TecnoLógicas

Performance Evaluation of Convolutional Networks on Heterogeneous Architectures for Applications in Autonomous Robotics

Evaluación de desempeño de redes convolucionales sobre arquitecturas heterogéneas para aplicaciones en robótica autónoma

Joaquín Guajo[1];
Cristian Alzate-Anzola[2];
Luis Castaño-Londoño[3];
David Márquez-Viloria[4]

[1] Instituto Tecnológico Metropolitano, Medellín-Colombia, joseguajo152012@correo.itm.edu.co
[2] Instituto Tecnológico Metropolitano, Medellín-Colombia, cristianalzate224500@correo.itm.edu.co
[3] Instituto Tecnológico Metropolitano, Medellín-Colombia, luiscastano@itm.edu.co
[4] Instituto Tecnológico Metropolitano, Medellín-Colombia, davidmarquez@itm.edu.co

ISSN-p: 0123-7799
ISSN-e: 2256-5337

Vol. 25, **nro.** 53, e2170, 2022

Recibido: 12 octubre 2021
Aceptado: 15 febrero 2022
Disponible: 29 abril 2022

Cómo citar / How to cite

J. Guajo; C. Alzate-Anzola; L. Castaño-Londoño; D. Márquez-Viloria, "Performance Evaluation of Convolutional Networks on Heterogeneous Architectures for Applications in Autonomous Robotics," *TecnoLógicas*, vol. 25, nro. 53, e2170, 2022. https://doi.org/10.22430/22565337.2170

Abstract

Humanoid robots find application in human-robot interaction tasks. However, despite their capabilities, their sequential computing system limits the execution of computationally expensive algorithms such as convolutional neural networks, which have demonstrated good performance in recognition tasks. As an alternative to sequential computing units, Field-Programmable Gate Arrays and Graphics Processing Units have a high degree of parallelism and low power consumption. This study aims to improve the visual perception of a humanoid robot called NAO using these embedded systems running a convolutional neural network. The methodology adopted here is based on image acquisition and transmission using simulation software: Webots and Choreographe. In each embedded system, an object recognition stage is performed using commercial convolutional neural network acceleration frameworks. Xilinx® Ultra96™, Intel® Cyclone® V-SoC and NVIDIA® Jetson™ TX2 cards were used, and Tinier-YOLO, AlexNet, Inception-V1 and Inception V3 transfer-learning networks were executed. Real-time metrics were obtained when Inception V1, Inception V3 transfer-learning and AlexNet were run on the Ultra96 and Jetson TX2 cards, with frame rates between 28 and 30 frames per second. The results demonstrated that the use of these embedded systems and convolutional neural networks can provide humanoid robots such as NAO with greater visual recognition in tasks that require high accuracy and autonomy.

Keywords

Convolutional neural networks, field programmable gate array, system-on-a-chip, high-level synthesis, humanoid robot.

Resumen

Los robots humanoides encuentran aplicación en tareas de interacción humano-robot. A pesar de sus capacidades, su sistema de computación secuencial limita la ejecución de algoritmos computacionalmente costosos, como las redes neuronales convolucionales, que han demostrado buen rendimiento en tareas de reconocimiento. Como alternativa a unidades de cómputo secuencial se encuentran los Field Programmable Gate Arrays y las Graphics Processing Unit, que tienen un alto grado de paralelismo y bajo consumo de energía. Este trabajo tuvo como objetivo mejorar la percepción visual del robot humanoide NAO utilizando estos sistemas embebidos que ejecutan una red neuronal convolucional. El trabajo se basó en la adquisición y transmisión de la imagen usando herramientas de simulación como Webots y Choreographe. Posteriormente, en cada sistema embebido, se realizó una etapa de reconocimiento del objeto utilizando *frameworks* de aceleración comerciales de redes neuronales convolucionales. Luego se utilizaron las tarjetas Xilinx Ultra96, Intel Cyclone V-SoC y Nvidia Jetson TX2; después fueron ejecutadas las redes Tinier-Yolo, Alexnet, Inception V1 y Inception V3 transfer-learning. Se obtuvieron métricas en tiempo real cuando Inception V1, Inception V3 transfer-learning y AlexNet fueron ejecutadas sobre la Ultra96 y Jetson TX2, teniendo como intervalo entre 28 y 30 cuadros por segundo. Los resultados demostraron que el uso de estos sistemas embebidos y redes neuronales convolucionales puede otorgarles a robots humanoides, como NAO, mayor reconocimiento visual en tareas que requieren alta precisión y autonomía.

Palabras clave

Redes neuronales convolucionales, matriz de puertas lógicas programable en campo, sistema en chip, síntesis de alto nivel, robot humanoide.

1. INTRODUCTION

Humanoid robots are used in assistance-type applications related to domestic, educational, and therapeutic services, among others [1]. This use arises from the sensations of comfort generated in humans in the interaction with this type of robot [2]. To produce these interactions, they have sensory devices that provide them with the ability to perceive and understand the environment. In particular, the visual perception system of humanoid robots is composed of cameras integrated into their structure, which gives them a field of vision [3].

Thanks to said field of vision, they can be used in applications oriented to object recognition in real time. For instance, the use of the NAO humanoid robot acquisition system and machine vision techniques for recognition and operation of a tool by programming by demonstration is presented in [4]. In that paper, the object to be recognized is captured by the visual perception system. The image is processed by a CPU, in which the execution is performed sequentially. The sequential processing of the image pixels produces a delay in the execution time in the computer system of the humanoid robot, limiting its responsiveness in decision-making for tasks involving object recognition [5]. This limitation is also shown in the implementation of classical computer vision techniques by [6], [7].

