High performance inference of gait recognition models on embedded systems

Paula Ruiz-Barroso[a], Francisco M. Castro[a,*], Rubén Delgado-Escaño[a], Julián Ramos-Cózar[a], Nicolás Guil[a]

[a]*Department of Computer Architecture, University of Málaga, Spain*

Abstract

Edge computing is gaining importance in the realm of Deep Learning, particularly after powerful devices such as recent heterogeneous embedded systems have demonstrated remarkable skills for accelerating their challenging computational requirements. In this work, we evaluate different hardware and software optimizations applied to state-of-the-art gait recognition approaches deployed on two Jetson devices with very different hardware capabilities. Specifically, we have selected three models with different characteristics in order to provide an in-depth deployment evaluation. This way, a 2D convolution-based model allows us to evaluate devices performance when a huge number of parameters must be managed. A model based on 3D convolutions is deployed to study devices capability to perform these kinds of operations. Finally, a novel model with a small number of parameters but with a huge number of activations is also evaluated. Obtained results show that different hardware and software optimizations are able to improve up to 5.4× energy consumption and 4.2× execution time w.r.t. baseline deployment, depending on the model and target device[1].

Keywords: Embedded systems, Deep Learning, Gait Recognition, Model Optimization

1. Introduction

Gait is a kind of biometric feature, like fingerprints or iris, that allows people identification using their way of walking. Gait has been studied from medical viewpoints [1, 2], human-computer interaction [3, 4], performance analysis in sport [5], etc. In this paper, we focus on its use as biometric feature for people identification as done in many previous works [6]. Its main advantages are that subjects can be identified without collaborating with the system and it can be performed at a certain distance. Therefore, over the last years, great effort has been put into gait recognition [7, 8, 9, 10] using many different input data types, such as silhouettes [11], Gait Energy Images [12], optical flow [13] or inertial sensors [14]. With the advent of deep learning, gait recognition approaches took advantage of this

*Corresponding author. *Postal address:* Bulevar Louis Pasteur, 35, Office 2.3.8a, 29071, Malaga, Spain. *Phone number:* +34952133388

[1]Code will be available at `https://github.com/PaulaRuizB/Embedded-Gait`

learning paradigm to propose new approaches to obtaining accuracies never seen before in the field of gait.

Typically, the deployment of any deep learning (DL) application needs the use of high-performance architectures to fulfil both inference latency and energy consumption requirements. This is due to the huge amount of computations required by the models. For example, widely used ResNet models [15] require between 1.8 to 7.3 GFLOP (billions of floating point operations) per each input sample. Modern Vision Transformers (ViT) [16] can reach peaks of 5.1 TFLOP using up to 2.44 billion parameters. This way, discrete Graphics Processing Units (GPUs), which are connected to the CPU through an I/O bus, are commonly used to build serving systems that attend inference requests coming from many users [17, 18]. However, when these applications have to work in embedded platforms, energy consumption is a paramount issue that must also be taken into account. In these platforms, energy-efficient CPUs and accelerators are integrated on the same chip achieving a good trade-off between performance and consumption [19, 20].

Different techniques can be applied to deep learning models in order to reduce both the computational and energy consumption requirements. Thus, using knowledge transfer, a so-called *teacher* model can be compressed by generating a smaller *student* model that achieves a similar accuracy [21]. In addition, pruning techniques [22] can reduce the number of arithmetic operations by removing filters and even layers of the model. Finally, more simple data representation can be used using quantization methods [23]. Thus, instead of using 32-bit floating-point data representation for inference, the model could employ 16-bit floating-point or even 8-bit integer data. As previous techniques modify the original model or their parameter precision, the original accuracy also changes.

Additional improvements that take advantage of the underlying system architecture can be also applied. Concretely, batching and streams can be employed on GPUs [24, 25]. Thus, batching groups several independent inferences that could be performed in parallel while streams enable the concurrent execution of inferences on GPUs if hardware resources are available. Both techniques can increase the application throughput but also can harm inference latency.

In this work we use two embedded heterogeneous architectures: NVIDIA Jetson Nano 4GB and NVIDIA Jetson AGX Xavier to deploy gait recognition models. This way, we can study the impact of all the proposed model improvements in architectures offering very different capabilities. Thus, Jetson Nano is an low-cost simple device for deploying deep learning models in limited hardware, while Jetson AGX Xavier is an advanced level device that includes specialized deep learning accelerators (DLAs) and more powerful CPUs and GPU at a higher cost.

Focusing on the deep learning models, we use three state-of-the-art models relying on the three most common kinds of approaches to gait recognition: *(i)* based on 2D convolutions with a huge amount of parameters that work with spatial information, (ii) based on 3D convolutions and a medium amount

of parameters managing spatio-temporal information and, (iii) based on independent processing of sample frames, which produces a huge amount of activations with a reduced number of parameters.

As we want to explore the impact of optimizations of the original models on different embedded platforms, several versions of the models have been studied with the aim of reducing both inference time and energy consumption requirements while recognition accuracy is kept high.

Thus, the main contributions of this paper can be summarized in:

- A combined energy consumption and performance analysis on two widely used embedded platforms using three state-of-the-art gait recognition models with different intrinsic characteristics together with a comparison among models and devices.
- A thorough study of software (pruning) and hardware (quantization) optimizations that can be applied to deep learning models according to the target devices and architectures (CPU, GPU and DLA).
- A simplification of 3D convolutions based models to 2D convolutions that reduce computational complexity and can be deployed without restrictions on both embedded platforms.
- An in-depth study about the benefits of using more than one sample (batch) at the same time during inference.

The rest of the paper is organized as follows. After presenting some background in Sec. 2, Sec. 3 summarizes the related work. Our methodology is described in Sec. 4. Then, in Sec. 5, we define the experimental setup and discuss the results and, finally, Sec. 6 concludes the paper.

2. Background

2.1. Embedded architectures

Nowadays, energy consumption has become an important concern in computation systems design. Performance metrics based on GFLOPS (Giga Floating Point Operations per Second) have been replaced for energy-aware figures of merit, such as the GFLOPS per Watt (GFLOPS/W). Similarly, Energy Delay Product (EDP) is often employed together with execution time value. The importance of energy minimization is even more crucial in embedded systems, where battery life has to be taken into account. For this reason, embedded systems manufacturers include new features in their boards such as several CPU cores, GPGPU accelerators and DFVS (Dynamic Frequency and Voltage Scaling) [26].

Compared with Field Programmable Gate Arrays (FPGAs), these systems can be programmed more easily by far [27]. CUDA [28] and OpenCL [29], only to mention a couple of APIs, have

contributed to making programming a GPU similar to programming a CPU. An interesting and simple approach to performance and energy optimization in these platforms consists of delegating to the GPU the most computational demanding tasks as they typically run more efficiently than on CPU.

Current embedded systems usually include a number of CPU cores highly coupled with some GPU accelerators. In Tab. 1 we show several embedded heterogeneous platforms together with the number of their computational resources and operating frequency values. As can be observed, all these platforms include CPU and GPU cores. This way, applications can use both computational resources attending to the characteristics of their computing kernels. Typically, DL models demand high memory bandwidth and computing intensity and, consequently, DL models are deployed in GPU since it is specifically designed for these types of workloads. In addition, GPU has better relation (lower values) in GFLOPS/W than CPU.

In this work, we have employed two different platforms: NVIDIA Jetson Nano 4GB and Nvidia Jetson AGX Xavier. Both platforms were designed to be used as low power consumption systems for embedded computer vision applications with a reasonable good performance. They offer a wide range of operating frequencies for their main devices: the CPU, the GPU and the eMMc. Additionally, the CPU cores can be connected or disconnected individually. All of these features allow a high degree of hardware customization and, consequently, accomplish a high variety of performance and power consumption trade-offs. Specifically, Jetson Nano includes a quad-core ARM Cortex-A57 processor and a Maxwell GPU with 128 cores accompanied by 4 GB of main memory shared by the ARM processors and the GPU. On the other hand, Jetson AGX Xavier includes an octa-core NVIDIA Carmel ARM v8.2 and a Volta GPU with 512 cores and 64 Tensor Cores accompanied by 32 GB of main memory shared by the ARM processors and the GPU. Moreover, Jetson AGX Xavier also includes two deep learning accelerators (DLAs) and one vision accelerator (VA). Note that DLA is slower and smaller than the GPU, therefore it is expected that execution time increases using DLA and GPU with respect to performing computation on just GPU. Thus, we have used an entry-level platform that includes enough hardware to deploy deep learning models with acceptable execution times and, an advanced platform which includes specific inference hardware to obtain better execution times.

2.2. Deep learning models

A deep learning model is a set of mathematical operations represented as a graph composed of different layers that encode the operations performed to the input data. Typical operations are convolutions, poolings, activation functions or dense multiplications. Once the architecture or set of layers of the approach is defined, the weights must be optimized in order to produce good outputs for

Board Name	CPU	GPU	GFLOPS$_{HP}$	GFLOPS$_{SP}$	GFLOPS$_{DP}$
Jetson AGX Xavier	8x NVIDIA Carmel ARM v8.2@2.03GHz	512-core Nvidia Volta with 64 Tensor Cores@1.21GHz	11000	5500	2750
Jetson Xavier NX	6x NVIDIA Carmel ARM v8.2@1.9GHz	384-core Nvidia Volta with 48 Tensor Cores@1.1GHz	6000	3000	1500
Jetson TX2	4x A57 @2GHz + 2x NVIDIA Denver2@2GHz	Pascal 256@1300MHz	1500	750	23.44
Jetson Nano	4x A57@1.43GHz	Nvidia Maxwell 128@921 MHz	471.6	235.8	7.37
Raspberry Pi 4 Model B	4x A72@1.5GHz	VideoCore VI@500MHz	64	32	8

Table 1: Main characteristics of modern embedded computing boards with heterogeneous architecture. Theoretical GFLOPS for Half Precision (HP), Single Precision (SP) and Double Precision (DP) are included.

the desired task. This optimization is carried out during the training process using a set of samples that guides the optimization process. Then, once the model is trained, it can be used at inference time to produce outputs for new data.

In order to accelerate the inference step, deep learning models are prepared for inference by removing unnecessary layers (e.g. dropout or batch normalization) that only are used during training, or applying quantization to reduce the arithmetic precision to increase the number of operations per unit of time. This way, the number of layers, operations and memory are reduced what indirectly allows to deploy those models into a wide range of devices with limited computational resources. In this sense, libraries such as TensorFlow Lite [30] or TensorRT [31] play an important role optimizing models for deploying on CPU and GPU, respectively. In our case, we will employ TensorRT to optimize models to be mapped on GPU and DLA. Also, some experiments are performed on CPU using TensorFlow Lite.

2.3. Deployment libraries

Nowadays, there exist many libraries for efficient deployment of deep learning models. For NVIDIA devices, NVIDIA TensorRT [31] is the library released for optimizing deep learning models. It incorporates a deep learning inference optimizer and a runtime that allows obtaining inference models with low latency and high throughput using techniques like quantization, layer fusion or memory bandwidth optimizations. These optimized models can be deployed in hyperscale data centers, embedded devices or automotive platforms.

For ARM CPUs like the ones include in Jetson devices, TensorFlow Lite (or TFLite) [30] is a *de facto standard*. It is an open-source and cross-platform deep learning framework that converts a pre-trained model in TensorFlow to a format optimized for maximizing inference speed and minimizing model storage using techniques such as quantization or layer fusion. This converted model can be deployed on any device with an ARM CPU.

