectively. Moreover, the CMX, which serves as a shared L1 cache in the VPU, is used together by the 16 SHAVE cores. In order to avoid memory access conflicts, strict scheduling of memory access by the LEON core is needed,[53] which might reduce the memory bandwidth from the CMX and hence the throughput.

Table 3.3: Processing Time per Event Comparison based on Various Benchmarks

Benchmark	**TPU**	**VPU**	**CPU**
`conv_kernel`	1.3 - 1.4 ms	2.5 - 17.9 ms	0.25 - 2.15 ms
`conv_channels`	1.3 - 2.2 ms	2.5 - 6.3 ms	0.4 - 10.2 ms
`conv_stride`	0.6 - 1.4 ms	2.2 - 2.6 ms	0.2 - 0.3 ms
Naive Inception	8.5 ± 0.9 ms	8.5 ± 0.3 ms	3.7 ± 0.3 ms
Dim. red. Inception	4.8 ± 0.5 ms	5.1 ± 0.1 ms	1.7 ± 0.1 ms
Angle Model	1.2 ± 0.1 ms	4.2 ± 0.1 ms	2.7 ± 0.1 ms
PID Model	4.7 ± 0.2 ms	5.7 ± 0.1 ms	7.7 ± 0.1 ms
Matrix mult.	$0.4 - 76.6$ ms	$2.2 - 10.3$ ms	-
Max. Pooling	0.45 ± 0.01 ms	2.13 ± 0.02 ms	0.16 ± 0.01 ms
Avg. Pooling	0.46 ± 0.01 ms	2.14 ± 0.02 ms	0.15 ± 0.01 ms
Act. functions			
sigmoid	0.13 ± 0.03 ms	0.41 ± 0.19 ms	0.02 ± 0.01 ms
softmax	0.01 ± 0.04 ms	0.49 ± 0.25 ms	0.05 ± 0.01 ms
MobileNet	0.95 ± 0.02 ms	6.73 ± 0.02 ms	1.87 ± 0.03 ms
VGG16	16.3 ± 0.1 ms	9.2 ± 0.1 ms	18.4 ± 0.1 ms
VGG19	27.6 ± 0.2 ms	11.2 ± 0.2 ms	24.0 ± 0.2 ms
ResNet50	22.3 ± 0.2 ms	11.4 ± 0.1 ms	9.9 ± 0.2 ms

For models smaller than 8 MB, i.e. Angle Model, PID Model and MobileNet, the TPU has also a better throughput than the VPU, because in this case the whole model weights fit into the on-chip memory and the full bandwidth of the TPU SRAM can be used.

On the other hand, for models larger than 8 MB, i.e. VGG16, VGG19 and ResNet50, parts of the model are stored in an external RAM, because there is no RAM on chip. This reduces the data bandwidth of the TPU significantly and therefore the computation is limited by the data bandwidth.

For both the TPU and the VPU, the processing time per event increases linearly with each added layer, as shown in Subsubsection 3.1.2.4. The slope is larger for the VPU than for the TPU, because as shown in the previous benchmarks, the computing time per layer is smaller for the TPU. However, the slope for the TPU becomes steep if the next layer should be stored in an external RAM, reducing the memory access speed.

In conclusion, the TPU is more performant for smaller models and the VPU for larger models.

Power Consumption

Power consumption in computing architectures is mainly caused by dataflow and memory access, with memory accesses from storages in proximity to the processing elements, e.g. SRAM, consuming less power than memory accesses from storages further away from the processing elements in terms of memory hierarchy, e.g. DRAM.[75]

Table 3.4: Power Consumption Comparison based on Various Benchmarks

Benchmark	**TPU**	**VPU**
`conv_kernel`	1.4 - 1.4 W	1.4 - 2.0 W
`conv_channels`	1.1 - 1.3 W	1.4 - 1.8 W
`conv_stride`	1.0 - 1.1 W	1.4 - 1.4 W
Naive Inception	1.167 ± 0.001 W	1.487 ± 0.007 W
Dim. red. Inception	1.166 ± 0.002 W	1.508 ± 0.003 W
Angle Model	1.240 ± 0.005 W	1.605 ± 0.004 W
PID Model	1.243 ± 0.019 W	1.841 ± 0.009 W
Max. Pooling	1.05 ± 0.01 W	1.46 ± 0.01 W
Avg. Pooling	1.04 ± 0.01 W	1.42 ± 0.01 W

As shown in the Table 3.4, the power consumption of the TPU is lower than of the VPU for benchmarks with a size less than 8 MB. Since the systolic architecture of the TPU uses one memory access for multiple computations, there are in average less numbers of memory accesses for the TPU than for the VPU. For the VPU, model weights are stored in the 512 MB DRAM, whose memory access is energy expensive.

Another possible factor for the higher power consumption of the VPU is the existence of the RISC processor, which performs sophisticated tasks such as regulating the I/O of data and scheduling of memory access.

The benchmark models that are tested in Table 3.4 are however less than 8 MB in size, and therefore fit in the on-chip memory of the TPU. The power consumption for the inference of models larger than 8 MB are not yet measured.

