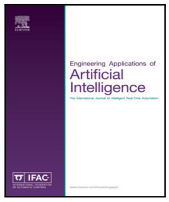
Engineering Applications of Artificial Intelligence 117 (2023) 105604

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



Benchmarking edge computing devices for grape bunches and trunks detection using accelerated object detection single shot multibox deep learning models

Sandro Costa Magalhães [a,b,*], Filipe Neves dos Santos [b], Pedro Machado [c], António Paulo Moreira [a,b], Jorge Dias [d,e]

[a] INESC TEC – Instituto de Engenharia, Tecnologia e Ciencia, Campus da FEUP, Rua Dr. Roberto Frias s/n, Porto, 4200-465, Porto, Portugal
[b] Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias s/n, Porto, 4200-465, Porto, Portugal
[c] Computational Intelligence and Applications group (CIA), Department of Computer Science, School of Science and Technology, Nottingham Trent University, Clifton Campus, Nottingham, NG11 8NS, United Kingdom
[d] Khalifa University Center of Autonomous Robotics Systems (KUCARS), Khalifa University of Science, Technology and Research (KU), 127788, Abu Dhabi, United Arab Emirates
[e] Department of Electrical Engineering and Computers, University of Coimbra, Rua Silvio Lima, Coimbra, 3030-290, Coimbra, Portugal

ARTICLE INFO

MSC:
62M45
62P30
68Q85

Keywords:
Embedded systems
Heterogeneous platforms
Object detection
SSD resNet
RetinaNet resNet

ABSTRACT

Purpose: Visual perception enables robots to perceive the environment. Visual data is processed using computer vision algorithms that are usually time-expensive and require powerful devices to process the visual data in real-time, which is unfeasible for open-field robots with limited energy. This work benchmarks the performance of different heterogeneous platforms for object detection in real-time. This research benchmarks three architectures: embedded GPU—Graphical Processing Units (such as NVIDIA Jetson Nano 2 GB and 4 GB, and NVIDIA Jetson TX2), TPU—Tensor Processing Unit (such as Coral Dev Board TPU), and DPU—Deep Learning Processor Unit (such as in AMD-Xilinx ZCU104 Development Board, and AMD-Xilinx Kria KV260 Starter Kit).

Methods: The authors used the RetinaNet ResNet-50 fine-tuned using the natural VineSet dataset. After the trained model was converted and compiled for target-specific hardware formats to improve the execution efficiency.

Conclusions and Results: The platforms were assessed in terms of performance of the evaluation metrics and efficiency (time of inference). Graphical Processing Units (GPUs) were the slowest devices, running at 3 FPS to 5 FPS, and Field Programmable Gate Arrays (FPGAs) were the fastest devices, running at 14 FPS to 25 FPS. The efficiency of the Tensor Processing Unit (TPU) is irrelevant and similar to NVIDIA Jetson TX2. TPU and GPU are the most power-efficient, consuming about 5 W. The performance differences, in the evaluation metrics, across devices are irrelevant and have an F1 of about 70 % and mean Average Precision (mAP) of about 60 %.

1. Introduction

Computer vision classifiers are largely explored in multiple robotics systems, such as agricultural ones. These systems allow robots to perform visual localisation by visually detecting natural landmarks like tree trunks (Mendes et al., 2016) or to detect objects for other purposes such as grasping or harvesting (Magalhães et al., 2021; Moreira et al., 2022).

The rise of Artificial Intelligence (AI) and the continuous generation of big data is creating computational challenges. Central Processing Units (CPUs) are not enough to efficiently run state-of-the-art AI algorithms or process all the data generated by a wide range of sensors. World-leading processing technology companies (such as NVIDIA, AMD, Intel and ARM) have been looking closely into the new requirements. They have been pushing the boundaries of technology to deliver efficient and flexible processing solutions.

Heterogeneous computing refers to the use of different types of processor systems in a given scientific computing challenge.

Heterogeneous platforms are composed of different types of computational units and technologies. Such media can be composed of

* Corresponding author at: INESC TEC – Instituto de Engenharia, Tecnologia e Ciencia, Campus da FEUP, Rua Dr. Roberto Frias s/n, Porto, 4200-465, Porto, Portugal.
E-mail addresses: sandro.a.magalhaes@inesctec.pt (S.C. Magalhães), fbsantos@inestec.pt (F.N. dos Santos), pedro.machado@ntu.ac.uk (P. Machado), amoreira@fe.up.pt (A.P. Moreira), jorge.dias@ku.ac.ae (J. Dias).

https://doi.org/10.1016/j.engappai.2022.105604
Received 22 August 2022; Received in revised form 18 October 2022; Accepted 2 November 2022
Available online 19 November 2022
0952-1976/© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

multi-core CPUs, GPUs and FPGAs acting as computational units and offering the flexibility and adaptability demanded by a wide range of application domains (de Andrade, 2018). These computational units can significantly increase the overall system efficiency and reduce power consumption by parallelising concurrent operations that require substantial CPU resources over long periods.

Accelerators like GPUs and FPGAs are massive parallel processing systems that enable accelerating portions of code that are parallelisable. Combining CPUs with GPUs and FPGAs help improve the efficiency (speed of executing algorithms) by assigning different computational tasks to specialised processing systems. GPUs are optimised to perform matrix multiplications in parallel, which is the major bottleneck in video processing and computer graphics. Nevertheless, GPUs also introduce hardware and environmental limitations (e.g. high-power consumption and architectural limitations) (Intel, 2020). Convolutional Neural Networks (CNNs) are massively parallel in their nature and not suitable for matrix representation because each neuron can be considered a node containing several sequential mathematics operations. Despite of very optimised to execute parallel operations, GPUs architecture is inspired by CPU. Application-Specific Integrated Circuits (ASICs) are synthesised FPGAs' designs that aim to optimise and specify the operations executions. ASICs are more compact and, if designed for processing CNN algorithms, so fast as FPGAs. ASICs can be designed to work as single devices or connected to external systems.

In Deep Learning (DL) applied to visual problems, CNNs are the most common Artificial Neural Networks (ANNs). These networks' architecture is mainly composed of sequential convolution layers that are trained to extract relevant features from images. CNNs are frequently applied to classification, object detection and segmentation problems. In the scope of object detection, the most used CNNs architectures are Single-Shot Multibox Detector (SSD) (Liu et al., 2016), faster R-CNN (Wang and Peng, 2019), and You Only Look Once (YOLO) (Redmon et al., 2016; Redmon and Farhadi, 2018). Faster R-CNN is the most precise model to detect objects but processes the image in two stages, making inference slower. SSD and YOLO are both single-shot architectures, i.e., they only process the image once using feature maps, repositioning the object bounding boxes, and making their classification. Some authors have been exploring single-shot architectures to detect fruits and other objects in open-field environments (Magalhães et al., 2021; Sozzi et al., 2022; Zhao et al., 2022; Olenskyj et al., 2022; Terra et al., 2021; Magalhães et al., 2022). Inside this group of architectures, YOLO models are undoubtedly the most common deep neural network (Magalhães et al., 2021; Sozzi et al., 2022; Zhao et al., 2022; Olenskyj et al., 2022). Because, They are fast and can achieve near real-time speed easily under regular computing hardware (Zhao et al., 2022), without big degradation of the metric when compared with other equivalent ANNs (Magalhães et al., 2021). However, they may have difficulty detecting some objects, which can be resolved by bigger and more capable CNN architecture. Transformers are also an upcoming DL architecture for object detection with successful results (Olenskyj et al., 2022). Despite this analysis, most authors benchmark their works against powerful and high-consuming hardware not suitable for embedded or robotics applications (Magalhães et al., 2021; Sozzi et al., 2022).

For overcoming the restrictions of real-time classification and power consumption, many researchers have studying small-size and effective DL architectures, like Tiny-YOLO (Redmon et al., 2016; Redmon and Farhadi, 2018), YOLACT (Bolya et al., 2019), and many other architectures (Howard et al., 2017; Sandler et al., 2018; Liu et al., 2016), that can be implemented in more cost-effective GPUs or even in CPUs. Alternatively, other researchers are studying low-power and efficient devices that may run parallelisable deep neural networks (Puchtler and Peinl, 2020). These devices are generically characterised as embedded devices and are from many types and architectures: GPUs, FPGAs, and ASICs, more commonly, Coral TPUs and Intel Neural Compute Sticks (NCSs) (see more details at Section 2.1). Another common technique used by some researchers is quantisation (Yang et al., 2019). By default, Artificial Neural Networks (ANNs) are trained in FP32, but the optimisation algorithms are iterative and often converge to high-resolution precise values that are time-consuming to compute and meaningless for the classification process. The quantisation technique allows reducing the ANN resolution to INT8 by rescaling the FP32 weights, improving the time of inference and, sometimes, the accuracy. The merge of different strategies to optimise the execution of ANNs can create highly efficient DL models that can process images at thousands of frames per second (FPS).

Researchers have essentially focused on embedded GPUs from the NVIDIA Jetson family, using NVIDIA Jetson Nano, NVIDIA Jetson TX2 and NVIDIA Jetson AGX Xavier. Zhao et al. (2019) benchmarks two DL models, Tiny-YOLO and DNET, under NVIDIA Jetson TX2 and NVIDIA GTX Titan X. The authors could have a low accuracy drop (about 1 %) in the quantisation process for the NVIDIA Jetson TX2. The inference speed was about ten times slower in the NVIDIA Jetson TX2 (running at 18 FPS), as expected, but consumed 20 times less power, consuming only about 8 W. Suzen et al. (2020), Chiu et al. (2020), Rahmaniar and Hernawan (2021) and Martinez et al. (2021) also benchmark DL models efficiency between NVIDIA Jetson embedded boards. The NVIDIA Jetson AGX Xavier was the fastest board in the family but also the most power-expensive. On the other side, the NVIDIA Jetson Nano is less power-consuming but slower. The most benchmarked DL models are SSD MobileNet networks family and YOLO family. Both are small-size networks that have fewer convolution layers and retain fewer images' features. Martinez et al. (2021) run a YOLACT at 66 FPS in an NVIDIA Jetson AGX Xavier and at 16 FPS in an NVIDIA Jetson TX2, revealing the substantial hardware improvement of the most recent NVIDIA Jetson board. Chiu et al. (2020), Rahmaniar and Hernawan (2021) benchmark SSD MobileNet v2 in the three boards and NVIDIA Jetson TX2 was the fastest with 26 FPS. Suzen et al. (2020) also benchmarked the Raspberry Pi4, but it was slow and inefficient.

Despite the reasonable power-consumption improvement, Jetson GPUs have a similar architecture to traditional NVIDIA GPUs, sharing some of their limitations. So, some researchers started exploring the highly efficient FPGAs. The most commonly explored FPGAs in the literature now belongs to AMD-Xilinx, particularly to the AMD-Xilinx Zynq family. Venieris and Bouganis (2017), Chen et al. (2019) compared a Zynq FPGA against a GPU. Venieris and Bouganis (2017) benchmark multiple CNNs between AMD-Xilinx Zynq-7045 and a NVIDIA Tegra X1. In all the cases, the FPGA was at least twice faster. Chen et al. (2019) benchmarked an AMD-Xilinx ZedBoard against a NVIDIA GTX 1080Ti in the ImageNet dataset (Russakovsky et al., 2015), using a ResNet-18 classifier. During the quantisation process, Chen et al. (2019) could improve the network's accuracy and efficiency, running it at 20 FPS and saving 100 times less power (consumes about 2.58 W). Lin et al. (2021) compared a quantised INT8 MobileNet classifier running at the FPGA's DPU (the FPGA main core for processing DL models, Section 2.1) against multiple AMD-Xilinx FPGAs in the literature. Their main study focused on the AMD-Xilinx ZCU104, which executed the algorithm at 376 FPS while consuming only 5 W. Zhao et al. (2021) benchmarked an AMD-Xilinx ZCU104 against an Amazon Cloud FPGA EC2, using an YOLO INT8. Both devices reached similar results, with up to 13 FPS in the Penn Treebank dataset. Also Jain et al. (2021) benchmarked multiple FPGAs using a Tiny-YOLO INT8 and reached an inference speed between 12 FPS to 23 FPS at the AMD-Xilinx XC7Z035.