3. Related Work

3.1. Gait recognition

Gait can be considered as a biometric feature for human identification at a distance without requiring the collaboration of the subject to be identified. Gait recognition has been an active research topic for the past years mainly due to its surveillance applications such as crime prevention or forensic identification. Typically, gait recognition approaches rely on appearance and period-based representations such as silhouette-based methods (Gait Energy Image [7, 8]) or extensions such as frequency (Frequency-Domain Feature [32]), entropy (Gait Entropy Image [33]) or even optical flow (Gait Flow Image [34]). However, silhouette data is not invariant to changes in the body shape or clothing of the subjects. Moreover, it is affected by dynamic backgrounds, illumination changes and other factors that penalize people segmentation, producing low-quality silhouettes that result in poor recognition results. Other works rely on different sensors [35], such as floor-sensors [36], accelerometers [37] or wave-sensors [38]. However, these kinds of sensor-based systems are vulnerable to adversarial perturbations [39, 40]. An alternative to silhouette and sensor-based systems are skeleton-based [41, 42] or model-based [43, 44, 45] approaches. However, these kinds of data are very challenging to acquire especially for low image resolution [44]. [13] uses optical flow obtained from raw data frames to build a deep learning model. An in-depth evaluation of different CNN architectures based on optical flow maps is presented in [46]. Finally, [47] presents a multitask CNN with a combined loss function with multiple kinds of output labels.

Another line of research focuses on different aspects of gait recognition, such as cross-view [48, 49]. [50] proposes a gait-related loss function on a simplified spatial transformer network [51] to learn discriminative gait features, [52] proposes a view-resistant approach using optical flow, and [53] propose a method operating on multiple scales. [54] propose a view-invariant gait representation framework for cross-view gait recognition using the spatio-temporal motion characteristics. Different from previous approaches that use an image-like gait template or a sequence of images [55, 45], [11, 56, 57, 58, 59] treat gait as a set of independent frames obtaining a method that is immune to frame permutation and can integrate frames from different videos. Finally, some works use gait to extract information, e.g. age [47, 60], or to produce gait patterns, e.g. part movements [61].

Recently, several works start to rely on multiple input modalities [62, 63, 64, 65, 66]. Besides optical flow [67, 52], the most commonly-used modalities are infrared [68], pose [10] and depth [69].

3.2. Model optimizations and deployments

In recent years, the deployment of deep learning applications on embedded platforms has become increasingly important. Typically, these platforms are employed just for inference due to their limited

computational capabilities [70, 71, 72] although some authors have proposed solutions where several of them are used for collaborative training [73, 74, 75].

Many contributions have been made to increase inference throughput on embedded platforms applying techniques such as pruning, quantization and matrix factorization, among others. Extensive surveys of the state-of-the-art in model compression techniques can be found in [76, 22]. Next, we review some of the most relevant co ntributions in this field.

Pruning removes redundant parameters or neurons that do not contribute notably to the accuracy of results. There are multiple pruning options such as weight-by-weight, filter-by-filter or layer-by-layer. Thus, [77] proposes progressive channel pruning to accelerate CNNs. Similarly, [78] presents a network pruning approach that identifies structural redundancy in a CNN using a graph establishment per layer and two graph-related quantities find out and prune filters of redundant layers. AutoPruner [79] is a channel selection layer that can be attached to any convolution layer to find out less important filters during training and prune them automatically. Play and Prune [80] is a framework that jointly prunes and fine-tunes CNN model parameters keeping original accuracy.

Quantization reduces the number of computations using simpler data representation. [81] presents two methods to optimize the training process using a quantization procedure. The first one is the introduction of trainable scale thresholds per filter and, the second one is based on mutual re-scaling of consecutive depthwise separable convolutions and normal convolutions. [82] proposes a quantization-friendly separable convolution architecture. [72] applies post-training quantization. [83] uses 16-bit quantization along with Tucker decomposition.

Matrix factorization consists in computing a compact version of the full parameter matrix that enables to face the overparametrization problem and to remove redundancy while keeping the key knowledge latent in the model. For example, [84] proposes a sparse matrix factorization method that replaces overparameterized representations of models by multiplications of sparse matrices. [85] employs Singular Value Decomposition (SVD), a particular model of matrix factorization, to approximate the associative memory of Hopfield neural network. [86] presents a method based on Tucker-2 decomposition to compress convolutional weights to deploy compressed models in mobile devices.

Some papers include more than one optimization technique. Thus [87] uses quantization and pruning techniques separately while [88] jointly implements sparsity training, channel and layer pruning, and quantization on YOLOv4.

Furthermore, some authors proposed the concurrent use of the available processing elements to reduce the inference time of deep learning models. Thus, several light-weighted frameworks that try to reduce data sharing overhead and achieve a balanced partition of model computation between processors have been proposed [89],[90]. In these works, authors employ heterogeneous architectures, as Snapdragon 855 [91] and Kirin [92], where the computational power of GPU and CPU is similar.

This way, the partitioning of the model between GPU and CPU can be beneficial despite overheads incurred by input data remapping and the required synchronization for data coherence. However, as it is shown in Section 5.4.3, the computational power of CPU in NVIDIA Jetson and Xavier platforms is much lower than on GPU, discouraging the inference co-execution of the gait models on these processing units.

Finally, there are multiple software development kits used for optimizing deep learning models such as TensorRT or TensorFlow Lite for GPU or CPU, respectively. On Jetson AGX Xavier, [93] proposes an RGB-D segmentation approach that is optimized using TensorRT. Similarly, [71] presents a parallelization methodology in TensorRT to maximize the performance of a DL application using GPU and DLAs. They use various types of parallelism as multithreading, multistream, pipelining on the inference network and also partial network duplication. On Jetson TX2, [72] uses TensorRT with different object detection networks to optimize and post-training quantization. [94] shows comparative tests of accuracy, execution time and energy efficiency of a large range of vision kernels and neural networks on several embedded platforms using TensorRT among other tools. [95] studies the performance achieved by Jetson Nano using a collection of personal-scale sensory deep learning models.

Focusing on TensorFlow Lite, we can differentiate two targets: microcontrollers and smartphones. In the first case, as explained by the authors on [96], they were looking for a simple way to convert TensorFlow models into optimized models for a very different set of hardware, reducing as much as possible the necessary dependencies, with a high degree of optimization and easily usable and adaptable. [97] demonstrates, on the one hand, the viability of the TensorFlow Lite library to perform a typical operation, a fully-connected layer, in a time that can be assumed by a microcontroller. On the other hand, it also checks the impact of the quantization process on the execution of this operation. In smartphone implementations, [98] proposes the use of this tool as part of a pipeline based on fine-tuning, performing the recognition on the smartphone itself, which means a reduction in transfer times and an increase in the security of the system.

4. Methodology

In this section, we describe the process followed to deploy deep learning modes on different Jetson devices. Then, we describe these deep models in detail together with all possible optimizations (hardware or software) applied in this work.

Fig. 1 shows a sketch of our optimization and deployment process. We start with pre-trained deep learning models for gait recognition. Then, two optimization processes, are applied: quantization and pruning. We propose an approach where complete neurons or filters are pruned so that the number of parameters and activations are reduced. This way, the memory storage and computing

requirements of the new model are lower. We also quantize the weights of the original model, expressed in FP32 format, to two new formats, namely, FP16 and INT8. This way, both memory demand and computation time are further reduced. After these two optimization steps, we obtain an optimized model ready for its deployment using TensorRT for GPU and DLA or TensorFlow Lite (TFLite) for CPU.

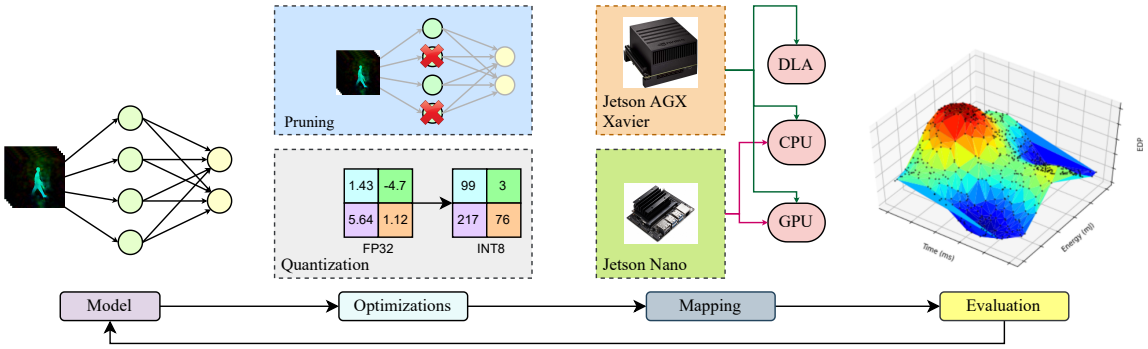

Figure 1: General process of the optimization and deployment of deep learning models on Jetson devices.

4.1. Model description

In this work, we use state-of-the-art deep learning approaches designed for two of the most widely used gait recognition datasets. Since we want to explore the optimization and deployment capabilities of different devices, we have selected three different models that use two kinds of input data.

4.1.1. Input data

Optical Flow Optical Flow (OF) [99] is defined as the motion pattern in a scene caused by the relative motion between an observer and the scene in two instants of time. OF is a motion-centric representation that obviates appearance and focuses on describing a subject by a set of local and subtly varying motions. For this reason, OF has shown excellent results in the characterization of gait [62]. Its representation is divided into two components or channels, one for the y-axis and the other for the x-axis, where most of the gait motion flow is concentrated.

Silhouettes Silhouettes are the result of segmenting a person in the foreground from the background of a frame or set of frames. The result should be one or more silhouettes of people appearing in a video represented as binary image masks. This type of descriptor, applied to gait recognition, gives information about the subject removing its appearance information, making it robust to clothing changes.

4.1.2. Gait recognition models

2D-CNN Model based on 2D convolutions is introduced in [62]. It follows the most traditional architecture for this type of problem, that is, a linear pipeline of convolutional and pooling layers. An overview of the model is shown in Fig. 2. It is composed of four Conv2D blocks, each one equipped with a Conv2D layer and a Max Pooling layer. The number of convolutional filters grows according to the depth of the layer, thus, the deeper layer, the more filters. Finally, it has three Fully-Connected layers. More information about the architecture is shown in Tab. 2. This model is composed of eleven layers with a medium number of parameters (6.83M) compared with the other two models. However, the number of activations is very limited, only 0.68M, which is the smallest one together with the total number of floating-point operations of the model, that is, 2.03 GFLOP. Thus, this model is not very demanding of computational resources.

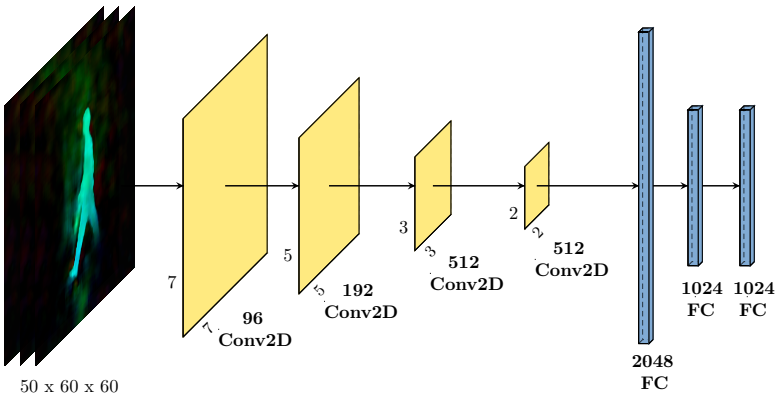

Figure 2: **2D-CNN architecture**. Each Conv2D block is composed of a Conv2D layer and Max Pooling layer. The amount of filters and kernel shape are indicated for each layer. Blue boxes represent fully-connected (FC) layers with the number of dense units.