Integrability to the FPGA

However, the integrability from the FPGA to both the TPU and the VPU cannot be guaranteed without using a host microprocessor with an operating system.

Effort for Model Building, Development and Optimization

One can use any integrated development environment (IDE) that supports Python and C++/C for the building and deployment neural network models in the case of both hardware accelerators TPU and VPU.

The Radiation Environment

The TID irradiation tests [44] mentioned in Section 2.2 show that the TPU is suitable for 5-year LEO missions. Hence, the TPU is expected to be sufficiently radiant tolerant for the mission, since AFIS takes place in LEO as well.

The NASA tests on heavy ions and protons [45] mentioned in Chapter 3 show that while the number of SEFIs (Single Event Functional Interrupts) per day due to heavy ions are the same for both the TPU and the VPU, the number of SEFIs due to protons for the VPU is an order higher than the TPU. A high flux of protons are expected during the mission, so the VPU must still be verified of radiation tolerance for our purposes.

Thoughts on the Mythic M1076 AMP and the Hailo-8 AI Accelerator

Unfortunately, these hardware accelerators cannot be tested due to market unavailability, but some expectations can be made based on their computing architecture concept.

Regarding throughput performance and power consumption of the M1076, since there is no transmission bottleneck for the access of weight data and the computation is located where the weights are, the throughput performance of the AMP can be potentially very high in comparison to the TPU and the VPU. However, the analog-to-digital (ADC) and digital-to-analog (DAC) converters that are required to do analog computation might consume a lot of power and influences the data bandwidth, which are in form of input and output voltages. Moreover, the AMP might be radiation tolerant because the AMP was originally designed for GPS satellite applications.[76]

The Hailo-8 AI Accelerator can also have a high throughput and low power consumption, because the idea of the model compiler is to modify the computing architecture so that it is optimized for the neural network model, like a neural-network specialized FPGA. Optimally, memory accesses should be minimized and memory addressing of data should not take a long time. The Hailo-8 promises a very high TOPS, but as mentioned before this should be taken with a grain of salt because this really depends on the data bandwidth.

Methods of Neural Network Optimization for Hardware Accelerators

Based on the results of the benchmarks run in Section 3.1, the neural network hyperparameters that deliver optimal throughput for each hardware accelerator are compiled in the following.

Optimal Model Specifications for the TPU

For the TPU, it is of significant importance that the model size is less than 8 MB, so that all of the model fits inside the on-chip SRAM, otherwise the whole model would be stored in an external memory, which will reduce the throughput due to the significantly lower data bandwidth.

From Subsubsection 3.1.2.1, it can be learned that for dense layers, both the input vector size and the number of neurons should be less than 2^{12}. If the input vector size is larger than 2^{12}, then the number of MACs in calculating one element of the output vector will be larger than 2^{12} and therefore the calculation will need an extra clock cycle, so the time to complete the layer computation increases and the throughput decreases. If the number of neurons is larger than 2^{12}, then the bias vector addition size will be larger than 2^{12}. The MXU will not be able to execute the bias addition in one clock cycle.

From Figure 3.13, it can be learned that for optimized convolutions in the TPU, as these are also converted into matrix multiplications by the TPU, the following requirements should be fulfilled

- the number of MACs calculated for a single element in the output, which is equal to $c_I \cdot k^2$, where c_I is the number of input channels of the convolutional layer and k is the convolutional kernel size, must be less than or equal to 2^{12}. This quantity is simultaneously the inner dimensions [2] of the converted matrix multiplication.

- $h_O \times w_O$ and c_O, where h_O and w_O are the output height and widths of the convolutional layer and c_O is the number of output channels, which is also equal to the number of convolution filters, must be less than or equal to 2^{12}. This quantity is simultaneously the outer dimensions[3] of the converted matrix multiplication.

From Figure 3.16, using convolution stride > 1 reduces power consumption and increases the throughput due to the reduced number of convolution operations.

From Subsubsection 3.1.2.3, maximum and average pooling are both equally efficient for the TPU.

From Figure 3.19, the evaluation of non-linear activation functions ReLU, softmax, and tanh do not introduce significant additional computation time, so the application of these functions is preferred in the TPU.

Due to its systolic architecture, the order of operations done in a single data cycle is fixed. For example, an activation layer put before a pooling layer might not be effectively processed by the TPU, as the pooling unit is located before the activation unit in the dataflow, which means that the data has to undergo two data cycles.

[2] For a matrix product AB, where A has the size $m \times n$ and B has the size $n \times o$, n is called the inner dimension, m and o are called the outer dimensions

[3] See footnote 2

Optimal Model Specifications for the VPU

Since most neural networks rarely reach 512 MB, almost all models can be fully accommodated in the VPU DRAM completely.

From Subsubsection 3.1.2.1, it can be learned that for dense layers, the VPU can optimally function if the input vector sizes and the number of neurons per layer are less than 2^{12} since the VRFs of the 16 SHAVEs can accommodate only 2^{12} FP16 values, like the TPU.

From Figure 3.13, it can be learned that for convolutions, the kernel sizes must be less than 16, so that the input feature map of one convolution can fit in one VRF of a SHAVE core.