Researchers are also looking for some ASICs to execute the neural networks because they can become cheaper, smaller, and easier to integrate with other systems. The most common ASICs are Google Coral TPUs and Intel NCSs. Puchtler and Peinl (2020) benchmarked Coral Edge TPU USB Accelerator and Intel NCS 2 using a SSD MobileNet v2 INT8 against an NVIDIA Jetson Nano and Raspberry Pi 4 with a SSD MobileNet v2 with weights in FP16. The ASICs were the fastest devices, reaching inference framerates of 55 FPS in the Coral Edge TPU USB Accelerator and 23 FPS at the Intel NCS 2. Raspberry Pi

4 was the slowest device, inferring at 4 FPS, and the Jetson Nano inferred at 15.90 FPS. The authors did not do any power consumption analysis. Also Aguiar et al. (2021), Kovács et al. (2021) evaluated the performance and efficiency of Coral Edge TPU USB Accelerator.

As illustrated in the revised literature, researchers are constantly looking to improve the DL models' speed and accuracy to meet real-time constraints, but most of the work focuses essentially on improving the models' architecture and not their intrinsic properties such as their high parallelisation ratio (Redmon et al., 2016; Redmon and Farhadi, 2018; Bolya et al., 2019; Howard et al., 2017; Sandler et al., 2018; Liu et al., 2016). Moreover, many works essay their algorithms in high-performance devices never used in robotics and mobile applications (Magalhães et al., 2021; Moreira et al., 2022). Some authors argue that some models in embedded devices (Martinez et al., 2021; Chiu et al., 2020; Rahmaniar and Hernawan, 2021; Venieris and Bouganis, 2017; Lin et al., 2021; Zhao et al., 2021), but it is not clear which kind of device is more suitable for the target application. Therefore, our work aims to perform a wide benchmark between heterogeneous platforms for evaluating the performance in the evaluation metrics and time and power efficiency of these edge computing devices in robotics applications for running DL models, giving continuity to Aguiar et al. (2021)'s work. The authors will focus only on using the RetinaNet ResNet-50 (Lin et al., 2020; Humbarwadi, 2020) fine-tuned in the VineSet dataset (Aguiar et al., 2021; Aguiar and Magalhães, 2021) and compare them using multiple pointwise models (FP32, FP16, INT8) and heterogeneous platforms. The used embedded devices were two GPUs with 1000 TFLOPS (NVIDIA Jetson Nano 2 GB and 4 GB – Jetson Nano), one GPU with 2000 TFLOPS (NVIDIA Jetson TX2 – TX2), one TPU (Coral Dev Board TPU — TPU), and DPUs (AMD-Xilinx ZCU104 Development Board – ZCU104 – and AMD-Xilinx Kria KV260 Starter Kit – KV260). To the author's knowledge, this is the first study involving a big object detection model like RetinaNet ResNet-50 and benchmarking the AMD-Xilinx Kria KV260.

The authors aim to assess the RetinaNet using ResNet-50 to near-real-time applications. Although the proposed method is suitable for farming application, this might not be the case for other use-cases. Because farming robots typically run at speeds of $0.5\,\mathrm{m\,s^{-1}}$ when operating in vineyards. Using a camera vision sensor with a field of view of 45° that operates at 0.5 m from the grapevine, this sensor can see 0.5 m of the grapevine. Thus, if the ANN could infer the images at 5 FPS, then the processed images will have an overlap between frames of about 0.4 m (i.e. 80 %), which should be sufficient for object detection and tracking.

Therefore, the current work aims to innovate in the following aspects:

- in the authors' knowledge, this is the first research to apply and study a big and complex object detection model like RetinaNet ResNet-50 in heterogeneous platforms;
- a larger benchmark towards object detection using many different heterogeneous platforms, when compared with the reviewed literature, containing embedded GPUs, ASICs (i.e. TPU) and embedded FPGAs (including the new AMD-Xilinx KV260, designed for robotics applications).

The next sections of this manuscript are structured as follows. In Section 2, the author will explore the different used heterogeneous platforms, stating their features and limitations, as well as the required software to deploy the ANNs for the different devices. In the same section, the authors also state the assumptions made and the methodology. In Section 3, the time and power efficiency and performance results in the evaluation metrics are presented. These results are deeply discussed in Section 4, comparing between them and with the revised literature. Section 5 summarises the experiences and the main conclusions, framing them with future required work.

2. Materials and methods

The current section details the methodology and the required material to reproduce this experience. Once this is a DL study, it requires a dataset and a DL model. The deep DL was built and trained in TensorFlow 2.8 Keras.[1] Because the authors used heterogeneous platforms, additional libraries were required to optimise the models for the specific platforms architectures: Vitis-AI 1.4, Edge TPU Compiler, and TF-TRT.[2]

2.1. Heterogeneous platforms

The current research topic aimed to benchmark heterogeneous platforms, looking for faster inference devices, minimising the accuracy drop. The authors compared three embedded GPUs with 1000 TFLOPS and 2000 TFLOPS (Jetson Nano 2 GB, Jetson Nano 4 GB, and TX2), DPUs, recurring to FPGAs (ZCU104, KV260), and TPU (Coral Dev Board TPU). For optimisation purposes, each platform required its compiler to improve operations performance in the hardware and thus the inference speed. Additionally, the RTX3090 was used to train the DL model and baseline the benchmark with a powerful and efficient GPU.

Besides the dedicated hardware, all the used boards also have a Processing System (PS) to coordinate the desired tasks and manage the operating system. The PS can have multiple architectures, but AMD64 and ARM64 are the most common in the current state-of-the-art.

2.1.1. NVIDIA GPUs and TF-TRT

Four NVIDIA GPUs were used for the current benchmark. NVIDIA RTX3090[3] is a powerful GPU designed with Ampere Architecture and GB24 of Video Random Access Memory (VRAM). Its powerful features allow the GPU to train deep neural networks quickly and with big training batches. Because the NVIDIA RTX3090 is very powerful and efficient, any straight benchmark of speed inference cannot be made, but it could work as a reference GPU for the evaluation. Besides, it is unsuitable for embedded applications because of its high power-consumption ratios, until 350 W.

The NVIDIA Jetson GPUs were designed as embedded devices to assemble in low-power systems like robots. The two Jetson Nano[4] have similar architecture but differ in the amount of available RAM (GB2 and 4 GB). TX2[5] is the second generation of Jetson Nano with a TX2 GPU against TX1 GPU. In all of these boards, the available RAM is shared between the GPU and CPU.

Although all the GPUs are compatible with TensorFlow 2 Keras models, they only reach their maximum performance and efficiency when the DL models are optimised for their architecture and specialised CUDA and Tensor cores. NVIDIA deployed CUDA cores and Tensor cores that aim to optimise parallel and matrices operations for maximum performance with CNNs. TF-TRT is an NVIDIA library that operates with TensorFlow and TensorRT (TRT) and is responsible for analysing the ANN graph and inferring the best transformations for

[1] See TensorFlow, 2022, TensorFlow, URL: https://www.tensorflow.org/. Last accessed on 05/08/2022 and Keras, 2022, Keras, URL: https://keras.io/. Last accessed on 05/08/2022.

[2] See AMD-Xilinx, 2022, Vitis-AI, URL: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html. Last accessed on 05/08/2022; Coral, 2022, Edge TPU Compiler, URL: https://coral.ai/docs/edgetpu/compiler/. Last accessed on 05/08/2022; and NVIDIA, 2022, Deep Learning Frameworks Documentation, URL: https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html. Last accessed on 05/08/2022, respectively.

[3] See NVIDIA, 2022, GeForce RTX3090 Family, URL: https://www.nvidia.com/en-eu/geforce/graphics-cards/30-series/rtx-3090-3090ti/. Last accessed on 05/08/2022.

[4] See NVIDIA, 2022, Jetson Nano 2 GB Developer Kit, URL: https://developer.nvidia.com/embedded/jetson-nano-2gb-developer-kit. Last accessed on 05/08/2022; and NVIDIA, 2022, Jetson Nano Developer Kit, URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. Last accessed on 05/08/2022.

[5] See NVIDIA, 2022, Harness AI at the Edge with the Jetson TX2 Developer Kit, URL: https://developer.nvidia.com/embedded/jetson-tx2-developer-kit. Last accessed on 05/08/2022.

Fig. 1. Sample of images in the dataset (Aguiar and Magalhães, 2021) with the respective ground truth bounding boxes in blue squares. (a) Thermal image of vines' trunks; (b) image of vines' trunks without infra-red filter; (c) image of bunches of medium-size grapes; (d) image of bunches of corn-size grapes; (e) image of vines' trunks.

speed efficiency using the dedicated cores. Besides these operations, TF-TRT also allows to change the network's graph resolution between FP32, FP16, and INT8 (the last one through quantisation). The advantage of TF-TRT against TRT is that the first one is compatible with TensorFlow and allows to have a hybrid solution when some operations cannot be converted to a TRT graph. Therefore, the main graph can have some operations executed in TensorFlow, and others executed inTRT.

2.1.2. AMD-Xilinx FPGAs and Vitis-AI

Field Programmable Gate Arrays (FPGAs) are integrated circuits that can be reconfigured to meet the designer's needs. Due to its high-reconfiguration capability, FPGAs can be useful for executing parallelizable algorithms while keeping the power consumption low. These boards always have two main components Processing System (PS) and Programmable Logic (PL). The PS is responsible for managing the operations and memory in the FPGA, while PL concerns to the reconfigurable integrated circuits. AMD-Xilinx deployed the DPU cores (AMD-Xilinx, 2022), a proprietary programable engine dedicated for CNN. This unit has a register configure module, a data controller module, and a convolution computing module. The DPU Intellectual Property (IP) can be integrated as a block in the PL with direct access to PS.

For the current benchmark, the authors chose two FPGAs, ZCU104 and KV260. Both boards have similar architecture and compatibility, but KV260 is newer, more compact and designed thinking in robotics applications. ZCU104 has two DPU cores, while KV260 has only one. These two DPUs allow the ZCU104 to simultaneously process two neural network graphs.

For executing the models in the DPU, the graph should be quantised in INT8 weights and converted to a readable DPU format. Vitis-AI is a fully integrated system in a Docker[6] environment created by AMD-Xilinx to manage this process. Vitis-AI is characterised as a comprehensive AI inference development platform for AMD-Xilinx devices. Among other features, Vitis-AI processes TensorFlow, Pytorch, and Caffe models using specific quantisers for the FPGA's design. Vitis-AI compiles and optimises the quantised models for the DPU architecture. This environment also has additional tools to optimise and debug the compiled neural network, such as pruning and profiling tools.

2.1.3. Coral TPU and Edge TPU compiler

TPU is an AI accelerator ASIC designed by Google to optimise the execution of ANN. This ASIC was made compatible with TensorFlow and accepts DL models build with the lite version of TensorFlow (TFLite). Similarly to FPGA, the ANNs running in edge computing TPUs should be quantised to make the models fully compatible with the ASIC architecture.