3D-CNN Model based on 3D convolutions developed in [62], which is designed to capture temporal information from videos. The model, shown in Fig. 3, consists of seven Conv3D layers, whose number of filters grows according to the depth of the layer in the architecture. Finally, a Fully-Connected layer is appended to the end of the model. This model allows assessing a deployment that employs 3D convolutions which are far less optimized than 2D convolutions in all frameworks and hardware devices. As shown in Tab. 2, it is composed of a high number of parameters that produce a reduced number of activations. However, the number of GFLOP is small compared to the number of parameters, probably due to the cuDNN implementation of 3D convolutions.

GaitSet GaitSet [56] has been chosen because it is a well-known approach that obtains state-of-the-art on CASIA-B since this model is robust against permutations of the frames, changes in the clothing

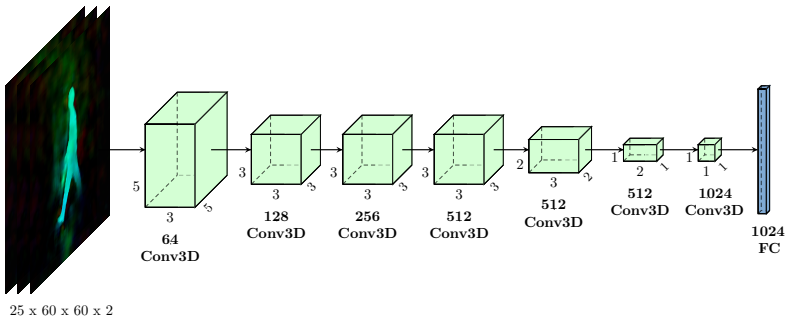

Figure 3: **3D-CNN architecture**. Each cuboid represents a Conv3D layer. The amount of filters and kernel shape are indicated in each Conv3D. The blue box represents a fully-connected (FC) layer, with the number of dense units.

and the viewpoints. GaitSet is divided into two branches, as shown in Fig. 4: the main pipeline (violet boxes) and a second branch (green boxes) called Multilayer Global Pipeline (MGP) whose objective is to take information sets at different levels of the main branch. All this information is concatenated and passed to a set of independent fully-connected layers called Horizontal Pyramid Mapping (HPM) to compensate the fact that the size of the discriminating parts varies from one image to others. From a deployment point of view, this network is interesting since, as shown in Tab. 2, it is composed of thirty layers with a small number of parameters, only 2.59M, but requires a huge amount of memory for the activations (17.69M). Focusing on the number of operations, this model has the highest GFLOP (6.53) compared with the two others, requiring more hardware resources to be executed in real-time.

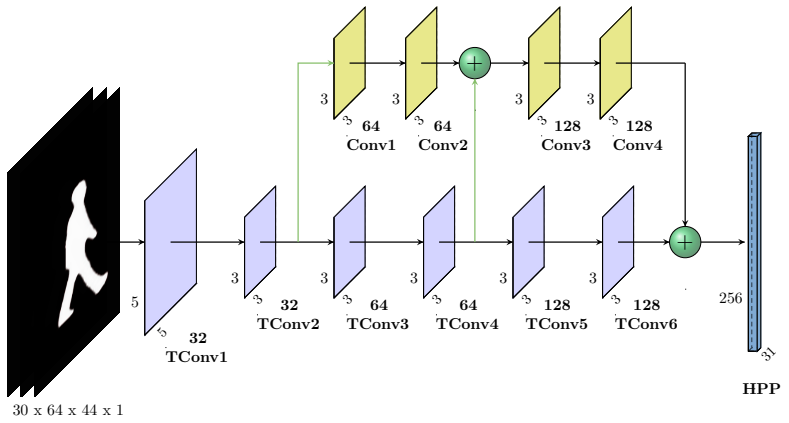

Figure 4: **GaitSet architecture**. Each violet block represents a Temporal Conv block, composed of a Con2D layer, a Max Pooling layer and a leaky ReLU activation. Each green block is a Conv block with a Conv2D layer and leaky ReLU activation, and the blue block is a horizontal pyramid pooling (HPP). Vertical green arrows represent a set pooling operation (SP) applied to feature maps coming from different samples. The amount of filters and kernel shape are indicated for every block.

Model	Dataset (Input)	N. Layers	N. Parameters	N. Activations	GFLOP
2D-CNN	TUM-GAID (OF)	11	6.83M	0.68M	2.03
3D-CNN	TUM-GAID (OF)	8	9.90M	1.86M	2.69
GaitSet	CASIA-B (Silhouettes)	30	2.59M	17.69M	6.53

Table 2: **Models comparison**. Each row represents a different CNN model. The dataset (and input mode) used, the number of layers, the number of parameters and activations (in millions) and the number of floating-point operations (in GFLOP) are reported in each column.

4.2. Hardware optimizations

Hardware optimizations depend on the target devices where the model is going to be deployed. This way, each device can include different processing units (e.g. CPU, GPU, DLA) with unlike computational capabilities to deal with different data precisions (e.g. FP32, FP16, INT8). Thus, our baseline experiments study the inference of the different model on the three types of processing elements using FP32 data representation.

Then, two different hardware optimizations are applied: quantization and concurrent inference. Regarding the first issue, we use more compact numerical representations, that is FP16 and INT8, in order to reduce the memory consumption and accelerate the execution. However, this data conversion can harm model accuracy since the numerical precision of the weights and activations is lower. This way, an additional optimization of the resulting model is required to improve final model accuracy. These optimizations are performed with TensorRT framework. Note that, Jetson Nano does not include DLAs and INT8 data format is not supported, thus, those hardware optimizations cannot be explored in this device. Finally, in order to take advantage of the CPU included in both Jetson devices, TensorFlow Lite is also used to convert and optimize the complete deep models for CPU execution. In this case, only FP32 and INT8 precision are supported by the CPU.

Regarding concurrent model deployment, some experiments (see Section 5.4.3) have shown that gait model inference is much slower on CPU than on GPU, discouraging the concurrent use of CPU to deploy our models. Thus, we just use specific deep learning accelerators (DLA and GPU) included in Jetson Xavier to perform experiments where model inference is concurrently executed on them.

4.3. Software optimizations

We use filter-wise pruning [22, 78, 79] to reduce inference costs of memory, time and energy. This way, models can be run on very cheap devices with low memory capacities. However, removing sparse weights typically does not improve the execution time and energy consumption. Nevertheless, removing complete filters saves both, memory and execution time. Our filter-wise pruning uses L_pnorm to remove filters that less affect the accuracy of the classification. L_pnorm is a typical metric to determine which filters can be eliminated [22]. In our case L_pnorm is calculated as $L_pnorm = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$,

with $p = 1$, where x is a vector, n is the number of elements of a filter and p is a real number. This way, we calculate the $L_1 norm$ for each filter of each convolutional layer and then, filters whose L1 value is lower than a predefined percentile threshold are removed. We ensure that every layer has at least one filter with the objective of not removing complete layers and maintaining the architecture of the original models. Finally, once those less important filters are removed, we perform a fine-tuning step to stabilize the model and improve its behaviour.

Note that magnitude-based weight pruning techniques [22, 100, 101, 102], which focus on removing sparse weights to save memory, has been discarded of our analysis because their impact in the reduction of both execution time and energy consumption is negligible.

5. Experimental results

In this section, we first present the optimization process and the gait recognition datasets that we use in our experiments: TUM-GAID and CASIA-B. Then, some implementation details are shown, and, finally, we perform a wide range of experiments using the models and optimizations proposed in Sec. 4 to analyze their impact on energy and time consumption. In all experiments, we report the energy consumption and execution time together with the accuracy of the model. Energy consumption is provided by internal sensors incorporated on both Jetson boards. We use a sampling frequency of 500Hz which is the maximum value supported by the sensors. For all experiments, we provide the average values measured for 20 forward steps after 5 warm-up predictions. Note that we only measure the real computations, the initialization of variables and frameworks is ignored. To measure the hardware performance we use the Energy Delay Product (EDP) metric, which is the product of energy and execution time. This way, EDP takes into account not only the energy consumption value but also the computation time. Thus, the lower EDP value, the better.

Experiments are divided into three sections: baseline experiments, hardware optimizations and software optimizations. On software optimizations, in addition to pruning, we also show a way to adapt models based on 3D convolutions to use only 2D convolutions to save energy and inference time. Moreover, we have been carried out a study to observe how the batch size affects the energy and time consumption during the inference process.

5.1. Datasets

5.1.1. TUM-GAID

For experiments with 2D-CNN and 3D-CNN models, we use TUM Gait from Audio, Image and Depth (TUM-GAID) dataset [103]. TUM-GAID contains simultaneous RGB, depth and audio videos. It includes 305 subjects performing two walking trajectories (left to right and vice versa) in an indoor environment. Two recording sessions were performed, one in January with winter clothing and the

second in April with different clothes. Three walking conditions are considered: normal walk, carrying a backpack of approximately 5 Kg and wearing coating shoes, as the ones used in cleanrooms for hygiene conditions. So that, for each subject there are six sequences of normal walking, two sequences carrying a bag and two sequences wearing coating shoes. In addition, to investigate the effect of time variation, 32 subjects were recorded a second time, so they have 10 additional sequences. Therefore, TUM-GAID dataset contains 3400 videos. A Microsoft Kinect sensor is used which provides a video stream with a resolution of 640×480 pixels with a frame rate of approximately 30 FPS. To standardize the experiments, the authors propose to divide the dataset: 100 subjects are used for training, 50 subjects are used for validation and finally, 155 subjects for testing.

5.1.2. CASIA-B

For GaitSet experiments, we use CASIA Gait Dataset [104], part B (CASIA-B). It contains 124 subjects performing walking trajectories in an indoor environment from 11 viewpoints, every one is separated by 18 degrees with respect to the previous one. Three situations are considered: normal walk, wearing a coat and carrying a bag. For each subject, there are six sequences of normal walking, two sequences carrying a bag and two sequences carrying a coat. Therefore, TUM-GAID dataset contains 13640 videos with a resolution of 320×240 pixels. Following the authors indications, 74 subjects are used for training and the rest of 50 subjects are used for testing.

5.2. Implementation Details

For 2D-CNN and 3D-CNN models, during training, the weights are learnt using SGD (Stochastic Gradient Descent) with momentum equals 0.9. Weight decay is set to $5 \cdot 10^{-6}$. The learning rate is initially set to 10^{-3} and divided by 0.2 when the validation error gets stuck. At each epoch, a mini-batch of 150 samples is constructed. Models are trained during 100 epochs using Crossentropy loss. GaitSet, is trained with Adam using a constant learning rate of 0.0001 for 80000 iterations. We use Triplet loss with a margin of 0.2 and a batch size of 128 samples and 30 frames. During fine-tuning, models are trained during 30 epochs using the same hyperparameters. We implement our models using the Keras version of TensorFlow 2.4 [105], TensorRT 7.1.3 and cuDNN 8.0.

5.3. Baseline experiments

In this section, we present baseline models, without any optimization, that are deployed on the GPU of both Jetson devices. As an inference requires a first stage, called feature extractor (FE), to calculate the feature descriptor (the output of the model) and, then, a second stage to apply a classifier that proposes the identity, we take performance values for both stages. In order to obtain the feature vector, we deploy both the original TensorFlow models and their converted versions to TensorRT using FP32 representation without pruning. Regarding the classification stage, we study

the impact of the two most common classifiers used in inference, i.e. K-Nearest Neighbour (KNN) and Softmax (SM).

5.3.1. Time and Energy study

The obtained results can be observed in Tab. 3. The better values achieved by TensorRT models for each platform and dataset are shown in bold. Note that the cells containing 'N/A' (not applicable) mean that energy cannot be measured due to limitations in the sampling rate of the energy sensor. KNN and SM suffixes refer to the classifier used. TRT and TF refer to the framework used during inference, that is, TensorRT converted models or original TensorFlow models respectively.