Also, based on 3.13, the product of the square of the kernel size and the number of filters should be less than 2^{12}, since the VRFs of the 16 SHAVEs can accommodate in total 2^{12} FP16 values.

Like the TPU, stride > 1 for the convolution does not affect the power consumption based on 3.17, but increases the throughput as shown in Figure 3.16 due to the reduced number of convolution operations due to downsampling. Hence, all convolutional strides can be used according to one's purpose.

According to Figure 3.19, the use of activation functions ReLU and tanh are preferred for the VPU because they introduce no significant additional computation time due to their computational simplicity.

A Small Optimization Step in the PID Model

All the layers in the PID model fit within the aforementioned specifications for the TPU and the VPU, because the output dimensions of these layers are all below 2^{12}, except for the layer `dense` (See Appendix on PID Model Architecture). The input size vector for this layer is 5120, which is larger than 2^{12}. This large vector size is due to the flattening of the output of the preceding layer `max_pooling2d_3` $(4, 20, 64)$. Hence, I would suggest to apply a 1×1 convolutional layer with 32 or lower filters as a means of dimensionality reduction layer, like in the Inception module.

The effects of this change can be seen in Figure 3.29. As expected, the addition of this 1×1 convolutional layer after the `max_pooling2d_3` layer in the PID model improves the performance of both accelerators. The effect is significantly stronger for the TPU, because as in Subsection 3.1.2.1, the processing time per event increase for exceeding the dimensions of the processing array is higher than for the VPU and simultaneously the addition of a convolutional layer adds up to the processing time of the VPU more than to the processing time of the TPU. As depicted in Figure 3.21, in general the TPU excels in calculating convolutions compared to the VPU.

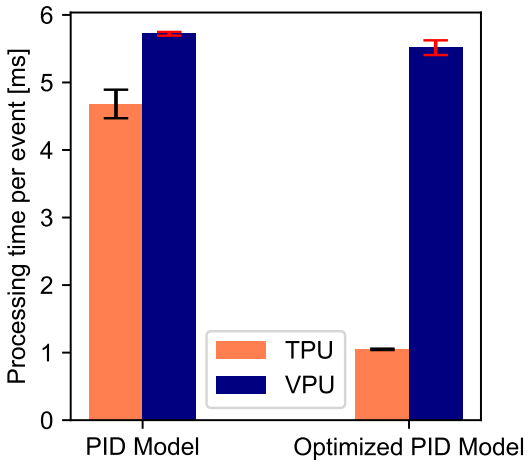


Figure 3.29: The change in processing time per event after adding a 1×1 convolutional layer with 32 filters after `max_pooling2d_3` to reduce dimensionality. The effect on the TPU is more significant than on the VPU.

Model Compression

Since the TPU only supports INT8 weights and the VPU only FP16 weights, INT8 and FP16 quantization of models are required. However, one must be careful, since quantizing the weights after training could reduce the classification accuracy of the model. Therefore, in most cases, it would be best to quantize the weights already during training, called quantization-aware training.

As shown in Subsubsection 3.1.2.8, pruning has no effect for both hardware accelerators and for both models under 8 MB and above 8 MB. This is expected for the TPU, because no computation units that evaluate boolean algebra are available, hence preventing it from determining which operations to execute in a model. However, a better performance was expected for the VPU after model pruning since the load-store units (LSU) are able to generate memory addresses solely for non-zero weights.[77] This is not observed and hence a further investigation in the model compilation process of OpenVINO Model Optimizer is needed.

Knowledge distillation could benefit both the TPU and the VPU. For the TPU, the main factor that can influence the throughput is if the model size is larger than 8 MB. If so, a student model with a size smaller than 8 MB for this large model can be used. For the VPU, the main factor that influences the throughput is not the size, but the depth of the model. Hence, a student model that has a shallower depth can be used for distillation from this larger model. Clearly, the distillation is most optimal

if the optimizal model specifications for each hardware accelerator is considered when building the student model. However, it is to be noted, that knowledge distillation takes large training epochs[78], so one should consider carefully whether this compression approach should be done, taking into account the development time cost.

Optimizing Models for the M1076 AMP and the Hailo-8 AI Accelerator

As an analog signal processor, the Mythic M1076 is susceptible to electrical noise that may affect the computation accuracy. Therefore it is crucial to design robust classification models that take into account noise fluctuation during training. To optimize model deployment in the AMP, [32] suggests to design shallower but wider neural networks to minimize the digital-to-analog signal conversions and maximize the reuse of data that is converted to analog.

For the Hailo-8 AI Accelerator, the microarchitecture of the tiles is unknown, and therefore the optimal dimensions for dense and convolutional layer operations are also unknown.