Deep learning algorithms have been executed on the sequential computing system of some types of humanoid robots. These algorithms are computational models that replicate human cognitive ability and contribute to the use of robots in everyday environments. Among these algorithms are Convolutional Neural Networks (CNNs), which have proven to be relevant in projects where object recognition, localization, and detection are integrated, becoming efficient in terms of accuracy in the completion of these tasks [8]. Different architectures of CNNs have been developed, such as AlexNet, VGGNet, ResNet, among others [9] - [11]. These networks were trained with millions of images, which improved the accuracy of the network in recognizing objects in everyday environments [12].

Despite their advantages, CNNs present a high computational cost when implemented in the sequential computing system of humanoid robots. Therefore, developments have been implemented in external computational systems based on higher-capacity CPUs connected to the robot's computational architecture, usually through Ethernet [13]. Two studies [14], [15] concluded that the limitations in robot image processing were overcome by integrating an external computational system. However, the robot's autonomy was affected by the continuous connection, generating dependence in tasks that require free movement in its environment. The limitations, in terms of responsiveness and autonomy, of humanoid robots with sequential operation computational architectures present a challenging problem for the implementation of CNNs on GPU- or FPGA-based embedded systems. Nevertheless, these devices have a high degree of parallelism and low power consumption in image and video processing applications, which could provide the humanoid robot with greater processing capacity and autonomy in tasks involving higher perception and deeper understanding of the environment.

An implementation of a CNN on external computational systems is presented in [16]. The proposed system is based on the execution of a CNN SSD mobilenet DNN on the CPU-centered Intel® NUC7i7BNH (NUC) and Jetson TX2 computational systems for an application focused on pedestrian detection on the NUgus humanoid robot. The DNN implementation on the NUC was performed on the CPU due to the incompatibility between Tensorflow and OpenCL, which prevented the implementation on the GPU side. Regarding the development on the Jetson TX2, the DNN implementation is performed on the GPU through CUDA. The results showed that the CPU of the NUC is faster than the GPU of the Jetson TX2 when executing the DNN: the NUC took 0.17 s; and the Jetson TX2, 0.57 s.

In terms of power consumption, the NUC CPU consumes 40.52 W, and the Jetson TX2, around 9.8 W. The results showed high-power consumption and inference times shorter than what is established for a real-time application.

Different studies have addressed the implementation of CNNs on FPGAs. In [17], the authors explore weight and node-level parallelization over convolutional layer computations.

The system uses maximum resources through data reuse and concatenation. Decomposition of input data into convolutional computations is also an approach implemented in that study; each iteration reuses the arithmetic computation units to process, in different fragments, the split data. The data is transmitted between the FPGA and the memory through the Direct Memory Access (DMA) unit and Axi-Stream transmission interfaces. This transmission in [18] includes partitioning techniques and embedding the weights in local memory and parallelization techniques. These parallelizations and the batch-based method reduced the memory bandwidth required in CNN [19] matrix multiplications. The opposite case occurs in the implementation of the Winograd algorithm proposed by [20], where less computational resources are used, but more pressure is put on memory bandwidth.

On the other hand, the use of development environments has allowed researchers to control memory usage and parallelization techniques on these reconfigurable devices. In [21], [22] information from a trained CNN network is synthesized in hardware through the Vivado HLS high-level synthesis tool. The tool defines how the Intellectual Property Core (IP Core) that will contain the information of the images to be classified should be generated. The IP Core is generated using internally developed high-level wrappers to facilitate communication with direct access to the DMA memory. In [19] is presented a similar study, but they proposed an analytical design model called Roofline in their development. The model can be used to quantitatively analyze the computational performance and memory bandwidth required for any solution of a CNN design.

Regarding the implementation of CNNs on GPUs, different acceleration techniques have been proposed for video and image processing. In [23], an acceleration method based on the treatment of binary weights is proposed, focusing on optimizing the arithmetic kernel in the storage of the weights. A similar approach is presented in [24], where the acceleration is performed through a resistive random-access memory (i.e., ReRAM). This accelerator architecture was adapted for bit-by-bit convolution.

Previous studies showed a high-power consumption. In [25], a programmable many-core accelerator reduces said consumption for a CNN network architecture. The accelerator is called PACENet and consists of a neural network kernel-specific instruction set architecture and six pipeline stages to accelerate the convolution layer, Relu activations, Maxpool layer, and fully connected layer. The accelerator design is similar. However, two scheduling algorithms were implemented in two stages. The first stage is focused on image combination to accelerate the feed-forward process of the CNN; and the second, on a memory-light cost algorithm for accelerating an arbitrarily large CNN model for a memory-limited GPU device.

In order to enhance the performance of the GPU system, accelerators have been developed through the parallel computing architecture CUDA (Computed Unified Device Architecture) in the execution of CNNs. In [26], CUDA is used to create a mechanism to improve the network execution, which consists of integrating the data of the network nodes and the dynamic adjustment of the smoothing factor of the basis function. Other authors [27] implemented a C++ library on CUDA to accelerate the training and classification process of CNN and NVIDIA cuBLAS libraries to exchange the mathematical vector and functional operations.