Regarding classifiers it can be observed that models using KNN spend more energy and take longer execution time than the corresponding model using SM because KNN computes the distance between each gallery and probe feature vector. On the other hand, Softmax is implemented with just an additional layer at the end of the model. These differences are more pronounced for GaitSet model as this model has longer features vectors. For example in Xavier execution time using Softmax is 1.5× faster than using KNN and consumes 1.08× less energy. Thus, only models using SM classifier will be further discussed.

In Xavier, for TUM-GAID dataset, we can conclude 2D-CNN-SM-TRT models are faster (2.61 ms) than 3D-CNN-SM-TRT (5.94 ms). However, the 3D-CNN model achieves better accuracy. We also see TRT is able to reduce the inference time of 2D-CNN model from 153.91 ms (TensorFlow model) to 2.61 ms (TRT model). Very important time reduction also happens for 3D-CNN models. Regarding CASIA-B datatset, GaitSet-SM-TRT is the slowest one, taking 7.18 ms to perform one inference. Inference energy consumption is also low, taking values of 1.68, 3.43 and 4.90 millijoules for 2D-CNN, 3D-CNN and Gaitset models respectively.

Analyzing results in Nano, it can be observed that execution times of models are longer than the corresponding one in Xavier. Thus 2D-CNN, 3D-CNN and GaitSet models are 5.2×, 9.6× and 7.4× slower than corresponding ones in Xavier platform. Regarding energy consumption, values for 3D-CNN and GaitSet models are close to each other and they are around 4× higher than those obtained by 2D-CNN. In addition, inference on Nano requires more energy than in Xavier since the latter one has a more recent GPU that spend fewer watts per FLOP.

Note that, 2D-CNN TensorFlow model is not included in the table since it cannot be executed on Jetson Nano due to memory limitations.

Focusing on accuracy, KNN and Softmax achieve similar results but the latter provides shorter execution times consuming less energy. Comparing the two available models for TUM-GAID, 3D-CNN model obtains the best results. Obviously, accuracy values for similar models in different platforms are equal.

Device	Dataset	Model	Energy (mJ)			Time (ms)			EDP	Acc
			FE	Classifier	Total	FE	Classifier	Total		
Xavier	TUM-GAID	2D-CNN-KNN-TRT	1.67	0.02	1.69	2.59	0.98	3.57	6.02	92.03
		3D-CNN-KNN-TRT	3.45	0.02	3.47	5.92	0.98	6.90	23.92	**95.80**
		2D-CNN-SM-TRT	1.68	N/A	**1.68**	2.59	0.02	**2.61**	**4.38**	92.90
		3D-CNN-SM-TRT	3.43	N/A	3.43	5.92	0.02	5.94	20.37	**95.80**
		2D-CNN-SM-TF	15.18	N/A	15.18	153.35	0.56	153.91	2328.40	92.90
		3D-CNN-SM-TF	7.41	N/A	7.41	105.35	0.37	105.72	781.06	95.80
	CASIA-B	GaitSet-KNN-TRT	5.26	0.03	5.29	7.11	3.78	10.89	57.62	79.39
		GaitSet-SM-TRT	4.90	N/A	**4.90**	7.11	0.07	**7.18**	**35.19**	**79.65**
		GaitSet-SM-TF	34.89	N/A	34.89	266.52	2.69	269.21	9298.80	79.65
NANO 4GB	TUM-GAID	2D-CNN-KNN-TRT	8.76	0.75	9.51	13.54	12.86	26.40	251.15	92.03
		3D-CNN-KNN-TRT	31.44	0.75	32.19	57.03	12.86	69.89	2250.00	**95.80**
		2D-CNN-SM-TRT	8.86	N/A	**8.86**	13.54	0.08	**13.62**	**120.67**	92.90
		3D-CNN-SM-TRT	31.37	N/A	31.37	57.03	0.08	57.11	1791.54	**95.80**
		3D-CNN-SM-TF	77.99	N/A	77.99	621.71	0.89	622.60	48486.84	95.80
	CASIA-B	GaitSet-KNN-TRT	28.71	7.25	35.96	52.39	51.04	103.43	3719.33	79.39
		GaitSet-SM-TRT	29.10	N/A	**29.10**	52.39	0.54	**52.93**	**1540.12**	**79.65**
		GaitSet-SM-TF	90.65	N/A	90.65	517.48	5.25	522.73	46910.52	79.65

Table 3: **Baseline study: time and energy**. Each row represents a different model on Xavier or Nano with CASIA-B or TUM-GAID dataset using TensorRT (TRT) or TensorFlow (TF) and two classifiers, K-Nearest Neighbour (KNN) and Softmax (SM). Columns represent energy and time consumption for GPU (divided into Features Extraction (FE) and classifier) together with EDP value (Energy Delay Product). Finally accuracy is included in the last column. Note: N/A values cannot be measured due to the limitations of the sensor precision. Best results using TensorRT are marked in bold.

5.3.2. Peak Energy Consumption

In this section, we compare peak energy consumption in the following models: GaitSet-SM-TRT 2D-CNN-SM-TRT and 3D-CNN-SM-TRT using Xavier and Nano GPU. For simplicity, these models will be denominated as GaitSet, 2D-CNN, and 3D-CNN from now on.

As we can see in the rightmost part of Fig. 5 and Fig. 6, Xavier has between 5× and 9× lower peak consumption than Nano, depending on the model. In our opinion, this behaviour is due to Xavier is equipped with a Volta GPU, which is a newer generation than the Maxwell GPU included on Jetson Nano. Focusing on the models, 2D-CNN has a lower maximum energy consumption than 3D-CNN and GaitSet. Both 3D-CNN and GaitSet have a similar maximum energy consumption on Xavier and Nano.

5.4. Hardware optimizations

5.4.1. Data precision on GPU

In Tab. 4 we observe energy and time consumption during feature extraction using the different data precisions managed by Xavier and Nano GPUs. EDP value and accuracy are also included. The

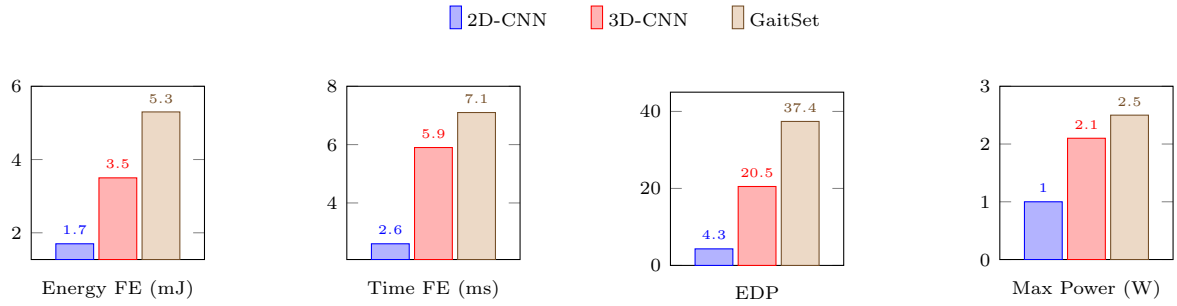

Figure 5: **Xavier baseline study: peak energy consumption.** Each colour represents a different model on Xavier. From left to right: energy and time consumption, EDP value (Energy Delay Product) for GPU and finally, maximum power consumption.

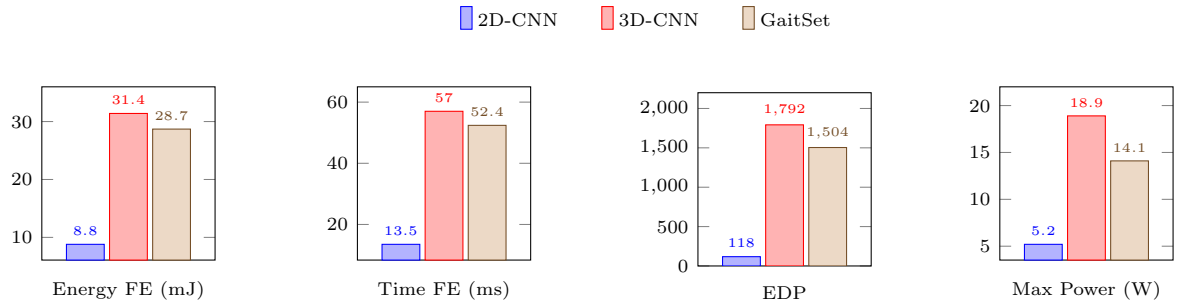

Figure 6: **Nano baseline study: peak energy consumption.** Each colour represents a different model on Nano. From left to right: energy and time consumption, EDP value (Energy Delay Product) for GPU and finally, maximum power consumption.

last two columns show the EDP reduction ratio (EDP_{rr}) and the increase or decrease in the accuracy (Diff. Acc.) for FP16 or INT8 with respect to FP32 precision. Note that Jetson Nano only supports FP16 and FP32.

Focusing on the execution time of the feature extractor, the behaviour is the expected one since the lower data precision the faster execution times. Only 3D-CNN model obtains non-expected execution times for INT8 on Xavier and FP16 on Jetson Nano. In both cases, the execution times are similar to the ones obtained with FP32. To clarify this behaviour, we analyzed called kernels with NVIDIA Profiler and we discovered that, for those cases, FP32 kernels are launched due to TensorRT internal policies. Therefore, execution times are the same as the ones obtained with FP32. Comparing the execution times for a given model, 2D-CNN achieves an unexpected improvement from FP32 to FP16, which is bigger than the 2× expected reduction. This is due to the improved version of convolutional kernels used by cuDNN which take advantage of tensor cores. For the other models, this improvement in execution time is less remarkable and within expected values. For INT8, executions times are, in general, worse than the anticipated value since, analyzing launched functions with NVIDIA Profiler,

we observed that not all kernels of the model are executed with INT8 precision. Moving to the energy consumption column, it has a behaviour similar to that of execution time since it is highly correlated with inference latency.

Comparing EDP reduction ratios (EDP_{rr}) among different data precision, we can observe those reduction ratios for GaitSet and 2D-CNN on Xavier are higher, achieving a value of 22× in 2D-CNN for INT8 with respect to FP32. For Jetson Nano, it can be observed there is a clear improvement in EDP when FP16 is employed. However, similarly to Xavier platform, no improvement is obtained for 3D-CNN model.

Focusing on accuracy values, different data representations obtain very similar scores, independently of the employed precision model, remaining even identical in some cases for FP32 and FP16. From this, we can conclude that TensorRT optimizations allow us to reduce inference latency, especially for 2D-CNN and GaitSet model, with FP16 or INT8 while preserving accuracy. However, the achieved improvement for the 3D-CNN model is low.

Device	Model	Precision	Energy FE (mJ)	Time FE (ms)	EDP	Acc	EDP_{rr}	Diff. Acc
Xavier	2D-CNN	FP32	1.668	2.593	4.324	92.03	-	-
		FP16	0.511	0.783	0.400	92.79	10.814x	+0.75
		INT8	0.308	0.619	0.190	92.35	22.714x	+0.32
	3D-CNN	FP32	3.447	5.941	20.477	95.80	-	-
		FP16	2.423	4.159	10.079	95.80	2.032x	0
		INT8	3.438	5.903	20.297	95.69	1.009x	-0.11
	GaitSet	FP32	5.259	7.112	37.399	79.37	-	-
		FP16	2.587	4.193	10.849	79.37	3.447x	0
		INT8	1.894	3.495	6.621	79.24	5.649x	-0.13
NANO 4GB	2D-CNN	FP32	8.763	13.538	118.641	92.03	-	-
		FP16	4.742	7.540	35.751	92.79	3.319x	+0.75
	3D-CNN	FP32	31.438	57.025	1792.739	95.80	-	-
		FP16	29.795	56.736	1690.412	95.80	1.061x	0
	GaitSet	FP32	28.713	52.387	1504.179	79.37	-	-
		FP16	18.992	37.905	719.897	79.37	2.089x	0

Table 4: **Hardware optimizations: data precision study**. Each row represents a different model for Xavier or Nano with different data precision. Columns represent energy and time consumption together with EDP value (Energy Delay Product) for GPU and accuracy. Finally, EDP_{rr} (Energy Delay Product reduction ratio) and difference of accuracy with respect to the corresponding FP32 model are included in the last two columns.