Chapter 4

Conclusion and Outlook

The purpose of the Antiproton Flux in Space (AFIS) mission is to measure the flux of geomagnetically trapped low-energetic antiprotons in the South Atlantic Anomaly (SAA), which is a region where the inner van Allen radiation belt dips down to an altitude of 200 km, due to the non-concentricity of the magnetic and geographic poles of the Earth. Due to the high event rate of at least the order of 10^4 events per second and the limited downlink bandwidth for small satellites, on-board data processsing in the form of an event trigger concept is required, simultaneously increasing the signal-to-background ratio from the order of 10^{-7}. For this, an FPGA is chosen due to its parallel computing architecture and reconfigurability. There can be a case when certain operations introduce computational bottlenecks or cannot be implemented due to the lack of available resources in the FPGA, particularly when using neural network based applications that contain matrix multiplications and convolutions, which can require a large number of processing elements per input byte. To mitigate this problem, additional hardware can be integrated to support the on-board data processing through a separate module called the Scientific Computation Module. The utilization of a CPU or microcontroller is not suitable due to their sequential computing architecture, which is not suitable for highly parallelizable operations such as matrix multiplications and convolutions. Moreover, a CPU usually has a high power consumption, which can be non-compliant to the power constraints of small satellite applications. A GPU has parallelized architecture but usually also has a high power consumption. One of the remaining options is to integrate a hardware accelerator, which are specialized for matrix multiplications and convolutions, from a different module.

In this work, I evaluate the advantages and drawbacks of such hardware accelerators based on the throughput, power consumption, integrability to the FPGA and suitability of algorithms. Initially, literature research was made for several feasible hardware accelerators and the computing architectures were analyzed. The TPU (Tensor Processing Unit) has a systolic architecture, whose principle is to maximize the number of computations per memory access. It has a small on-chip SRAM of the size 8 MB and is specialized for neural networks. The radiation tests show the suitability of the TPU for LEO missions, which is the type of mission AFIS is. The VPU (Vision Processing Unit) has a hybrid architecture of RISC and VLIW (Very Long Instruction Word). The

VLIW architecture reduces the complexity of the hardware by making it necessary for its compiler to pack instructions that are independent from each other into long words, so that the hardware is not required to check instruction dependencies. The VPU implements memory hierarchy, meaning that it has a system of registers, caches and a DRAM. The VPU specializes not only for neural networks, but also contains specialized hardware architectures for image processing and computer vision. Mythic M1076 AMP and Hailo-8 AI Accelerator were not available in the market at the time of the creation of this thesis. For all accelerators, the literature lacked information regarding the command protocols, which is important for the integration from the FPGA. Afterwards, several benchmark models were built and run in the USB modules of the TPU and VPU and simulated proton events were used as input data. The results show that in general, TPU is suitable for the inference of small neural network models due to its small on-chip SRAM and the VPU is suitable for larger neural network models due to its larger 512 MB DRAM and image processing. By evaluating and running these benchmark models, the optimal hyperparameters such as number of neurons on a dense layer, kernel sizes, number of filters and stride on a convolutional layer that lead to the best throughput and/or lowest power consumption can be deduced. There are several limitations associated with the conducted measurement, such as refraining the use of antiproton events for benchmarking, which would have increased the realism of this process. The method used to measure processing time for the hardware accelerator is based on host CPU time and not from the time measured from the accelerator itself. Hence, inaccuracies can be produced. Moreover, using a USB voltmeter to measure the power consumption of the USB modules delivers only a rough approximation of the power consumption of the corresponding hardware accelerators. Since microcontrollers are included in the USB module, the measurement values also include the power consumption of the microcontrollers.

Several possible optimization methods for each hardware accelerator have been investigated in this work. I have done a small optimization step on the PID model, I added a 1×1 layer for dimensionality reduction so that all layers have the specifications mentioned in Section 3.2 that optimizes the performance of each hardware, but without retraining it. A significant throughput improvement improvement can be seen for the TPU. For the VPU, only a small improvement in the throughput can be observed. In the future, the model with the dimensionality reduction should be retrained and evaluated of the model accuracy. The effect of model compression through magnitude-based pruning, i.e. setting weights to zero based on their magnitude, is also observed. For both TPU and VPU, the benchmarking results show no improvement in the throughput. This means that we cannot optimize the throughput for the TPU and the VPU through magnitude-based pruning. Knowledge distillation of existing neural network models can also be realized and evaluated in the future. The student model shall adhere to the specifications that optimize the hardware performance. For the

TPU, it has to be ensured that the student model size is less than 8 MB, whereas shallow but wide student models are preferred for the VPU. In the end, accuracy, throughput and power consumption of the optimized models are the key factors for determining the best optimization technique.

If the AMP M1076 and the Hailo-8 AI Accelerator are available in the market in the future, similar benchmarking tests can be performed for comparison. This benchmarking can be performed for the FPGA as well. Additionally, radiation testing can be also performed for them, if no data can be found regarding this.

Finally, the primary challenge for the application of hardware accelerators for AFIS is the interfacing to the FPGA, due to the lack of information regarding this topic. For the future development of the Scientific Computation Module, a further investigation of the command protocols can be carried out. If this information remains elusive, the usage of a host microprocessor for the hardware accelerator can be used as an alternative. This host microprocessor can take the form of a softcore microprocessor emulated in the FPGA or a hardcore microprocessor in its physical form integrated in the SCM. If this hurdle is overcome, the use of hardware accelerators combined with model optimization can achieve high throughput and low power consumption for neural network inference.