The whole design and management of the model are made with TensorFlow and TFLite. The compatible model to the TPU is got in TFLite by the Edge TPU Compiler.

The authors used the Coral Dev Board TPU which is an embedded board with a PS and a TPU system on-module (SoM) attached.

2.2. Dataset

The different classification models were benchmarked using the VineSet (Aguiar and Magalhães, 2021) dataset composed of 428 498 images of 300 × 300 px, manually labelled and gathered from multiple sources (stereo cameras, high-quality cameras, and thermal cameras). Furthermore, the VineSet is composed of natural vineyards images split into the following three classes: vines' trunks, bunches of berry-corn size grapes, and bunches of berry-closed grapes. Fig. 1 illustrates some images inside the dataset.

The dataset was split into three batches: train set (411 360 images), validation set (8569 images), and test set (8569 images). For consistency in the results with real-world data, the augmentation images in the test set were removed, resuming in 1125 images.

2.3. RetinaNet

RetinaNet (Lin et al., 2020; Humbarwadi, 2020) is a state-of-the-art DL model for object detection in the class of one-stage detectors. This DL model is very similar to an SSD ANN (Liu et al., 2016). After the input layer, a backbone will process the different feature maps and extract the image's features. The backbone is some CNN but ResNet-50 is the implemented backbone in the presentation article (Lin et al., 2020). Following the backbone, a FPN (Lin et al., 2017) is used. These layers follow a top-down architecture (Fig. 2) and recover the information processed by the CNN, aiming to improve the box classification and regression performance (Lin et al., 2017). The main improvement of RetinaNet against SSD DL models is the implementation of a new custom loss function, focal loss (Lin et al., 2020), that aims to prioritise the correct detection and classification of the objects, True Positive (TP), against the correct not detection of objects, True Negatives (TN).

Given the improvements in the state-of-the-art provided by RetinaNet ResNet-50 against SSD networks, and because these ANNs usually provide better results than YOLO (Magalhães et al., 2021; Tan et al., 2021; Morera et al., 2020), the authors of this benchmark chose to use RetinaNet ResNet-50 as initially stated by Lin et al. (2020). The authors used a previous model already implemented in TensorFlow 2 Keras by Humbarwadi (2020), making the necessary changes to the architecture to make it compatible with all the heterogeneous platforms. The model had to be implemented using a functional strategy,[7] but pre-processing and post-processing layers were kept in the sub-modelling format because they were not converted or recompiled for

[6] See Docker, 2022, Docker, URL: https://www.docker.com/. Last accessed on 05/08/2022.

[7] See Tensorflow, 2022, The Functional API, https://www.tensorflow.org/guide/keras/functional. Last accessed on 05/08/2022.

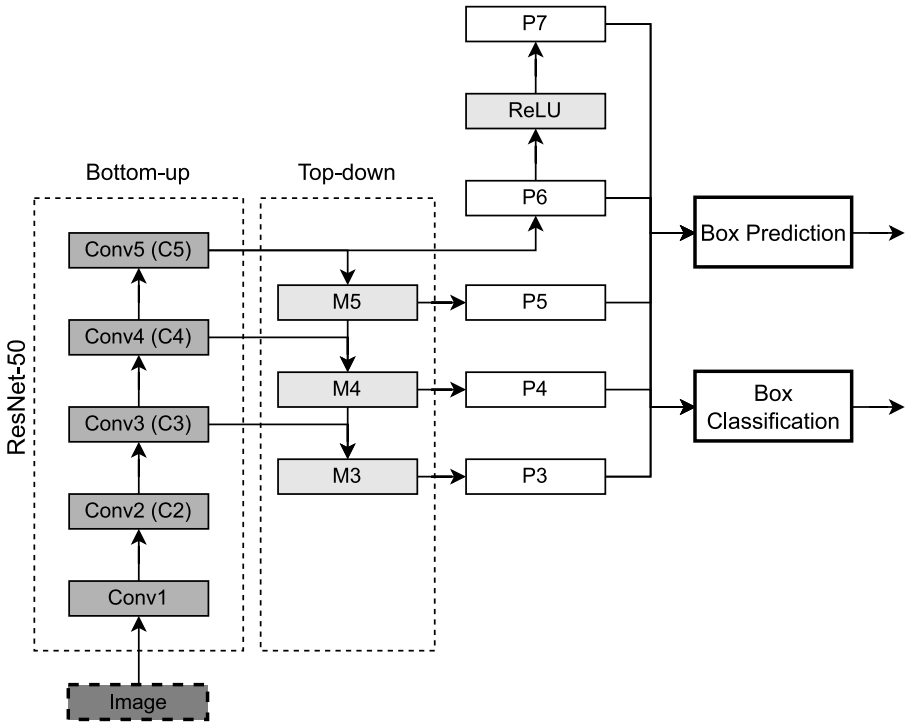


Fig. 2. Overview of a simplified diagram of the RetinaNet ResNet-50. $Conv_i$ are convolutional layers; M_i are intermediate layers composed by upsampling, additions and convolutions to generate FPN output layers P_i; P_i are convolution layers for the output of FPN.

any heterogeneous platform. Instead, these layers were reimplemented. The ResNet-50 (He et al., 2016) was configured with the same pre-trained weights used by the ImageNet dataset (Russakovsky et al., 2015) to ensure consistency and avoid deterioration of processing speed.

Vitis-AI has some operations constraints for compiling the DL model to the FPGAs. These constraints were found at Rectified Linear Unit (ReLU) operations that should be immediately preceded by another operation, like a convolution or a mathematical operation. This compromises the compilation of the network, mainly between P6 and P7 of the FPN (Fig. 2), because an output for the regression and classification layers are required at the convolution 2D P6 and the convolution 2D P7. Therefore, an additional convolutional 2D layer was added at P6, cloning the initial P6 convolution 2D layer (Fig. 3). In this way, the Vitis-AI compiler can further compile all layers of the model's core at the DPU; otherwise, a split of the architecture could happen, and some operations could be executed at the CPU.

The changed version of RetinaNet ResNet-50 (Fig. 3) was trained by fine-tuning until the convergence of the train loss function. The training algorithm used the focal loss function and the Stochastic Gradient Descendent (SGD) optimiser. For better adjustment of the learning rate and momentum values, the authors used the Keras Tuner library (O'Malley et al., 2019) with the Hyperband algorithm (Li et al., 2018) to search for the best values that optimise the validation loss. During this stage, only two batches of the dataset are used, the train set for training the model and the validation set for evaluating the model performance in the evaluation metrics and tracking the model's overfitting. The model was trained in the GPU RTX3090.

2.4. Deploying RetinaNet ResNet-50 for heterogeneous platforms

The main aim of this study is to assess the performance reliability in the evaluation metrics of DL models in heterogeneous platforms and assess their effectiveness for real-time object detection.

Deploying the RetinaNet for each heterogeneous platform is very similar but requires the use of proprietary libraries. Therefore, the steps to deploy a model for each device are:

1. RetinaNet ResNet-50 fine-tuning in the VineSet train set;
2. Quantise the model to INT8 (optional, depends on the platform);
3. Deploy the model to a platform's compatible format

The first step implies the train of the ANN, which is the same for all platforms and happens in TensorFlow 2 in the RTX3090, as stated in Section 2.2. Because pre-processing and post-processing layers cannot be compiled in some devices, only the core of the ANN is used in the following steps. Whenever required, these layers are implemented.

After training, the model is manipulated using the proprietary specific libraries. TPU and FPGAs require the use of quantisation. The quantisation can be aware of training of be agnostic to it, happening when the model has already converged. For compatibility issues, only post-training quantisation is compatible with all devices. Therefore, a dataset calibration was derived from the train set to quantise the ANN weights and calibrate them to the input calibration data. Because any train is being performed, the calibration set did not require the ground truth labels. However, compatible with quantised networks, RTX3090 and Jetson GPUs did not require them. Besides, as we could conclude in Section 3 Jetson devices could not generate quantised models of RetinaNet ResNet-50.

The last step is to optimise the ANN nodes to the hardware where they run. That is made with proprietary compilers, namely TF-TRT for GPUs, Edge TPU Compiler for TPU, and Vitis-AI for FPGAs. A full comprehensive tutorial for deploying the RetinaNet ResNet-50 at AMD-Xilinx FPGAs is published in Magalhães et al. (2022).

The deploying of ANNs is heterogeneous devices also require the implementation of pre-processing and post-processing layers whenever required. Because these layers were removed after training, these layers were reimplemented for each device in Python using OpenCV library.

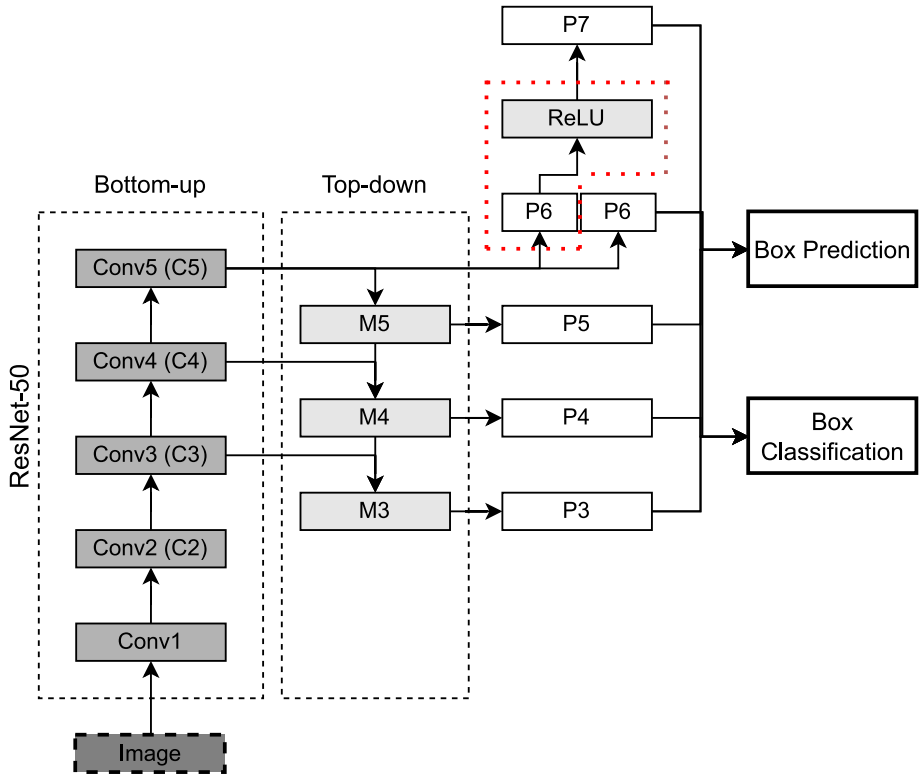


Fig. 3. Overview of a simplified diagram of a changed version of RetinaNet ResNet-50 for FPGA compatibility. $Conv_i$ are convolutional layers; M_i are intermediate layers composed by upsampling, additions and convolutions to generate FPN output layers P_i; P_i are convolution layers for the output of FPN.

It is important to realise that this work only focuses on the core of the DL model. Pre-processing and post-processing tasks are not being optimised and are being executed in the devices CPUs because of some limitations of some operations with the compilers.

2.5. Evaluation

The network's performance and efficiency in the different devices were evaluated at two levels: results in accuracy and inference speed.