The developments of CNN accelerators on FPGA- and GPU-based embedded systems described in the previous paragraphs demonstrated high processing capacity and low power consumption. However, the use of these systems presents an open research topic regarding performance evaluation when they are applied to humanoid robots with sequential processors integrated into their structure. These processors present limitations in their processing capacity when they execute sophisticated algorithms of deep learning techniques, restricting the robot in performing tasks related to object recognition in real time.

This paper proposes a visual perception enhancement system for humanoid robots using FPGA- or GPU-based embedded systems running convolutional neural networks. This study focuses on solving three problems: (1) reducing execution time and improving object detection accuracy in an everyday environment while maintaining robot autonomy; (2) creating a system that can be replicable to humanoid robots such as Pepper and Robotis OP3; and (3) producing a development that can be used to integrate heterogeneous architectures with a high degree of parallelism, low power consumption and small size that execute a CNN and can be easily integrated into the structure of any humanoid robot. The rest of this paper is organized as follows. Section 2 details the method and describes the acquisition of the image, communication with embedded systems, and the use of CNN acceleration frameworks for each heterogeneous architecture. Section 3 presents the results and discussion. Finally, Section 4 draws the conclusions.

2. METHOD

The goal of the proposed methodology is to design a visual perception enhancement system for humanoid robots based on an external computational system that executes a CNN.

For this system, FPGA- and GPU-based embedded computational systems were evaluated. These heterogeneous architectures are small and have a high degree of parallelism and low power consumption. In this study, Intel Cyclone V SoC and Xilinx Ultra96-V2 cards were used for FPGA evaluation, and a Nvidia Jetson TX2 development board was used for GPU evaluation. In addition, an application focused on the classification of toys for early childhood education is developed through a transfer-learning on the Inception-V3 network from the INSTRE dataset. The inference is performed on the Ultra96 FPGA embedded system. Table 1 shows the embedded systems' features, along with the acceleration framework and the CNN executed here.

Table 1. Embedded systems, acceleration frameworks and implemented CNNs
Source: Created by the authors.

Hardware type	Development board	Acceleration framework	CNN
FPGA	Cyclone V-SoC	PipeCNN	AlexNet
		Finn	Tinier-YOLO
	Ultra96	Vitis-DPU	Inception-V1
		Dnndk	Inception-V3 transfer learning
GPU	Jetson TX2	Darknet	Tinier-YOLO
		TensorRT	Inception-V1
			AlexNet

The study case in this article is a visual perception system for the NAO humanoid robot. However, since there is an Ethernet connection between the robot and the embedded system, the proposed system can be replicable in humanoid robots such as Pepper and Robotis OP3.

This study was conducted using a virtual NAO humanoid robot, and the virtual environment was created by means of Cyberbotics Ltd. Webots [28], a mobile robotic simulation software that provides a rapid prototyping environment for modeling, programming, and simulating mobile robots.

For the acquisition and transmission of the virtual image provided by Webots, Naoqi SDK and Choregraphe [29] programming software was used. For the transmission of the video from the robot, the encoding and sending of packets was performed using a TCP/IP socket.

The video reception stage and the execution of the CNN are performed on each commercial embedded system selected here. Finally, for comparative purposes, an implementation focused on the execution of classical machine learning models on the computational system of the humanoid robot NAO is performed using a processor that has the same characteristics as NAO's CPU. This hardware consisted of a Dell Inspiron Mini laptop containing a 1.6-GHz Intel Atom Z530 processor. The use of a processor with the same characteristics as NAO's CPU facilitates testing without the presence of or powering up the humanoid robot. This paper combines the capabilities of CNNs, and heterogeneous architectures based on FPGA and GPU to improve the visual perception of humanoid robots such as NAO. Starting from the acquisition of the image of the object by the robot, it is necessary to have clear tools with which to simulate a virtual environment. For this application, Webots, Choreographe and python were used as programming languages for the generation of the virtual environment and transmission of the image. Once the image is acquired, it is sent to each embedded system, and, using acceleration frameworks for FPGA and GPU, the processing and classification or detection of the image is performed. At this point, understanding CNNs and knowledge of heterogeneous architectures become necessary to reproduce the proposed methodology. Figure 1 shows the diagram of the proposed system methodology.