The Neural Network Architecture of the Angle Model

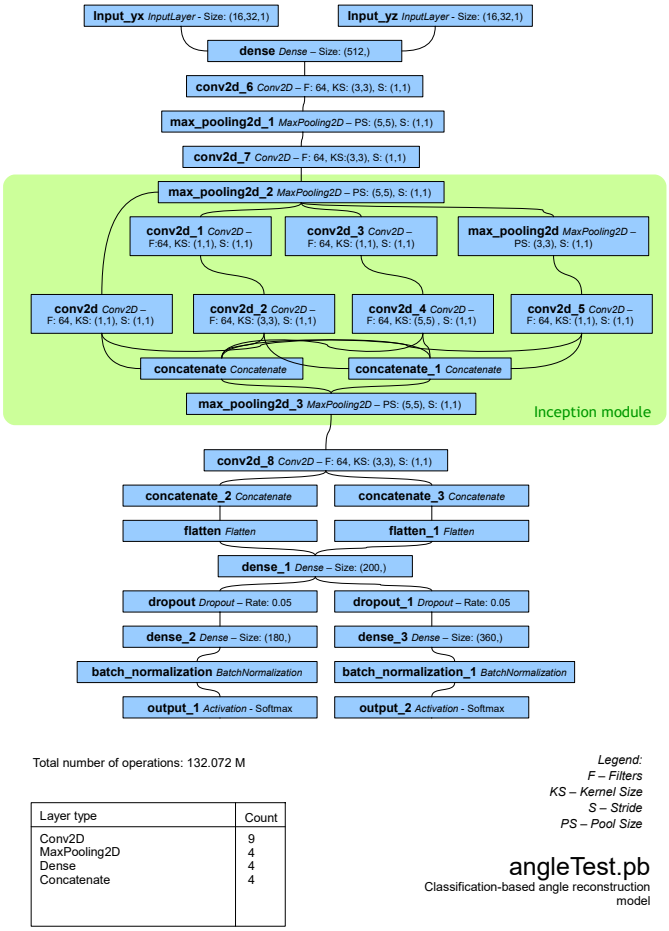


Figure 1: The neural network architecture of the angle model

The Neural Network Architecture of the PID Model

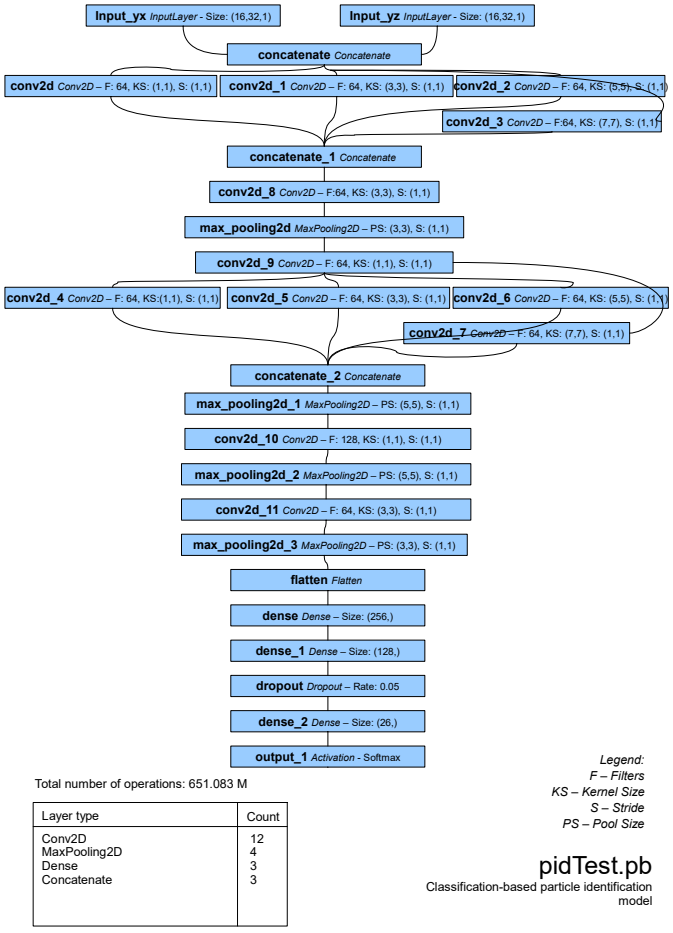


Figure 1: The neural network architecture of the PID model

Bibliography

[1] P. A. M. Dirac. »The Quantum Theory of the Electron«. In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 117.778 (1928), pp. 610–624.

[2] C. D. Anderson. »The Positive Electron«. In: *Physical Review* 43.6 (1933), pp. 491–494.

[3] O. Chamberlain et al. »Observation of Antiprotons«. In: *Physical Review* 100.3 (1955), pp. 947–950.

[4] R. L. Golden et al. »Evidence for the Existence of Cosmic-Ray Antiprotons«. In: *Phys. Rev. Lett.* 43 (16 Oct. 1979), pp. 1196–1199. DOI: 10.1103/PhysRevLett.43.1196. URL: https://link.aps.org/doi/10.1103/PhysRevLett.43.1196.