The authors only considered the model's core to assess the inference. Because the pre-processing and post-processing layers are running at the devices' CPU, the authors could had made some unfair comparisons if these layers were used. Besides, in some platforms, these layers could be optimised to increase the level of parallelism, using GPU or PL.

The platforms' speed of inference was only assessed in the permanent stage of the platforms. For reaching the permanent stage, the platforms were required to infer 50 random images. During this stabilisation stage, the hardware prepares for inference, and the inference times may oscillate. The inference time is counted as the average inference value (t_{avg}) between all the images (N images) in the dataset (Eq. (1)).

$$t_{avg} = \frac{\sum_i^N t_i}{N} \tag{1}$$

Additionally, the model is also assessed in terms of results accuracy. Because the authors used post-training quantisation and different quantisation approaches, it was expected differences between results and some degradation relative to the FP32 model. For assessing this performance, the authors used the Precision (Eq. (2)), Recall (Eq. (3)), F1 (Eq. (4)), and mAP. The mAP is computed through the Precision × Recall curves and corresponds to the area under the curve.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

Because we are considering an object detection problem, the matching between the detection and the ground-truth is made using the Intersection over Union (IoU) ratio. In the current work, if the IoU between two labels is higher than 50 %, than the detection is a True Positive (TP), otherwise is a False Positive (FP). The ground truths that do not have any matching detection are reported as False Negatives (FNs).

Despite the inference efficiency and framerate, in heterogeneous systems is also relevant to assess the devices' power consumption on standby and while inferring. Heterogeneous platforms are usually applied to mobile systems powered by batteries and should perform for long periods. Therefore, the good selection of a power-effective device may be critical. The devices' power consumption was measured at the power input of the board using a Fluke 175 True RMS multimeter.[8] Because this multimeter cannot compute the power directly, that was made in two steps mathematically. A devices power consumption is given $P = V \cdot I$ (W), where V is the powering voltage in Volt and I is the consumed current in Ampere. In the first stage, the authors measured the powering voltage, in parallel, during standby and while inferring. After, they measured the current, assembling the multimeter in series and under the same conditions.

3. Results

As stated before, the authors are using the RTX3090 as the reference platform to benchmark the RetinaNet ResNet-50 model with the other heterogeneous platforms. The RTX3090 is a high-performing and power-consuming device, therefore, the presented values are only reference values, and no straight comparison should be made, mainly in

[8] See Fluke, 2022, Fluke 175 True-RMS Digital Multimeter, URL: https://www.fluke.com/en-gb/product/electrical-testing/digital-multimeters/fluke-175. Last accessed on 05/08/2022.

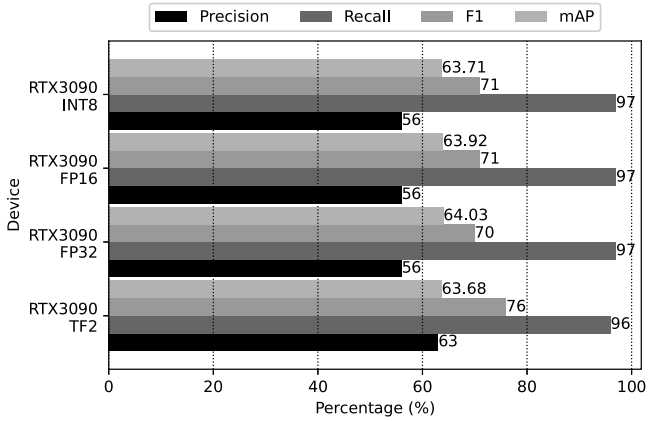


Fig. 4. Inference performance in the evaluation metrics in the reference GPU considering RAW TensorFlow 2 and the optimised models for NVIDIA Tensor cores. INT8 report to the model's weights quantised into 8-bit integers, FP16 to weights into 16-bit floating-point, and FP32 to weights into 32-bit floating-point.

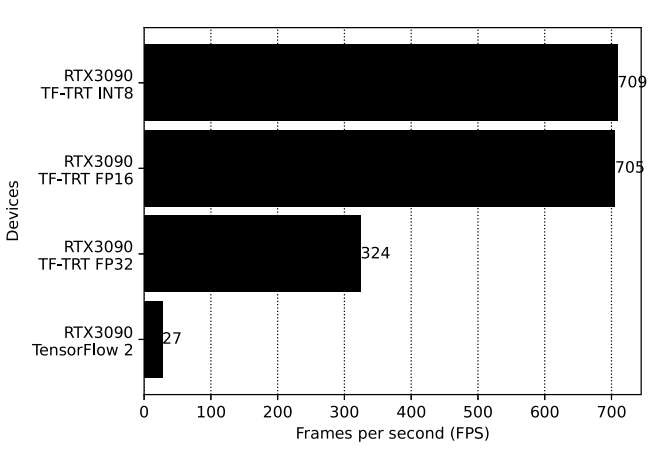


Fig. 5. Processing frame rate in the reference GPU NVIDIA RTX3090 considering RAW TensorFlow 2 and the optimised models for NVIDIA Tensor cores. INT8 report to the model's weights quantised into 8-bit integers, FP16 to weights into 16-bit floating-point, and FP32 to weights into 32-bit floating-point.

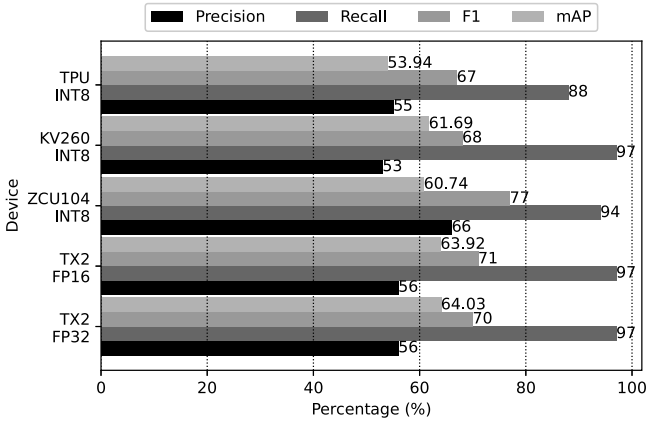


Fig. 6. Inference performance in the evaluation metrics in the edge computing devices. INT8 report to the model's weights quantised into 8-bit integers, FP16 to weights into 16-bit floating-point, and FP32 to weights into 32-bit floating-point.

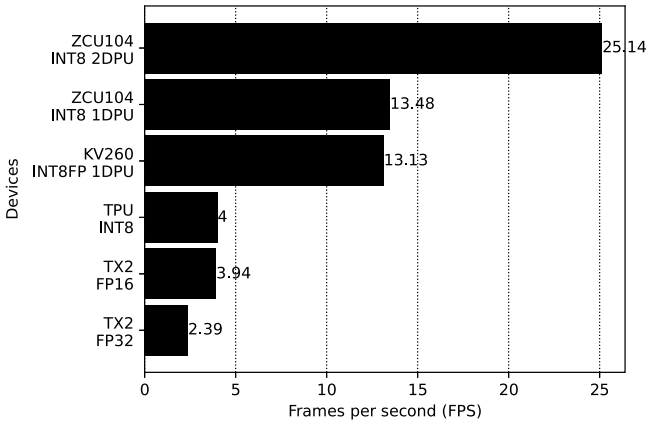


Fig. 7. Processing frame rate in the edge computing devices. INT8 report to the model's weights quantised into 8-bit integers, FP16 to weights into 16-bit floating-point, and FP32 to weights into 32-bit floating-point. FPGAs can have multiple DPU cores: 1DPU remains to the use of a single DPU and 2DPU is the simultaneous use of 2 DPU cores.

terms and speed of inference. Fig. 4 illustrates the model's accuracy in the test set. The model was compiled to optimise the hardware usage, mainly using Tensor cores. The RetinaNet got similar results in all its compiled versions but slightly better results in the default TensorFlow 2 model's version. This fact can be due to some detection's confidence drop after compilation (some detections were removed due to being inferior to the confidence threshold).

In Fig. 5 is clear the advantage of compiling the DL models for NVIDIA specifics hardware. Without modelling the weights' variables type, i.e., keeping the weights in FP32, TF-TRT could increase the inferences speed 10 times to TensorFlow 2. Reducing the weights resolution from FP32 to FP16, the models got 2.2 times faster than TF-TRT FP32 and 26 times faster than TensorFlow 2. Because RTX3090 is not optimised to operate with integers, the conversion to INT8 is meaningless.

Despite similarities, the performance in the evaluation metrics between the embedded platforms is different (Fig. 6). The model could not be assessed in any Jetson Nano due to memory and devices' limitations. The best performing device in the evaluation metrics was the TX2. This device could only compile FP32 and FP16 models because the device did not get enough memory to convert and quantise the model to INT8. TX2 got a good balance between precision and recall, which allowed for keeping F1. Conversely, the TPU was the worst performing device in the stated evaluation metrics. The quantisation process caused significant changes in the model's weights and loss of resolution, which reduced both precision and recall and, consequentially, F1. In the mid-term, the FPGAs compensates for the metrics' performance because when they reduce the recall, they increase the precision; or otherwise. The phenomena aid in keeping F1 stable between each other. The mAP follows the analysis made until now.

Fig. 7 illustrates the inference speed of the different devices in the study. The GPU was the slowest device between the heterogeneous platforms. The improvement of using FP32 against FP16 is in 1.6 times. The model could not be compiled and quantised to INT8. Conversely, FPGAs prove to be the fastest devices. While using one DPU these devices are 5.6 times faster than TX2 FP32 and 3.4 times faster than TX2 FP16 and TPU. Using the two DPUs from ZCU104, the inference reaches 25 FPS.

For better understanding of the effects of quantisation or type of variable changing, Figs. 8–10 illustrates the networks' performance in the evaluation metrics for each class. Bunch of berry-closed grapes

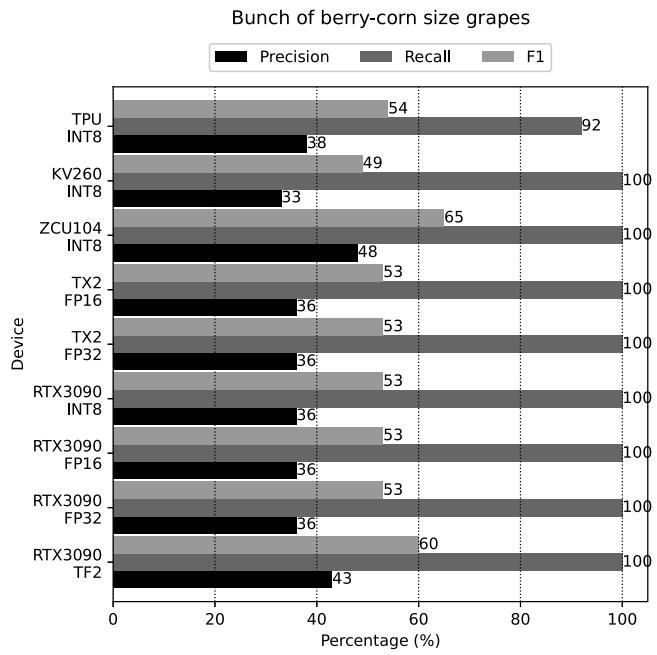


Fig. 8. Inference performance for the evaluation metrics in the different heterogeneous devices for the class of bunches of berry-corn size grapes. INT8 report to the model's weights quantised into 8-bit integers, FP16 to weights into 16-bit floating-point, and FP32 to weights into 32-bit floating-point. FPGAs can have multiple DPU cores: 1DPU remains to the use of a single DPU and 2DPU is the simultaneous use of 2 DPU cores.