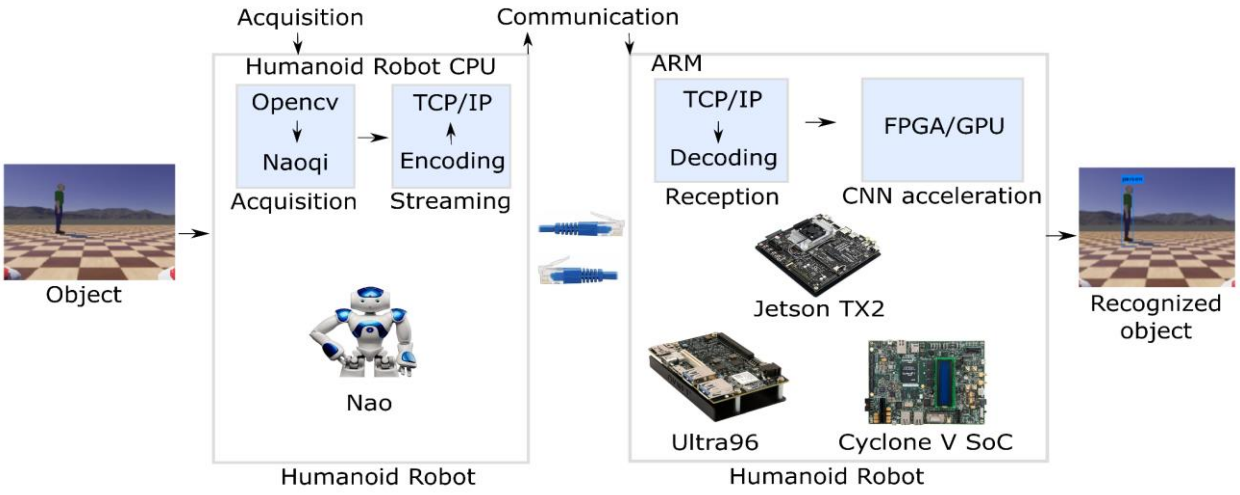

Figure 1. Diagram of the proposed system methodology. Source: Created by the authors.

The following subsections detail the image acquisition-communication system and the implementation of the CNNs on each development board using CNN acceleration frameworks. Finally, we present the design of a backpack for NAO, which contains a heterogeneous architecture and a battery to power it. This approach allows an external

computational system to provide NAO with increased accuracy while maintaining its autonomy and mobility.

2.1 Image acquisition and transmission

The video acquisition system for the NAO humanoid robot was simulated using Webots, and it was possible to create a virtual environment and subsequently perform a movement control using Choregraphe. Once the scene is captured by Choregraphe, video transmission to the FPGA or GPU embedded system is started.

2.1.1 Image acquisition

The image acquisition system was based on the use of a virtual environment containing objects to be recognized by the NAO humanoid robot that was also located in the recreated scene. This virtual environment was created using the Webots simulator, where the image of the recreated scene is sent to Choregraphe in order to edit interactive movements for the robot and start the transmission of the image to each embedded system. The communication between Webots and Choregraphe is established using naoqisim software, which allows the motion control of the NAO robot generated by Choregraphe to be displayed in the Webots simulator. For the visualization in Choregraphe of the image generated by Webots, NAOqi API is used, which contains libraries for the acquisition and communication of images by assigning an IP address and an Ethernet port. For this image display, the process performed initially imports the libraries opencv and vision definitions from naoqi for image acquisition.

From the function ALProxy the video transmitted by Webots is obtained from Choregraphe and is sent to the video monitor using the subscribeCamera function. Finally, the obtained image is displayed on the Choregraphe video monitor using the getImageRemote method. The pseudocode of the video acquisition is shown in Algorithm 1.

Algorithm 1: Get Webots image in Choregraphe

```
1  ip_robot = "string length"
2  port_robot = "int length
3  videoDevice = "ALProxy('image_webots', ip_robot, port_robot)"
4  captureDevice = "videoDevice.suscribeCamera()"
5  width = "image width"
6  height = "image height"
7  While True do
8      result = videoDevice.getImageRemote(captureDevice)
9      for i -> height
10         for j -> width
11             add image result in width x height
12         end
13     end
14     Result: show image in Choregraphe: result
15 end
```

2.1.2 Image transmission to the embedded system

In this stage, TCP was used as the packet transmission protocol. Using Python libraries, it was possible to encode the image and then send it through an IP address and an Ethernet port. The Base 64 library allowed the encoding and decoding of the image according to RFC 3548 standard. Using Base 64, the algorithms of Base 16, Base 32 and Base 64 were used to encode and decode arbitrary strings into text strings that could be sent over the network.

Finally, the image displayed in Choregraphe is transmitted to each embedded system. Algorithm 1 was adapted for the coding-decoding of the image and the transmission. The pseudocode is shown in Algorithm 2.

Algorithm 2: Video transmission to each embedded system

```
ip_robot = "string length"
port_robot = "int length
videoDevice = "ALProxy('image_webots', ip_robot, port_robot)"
captureDevice = "videoDevice.suscribeCamera()"
width = "image width"
height = "image height"
While True do
    result = videoDevice.getImageRemote(captureDevice)
    for i -> height
        for j -> width
            add image result in width x height
        end
    end
    result = encoded(result)
    send by socket(result)
    Result: video transmission result
end
```

2.1.3 Image reception on the embedded system

For the reception of the virtual image provided by Webots on the FPGA and GPU, the libraries described in the previous section were used. Through the setsockopt string function, the received data are manipulated and converted to a series of characters string.

Subsequently, this series of characters is decoded and converted to the positional numbering system Base 64. Finally, the decoded image is read from the buffer stored in the memory using the OpenCV function cv2.imdecode. This pseudocode is shown in Algorithm 3.

2.2 Implementation of CNNs on acceleration frameworks

The CNN implementation on FPGA- and GPU-based embedded systems was performed using available acceleration frameworks. For the execution of the CNNs on FPGAs, the PipeCNN, FINN, and DPU frameworks were used. For the GPU, the Darknet and TensorRT frameworks were implemented. For our application, pre-trained models were used to evaluate the performance of these heterogeneous architectures and include them in the visual perception enhancement system for humanoid robots.