[5] Boudaud, M. et al. »A new look at the cosmic ray positron fraction«. In: *AA* 575 (2015), A67. DOI: 10.1051/0004-6361/201425197. URL: https://doi.org/10.1051/0004-6361/201425197.

[6] Antje Putze. »Direct cosmic-ray detection«. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 662 (2012). 4th International workshop on Acoustic and Radio EeV Neutrino detection Activities, S157–S163. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2010.11.133. URL: https://www.sciencedirect.com/science/article/pii/S016890021002680X.

[7] Oscar Adriani et al. »The discovery of geomagnetically trapped cosmic ray antiprotons«. In: *Astrophysical Journal - ASTROPHYS J* 737 (July 2011).

[8] Michio Fuki, Ayako Kuwahara and Nozomi Sawada. »Calculation of Cosmic-Ray Proton and Anti-Proton Spatial Distribution in Magnetosphere«. In: *International Cosmic Ray Conference* (Jan. 2003).

[9] R. Selesnick et al. »Geomagnetically trapped antiprotons«. In: *Geophysical Research Letters - GEOPHYS RES LETT* 34 (Oct. 2007). DOI: 10.1029/2007GL031475.

[10] J A Van Allen et al. »OBSERVATION OF HIGH INTENSITY RADIATION BY SATELLITES 1958 ALPHA AND GAMMA«. In: *Jet Propulsion* 28 (Sept. 1958). DOI: 10.2514/8.7396. URL: https://www.osti.gov/biblio/4300875.

[11] T. Pöschl. »Modeling and Prototyping of a Novel Active-Target Particle Detector for Balloon and Space Applications – Master's Thesis«. Technical University of Munich, Mar. 2015.

[12] J.R Heirtzler. »The future of the South Atlantic anomaly and implications for radiation damage in space«. In: *Journal of Atmospheric and Solar-Terrestrial Physics* 64.16 (2002). Space Weather Effects on Technological Systems, pp. 1701–1708. ISSN: 1364-6826. DOI: https://doi.org/10.1016/S1364-6826(02)00120-7. URL: https://www.sciencedirect.com/science/article/pii/S1364682602001207.

[13] F.G. Hartjes and R. Wigmans. »Scintillating plastic fibres for hadron calorimetry«. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 277.2 (1989), pp. 379–385. ISSN: 0168-9002. DOI: https://doi.org/10.1016/0168-9002(89)90766-3. URL: https://www.sciencedirect.com/science/article/pii/0168900289907663.

[14] L. Meng. »Development of a CubeSat Detector for Measuring the Low-Energy Antiproton Flux in Low Earth Orbit – Diploma Thesis«. Technical University of Munich, 2014.

[15] Martin Losekamm et al. »The AFIS Detector: Measuring Antimatter Fluxes on Nanosatellites«. In: *Proceedings of the International Astronautical Congress, IAC*. Vol. 5. Sept. 2014. DOI: 10.13140/RG.2.1.4996.0405.

[16] C.R. Gruhn et al. »Bragg curve spectroscopy«. In: *Nuclear Instruments and Methods in Physics Research* 196.1 (1982), pp. 33–40. ISSN: 0167-5087. DOI: https://doi.org/10.1016/0029-554X(82)90612-7. URL: https://www.sciencedirect.com/science/article/pii/0029554X82906127.

[17] Liesa Eckert. *Integration and Calibration of the RadMap Telescope Detector Modules*. 2020.

[18] Holger Krag and Heiner Klinkrad. »Chapter 9 - Space debris protection«. In: *Safety Design for Space Systems (Second Edition)*. Ed. by Tommaso Sgobba et al. Second Edition. Butterworth-Heinemann, 2023, pp. 327–351. ISBN: 978-0-323-95654-3. DOI: https://doi.org/10.1016/B978-0-323-95654-3.00009-2. URL: https://www.sciencedirect.com/science/article/pii/B9780323956543000092.

[19] Ask an Astrophysicist. »The South Atlantic Anomaly«. In: *NASA* (). Archived from the original on November 5, 2007. Retrieved October 16, 2007. URL: https://web.archive.org/web/20071105155423/http://imagine.gsfc.nasa.gov/docs/ask_astro/answers/970303.html.

[20] Rainer Sandau, Klaus Brieß and Marco D'Errico. »Small satellites for global coverage: Potential and limits«. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 65.6 (2010). ISPRS Centenary Celebration Issue, pp. 492–504. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2010.09.003. URL: https://www.sciencedirect.com/science/article/pii/S0924271610000869.

[21] NVIDIA. *Autonomous Machines: The Future of AI*. https://www.nvidia.com/en-us/autonomous-machines/.

[22] Ravi Rao. *ASIC vs FPGA: A Comprehensive Comparison*. https://www.wevolver.com/article/asic-vs-fpga-in-chip-design.

[23] Martin J. Losekamm, Peter Hinderberger and Sebastian Rückerl. »Reliable Data Handling and Processing Systems for Small-Satellite Missions«. In: *2023 IEEE Aerospace Conference*. 2023, pp. 1–9. DOI: 10.1109/AERO55745.2023.10115789.