5.4.2. Deep Learning Accelerator

An additional hardware optimization available on Xavier can use both DLA and GPU to perform model inference, but DLA use is still limited. Thus, none of our models can be completely mapped on the DLA due to layer incompatibilities. For example, we have observed that dense layers with

more than 1024 filters are not supported by the DLA and are mapped in the GPU. For GaitSet, these limitations are so strict that any layer can be mapped on DLA. In our opinion, the reason is that model is composed of two different branches with mathematical reduction operations and reshaping of the activations. However, apart from layer incompatibilities stated by NVIDIA, the rest of mapping incompatibilities are not known beforehand since the layer scheduler of TensorRT is not public. Therefore, we must follow a 'try and error' process to find out which parts of the models can be mapped on the DLA. Finally, 3D convolutions cannot be mapped either, which makes impossible the deployment of the 3D-CNN model on this accelerator. This way, only 2D-CNN model is deployed simultaneously in both accelerators.

The results using DLA&GPU can be observed in Tab. 5. Note that, values shown in parentheses in the table indicate the reduction ratio for total energy, time and EDP with respect to GPU models (top three rows in Tab. 4). We observe that for FP16 and INT8 precisions the use of DLA makes the inference process slower but the total energy decreases. These results are expected since, as commented in Sec. 2.1, the DLA is slower than the GPU, however, it allows freeing GPU resources which can be employed for other computations. In addition, for FP32, the execution time is lower than using only GPU, indicating that DLA is very efficient computing kernels with FP32 data format.

Model	Precision	Enery FE (mJ)			Time FE (ms)	EDP	EDP_{rr}
		GPU	DLA	Total			
2D-CNN (DLA&GPU)	FP32	0.180	0.201	0.381 (x4.378)	1.461 (x1.774)	0.557 (x7.768)	-
	FP16	0.184	0.202	0.386 (x1.323)	1.525 (x0.513)	0.589 (x0.679)	x0.946
	INT8	0.193	0.087	0.280 (x.1.100)	1.071 (x0.578)	0.300 (x0.635)	x1.857

Table 5: **Hardware optimizations: DLA study**. Each row represents a different data precision. Columns represent energy and time consumption for DLA and GPU together with EDP value (Energy Delay Product). Finally EDP_{rr} (Energy Delay Product reduction ratio) with respect to FP32 data precision is included in the last column. Reduction ratio with respect to GPU models (top three rows in Tab. 4) is shown in parentheses for total energy, time and EDP.

5.4.3. Deployment on CPU

In this experiment, we are going to compare CPU and GPU implementations of the proposed models using TFLite library to perform the inference on CPU. This comparison will be made excluding the 3D-CNN network since 3D convolutions are not supported by TFLite. For brevity, these CPU experiments have been performed with a batch size of 1 in Jetson Nano. Results are summarized in Fig. 7. If we compare the inference times with its GPU analogues (Tab. 4) in FP32, 2D-CNN CPU is 20.5× slower and Gaitset is 18.7× slower. FP16 has not been included in the table because the CPU does not have specific hardware for it, so it reuses FP32 hardware, obtaining the same time and energy consumption. As an advantage over the GPU, this CPU does support INT8 type, although it is still slower than FP32 on GPU (9.5× in 2D-CNN and 14.6× in GaitSet). We see here

that no implementation on CPU can deal with input samples coming from video streaming at 25 fps, as inference takes more than 40ms. Regarding energy consumption, if we calculate the energy consumption per millisecond, we observe that Nano GPU, in FP32, consumes 2.6× less for 2D-CNN and 2.15× less for GaitSet.

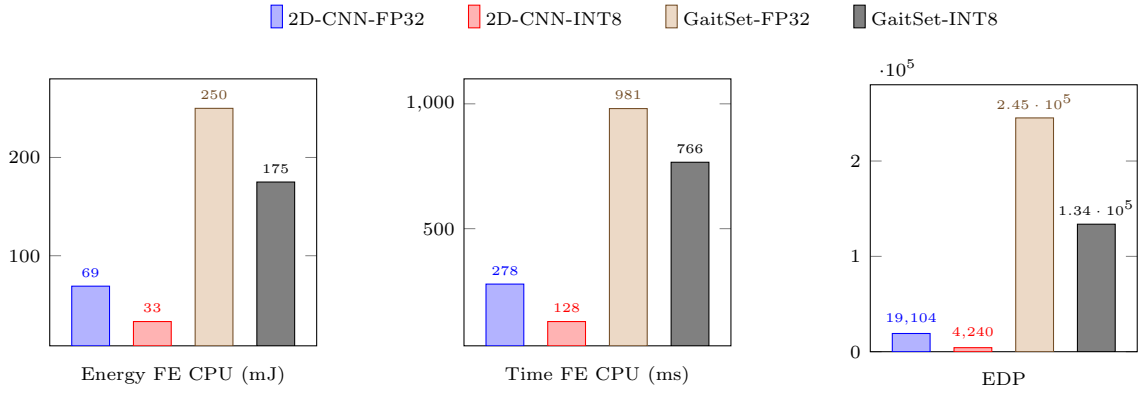

Figure 7: **Hardware optimizations: CPU study in Nano.** Each colour represents a different model or the same model with different data precision. X-axis represents energy and time consumption for CPU and EDP value (Energy Delay Product).

5.5. Software optimizations

5.5.1. Filter pruning

We evaluate five pruning ratios for each model using FP32 data precision deployed with TensorRT. Tab. 6 and Tab. 7 summarize the results of this experiment for Xavier and Nano, respectively. In this case, we report energy, time consumption, EDP and accuracy values. The last two columns show EDP reduction ratio and the decrease or increase of accuracy with respect to non-pruned models. The amount of pruned filters in convolutional layers is represented as a percentile over the total number of filters. Note that, for GaitSet, we cannot achieve the real percentile of pruned filters since some layers cannot be pruned because they are followed by mathematical operations which depend on the position of the filters. For these cases, we include the real percentile while the theoretical one is also indicated in parentheses.

Focusing on Xavier results (Tab. 6) the inference time is reduced as we remove filters but usually with a small decrease in the accuracy of the model. However, in some cases, the accuracy increases with higher pruning values because pruned networks have the possibility of escaping from local minima and slightly improving the accuracy.

Moving to Nano results shown in Tab. 7, the behaviour of all models is similar to Xavier and, the more pruned filters, the less execution time and the less energy consumption.

Note that, although we present the results of our pruning strategy, we have evaluated magnitude-based weight pruning techniques [22, 100, 101, 102]. Results show that inference time and energy consumption do not change although sparsity is increased due to zero weights are also computed. Note that for brevity, these results are not included.

Model	Percentile	Energy FE (mJ)	Time FE (ms)	EDP	Acc	EDP_{rr}	Diff. Acc
2D-CNN	0	1.668	2.593	4.324	92.03	-	-
	10	1.589	2.469	3.922	93.32	1.103x	+1.29
	20	1.536	2.388	3.667	92.25	1.179x	+0.22
	30	1.488	2.278	3.390	93.22	**1.276x**	**+1.19**
	40	1.560	2.216	3.457	92.68	1.251x	+0.64
	50	1.464	2.004	2,934	64.88	1.474x	-27.15
3D-CNN	0	3.447	5.941	20.477	95.80	-	-
	10	2.973	4.980	14.806	95.90	1.383x	+0.10
	20	2.924	4.929	14.411	95.69	1.421x	-0.11
	30	2.862	4.794	13.722	95.26	**1.492x**	**-0.54**
	40	1.789	3.382	6.050	1.51	3.385x	-94.29
GaitSet	0	5.259	7.112	37.399	79.39	-	-
	10	3.586	5.676	20.354	76.43	1.837x	-2.96
	19 (20)	2.809	4.778	13.423	75.64	2.786x	-3.75
	26 (30)	2.768	4.707	13.028	76.08	2.871x	-3.31
	36 (40)	2.584	4.566	11.798	76.63	3.170x	-2.76
	45 (50)	1.966	3.898	7.662	75.44	**4.881x**	**-3.95**

Table 6: **Software optimizations: filter pruning study on Xavier**. Each row represents a different model with its corresponding pruning percentile. Note that for GaitSet the theoretical percentile of pruned filters is shown in parentheses. Columns represent energy and time consumption together with EDP value (Energy Delay Product) for GPU and accuracy. Finally, EDP_{rr} (Energy Delay Product reduction ratio) and difference of accuracy with respect to the corresponding baseline model (percentile 0) are included in the last two columns. Best results are marked in bold.

5.5.2. Adapting 3D convolutions to 2D convolutions

In this experiment, we propose a variation of the 3D-CNN model since, as shown in previous experiments, 3D convolutions are slower and less optimized in the frameworks than 2D convolutions. This way, each 3D convolution has been transformed into a block of two 2D convolutions and a reshape layer in order to reproduce a similar behaviour. The main goal of this variation is to obtain a new model, called 3Dto2D-CNN, which is more optimized in terms of energy and time consumption than 3D-CNN but with better accuracy than 2D-CNN. Moreover, this model could be deployed on DLAs since it does not include 3D convolutions.

A comparison of the accuracy, inference time and energy consumption for this model and the original 3D-CNN and 2D-CNN models is shown in Tab. 8. On the one hand, using GPU in both Xavier and Nano, time and energy consumption and, consequently, EDP are lower using 3Dto2D-

Model	Percentile	Energy FE (mJ)	Time FE (ms)	EDP	Acc	EDP_{rr}	Diff. Acc
2D-CNN	0	8.763	13.538	118.641	92.03	-	-
	10	8.728	14.194	123.881	93.32	0.958x	+1.29
	20	8.292	12.624	104.685	92.25	1.133x	+0.21
	30	8.105	12.463	101.005	93.22	1.175x	+1.19
	40	7.824	11.666	91.267	92.68	**1.300x**	**+0.64**
	50	7.290	11.569	84.340	64.88	1.407x	-27.15
3D-CNN	0	31.438	57.025	1792.739	95.80	-	-
	10	27.996	50.732	1420.286	95.90	1.262x	+0.10
	20	27.191	49.705	1351.536	95.69	1.326x	-0.11
	30	27.366	48.729	1333.509	95.26	**1.344x**	**-0.54**
	40	13.295	25.068	333.284	1.51	5.379x	-94.29
GaitSet	0	28.713	52.387	1504.179	79.39	-	-
	10	21.878	41.838	915.353	76.43	1.643x	-2.96
	19 (20)	16.853	34.643	583.825	75.64	2.576x	-3.75
	26 (30)	16.598	34.436	571.572	76.08	2.632x	-3.31
	36 (40)	15.638	33.189	518.988	76.63	2.898x	-2.76
	45 (50)	12.012	27.857	334.605	75.44	**4.495x**	**-3.95**

Table 7: **Software optimizations: filter pruning study on Nano**. Each row represents a different model with its corresponding pruning percentile. Note that for GaitSet the theoretical percentile of pruned filters is shown in parentheses. Columns represent energy and time consumption together with EDP value (Energy Delay Product) for GPU and accuracy. Finally, EDP_{rr} (Energy Delay Product reduction ratio) and difference of accuracy with respect to the corresponding baseline model (percentile 0) are included in the last two columns. Best results are marked in bold.