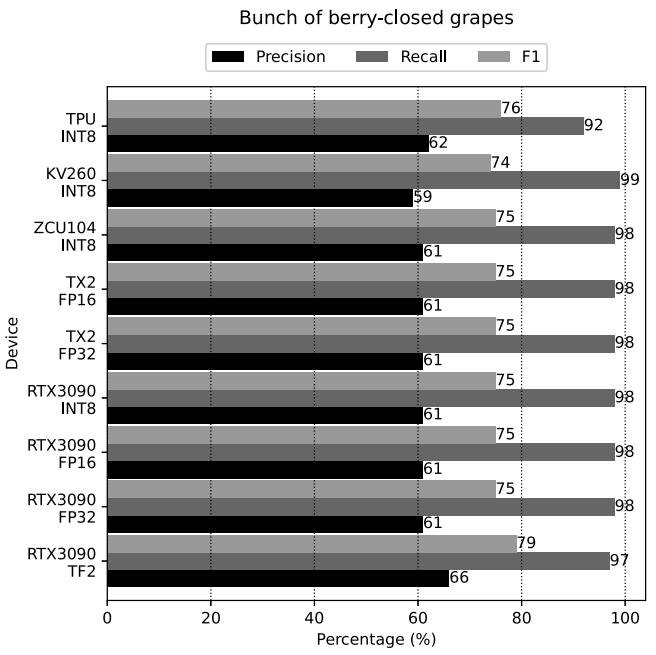


Fig. 9. Inference performance for the evaluation metrics in the different heterogeneous devices for the class bunches of berry-closed grapes class. INT8 report to the model's weights quantised into 8-bit integers, FP16 to weights into 16-bit floating-point, and FP32 to weights into 32-bit floating-point. FPGAs can have multiple DPU cores: 1DPU remains to the use of a single DPU and 2 DPU is the simultaneous use of 2 DPU cores.

(Fig. 9) is the most stable and predictable class. Changes in the network's weights do not make big changes in evaluation metrics' performance detection. Bunches of berry-corn size grapes and trunks have more difficult features (Fig. 11 and Appendix). Bunches of berry-corn size grapes are very small (these bunches appear just after inflorescence and are very similar) and have a colour similar to the background. Trunks are highly-variable in shape and size. The images also have many sources. Besides, the network confuses many masts in the vineyards as vines' trunks. The quantisation process in limited resources of TPU reduces the number of detections (Figs. 8 and 10), which reduces the TPU's recall (Eq. (3)). The reduction of the number of detections also reduces the number of FP and, consequently, the TPU's precision (Eq. (2)). ZCU104 also reveals the marginal case where quantisation reduces the network's noise and improves the detection performance of the evaluation metrics (Gong et al., 2014).

Using heterogeneous platforms in mobile systems, mainly powered by batteries, requires careful power consumption control. In the literature, these are the most common devices for mobile applications. Therefore, Fig. 12 provides power consumption for all devices. Only for inferring, all the devices consume a similar amount of energy, but they vary extremely for their operating system (standby) operations.

4. Discussion

Comparing all the benchmarked devices, it is still clear that when maximum performance in the evaluation metrics and time efficiency are required, using high-performance GPUs is the best option. However, it is important to mind that the current study does not benchmark other high-performing devices, like server-side FPGAs (like AMD-Xilinx Alveo family), but low-power heterogeneous devices that can be assembled to mobile systems like robots. The compilation of the network to the different devices did not severely change the model's performance in the evaluation metrics, despite some resolution reduction.

Inside edge computing devices, despite GPUs having the best performing results in the evaluation metrics, FPGAs were much faster.

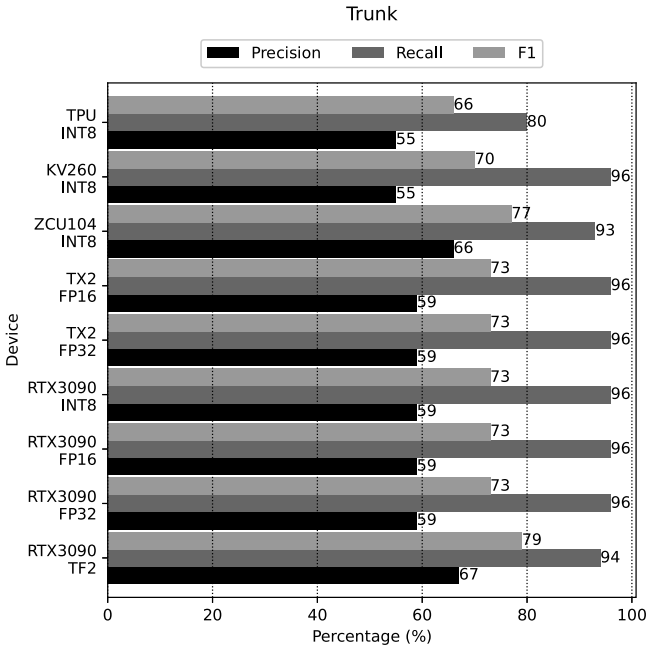


Fig. 10. Inference performance for the evaluation metrics in the different heterogeneous devices for the class of trunks. INT8 report to the model's weights quantised into 8-bit integers, FP16 to weights into 16-bit floating-point, and FP32 to weights into 32-bit floating-point. FPGAs can have multiple DPU cores: 1DPU remains to the use of a single DPU and 2DPU is the simultaneous use of 2 DPU cores.

Realise that during this study, only the model's core is being benchmarked, i.e., the authors are excluding pre-processing and post-processing layers. Therefore, due to its features, FPGAs could be more

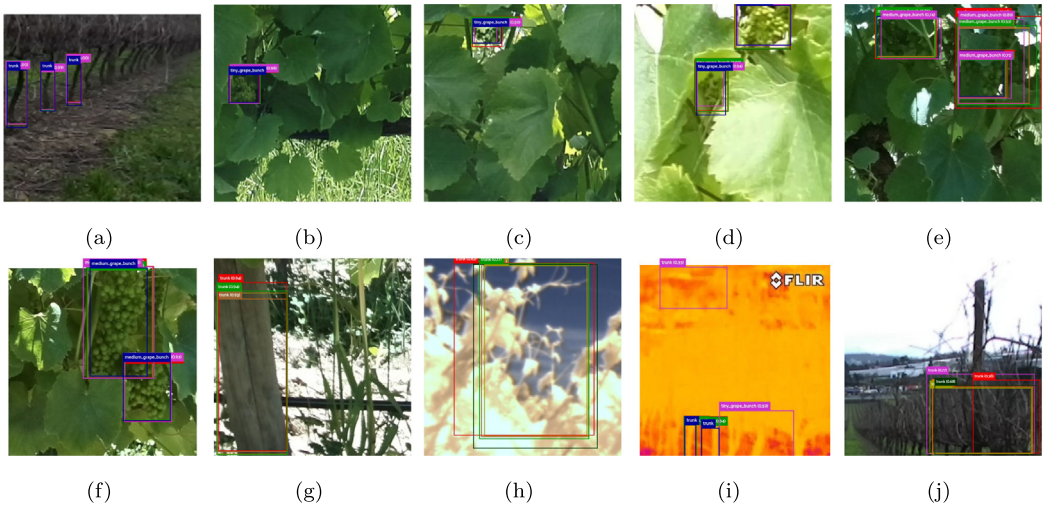


Fig. 11. Some sample images with the inference results. Details of this figure were added to Appendix in Figs. A.13 to A.22. Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.

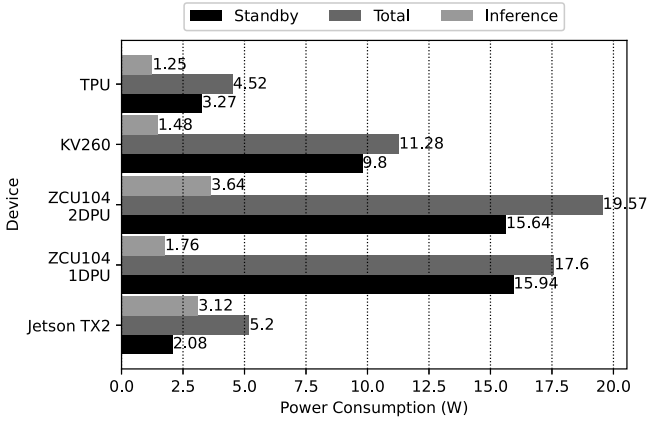


Fig. 12. Power consumption.

capable of parallelising these layers. Besides DPU, they also have the PL and an on-board GPU that can be used to optimise both blocks of layers.

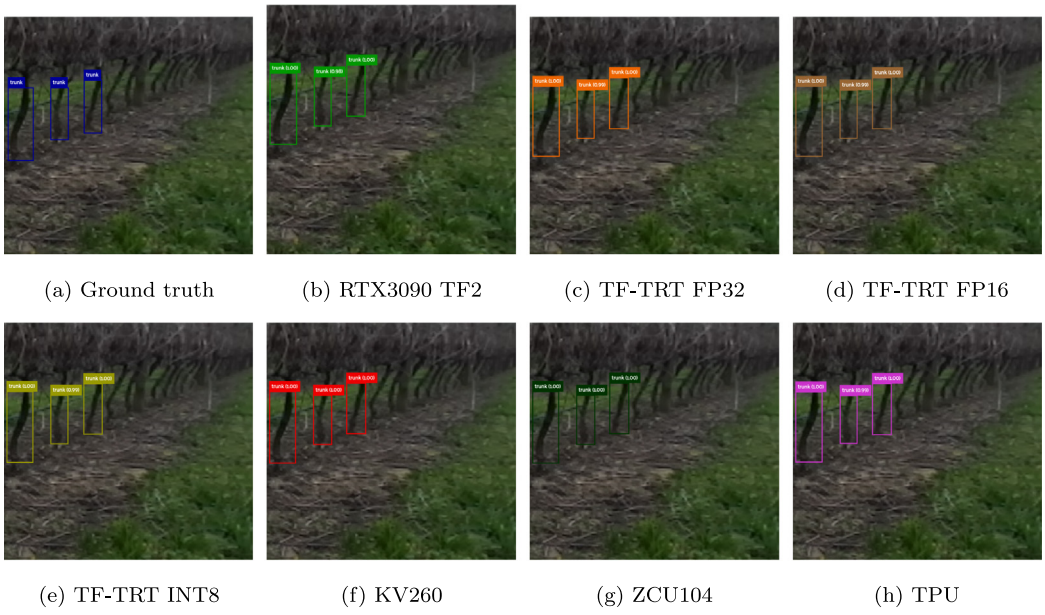
The authors also tried to benchmark NVIDIA Jetson Nano 2 GB and 4 GB, but their limited features impeded converting and compiling the model into TF-TRT. Because of that, these boards had to be excluded from this research analysis.

Fig. 11 illustrates some images of the test set with the respective detections registered for each device and ground truth. Details and extended versions of these images can be found in Appendix in Figs. A.13 to A.22. Generically, all the devices could perform well in detecting the target objects (most of the detections are clearly overlapped in the different samples). Fig. 11(e) shows one of the grapes being detected twice, which was a consequence of its size and because of being overlapped by a leaf. From the images 11(b) and 11(c) is possible to verify that berry-corn size grapes are the hardest object to detect. The reported issue is evident in Fig. 8, where the F1 score is generally lower than 60 %. However, this could not be an impact issue in practical applications once other landmarks can be used for robot localisation, for instance. Nevertheless, trunks' and berry-closed size grapes' detection is more important. The trunks' class is very important for obstacles and the robot's localisation, while the berry-closed size grapes are usually targeted for performing tasks like monitoring or harvesting. These two classes have detection ratios between 70 % to 80 % (Figs. 9 and 10), which are feasible for practical applications. Therefore, the low mAP of about 60 % illustrated in Figs. 4 and 6 can be induced by the low detection ratio of berry-corn size grapes. Figs. 11(f) to 11(j) depict some detection errors introduced by the different model's versions. The current detection ratios of the different ANNs' versions should conduct further improvements at two levels: optimise the neural network's structure and parameters and deeply review the dataset. Hyper-parameters such as the confidence threshold can be optimised (Magalhães et al., 2021). The metric results also reveal a possible misannotation of some objects that are being correctly identified, i.e., some objects like trunks could be successfully detected by the model, but they were not labelled in the ground truth.