[24] Doug Sinclair and Jonathan Dyer. *Radiation Effects and COTS Parts in SmallSats*. https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=2934context=smallsat.

[25] Xiaoxiao Jiang and Jun Tao. »Implementation of effective matrix multiplication on FPGA«. In: *2011 4th IEEE International Conference on Broadband Network and Multimedia Technology*. 2011, pp. 656–658. DOI: 10.1109/ICBNMT.2011.6156017.

[26] C. N. Hesse. »Analysis and Comparison of Performance and Power Consumption of Neural Networks on CPU, GPU, TPU and FPGA – Master's Thesis«. Stiftung Universitaet Hildesheim, June 2021.

[27] Christian Janiesch, Patrick Zschech and Kai Heinrich. »Machine learning and deep learning«. In: *CoRR* abs/2104.05314 (2021). arXiv: 2104.05314. URL: https://arxiv.org/abs/2104.05314.

[28] L. Meyer-Hetling. »A Neural-Network-Based Event Reconstruction for the RadMap Telescope – Master's Thesis«. Technical University of Munich, Oct. 2021.

[29] Stephan Guennemann. »Machine Learning (TUM), Lecture 8: Deep Learning II«. In: 2022.

[30] Yangjie Zhou et al. »Characterizing and Demystifying the Implicit Convolution Algorithm on Commercial Matrix-Multiplication Accelerators«. In: *2021 IEEE International Symposium on Workload Characterization (IISWC)*. 2021, pp. 214–225. DOI: 10.1109/IISWC53511.2021.00029.

[31] Haotong Qin et al. »Binary neural networks: A survey«. In: *Pattern Recognition* 105 (Sept. 2020), p. 107281. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2020.107281. URL: http://dx.doi.org/10.1016/j.patcog.2020.107281.

[32] Vivienne Sze. *How to Evaluate Efficient Deep Neural Network Approaches*. https://eems.mit.edu/wp-content/uploads/2020/06/2020_CVPR_evaluate_dnn.pdf.

[33] Geoffrey Hinton, Oriol Vinyals and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [statL].

[34] S. Francescato, S. Giagu, F. Riti et al. »Model compression and simplification pipelines for fast deep neural network inference in FPGAs in HEP«. In: *European Physical Journal C* 81.10 (2021), p. 969. DOI: 10.1140/epjc/s10052-021-09770-w. URL: https://doi.org/10.1140/epjc/s10052-021-09770-w.

[35] Coral. *Accelerator Module datasheet, Version 1.4*. https://coral.ai/static/files/Coral-Accelerator-Module-datasheet.pdf.

[36] H. T. Kung. »Why systolic architectures?« In: *Computer* 15 (1982), pp. 37–46. URL: https://api.semanticscholar.org/CorpusID:1858965.

[37] Maroju Sai Kumar, D. Ashok Kumar and P. Samundiswary. »Design and performance analysis of Multiply-Accumulate (MAC) unit«. In: *2014 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2014]*. 2014, pp. 1084–1089. DOI: 10.1109/ICCPCT.2014.7054782.

[38] Q-engineering. *Google Coral Edge TPU explained in depth*. https://qengineering.eu/google-corals-tpu-explained.html.

[39] Coral. *Edge TPU Compiler*. https://coral.ai/docs/edgetpu/compiler/.

[40] Norman P. Jouppi et al. *In-Datacenter Performance Analysis of a Tensor Processing Unit*. 2017. arXiv: 1704.04760 [cs.AR].

[41] Google. libedgetpu|. https://github.com/google-coral/libedgetpu/blob/master/tflite

[42] Google. *Tensor Processor Instruction Set Architecture*. https://patents.google.com/patent/US20180341484A1/en.

[43] Coral. *TensorFlow models on the Edge TPU*. https://coral.ai/docs/edgetpu/models-intro/.

[44] George Lentaris. *EXECUTIVE SUMMARY – CAIRS21 4000135491/21/NL/GLC/OV*. https://nebula.esa.int/sites/default/files/neb_study/2666/C4000135491ExS.pdf.

[45] Megan Casey (NASA). *Recent Radiation Test Results on COTS AI Edge Processing ASICs*. https://nepp.nasa.gov/docs/etw/2022/15-JUN-WED/0930-Casey-220220009215.pdf.

[46] Intel. *Intel® Movidius™ Myriad™ X Vision Processing Unit (VPU) with Neural Compute Engine*. https://www.intel.com/content/www/us/en/products/docs/processors/movidvpu/myriad-x-product-brief.html.

[47] Intel. *Myriad™ X: Evolving low power VPUs for Deep Neural Networks*. https://community.intel.com/t5/Blogs/Tech-Innovation/Artificial-Intelligence-AI/Myriad-X-Evolving-low-power-VPUs-for-Deep-Neural-Networks/post/1335627.

[48] Evangelos Petrongonas et al. »ParalOS: A Scheduling Memory Management Framework for Heterogeneous VPUs«. In: *2021 24th Euromicro Conference on Digital System Design (DSD)*. 2021, pp. 221–228. DOI: 10.1109/DSD53832.2021.00043.