CNN than using 3D-CNN but higher than using 2D-CNN. On DLA, the same behaviour is observed by comparing 2D-CNN and 3Dto2D-CNN. However, the accuracy achieved by 3Dto2D-CNN model is higher than that obtained by 2D-CNN in both embedded platforms. Therefore, we can conclude that 3Dto2D-CNN is a balanced solution between accuracy and inference time and energy consumption, with the addition of being able to deploy on the DLA.

5.5.3. Batch size study

A batch is a set of inputs that can be processed independently. As more data is available for computation, more efficient kernels can be launched on the GPU. This way, higher throughput in samples per second can be achieved. Energy consumption can also benefit from using batch. However, since the launching of a new batch has to wait until the previous batch finishes, the time gap between samples belonging to consecutive batches increases as the batch size grows. This could be a problem if batches contain samples coming from a video streaming since they could not be processed at the required rate.

In Fig. 8 and Fig. 9, we can observe the results of time and energy as well as EDP per sample for each model from batch 1 to batch 8 for Xavier and Nano, respectively, using TensorRT. Focusing

Device	Model	Enery FE (mJ)			Time FE (ms)	EDP	Acc
		GPU	DLA	Total			
Xavier	2D-CNN	1.67	-	1.67	2.59	4.32	92.03
	3Dto2D-CNN	2.12	-	2.12	3.28	6.97	93,54
	3D-CNN	3.45	-	3.45	5.94	20.48	95.80
	2D-CNN-DLA	0.18	0.20	0.38	1.46	0.56	92.03
	3Dto2D-CNN-DLA	1.28	0.27	1.55	5.58	8.67	93,54
NANO 4GB	2D-CNN	8.76	-	8.76	13.54	118.64	92.03
	3Dto2D-CNN	12.55	-	12.55	22.35	280.65	93,54
	3D-CNN	31.44	-	31.44	57.03	1792.74	95.80

Table 8: **Software optimizations: conversion of 3D to 2D models study**. Each row represents a different model on Xavier or Nano. Columns represent energy and time consumption for DLA and GPU together with EDP value (Energy Delay product). Finally, accuracy is included in the last column.

on Xavier EDP value per sample for 2D-CNN, it has been reduced by more than a third for the best batch size (batch 8) with respect to batch 1. For 3D-CNN, the EDP value has been reduced by more than 1.5× (batch 2) while for GaitSet the reduction is not so significant (1.18× for batch 4). Focusing on Jetson Nano, the EDP values have been reduced between 1.2× and 1.5×. In our opinion, this small improvement for Jetson Nano is due to it has lower computational capabilities, which limits the number of parallel operations that can be performed. This way, Xavier shows better improvements since it includes more powerful hardware.

From these results, we can conclude that increasing the batch size always involves some reduction in execution time and energy consumption per sample and, generally, it has more effect on Xavier than on Nano. Note that, although execution time and energy per sample decreases, the time required by the model to produce an output increases with the batch size. Thus, in Fig. 8 and Fig. 9, we can obtain the execution time of batches just by multiplying the execution time per sample by the batch size. For instance, if a batch size of six is used for GaitSet in Xavier, the batch execution time is 39.6 ms. Therefore, this implementation will be able to process samples of up to six people appearing in a video at a rate of 25 FPS (40 ms between consecutive frames). However, if more samples must be packed into a batch, this frame rate cannot be sustained.

5.6. Final guidelines

In this section, we propose some guidelines, according to the obtained experience, to efficiently deploy deep learning models on Xavier and Nano. First of all, if possible, NVIDIA TensorRT must be used instead of pure TensorFlow. Although TensorFlow gives us the chance to map a model on an embedded system, the execution time and energy consumption are higher than using TensorRT. Focusing on Xavier, it benefits from INT8 representation and pruning, which improves energy and time consumption while maintaining good accuracy. DLA is only useful if the approach requires executing

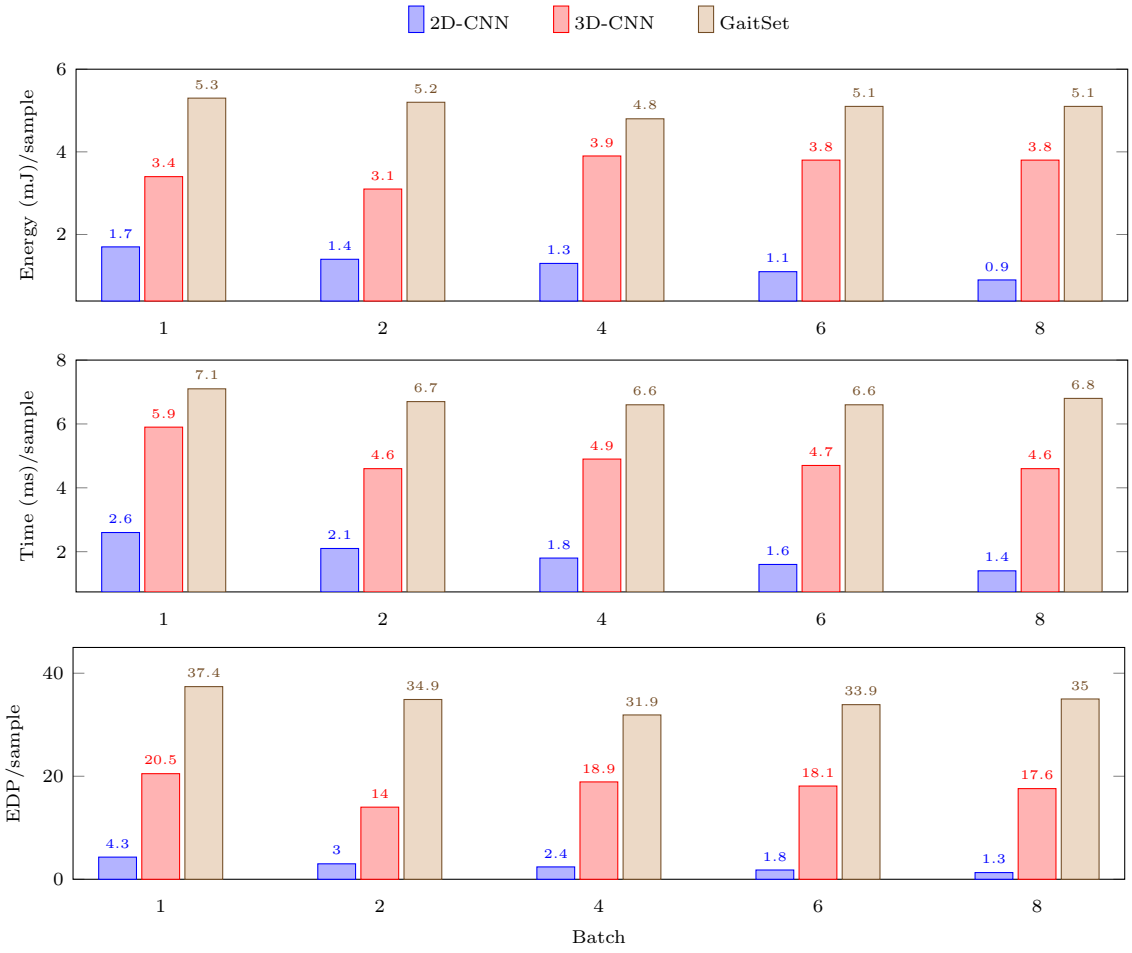

Figure 8: **Software optimizations: batch size study on Xavier.** Each colour represents a different model. X-axis represents a different batch from 1 to 8. From top to bottom, Y-axis represents energy and time consumption for GPU and EDP value (Energy Delay Product) per batch.

several models in parallel, deploying one model on each accelerator (DLA and GPU). Moving to Jetson Nano, it also benefits from pruning but, in this case, data representation must be FP16 due to hardware limitations. The use of 3D convolutions must be carefully studied in terms of accuracy and EDP, since this kind of layer is slow and resource-demanding. It could be a good idea to follow our proposal to adapt 3D convolutions to 2D convolutions to reduce energy and time consumption and take advantage of DLAs on Xavier. The use of DLA+GPU makes the inference process slower but the total energy decreases thus, on systems where real-time is easily achieved, it can be a good way to reduce energy consumption. Finally, if the latency of the application is not an issue, increasing the batch size reduces execution time and energy consumption per sample at a cost in inference latency.

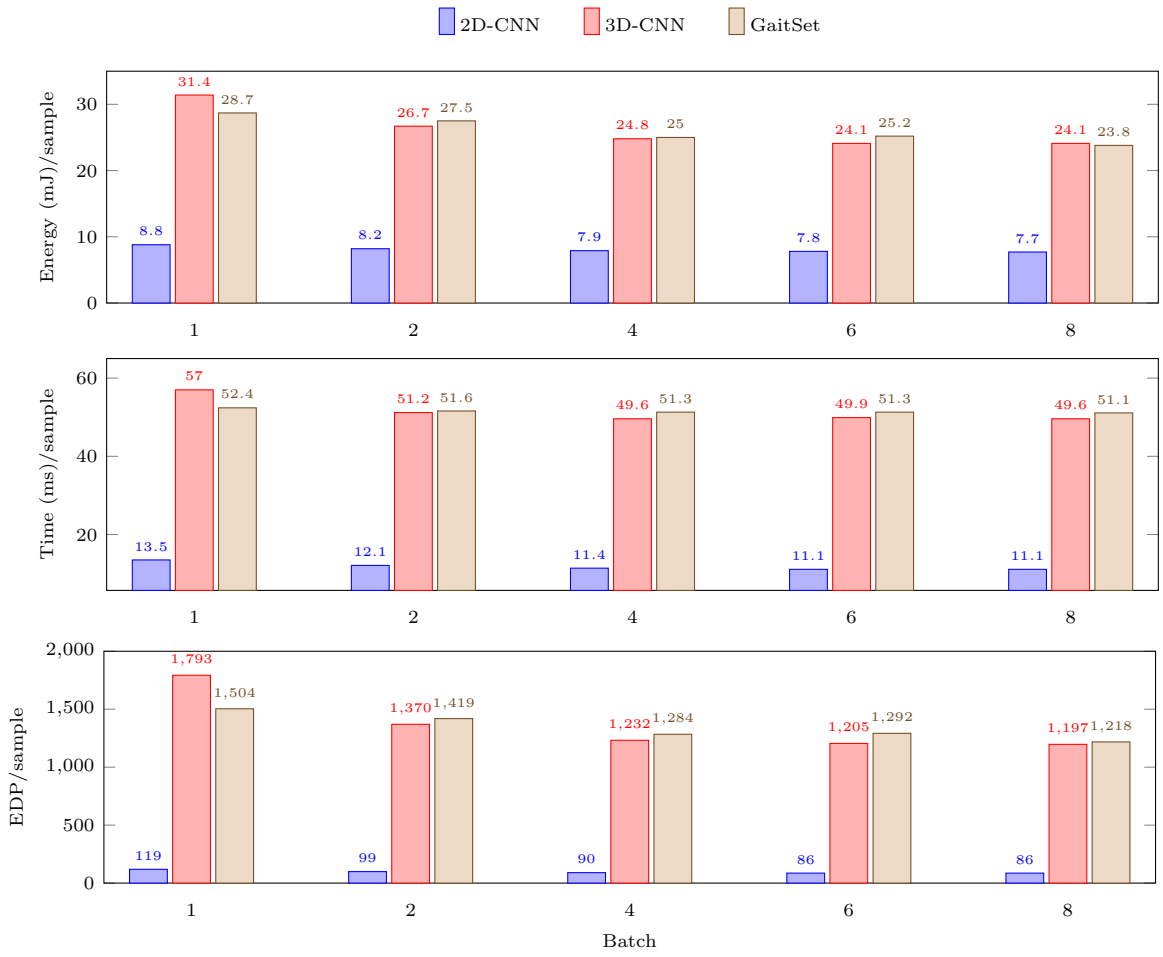

Figure 9: **Software optimizations: batch size study on Nano.** Each colour represents a different model. X-axis represents a different batch from 1 to 8. From top to bottom, Y-axis represents energy and time consumption for GPU and EDP value (Energy Delay Product) per batch.