Algorithm 3: Video reception on each embedded system

```
ip_robot = "string length"
port_robot = "int length
socket.bind(listen for ip_robot and port_robot)
setsockpt(transmitted image)
while True do
    result = base 64 decoding
    result = buffer reading by opencv
    Result: viewing the video result
end
```

2.3 Implementation of CNNS on FPGA boards

The implementation was performed for two different acceleration frameworks. The PipeCNN framework was adapted for the Cyclone V-SoC development system on the INTEL platform to develop this work. The Cyclone V-SoC device has an ARM processor that acts as a host and an FPGA that works as an accelerator by executing a kernel implemented with OpenCL. PipeCNN uses two parameters to control the hardware resource cost and improve execution time. These parameters are the size of the data vectorization and the number of parallel computing units. This framework also uses high-level methodologies but based on OpenCL code; thus, highly efficient, and configurable kernels can be adapted to a wide variety of CNN models. PipeCNN is an FPGA CNN accelerator developed in OpenCL. It is compatible with Caffenet (AlexNet), VGG-16 and ResNeT-50. The framework is reconfigurable, which makes it easily adaptable to different boards, and, being written in OpenCL, it allows an easy implementation between different platforms such as CPU and GPU.

The convolutional architecture defined in the PipeCNN framework is cascading, in which each layer is executed once the previous one is finished. Another optimization used by PipeCNN is Fixed-Point arithmetic representation instead of Floating-Point, reducing the resource consumption considerably in the FPGAs, although the accuracy of the CNNs is also reduced. This architecture is shown in Figure 2.

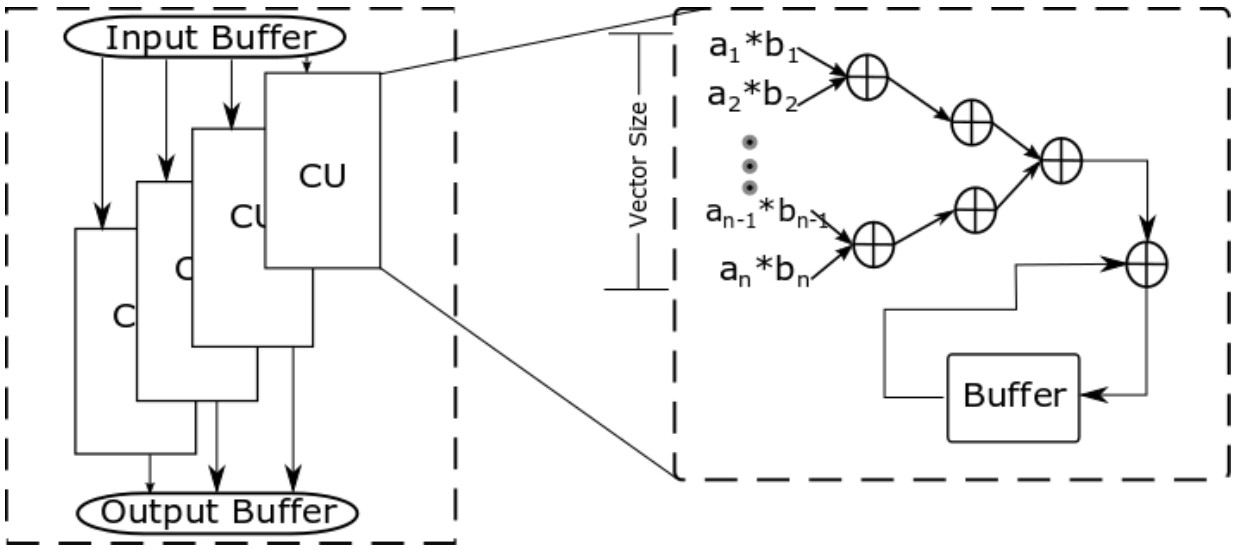

Figure 2. Hardware architecture of the PipeCNN framework. Source: Created by the authors.

For the implementation of PipeCNN, Intel® FPGA SDK for OpenCL software was downloaded. Subsequently, the software was configured, and the OpenCL libraries were compiled. This compilation generated two files: *.aocx, which is the FPGA binary, and an executable that loaded the image and sent it to the FPGA. These two files were loaded on the board and subsequently configured when the embedded system was booted.

In addition, the free software application Quantized Neural Network was adapted on a XILINX platform using a heterogeneous FPGA-based architecture. This application is based on the FINN framework presented by [30]. This framework allows the implementation of CNNs on Xilinx devices with a predefined architecture and high efficiency to focus more on the implementation. The implementation is done on an Ultra96 SoC board using the PYNQ framework. The PYNQ framework is used for rapid code development on the host. This framework allows high-speed applications to run side-by-side on hardware with Python-based software applications.

In the FINN framework, there are two types of CNN acceleration architectures implemented on FPGA; they are shown in Figure 3. The main difference between them is the fact that the DF architecture is built for a single CNN topology, weights and activations already defined. However, this is inefficient in FPGAs because in these architectures reconfigurable computing can be leveraged. This advantage is exploited in the MO architecture because, unlike its DF counterpart, it does not depend on network topology, and the block feeds back on itself and reconfigures itself based on the network topology. The DF architecture has an advantage over its MO counterpart because it is a dedicated and enhanced architecture only for a particular topology. Since the weights are in the same system, the processing time is much shorter, unlike MOs configured at each layer of the CNN.

In the MO architecture, we can implement CNN topologies that cannot be executed in the DF architecture because they consume more resources than the board has available. The MO architecture loads each of these layers sequentially on the FPGA and thus does not occupy the entire resource space, especially the BRAM.