[49] Intel. *Intel Unveils Neural Compute Engine Movidius Myriad X VPU Unleash AI Edge*. https://newsroom.intel.com/news/intel-unveils-neural-compute-engine-movidius-myriad-x-vpu-unleash-ai-edge/.

[50] Joseph Fisher et al. »Parallel processing: A smart compiler and a dumb machine«. In: *ACM SIGPLAN Notices* 19 (Apr. 2004), pp. 37–47. DOI: 10.1145/502949.502878.

[51] Sergio Rivas-Gomez et al. »Exploring the Vision Processing Unit as Co-Processor for Inference«. In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2018, pp. 589–598. DOI: 10.1109/IPDPSW.2018.00098.

[52] Brendan Barry et al. »Always-on Vision Processing Unit for Mobile Applications«. In: *IEEE Micro* 35.2 (2015), pp. 56–66. DOI: 10.1109/MM.2015.10.

[53] Mircea Horea Ionica and David Gregg. »The Movidius Myriad Architecture's Potential for Scientific Computing«. In: *IEEE Micro* 35.1 (2015), pp. 6–14. DOI: 10.1109/MM.2015.4.

[54] Nate Oh. *Intel Announces Movidius Myriad X VPU, Featuring 'Neural Compute Engine'*. https://www.anandtech.com/show/11771/intel-announces-movidius-myriad-x-vpu.

[55] Intel. *Intel Neural Compute Stick 2 – Product Brief*. https://cdrdv2.intel.com/v1/dl/getContent/749742.

[56] John von Neumann. *First Draft of a Report on the EDVAC*. Tech. rep. Archived from the original on March 14, 2013. Retrieved August 24, 2011. 1945. URL: https://www.rand.org/content/dam/rand/pubs/research_memoranda/2006/RM704.pdf.

[57] Mythic AI. *M1076 Analog Matrix Processor Product Brief*. https://mythic.ai/wp-content/uploads/2022/03/M1076-AMP-Product-Brief-v1.0-1.pdf.

[58] Mythic. *M1076 Analog Matrix Processor*. https://mythic.ai/products/m1076-analog-matrix-processor/.

[59] Hailo. *Hailo-8 AI Processor*. https://developer.hailo.ai/files/hailo-8-product-brief-en/.

[60] Sam Wechsler. *Inside the Google Coral TPU USB Stick*. https://www.youtube.com/watch?v=HzNf59GfbLE.

[61] ST. *STM32L011x3 STM32L011x4 Datasheet*. https://www.st.com/resource/en/datasheet/stm32l011d4.pdf.

[62] Ltd. Hangzhou Ruideng Technology Co. *Instruction for USB Tester with Full Colour Display*. http://ruidengkeji.com/inst/AT35.pdf.

[63] Mohammadreza Mohammadi et al. »Facial Expression Recognition at the Edge: CPU vs GPU vs VPU vs TPU«. In: *Proceedings of the Great Lakes Symposium on VLSI 2023*. GLSVLSI '23. ACM, June 2023. DOI: 10.1145/3583781.3590245. URL: http://dx.doi.org/10.1145/3583781.3590245.

[64] Swarnima Pandey. *How to choose the size of the convolution filter or Kernel size for CNN?* https://medium.com/analytics-vidhya/how-to-choose-the-size-of-the-convolution-filter-or-kernel-size-for-cnn-86a55a1e2d15.

[65] Jason Brownlee. *How Do Convolutional Layers Work in Deep Learning Neural Networks*. https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks.

[66] deepai.org. *Understanding Stride in Convolutional Neural Networks*. https://deepai.org/machine-learning-glossary-and-terms/stride.

[67] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[68] deepai.org. *Understanding the Inception Module in Deep Learning*. https://deepai.org/machine-learning-glossary-and-terms/inception-module.

[69] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].

[70] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[71] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[72] opengenus.org. *Understanding the VGG19 architecture*. https://iq.opengenus.org/vgg19-architecture/.

[73] opengenus.org. *Understanding ResNet50 architecture*. https://iq.opengenus.org/resnet50-architecture/.

[74] Coral. *Industries*. https://coral.ai/industries/.

[75] Vivienne Sze et al. »How to Evaluate Deep Neural Network Processors: TOPS/W (Alone) Considered Harmful«. In: *IEEE Solid-State Circuits Magazine* 12.3 (2020), pp. 28–41. DOI: 10.1109/MSSC.2020.3002140.

[76] Dave Fick. *Mythic's Flash-Based Analog Compute-in-Memory*. https://www.ict.tuwien.ac.at/staff/taherinejad/MiM/slides/MiM_Talk7_20210802.pdf. 2019.

[77] WikiChip. *SHAVE v2.0 - Microarchitectures - Intel Movidius*. https://en.wikichip.org/wiki/movidius/microarchitectures/shave_v2.0.

[78] Lucas Beyer et al. »Knowledge distillation: A good teacher is patient and consistent«. In: *CoRR* abs/2106.05237 (2021). arXiv: 2106.05237. URL: https://arxiv.org/abs/2106.05237.

Declaration

I hereby declare that I created this work all by my own and correctly cited every information that is based on the work of others.

Garching, January 2024

_____
Vernando Fransiscus Limodya