In the revised literature, no publication researched the application of RetinaNet ResNet-50 or SSD ResNet-50 FPN in heterogeneous devices. So, it is not possible directly compare our results with state-of-the-art results. Although the results show that our experience was slightly slower than state-of-the-art results, RetinaNet ResNet-50 is more complex than YOLO and SSD MobileNet. Given the fast inference times with high-performing rates, which sometimes are similar to YOLO results from the revised literature, the authors can conclude that the research from this work is suitable for near real-time applications.

Aguiar et al. (2021) also essayed the VineSet dataset using an SSD object detection model with two backbone feature extractors, MobileNet v1 and Inception v2, in a USB Coral Accelerator TPU. They reached a mAP of 66.96 % and 55.78 %, respectively, without the trunk's class. Given the conditions of not using the trunks' class, we can assume that these are similar results to ours, and the inclusion of trunks in the dataset may lead to the metrics' degradation. Therefore, we can induce that we are working near the limits of the dataset, requiring a deeper labelling review to identify possible misannotations. Besides, Aguiar et al. (2021) performed an inference threshold analysis to identify the best confidence score that optimises the metrics, while we are using a standard confidence score of 30 %.

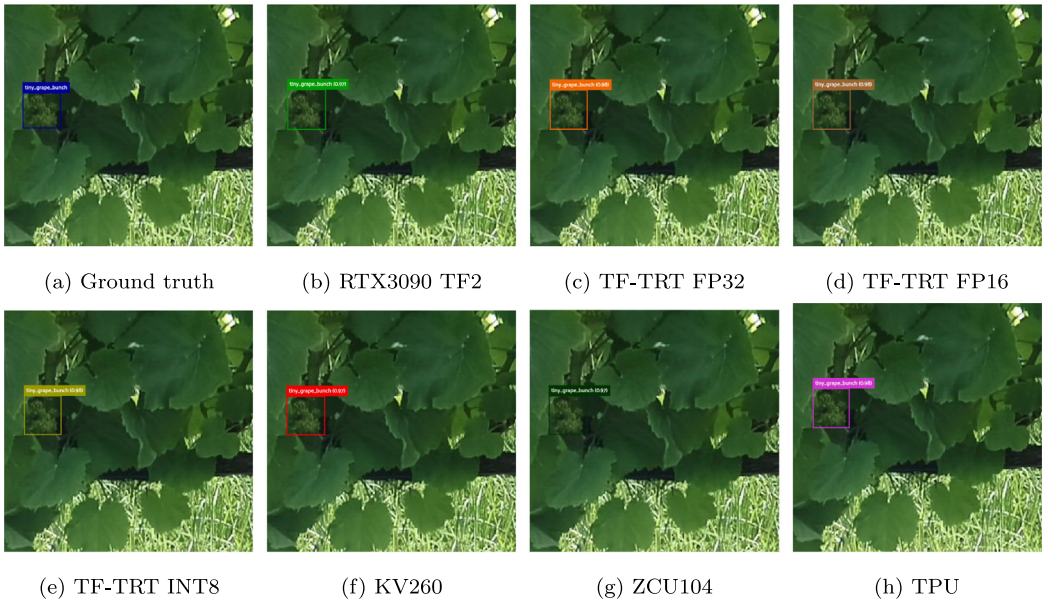
As expected, MobileNet networks aim to be faster and designed for mobile applications. Similarly, Inception networks are also less complex than ResNet networks and, because of that, faster. Inside a TPU, the networks reached 158.98 FPS and 38.36 FPS, respectively. Undoubtedly,



(a) Ground truth (b) RTX3090 TF2 (c) TF-TRT FP32 (d) TF-TRT FP16

(e) TF-TRT INT8 (f) KV260 (g) ZCU104 (h) TPU

Fig. A.13. Detailed sample image 11(a) from Fig. 11 from Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.



(a) Ground truth (b) RTX3090 TF2 (c) TF-TRT FP32 (d) TF-TRT FP16

(e) TF-TRT INT8 (f) KV260 (g) ZCU104 (h) TPU

Fig. A.14. Detailed sample image 11(b) from Fig. 11 from Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.

this previous work reached faster performances than ours with similar performances. However, it is unclear if there is any difference in the networks' performance between a USB Accelerator Edge TPU and the Dev Board TPU. The authors did not make a formal power consumption analysis but could infer an average power consumption of 2.5 W for the USB stick, ignoring all the power consumption for the computer maintenance and processing.

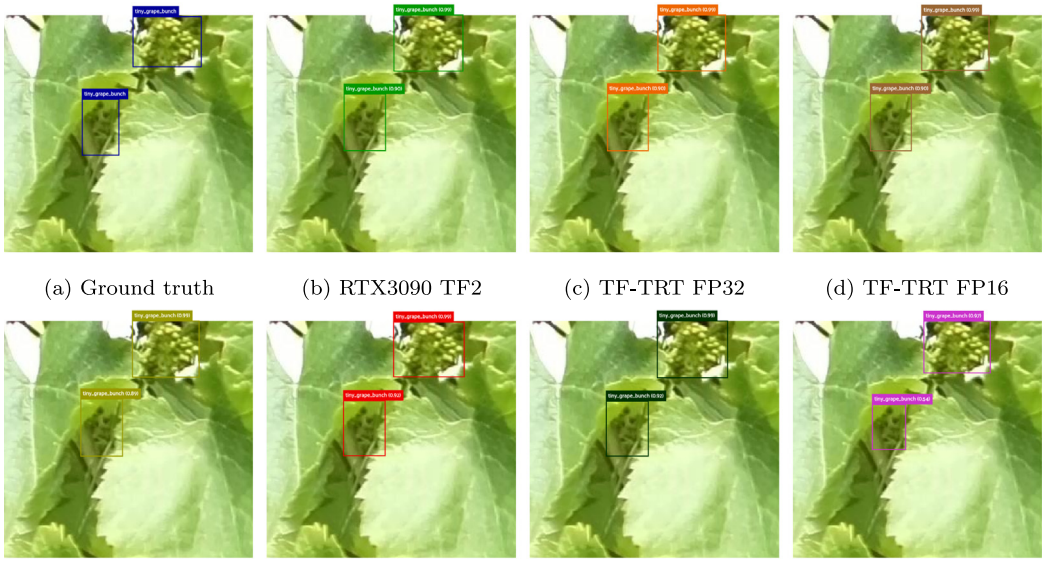
Considering our results, the TPU is the best solution when reducing the power is a demand, despite the small variations in the networks' performance in the evaluation metrics and the reduced inference speed. However, when applications are looking for a balance between power consumption and inference speed, KV260 has high potential. In all the cases, keep in mind that ZCU104 and KV260 have installed a standard PetaLinux[9] image provided by AMD-Xilinx. These images have

[9] See AMD-Xilinx, 2022, PetaLinux Tools, URL: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html, Last accessed on 05/08/2022.

(a) Ground truth (b) RTX3090 TF2 (c) TF-TRT FP32 (d) TF-TRT FP16

(e) TF-TRT INT8 (f) KV260 (g) ZCU104 (h) TPU

Fig. A.15. Detailed sample image 11(c) from Fig. 11 from Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.



(a) Ground truth (b) RTX3090 TF2 (c) TF-TRT FP32 (d) TF-TRT FP16

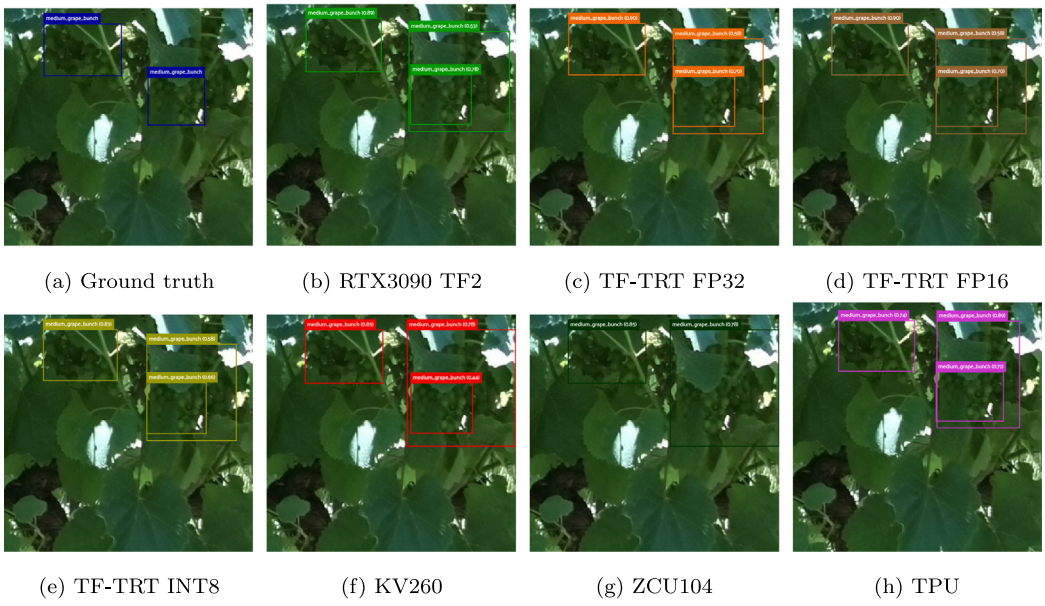
(e) TF-TRT INT8 (f) KV260 (g) ZCU104 (h) TPU

Fig. A.16. Detailed sample image 11(d) from Fig. 11 from Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.

all the FPGAs' resources active. Most of the resources are not necessary. Therefore, a deeper analysis with a better configured PetaLinux image can better assess the power consumption of FPGAs.