Figure 4 shows the Tinier-YOLO network architecture, which is implemented in the Ultra96 SoC. The input and output layers are executed in software (ARM) through Python, while the internal layers are executed in hardware (FPGA). The layers consist of operations such as convolutions and max pooling. The framework supports only quantized layers, meaning that weights and activations are represented from 1 to 3 bits. Tinier-YOLO is a modified version of the Tiny-YOLO object detection system. Tinier-YOLO is also trained with the PASCAL VOC database, but with 1 bit for weights and 3 bits for activations. Tinier-YOLO achieves 50.1 % mAP, while Tiny-YOLO achieves 57.1 % mAP.

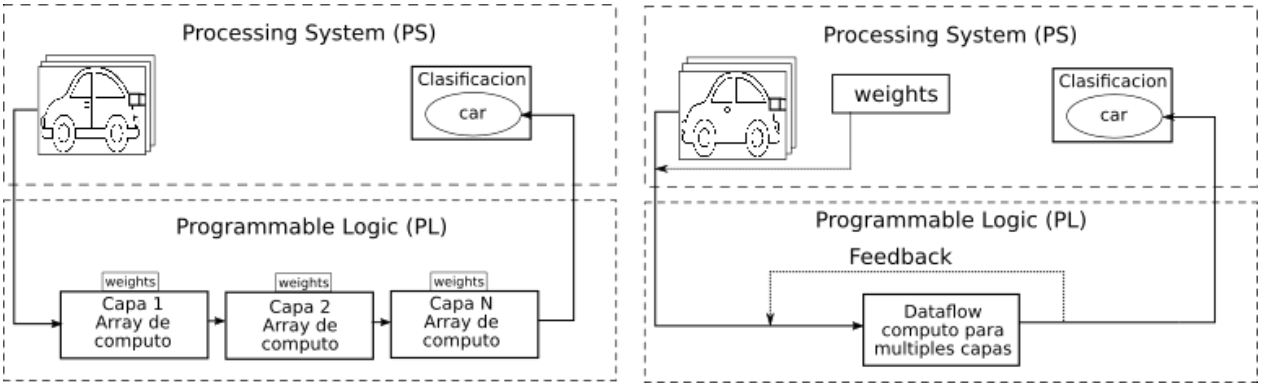

Figure 3. Types of FINN framework architectures: (a) DF architecture with defined layers and weights. (b) MO architecture with layers and weights for different accuracies and sizes. Source: Created by the authors.

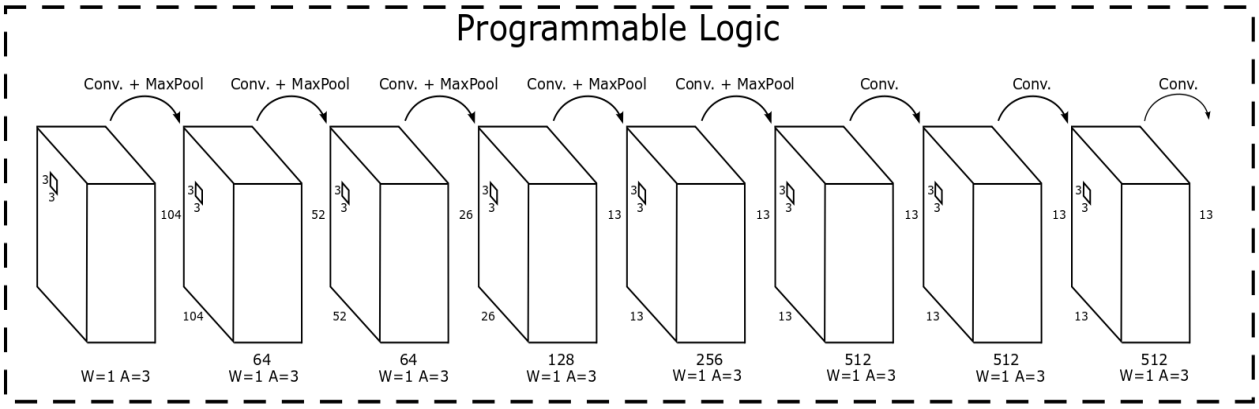

Figure 4. Tinier-YOLO network topology implemented with the QNN framework
Source: Created by the authors.

The Xilinx DPU was used for the execution of the CNN Inception-V1 on the Ultra96-V2 FPGA. This DPU is a programmable engine dedicated to executing each of the convolutional layers present in a CNN through a register configuration module, data controller module, and convolution computation module. This DPU module is integrated as a programmable logic (PL) unit, which is connected to the processing system (PS). Like the FINN framework, along with the DPU, PYNQ is used for host-side application development through an AX14-based interface. Vitis IA is used to convert the model trained in Tensorflow to *.elf format. This format contains the weight information, which is read by the ARM of the FPGA SoC; after that, the tasks are sent to the DPU for processing and transmitting the results back.

There are *.elf files for trained models such as Mnist, Resnet50 and YOLO v3. However, to make a comparison with the Jetson TX2, we selected the Inception-V1 CNN.

The following paragraphs present a system for the improvement of the visual perception of the NAO robot so that it can be used in pedagogical activities for early childhood education.

In this system, the first step is the recognition of toys by the robot in order to have a friendly interaction with children in learning activities. For this purpose, the Inception-V3 network was selected, which can distinguish up to 1000 classes, allowing the robot to better understand its interaction environment. This network does not detect objects in the image like Tinier-YOLO; however, in this application, we consider the number of classes a key factor, and, although some networks could have a better performance in this aspect, they need more hardware resources than those existing in the Ultra96. For toy classification, a dataset of 10 classes was constructed from the INSTRE database [31]. For each of the selected classes, a training set consisting of 70 to 100 images and a validation set between 30 and 40 were defined. The complete dataset consists of 810 training images and 356 validation images.