6. Conclusions

In this paper, we have presented a time, energy, and accuracy analysis of three state-of-the-art gait recognition models running on two embedded systems with very different computational power, namely, Jetson Nano and Jetson AGX Xavier. We have selected carefully the models so that they make use of a wide range of deep learning kernels that allow us to check the impact of the employed optimizations techniques on two embedded platforms and their real capabilities. Concretely, 2D-CNN model has a standard architecture where convolutions are bidimensional. 3D-CNN model performs most calculations using 3D convolutions, which are more computational demanding. Finally, GaitSet-CNN generates a big number of activations and also computes many pooling operations. This way, in our humble opinion, the recommendations proposed in our paper (Sec. 5.6) can be applied in any kind of gait recognition-based model in order to efficiently deploy it.

As final comments, we want to remark that specific accelerators like DLA of Xavier has shown some hardware limitations as, in our case, any complete model could be deployed. Thus, models must be carefully designed and in most of the cases, they should be deployed on DLA and GPU at the same time. However, this is not a limitation since both accelerators working together can be faster than using just a single one. Software optimizations like quantization or pruning have been tested, obtaining faster and less energy demanding models which keep the accuracy high. These two techniques are beneficial and it is recommended to apply them in all models. In addition, a study of batch size has been conducted. In general, we have observed that for all models, execution time and energy consumption are better as the batch size grows.

As future work, we plan to explore further optimizations together with partial deployment of models in different computational units included in devices like Xavier.

Acknowledgments

This work has been supported by the Junta de Andalucía of Spain (P18-FR-3130 and UMA20-FEDERJA-059) and the Ministry of Education of Spain (PID2019-105396RB-I00).

References

[1] B. J. West, N. Scafetta, Nonlinear dynamical model of human gait, Physical review E 67 (5) (2003) 051917.

[2] N. Scafetta, D. Marchi, B. J. West, Understanding the complexity of human gait dynamics, Chaos: An Interdisciplinary Journal of Nonlinear Science 19 (2) (2009) 026108.

[3] G. Gupta, S. Pequito, P. Bogdan, Re-thinking eeg-based non-invasive brain interfaces: Modeling and analysis, in: 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS), IEEE, 2018, pp. 275–286.

[4] Y. Xue, S. Rodriguez, P. Bogdan, A spatio-temporal fractal model for a cps approach to brain-machine-body interfaces, in: 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2016, pp. 642–647.

[5] S. L. Di Stasi, D. Logerstedt, E. S. Gardinier, L. Snyder-Mackler, Gait patterns differ between acl-reconstructed athletes who pass return-to-sport criteria and those who fail, The American journal of sports medicine 41 (6) (2013) 1310–1318.

[6] A. Sepas-Moghaddam, A. Etemad, Deep gait recognition: A survey, IEEE Transactions on Pattern Analysis and Machine Intelligence.

[7] W. Zeng, C. Wang, F. Yang, Silhouette-based gait recognition via deterministic learning, Pattern Recognition 47 (11) (2014) 3568 – 3584.

[8] J. Han, B. Bhanu, Individual recognition using gait energy image, IEEE PAMI 28 (2) (2005) 316–322.

[9] C. Wan, L. Wang, V. V. Phoha, A survey on gait recognition, ACM Computing Surveys (CSUR) 51 (5) (2019) 89.

[10] Z. Zhang, L. Tran, X. Yin, Y. Atoum, X. Liu, J. Wan, N. Wang, Gait recognition via disentangled representation learning, in: CVPR, 2019, pp. 4710–4719.

[11] H. Chao, Y. He, J. Zhang, J. Feng, Gaitset: Regarding gait as a set for cross-view gait recognition, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019.

[12] K. Shiraga, Y. Makihara, D. Muramatsu, T. Echigo, Y. Yagi, Geinet: View-invariant gait recognition using a convolutional neural network, in: Intl. conference on biometrics (ICB), 2016, pp. 1–8.

[13] F. M. Castro, M. J. Marín-Jiménez, N. Guil, N. P. de la Blanca, Automatic learning of gait signatures for people identification, in: IWANN, Vol. 10306, 2017, pp. 257–270.

[14] R. Delgado-Escaño, F. M. Castro, J. R. Cózar, M. J. Marín-Jiménez, N. Guil, An end-to-end multi-task and fusion cnn for inertial-based gait recognition, IEEE Access 7 (2018) 1897–1908.

[15] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: CVPR, 2016, pp. 770–778.

[16] X. Zhai, A. Kolesnikov, N. Houlsby, L. Beyer, Scaling vision transformers, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 12104–12113.

[17] F. M. Castro, N. Guil, M. J. Marín-Jiménez, J. Pérez-Serrano, M. Ujaldón, Energy-based tuning of convolutional neural networks on multi-gpus, Concurrency and Computation: Practice and Experience 31 (21) (2019) e4786, e4786 cpe.4786. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4786, doi:https://doi.org/10.1002/cpe.4786.
URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4786`

[18] Z.-Q. Zhao, P. Zheng, S.-T. Xu, X. Wu, Object detection with deep learning: A review, IEEE Transactions on Neural Networks and Learning Systems 30 (11) (2019) 3212–3232. doi:10.1109/TNNLS.2018.2876865.

[19] A. Saddik, R. Latif, M. Elhoseny, A. Elouardi, Real-time evaluation of different indexes in precision agriculture using a heterogeneous embedded system, Sustain. Comput. Informatics Syst. 30 (2021) 100506.

[20] M. Qasaimeh, K. Denolf, A. Khodamoradi, M. Blott, J. Lo, L. Halder, K. Vissers, J. Zambreno, P. H. Jones, Benchmarking vision kernels and neural network inference accelerators on embedded platforms, Journal of Systems Architecture 113 (2021) 101896. doi:https://doi.org/10.1016/j.sysarc.2020.101896.
URL https://www.sciencedirect.com/science/article/pii/S1383762120301697

[21] B. Jaiswal, N. Gajjar, Deep neural network compression via knowledge distillation for embedded applications, in: 2017 Nirma University International Conference on Engineering (NUiCONE), 2017, pp. 1–4. doi:10.1109/NUICONE.2017.8325620.

[22] T. Liang, J. Glossner, L. Wang, S. Shi, X. Zhang, Pruning and quantization for deep neural network acceleration: A survey, Neurocomputing 461 (2021) 370–403. doi:https://doi.org/10.1016/j.neucom.2021.07.045.
URL https://www.sciencedirect.com/science/article/pii/S0925231221010894

[23] M. Mathew, K. Desappan, P. Kumar Swami, S. Nagori, Sparse, quantized, full frame cnn for low power embedded devices, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2017.

[24] Y. Kochura, Y. Gordienko, V. Taran, N. Gordienko, A. Rokovyi, O. Alienin, S. Stirenko, Batch size influence on performance of graphic and tensor processing units during training and inference phases, ArXiv abs/1812.11731.

[25] M. Kim, Guaranteeing that multi-level prioritized dnn models on an embedded gpu have inference performance proportional to respective priorities, IEEE Embedded Systems Letters (2021) 1–1doi:10.1109/LES.2021.3129769.

[26] H. Xu, R. Li, C. Pan, K. Li, Minimizing energy consumption with reliability goal on heterogeneous embedded systems, Journal of Parallel and Distributed Computing 127 (2019) 44–57. doi:https://doi.org/10.1016/j.jpdc.2019.01.006.
URL https://www.sciencedirect.com/science/article/pii/S0743731519300243

[27] A. HajiRassouliha, A. J. Taberner, M. P. Nash, P. M. Nielsen, Suitability of recent hardware accelerators (dsps, fpgas, and gpus) for computer vision and image processing algorithms, Signal Processing: Image Communication 68 (2018) 101–119.

doi:https://doi.org/10.1016/j.image.2018.07.007.
URL https://www.sciencedirect.com/science/article/pii/S0923596518303606

[28] NVIDIA, Cuda sdk code samples (May 2018).
URL https://www.nvidia.com/object/cuda_get_samples_3.html

[29] Khronos Group, OpenCL 2.0 API Specification (October 2014).
URL https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf

[30] TensorFlow Lite, https://www.tensorflow.org/lite/, accessed: 20th January 2022.

[31] TensorRT, https://developer.nvidia.com/tensorrt, accessed: 20th January 2022.

[32] Y. Makihara, R. Sagawa, Y. Mukaigawa, T. Echigo, Y. Yagi, Gait recognition using a view transformation model in the frequency domain, in: ECCV, Springer, 2006, pp. 151–163.

[33] K. Bashir, T. Xiang, S. Gong, Gait recognition using gait entropy image, in: 3rd International Conference Imaging for Crime Detection and Prevention, IET, 2009.

[34] T. H. Lam, K. H. Cheung, J. N. Liu, Gait flow image: A silhouette-based gait representation for human identification, Pattern recognition 44 (4) (2011) 973–987.

[35] Y. Zhao, S. Zhou, Wearable device-based gait recognition using angle embedded gait dynamic images and a convolutional neural network, Sensors 17 (3) (2017) 478.

[36] K. Nakajima, Y. Mizukami, K. Tanaka, T. Tamura, Footprint-based personal recognition, IEEE Transactions on Biomedical Engineering 47 (11) (2000) 1534–1537.

[37] R. Delgado-Escaño, F. M. Castro, J. R. Cózar, M. J. Marín-Jiménez, N. Guil, E. Casilari, A cross-dataset deep learning-based classifier for people fall detection and identification, Comput. Methods Programs Biomed. 184 (2020) 105265.

[38] Z. Meng, S. Fu, J. Yan, H. Liang, A. Zhou, S. Zhu, H. Ma, J. Liu, N. Yang, Gait recognition for co-existing multiple people using millimeter wave sensing, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019.

[39] V. U. Prabhu, J. Whaley, Vulnerability of deep learning-based gait biometric recognition to adversarial perturbations, in: CVPR Workshop on The Bright and Dark Sides of Computer Vision: Challenges and Opportunities for Privacy and Security (CV-COPS 2017), 2017.

[40] Z. He, W. Wang, J. Dong, T. Tan, Temporal sparse adversarial attack on gait recognition, arXiv preprint arXiv:2002.09674.

[41] W. Sheng, X. Li, Siamese denoising autoencoders for joints trajectories reconstruction and robust gait recognition, Neurocomputing.

[42] V. Narayanan, B. M. Manoghar, V. S. Dorbala, D. Manocha, A. Bera, Proxemo: Gait-based emotion learning and multi-view proxemic fusion for socially-aware robot navigation, arXiv.

[43] Y. Feng, Y. Li, J. Luo, Learning effective gait features using lstm, in: Proc. ICPR, IEEE, 2016, pp. 325–330.

[44] R. Liao, S. Yu, W. An, Y. Huang, A model-based gait recognition method with body pose and human prior knowledge, Pattern Recognition.

[45] M. M. Hasan, H. A. Mustafa, Multi-level feature fusion for robust pose-based gait recognition using rnn, International Journal of Computer Science and Information Security (IJCSIS).

[46] F. M. Castro, M. J. Marín-Jiménez, N. Guil, S. López-Tapia, N. P. de la Blanca, Evaluation of CNN architectures for gait recognition based on optical flow maps, in: BIOSIG, 2017, pp. 251–258.