5. Conclusions

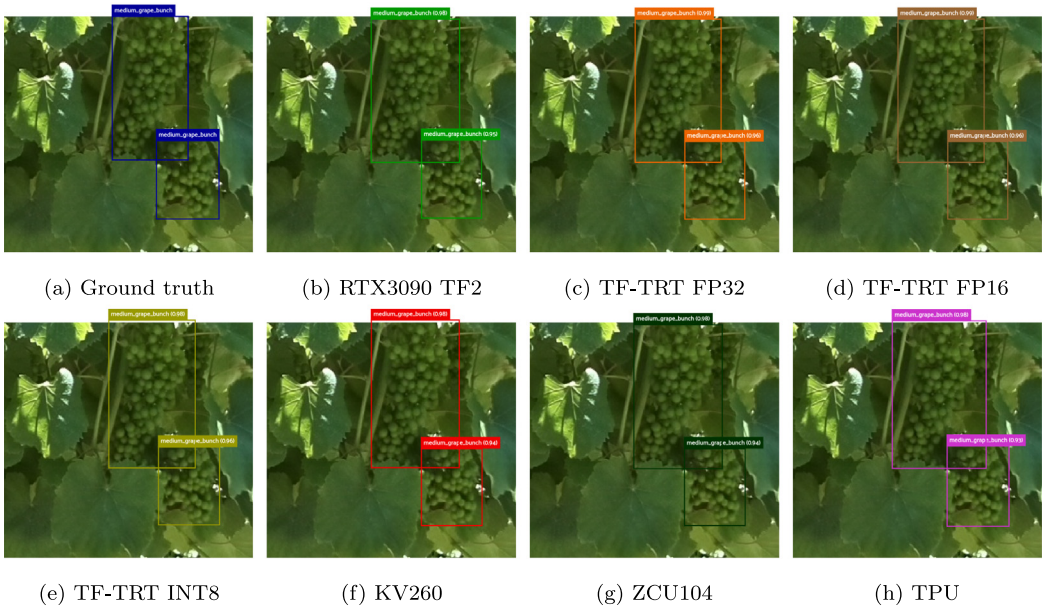
In this work, multiple heterogeneous platforms (i.e., GPU, TPU, and FPGA) were benchmarked using RetinaNet ResNet-50. The code used in this work is publicly available at GitLab INESC TEC, URL: https://gitlab.inesctec.pt/agrob/xilinx-acc2021. AMD-Xilinx ZCU104 performed better than the other benchmarked platforms because of its fast inference speed. Besides, ZCU104 also has the possibility to execute two models simultaneously. Furthermore, FPGAs offer more flexibility to implement and parallelise algorithms because of their onboard CPU, GPU and PL. TPU are better optimised and specified for running ANN (but more task restrictive), offering a lower power consumption. These



(a) Ground truth (b) RTX3090 TF2 (c) TF-TRT FP32 (d) TF-TRT FP16

(e) TF-TRT INT8 (f) KV260 (g) ZCU104 (h) TPU

Fig. A.17. Detailed sample image 11(e) from Fig. 11 from Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.



(a) Ground truth (b) RTX3090 TF2 (c) TF-TRT FP32 (d) TF-TRT FP16

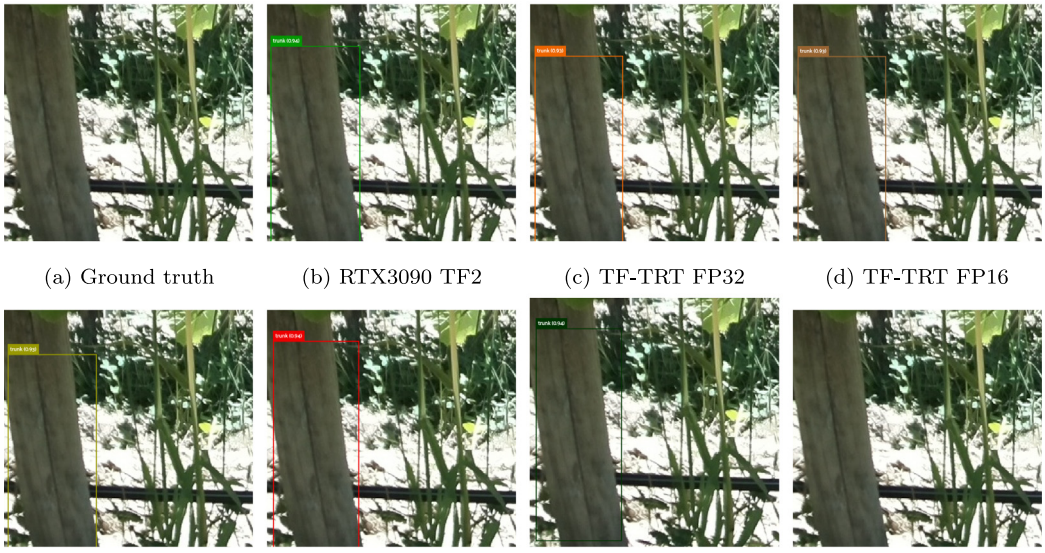
(e) TF-TRT INT8 (f) KV260 (g) ZCU104 (h) TPU

Fig. A.18. Detailed sample image 11(f) from Fig. 11 from Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.

devices may be the recommended option when saving power is crucial and the application is not time-restrictive.

Concerning the frameworks for ANNs' deploying, all of them have similar steps. Vitis-AI is the most complete but complex framework, becoming the hardest to use. Conversely, Edge TPU Compiler and TF-TRT are similar and easier to use, but they depend strongly on TensorFlow. Edge TPU Compiler is the easiest framework because it has cross-compiling capabilities, allowing to use of more powerful devices to deploy the model for the TPU. TF-TRT requires the model to be compiled on-device, highlighting the devices' limitations.

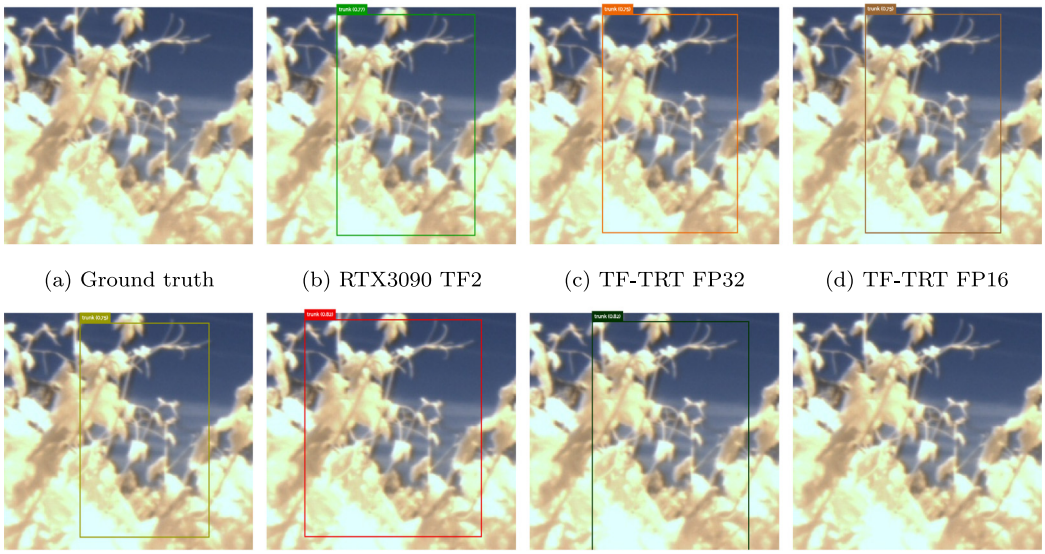
Future work intends to optimise the researched DL model by applying some optimisation strategies like pruning and exploring the use of binary neural networks. Besides, GPUs, TPUs and FPGAs have computational resources that could be considered to redesign and optimise RetinaNet ResNet-50 nodes to reach lower inference times. The authors will also evaluate other pre-processing and post-processing techniques



(a) Ground truth (b) RTX3090 TF2 (c) TF-TRT FP32 (d) TF-TRT FP16

(e) TF-TRT INT8 (f) KV260 (g) ZCU104 (h) TPU

Fig. A.19. Detailed sample image 11(g) from Fig. 11 from Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.



(a) Ground truth (b) RTX3090 TF2 (c) TF-TRT FP32 (d) TF-TRT FP16

(e) TF-TRT INT8 (f) KV260 (g) ZCU104 (h) TPU

Fig. A.20. Detailed sample image 11(h) from Fig. 11 from Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.

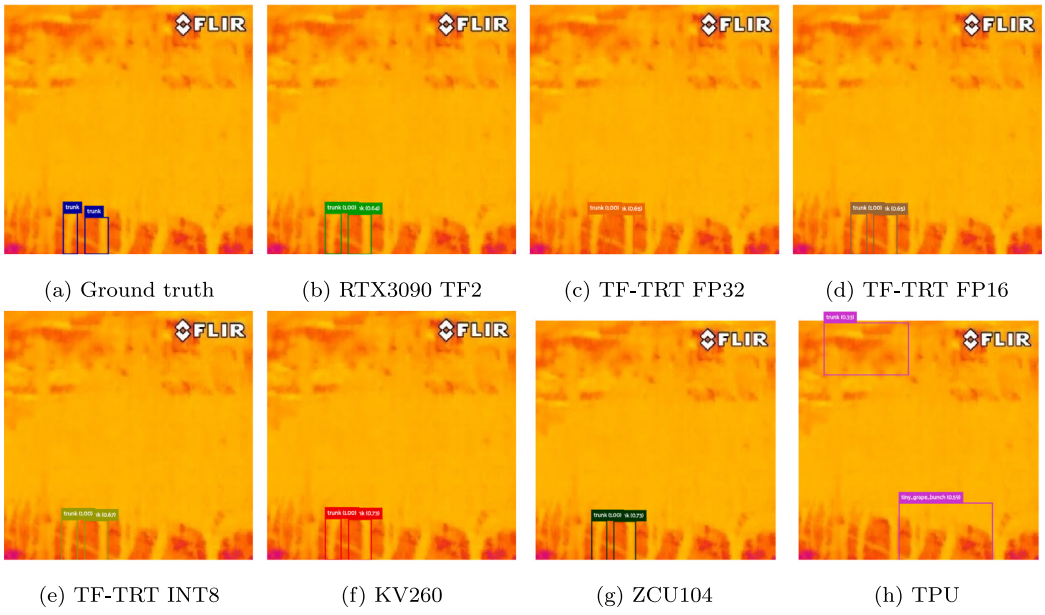
for reducing the inference time. Besides, the current work allows the authors to identify possible issues in the dataset labelling, therefore, a deep review of the dataset labels must be an important future step.

CRediT authorship contribution statement

Sandro Costa Magalhães: Conceptualisation, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Visualization, Funding acquisition. **Filipe Neves dos Santos:** Conceptualisation, Validation, Resources, Writing – review & editing, Supervision, Funding acquisition. **Pedro Machado:** Conceptualisation, Methodology, Validation, Investigation, Writing – original draft, Writing – review & editing, Supervision. **António Paulo Moreira:** Validation, Resources, Writing – review & editing, Supervision. **Jorge Dias:** Validation, Writing – review & editing, Supervision.

Declaration of competing interest

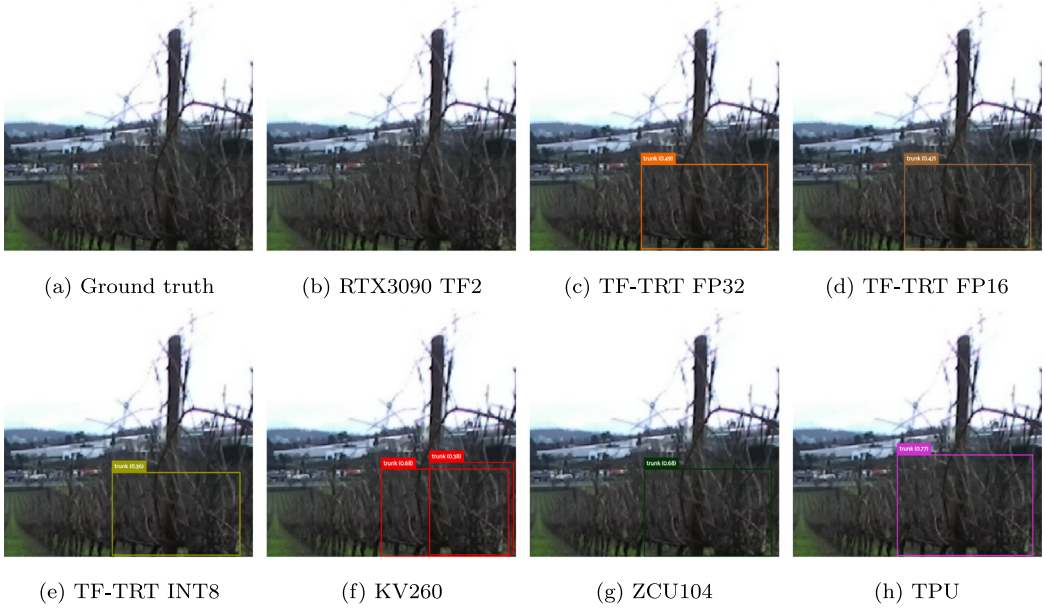
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.