The training for toy classification was performed using transfer-learning, which is a method that consists of reusing the previous training of a network by freezing some layers and selecting a lower learning rate to adapt the weights to the new training set. Transfer-learning can decrease the training time and use small databases because most of the features, such as edges, curves, and colors, are already defined by the weights trained by the first layers. In this paper, transfer-learning is used with the Inception-V3 convolutional network, which was originally trained with the Imagenet dataset to recognize 1000 types of classes.

This topology consists of 310 convolutional layers and fully-connected layers. For the training with transfer-learning, the first 172 convolutional layers of the network are frozen, and the output layer of Inception-V3 is a softmax activation with 1000 classes. In this study,

this layer is removed and replaced with a 10-class softmax layer, whereby this layer is trained together with the remaining dense layers for the application-centric classification of 10 types of toys.

This network was implemented on the Ultra96 using the DNNDK framework, which employs a DPU module that allows high computation and easy access for heterogeneous programming. DNNDK consists of several tools such as DECENT and DNNC. DECENT is responsible for reducing the size of CNNs in terms of data, i.e., a model that is stored in 32-bit floating point is quantized to 8 bits for each weight. By reducing this information, the size of the model is reduced, allowing an optimization in terms of computational efficiency, energy efficiency and less memory for the system, especially in bandwidth during data transmission from the host to the FPGA. DNNC is responsible for improving the resources used by the DPU by optimizing bandwidth and power consumption.

DNNC uses the debugged DECENT model and applies optimized compilation and transformation techniques, such as compute node merging, efficient instruction scheduling and reuse of on-chip memory features and weights.

2.4 Implementation of CNNs on GPU boards

Two frameworks were used for the acceleration of CNNs on the Jetson TX2 GPU. In the first one, Nvidia TensorRT, Inception-V1 and AlexNet networks were implemented.

TensorRT consists of two stages: training and inference. The training stage is mainly based on the use of Digits, where it is possible to manage and evaluate the data of the model by running it in the cloud or localhost. Once the training is complete, the trained weights are downloaded, and the inference is performed. In this study, the first stage was not performed, a previously trained model was obtained, and the optimization between the host and the device was performed for the inference stage.

TensorRT, CUDA and cuDNN were used for the inference stage. TensorRT is an optimizer of a model trained using CUDA and cuDNN; thus, it achieves low latency, which is ideal for real-time applications. This framework provides a quantization operation for the GPU inference engine. The computational latency is shortened due to floating arithmetic operations, and, in order not to reduce the model's mAP, the weights were quantized to 16-bits at the inference stage.

TensorRT modifies the size of the images before and after the inference process. Since this modification is computationally expensive on the CPU, the framework uses multithreading on the CPU to speed up the process. TensorRT creates two threads on each CPU core, and each thread processes one batch of data. The Jetson TX2 has 6 CPU cores, so TensorRT creates 12 threads. GPU inference can only run on a single thread; therefore, the framework takes inference as a mutual process, and the different threads must compete for the GPU. Finally, for the implementation of the Tinier-YOLO network, Darknet was used, which is an open-source framework for running convolutional neural networks where data are processed through C and CUDA for the computation between the CPU and the GPU.

2.5. Implementation of classical machine learning algorithms for early childhood education applications

The implementation of classical classifiers was performed using the same INSTRE database. The classifiers were implemented using Scikit-Learn instead of popular frameworks such as Keras and TensorFlow since the 32-Bit architecture of the NAO computing system does not allow the installation and configuration of these tools. This

restriction limits the execution of algorithms such as CNNs on the sequential system of the NAO robot, reducing its applicability in activities that require a greater understanding of the environment. Each pixel is used as one feature per frame, which means 200x200x3 features for the classification models.

Four classical classification algorithms have been implemented in the literature for binary and multi-class object classification: Logistic Regression, Naive Bayes, Decision Tree, and Random Forest. Processing images, these models perform well when hyperparameter optimization is performed; however, this task is complex and depends on the characteristics of the dataset. CNNs, using their hidden layers, have the ability to select the most important features without difficulty, so better results are obtained in terms of accuracy when the dataset is considerably large.

2.6. NAO backpack design

This section describes the design of a backpack for NAO integrated into the back of the humanoid robot. This extension of NAO contains an Ultra96-V2 FPGA that runs computationally expensive algorithms such as CNNs. The selection of the embedded system used for the backpack design was based on its results in terms of frames per second (FPS) and higher accuracy in object classification. Considering the above, the Ultra96 FPGA obtained the best performance when running Inception-V1 with 30.3 fps and an 88.9 % top-5 accuracy trained with the ImageNet database. Besides its results compared to those of the Jetson TX2, an important aspect is the size of the integrated system. The Ultra96 measures 8.5 cm x 5.4 cm, while the Jetson TX2 measures 17 cm x 17 cm.

Other authors [32] have proposed the implementation of a backpack for NAO that contains an ODROID XU4 running object classification algorithm. Nevertheless, when it ran ORB-SLAM2, it achieved 12 fps, which is below the benchmark for real-time applications.

The execution of deep learning algorithms presented memory problems because the hardware was a Cortex™-A7 Octa-core microprocessor, which is sequential and is limited in its capabilities when it executes high computational cost algorithms such as CNNs.