[47] M. J. Marín-Jiménez, F. M. Castro, N. Guil, F. de la Torre, R. Medina-Carnicer, Deep multi-task learning for gait-based biometrics, in: IEEE Intl. Conference on Image Processing (ICIP), 2017, pp. 106–110.

[48] C. Xu, Y. Makihara, X. Li, Y. Yagi, J. Lu, Cross-view gait recognition using pairwise spatial transformer networks, IEEE Transactions on Circuits and Systems for Video Technology.

[49] N. Takemura, Y. Makihara, D. Muramatsu, T. Echigo, Y. Yagi, Multi-view large population gait dataset and its performance evaluation for cross-view gait recognition, IPSJ Transactions on Computer Vision and Applications 10 (1) (2018) 4.

[50] Y. Zhang, Y. Huang, S. Yu, L. Wang, Cross-view gait recognition by discriminative feature learning, IEEE Transactions on Image Processing.

[51] M. Jaderberg, K. Simonyan, A. Zisserman, et al., Spatial transformer networks, in: NeurIPS, 2015, pp. 2017–2025.

[52] A. Sokolova, A. Konushin, View resistant gait recognition, in: Proceedings of the 3rd International Conference on Video and Image Processing, 2019, pp. 7–12.

[53] S. Luo, S. Feng, H. Pan, J. Yin, X. Zhang, A sequence-based multi-scale network for cross-view gait recognition, in: 2019 6th International Conference on Systems and Informatics (ICSAI), IEEE, 2019, pp. 1179–1183.

[54] M. H. Khan, M. S. Farid, M. Grzegorzek, A non-linear view transformations model for cross-view gait recognition, Neurocomputing.

[55] X. Wu, W. An, S. Yu, W. Guo, E. B. García, Spatial-temporal graph attention network for video-based gait recognition, in: Asian Conference on Pattern Recognition, Springer, 2019, pp. 274–286.

[56] H. Chao, K. Wang, Y. He, J. Zhang, J. Feng, Gaitset: Cross-view gait recognition through utilizing gait as a deep set, IEEE PAMI (2021) 1–1doi:10.1109/TPAMI.2021.3057879.

[57] C. Fan, Y. Peng, C. Cao, X. Liu, S. Hou, J. Chi, Y. Huang, Q. Li, Z. He, Gaitpart: Temporal part-based model for gait recognition, in: CVPR, 2020, pp. 14225–14233.

[58] S. Hou, C. Cao, X. Liu, Y. Huang, Gait lateral network: Learning discriminative and compact representations for gait recognition, in: ECCV, 2020, pp. 382–398.

[59] B. Lin, S. Zhang, X. Yu, Gait recognition via effective global-local feature representation and local temporal aggregation, in: ICCV, 2021, pp. 14648–14656.

[60] A. Sakata, Y. Makihara, N. Takemura, D. Muramatsu, Y. Yagi, Gait-based age estimation using a densenet, in: Proc. Asian Conf. on Computer Vision, Springer, 2018, pp. 55–63.

[61] C. Bouças, J. P. Ferreira, A. P. Coimbra, M. M. Crisóstomo, P. A. Mendes, Generating individual gait kinetic patterns using machine learning, in: International Conference on Applied Technologies, Springer, 2019, pp. 53–64.

[62] F. M. Castro, M. J. Marín-Jiménez, N. Guil, N. P. de la Blanca, Multimodal feature fusion for CNN-based gait recognition: an empirical comparison, Neural Computing and Applications (2020) 1–21.

[63] M. Hofmann, J. Geiger, S. Bachmann, B. Schuller, G. Rigoll, The TUM Gait from Audio, Image and Depth (GAID) database: Multimodal recognition of subjects and traits, J. of Visual Com. and Image Repres. 25 (1) (2014) 195 – 206.

[64] F. M. Castro, M. J. Marín-Jiménez, N. Guil, Multimodal features fusion for gait, gender and shoes recognition, Machine Vision and Applications 27 (8) (2016) 1213–1228.

[65] M. J. Marín-Jiménez, F. M. Castro, R. Delgado-Escaño, V. Kalogeiton, N. Guil, Ugaitnet: Multimodal gait recognition with missing input modalities, IEEE Transactions on Information Forensics and Security 16 (2021) 5452–5462.

[66] R. Delgado-Escaño, F. M. Castro, N. Guil, M. J. Marín-Jiménez, Gaitcopy: Disentangling appearance for gait recognition by signature copy, IEEE Access 9 (2021) 164339–164347.

[67] M. Jia, H. Yang, D. Huang, Y. Wang, Attacking gait recognition systems via silhouette guided gans, in: Proceedings of the 27th ACM International Conference on Multimedia, 2019.

[68] S. Choi, S. Lee, Y. Kim, T. Kim, C. Kim, Hi-cmd: Hierarchical cross-modality disentanglement for visible-infrared person re-identification, arXiv.

[69] P. Chattopadhyay, A. Roy, S. Sural, J. Mukhopadhyay, Pose depth volume extraction from rgb-d streams for frontal gait recognition, Journal of Visual Communication and Image Representation 25 (1) (2014) 53–63.

[70] V. Mazzia, A. Khaliq, F. Salvetti, M. Chiaberge, Real-time apple detection system using embedded systems with hardware accelerators: An edge ai application, IEEE Access 8 (2020) 9102–9114.

[71] E. J. Jeong, J. Kim, S. Tan, J. Lee, S. Ha, Deep learning inference parallelization on heterogeneous processors with tensorrt, IEEE Embedded Systems Letters.

[72] H.-H. Nguyen, D. N.-N. Tran, J. W. Jeon, Towards real-time vehicle detection on edge devices with nvidia jetson tx2, in: 2020 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), IEEE, 2020, pp. 1–4.

[73] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial intelligence and statistics, PMLR, 2017, pp. 1273–1282.

[74] J. Zhao, R. Mortier, J. Crowcroft, L. Wang, Privacy-preserving machine learning based data analytics on edge devices, in: Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society, 2018, pp. 341–346.

[75] Z. Tao, Q. Li, {eSGD}: Communication efficient distributed deep learning on the edge, in: USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), 2018.

[76] L. Deng, G. Li, S. Han, L. Shi, Y. Xie, Model compression and hardware acceleration for neural networks: A comprehensive survey, Proceedings of the IEEE 108 (4) (2020) 485–532. doi:10.1109/JPROC.2020.2976475.

[77] J. Guo, W. Zhang, W. Ouyang, D. Xu, Model compression using progressive channel pruning, IEEE Transactions on Circuits and Systems for Video Technology 31 (3) (2020) 1114–1124.

[78] Z. Wang, C. Li, X. Wang, Convolutional neural network pruning with structural redundancy reduction, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 14913–14922.

[79] J.-H. Luo, J. Wu, Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference, Pattern Recognition 107 (2020) 107461.

[80] P. Singh, V. K. Verma, P. Rai, V. P. Namboodiri, Play and prune: Adaptive filter pruning for deep model compression, arXiv preprint arXiv:1905.04446.

[81] A. Goncharenko, S. Alyamkin, A. Denisov, E. Terentev, Winning solution on lpirc-ll competition., in: CVPR Workshops, 2019, pp. 10–16.

[82] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, M. Aleksic, A quantization-friendly separable convolution for mobilenets, in: 2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2), IEEE, 2018, pp. 14–18.

[83] D. Kang, D. Kang, J. Kang, S. Yoo, S. Ha, Joint optimization of speed, accuracy, and energy for embedded image recognition systems, in: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2018, pp. 715–720.

[84] K. Wu, Y. Guo, C. Zhang, Compressing deep neural networks with sparse matrix factorization, IEEE Transactions on Neural Networks and Learning Systems 31 (10) (2019) 3828–3838.

[85] M. Masana, J. van de Weijer, L. Herranz, A. D. Bagdanov, J. M. Alvarez, Domain-adaptive deep network compression, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 4289–4297.

[86] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, D. Shin, Compression of deep convolutional neural networks for fast and low power mobile applications, arXiv preprint arXiv:1511.06530.

[87] K. Paupamah, S. James, R. Klein, Quantisation and pruning for neural network compression and regularisation, in: 2020 International SAUPEC/RobMech/PRASA Conference, IEEE, 2020, pp. 1–6.

[88] X. Ma, K. Ji, B. Xiong, L. Zhang, S. Feng, G. Kuang, Light-yolov4: An edge-device oriented target detection method for remote sensing images, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 14 (2021) 10808–10820.

[89] Y. Kim, J. Kim, D. Chae, D. Kim, J. Kim, Mlayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization, in: Proceedings of the Fourteenth EuroSys Conference 2019, EuroSys '19, 2019.

[90] F. Jia, D. Zhang, T. Cao, S. Jiang, Y. Liu, J. Ren, Y. Zhang, Codl: Efficient cpu-gpu co-execution for deep learning inference on mobile devices, in: Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services, MobiSys '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 209–221. doi:10.1145/3498361.3538932. URL https://doi.org/10.1145/3498361.3538932

[91] Sapdragon 855, https://www.qualcomm.com/products/application/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-855-mobile-platform, accessed: 1st September 2022.

[92] Mali-G76-Arm, https://www.arm.com/products/silicon-ip-multimedia/gpu/mali-g76, accessed: 1st September 2022.

[93] D. Seichter, M. Köhler, B. Lewandowski, T. Wengefeld, H.-M. Gross, Efficient rgb-d semantic segmentation for indoor scene analysis, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 13525–13531.

[94] M. Qasaimeh, K. Denolf, A. Khodamoradi, M. Blott, J. Lo, L. Halder, K. Vissers, J. Zambreno, P. H. Jones, Benchmarking vision kernels and neural network inference accelerators on embedded platforms, Journal of Systems Architecture 113 (2021) 101896.

[95] M. Antonini, T. H. Vu, C. Min, A. Montanari, A. Mathur, F. Kawsar, Resource characterisation of personal-scale sensing models on edge accelerators, in: Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things, 2019, pp. 49–55.

[96] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, et al., Tensorflow lite micro: Embedded machine learning on tinyml systems, arXiv preprint arXiv:2010.08678.

[97] K. Dokic, M. Martinovic, D. Mandusic, Inference speed and quantisation of neural networks with tensorflow lite for microcontrollers framework, in: 2020 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), IEEE, 2020, pp. 1–6.

[98] A. Zeroual, M. Derdour, M. Amroune, A. Bentahar, Using a fine-tuning method for a deep authentication in mobile cloud computing based on tensorflow lite framework, in: 2019 International Conference on Networking and Advanced Systems (ICNAS), IEEE, 2019, pp. 1–5.

[99] B. K. Horn, B. G. Schunck, Determining optical flow, Artificial intelligence 17 (1-3) (1981) 185–203.

[100] M. Zhu, S. Gupta, To prune, or not to prune: exploring the efficacy of pruning for model compression, arXiv preprint arXiv:1710.01878.

[101] G. Li, C. Qian, C. Jiang, X. Lu, K. Tang, Optimization based layer-wise magnitude-based pruning for dnn compression., in: IJCAI, 2018, pp. 2383–2389.

[102] J. Lee, S. Park, S. Mo, S. Ahn, J. Shin, Layer-adaptive sparsity for the magnitude-based pruning, arXiv preprint arXiv:2010.07611.

[103] M. Hofmann, J. Geiger, S. Bachmann, B. Schuller, G. Rigoll, The tum gait from audio, image and depth (gaid) database: Multimodal recognition of subjects and traits, Journal of Visual Communication and Image Representation 25 (1) (2014) 195–206.

[104] S. Yu, D. Tan, T. Tan, A framework for evaluating the effect of view angle, clothing and carrying condition on gait recognition, in: 18th International Conference on Pattern Recognition (ICPR'06), Vol. 4, IEEE, 2006, pp. 441–444.

[105] M. Abadi, others., TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).