(a) Ground truth (b) RTX3090 TF2 (c) TF-TRT FP32 (d) TF-TRT FP16

(e) TF-TRT INT8 (f) KV260 (g) ZCU104 (h) TPU

Fig. A.21. Detailed sample image 11(i) from Fig. 11 from Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.



(a) Ground truth (b) RTX3090 TF2 (c) TF-TRT FP32 (d) TF-TRT FP16

(e) TF-TRT INT8 (f) KV260 (g) ZCU104 (h) TPU

Fig. A.22. Detailed sample image 11(j) from Fig. 11 from Blue – ground-truth; light green – NVIDIA RTX3080 TF2; orange – NVIDIA RTX3090 TF-TRT FP32; brown – NVIDIA RTX3090 TF-TRT FP16; dark yellow – NVIDIA RTX3090 TF-TRT INT8; red – AMD-Xilinx Kria KV260; dark green – AMD-Xilinx ZCU104; pink – Coral Dev Board TPU.

Funding

Sandro Costa Magalhães was granted by the Portuguese funding agency, Fundação para a Ciência e Tecnologia (FCT), and the European Social Fund (ESF) under scholarship SFRH/BD/147117/2019. This work was supported by the European Union's Horizon 2020 Research and Innovation Program under Grant 101004085.

Data availability

Data and Software were made publicly available on open institutional repositories.

Appendix. Sample images of the dataset

For better readability of Fig. 11, this appendix supplements the same figure with one annotation kind per images in the Figs. A.13 to A.22

References

Aguiar, A.S., Magalhães, S., 2021. Grape bunch and vine trunk dataset for deep learning object detection. http://dx.doi.org/10.5281/ZENODO.5139598, [Dataset].

Aguiar, A.S., Magalhães, S.A., dos Santos, F.N., Castro, L., Pinho, T., Valente, J., Martins, R., Boaventura-Cunha, J., 2021. Grape bunch detection at different growth stages using deep learning quantized models. Agronomy 11 (9), 1890. http://dx.doi.org/10.3390/agronomy11091890.

AMD-Xilinx, 2022. DPU for convolutional neural network. URL https://www.xilinx.com/products/intellectual-property/dpu.html.

Bolya, D., Zhou, C., Xiao, F., Lee, Y.J., 2019. YOLACT: Real-time instance segmentation. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV). IEEE, Seoul, Korea (South), http://dx.doi.org/10.1109/iccv.2019.00925.

Chen, Y., Zhang, K., Gong, C., Hao, C., Zhang, X., Li, T., Chen, D., 2019. T-DLA: An open-source deep learning accelerator for ternarized DNN models on embedded FPGA. In: 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, Miami, FL, USA, http://dx.doi.org/10.1109/isvlsi.2019.00012.

Chiu, Y.-C., Tsai, C.-Y., Ruan, M.-D., Shen, G.-Y., Lee, T.-T., 2020. Mobilenet-SSDv2: An improved object detection model for embedded systems. In: 2020 International Conference on System Science and Engineering (ICSSE). IEEE, Kagawa, Japan, http://dx.doi.org/10.1109/icsse50014.2020.9219319.

de Andrade, H.S., 2018. Software Concerns for Execution on Heterogeneous Platforms (Ph.D. thesis). Chalmers University of Technology.

Gong, Y., Liu, L., Yang, M., Bourdev, L., 2014. Compressing deep convolutional networks using vector quantization. arXiv:1412.6115.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Las Vegas, NV, USA, http://dx.doi.org/10.1109/cvpr.2016.90.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861.

Humbarwadi, S., 2020. Object detection with RetinaNet. URL https://github.com/keras-team/keras-io/blob/master/examples/vision/retinanet.py Online.

Intel, 2020. What is a GPU? Graphics processing units defined. URL https://www.intel.co.uk/content/www/uk/en/products/docs/processors/what-is-a-gpu.html.

Jain, V., Jadhav, N., Verhelst, M., 2021. Enabling real-time object detection on low cost FPGAs. J. Real-Time Image Process. 19 (1), 217–229. http://dx.doi.org/10.1007/s11554-021-01177-w.

Kovács, B., Henriksen, A.D., Stets, J.D., Nalpantidis, L., 2021. Object detection on TPU accelerated embedded devices. In: Lecture Notes in Computer Science. Springer International Publishing, pp. 82–92. http://dx.doi.org/10.1007/978-3-030-87156-7_7.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A., 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. J. Mach. Learn. Res. 18 (185), 1–52, URL http://jmlr.org/papers/v18/16-558.html.

Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2017. Feature pyramid networks for object detection. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Honolulu, HI, USA, http://dx.doi.org/10.1109/cvpr.2017.106.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollar, P., 2020. Focal loss for dense object detection. IEEE Trans. Pattern Anal. Mach. Intell. 42 (2), 318–327. http://dx.doi.org/10.1109/tpami.2018.2858826.

Lin, G.-Z., Nguyen, H.M., Sun, C.-C., Kuo, P.-Y., Sheu, M.-H., 2021. A novel bird detection and identification based on DPU processor on PYNQ FPGA. In: 2021 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW). IEEE, Penghu, Taiwan, http://dx.doi.org/10.1109/icce-tw52618.2021.9603066.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C., 2016. SSD: Single shot MultiBox detector. In: Computer Vision – ECCV 2016. Springer International Publishing, pp. 21–37. http://dx.doi.org/10.1007/978-3-319-46448-0_2.

Magalhães, S.A., Castro, L., Moreira, G., dos Santos, F.N., Cunha, M., Dias, J., Moreira, A.P., 2021. Evaluating the single-shot MultiBox detector and YOLO deep learning models for the detection of tomatoes in a greenhouse. Sensors 21 (10), 3569. http://dx.doi.org/10.3390/s21103569.

Magalhães, S., Santos, F.N.D., Shyam, S., 2022. Grape detection using Vitis AI and RetinaNet. URL https://www.hackster.io/452741/grape-detection-using-vitis-ai-and-retinanet-7d0d71 Online.

Martinez, R.P., Schiopu, I., Cornelis, B., Munteanu, A., 2021. Real-time instance segmentation of traffic videos for embedded devices. Sensors 21 (1), 275. http://dx.doi.org/10.3390/s21010275.

Mendes, J., dos Santos, F.N., Ferraz, N., Couto, P., Morais, R., 2016. Vine trunk detector for a reliable robot localization system. In: 2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC). IEEE, Bragança, Portugal, http://dx.doi.org/10.1109/icarsc.2016.68.

Moreira, G., Magalhães, S.A., Pinho, T., dos Santos, F.N., Cunha, M., 2022. Benchmark of deep learning and a proposed HSV colour space models for the detection and classification of greenhouse tomato. Agronomy 12 (2), 356. http://dx.doi.org/10.3390/agronomy12020356.

Morera, A., Sanchéz, A., Moreno, A.B., Sappa, A.D., Vélez, J.F., 2020. SSD vs. YOLO for detection of outdoor urban advertising panels under multiple variabilities. Sensors 20 (16), http://dx.doi.org/10.3390/s20164587, URL https://www.mdpi.com/1424-8220/20/16/4587.

Olenskyj, A.G., Sams, B.S., Fei, Z., Singh, V., Raja, P.V., Bornhorst, G.M., Earles, J.M., 2022. End-to-end deep learning for directly estimating grape yield from ground-based imagery. Comput. Electron. Agric. 198, 107081. http://dx.doi.org/10.1016/j.compag.2022.107081.

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al., 2019. Kerastuner.

Puchtler, P., Peinl, R., 2020. Evaluation of deep learning accelerators for object detection at the edge. In: Lecture Notes in Computer Science. Springer International Publishing, pp. 320–326. http://dx.doi.org/10.1007/978-3-030-58285-2_29.

Rahmaniar, W., Hernawan, A., 2021. Real-time human detection using deep learning on embedded platforms: A review. J. Robot. Control 2 (6), http://dx.doi.org/10.18196/jrc.26123.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: Unified, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Las Vegas, NV, USA, http://dx.doi.org/10.1109/cvpr.2016.91.

Redmon, J., Farhadi, A., 2018. YOLOv3: An incremental improvement. arXiv:1804.02767.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L., 2015. ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. 115 (3), 211–252. http://dx.doi.org/10.1007/s11263-015-0816-y.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C., 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE, Salt Lake City, UT, USA, http://dx.doi.org/10.1109/cvpr.2018.00474.

Sozzi, M., Cantalamessa, S., Cogato, A., Kayad, A., Marinello, F., 2022. Automatic bunch detection in white grape varieties using YOLOv3, YOLOv4, and YOLOv5 deep learning algorithms. Agronomy 12 (2), 319. http://dx.doi.org/10.3390/agronomy12020319.

Suzen, A.A., Duman, B., Sen, B., 2020. Benchmark analysis of jetson TX2, jetson nano and raspberry PI using deep-CNN. In: 2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA). IEEE, Ankara, Turkey, http://dx.doi.org/10.1109/hora49412.2020.9152915.

Tan, L., Huangfu, T., Wu, L., Chen, W., 2021. Comparison of RetinaNet, SSD, and YOLO v3 for real-time pill identification. BMC Med. Inf. Decis. Mak. 21 (324), http://dx.doi.org/10.1186/s12911-021-01691-8.

Terra, F., Rodrigues, L., Magalhaes, S., Santos, F., Moura, P., Cunha, M., 2021. PixelCropRobot, a cartesian multitask platform for microfarms automation. In: 2021 International Symposium of Asian Control Association on Intelligent Robotics and Industrial Automation (IRIA). IEEE, http://dx.doi.org/10.1109/iria53009.2021.9588786.

Venieris, S.I., Bouganis, C.-S., 2017. fpgaConvNet: A toolflow for mapping diverse convolutional neural networks on embedded FPGAs. arXiv:1711.08740.

Wang, C., Peng, Z., 2019. Design and implementation of an object detection system using faster R-CNN. In: 2019 International Conference on Robots & Intelligent System (ICRIS). IEEE, Haikou, China, http://dx.doi.org/10.1109/icris.2019.00060.

Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., sheng Hua, X., 2019. Quantization networks. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Long Beach, CA, USA, http://dx.doi.org/10.1109/cvpr.2019.00748.

Zhao, H., Wang, C., Guo, R., Rong, X., Guo, J., Yang, Q., Yang, L., Zhao, Y., Li, Y., 2022. Autonomous live working robot navigation with real-time detection and motion planning system on distribution line. High Volt. http://dx.doi.org/10.1049/hve2.12221.

Zhao, H., Zhang, W., Sun, H., Xue, B., 2019. Embedded deep learning for ship detection and recognition. Future Internet 11 (2), 53. http://dx.doi.org/10.3390/fi11020053.

Zhao, X., Zhang, X., Yang, F., Xu, P., Li, W., Chen, F., 2021. Research on machine learning optimization algorithm of CNN for FPGA architecture. J. Phys. Conf. Ser. 2006 (1), 012012. http://dx.doi.org/10.1088/1742-6596/2006/1/012012.