Given the above, the proposed scheme integrates an external computational system that improves the visual perception of the NAO humanoid robot in tasks that require high performance, autonomy, and mobility. The following subsections present the details of the design.

2.7. Sizing the battery powering the embedded system

The battery sizing was based on the power required by the Ultra96 FPGA, the power consumed when executing a CNN and the autonomy in relation to the time the humanoid robot will be active. Regarding the last factor, the reference point in this study is [32], which reports that, despite the 60 minutes offered by the battery included in NAO, in the Robo Cup Soccer competition, the autonomy of the robot is only 30 minutes. Therefore, the operating time of the Ultra96 FPGA was estimated at 50 minutes of activity.

The input voltage of the Ultra96 is in the range from 8 V to 18 V, with a maximum current of 3 A. Although the FPGA only consumes 3.1 W in the inference stage, the power consumed is overestimated at 6 W. Taking these data into account, we proceed to calculate the power that should be delivered by the battery to be implemented based in (1), which is used to calculate the autonomy of a battery.

$$T = Wb / Wc \quad (1)$$

where T is the autonomy time in hours; Wb, the power delivered by the battery; and Wc, the power consumed. The estimated autonomy of 50 minutes in hours equals 0.833 h.

Considering the above, the power to be delivered by the battery is 4.998 W (Wb).

The next step was selecting the battery based on the calculation of the desired power delivery and autonomy. We selected a LiPo (lithium polymer) battery, which is a rechargeable battery composed of identical secondary cells arranged in parallel to increase the discharge current capacity. It offers a high discharge rate, light weight, and small size. These characteristics were suitable for the proposed scheme (i.e., a backpack for the robot) because they can provide autonomy to the robot without affecting its mobility.

Once the type of battery to be used was selected, a search for different LiPo batteries available in the market was carried out. Based on the fact that the power delivered by the battery must be 4.998 W, we selected a commercial LiPo battery that delivers this capacity and whose output voltage is within the input voltage range for the Ultra96 FPGA (i.e., 8 V–18 V). The chart in https://blog.ampow.com/lipo-battery-size-chart/, which details the electrical specifications of multiple LiPo batteries available in the market, was taken as a reference point. Since the delivered power must be 4.998 W and its dimensions are suitable for the application, a Lipo battery with 440 mAh of capacity and a voltage of 11.1 V was used here. The battery selection is based on the power calculation in (2):

$$P = V * I \tag{2}$$

where P is the power; V, the voltage; and I, the battery current. Assuming that the battery voltage is 11.1 V, and the current is 0.45 A, a power of 4.99 W is obtained.

After having the electrical specifications of the battery for the design, we searched for a commercial reference of a LiPo battery that met these power consumption needs. Although the electrical and size specifications of LiPo batteries are common among different manufacturers, the initial search retrieved the Gens Ace 450 mAh-11.1 V 3-cell battery, which was selected for this design.

3. RESULTS AND DISCUSSION

For the image acquisition and communication system, three different tests were performed varying the resolution of the acquired image and the encoding and decoding quality factor. In each test, we recorded data on the number of frames per second (fps) and the transmission rate measured in kbps (Kbits per second). Table 2 shows the results with different resolutions. It can be seen that, as the input image resolution increases, the frames per second decrease. However, the fps obtained by setting the image to the standard resolution of NAO's camera (i.e., 640x480) are much higher than those expected for a real-time application. By focusing the research work on the NAO humanoid robot, this resolution is taken as the entry point to the given data processing system for each heterogeneous architecture when running the CNN.

On the other hand, concerning CNN implementations in embedded systems, the results were obtained in frames per second and power consumption. To evaluate the performance of the object detection system implemented in FPGA, a CNN was employed in the PipeCNN framework using an Intel® FPGA board.

Table 2. Fps and Kbps in image transmission and reception. Source: Created by the authors.

Image resolution	Transmission		Reception	
	kbps	fps	kbps	fps
1920x1080 FullHD	91452,3	20	91121,1	19
640x480 VGA	10743,3	65	10534,6	63
224x224 ImageNet	11342,3	128	11134,6	123

The implemented CNN is a quantized version of AlexNet containing eight convolution layers executed on the FPGA. A result of 205 ms per image was obtained, which means that the FPGA can process 4.87 images per second. This implementation is not in real time because it is not in the range from 20 fps to 30 fps.

Comparing the results in frames per second, [33] obtained 66 fps when they implemented PipeCNN; [34], 70 fps when they accelerated a CNN using CUDA on a Jetson Tx2; [35], 864.7 fps when they implemented AlexNet on a Stratix-V; and [34], 11 fps, which is below what is required for a real-time application, when they implemented NCSDK on a Movidius.

Power consumption results were estimated with Intel Quartus® Prime Power Analyzer software. The maximum number of computing units that can be implemented on the Cyclone V SoC is four. It can be seen from Table 3 that the resources used in this design are below the total numbers contained in the board. This occurs because the board contains only 4192 LAB that are maxed out in this implementation. For this reason, it is not possible to increase the performance of the framework on the Cyclone V-SoC, unlike the implementation performed by [33] on the DE5-net platform, which does achieve real-time ranking since the platform is much larger in terms of resources.

Table 3. PipeCNN framework resources in the Cyclone V SoC Development Kit
Source: Created by the authors.

PipeCNN			
Resources	Used	Available	Utilization percentage
LUT	48173	110000	43.79 %
FF	66830	219144	30.50 %
RAM	285	514	55.45 %
DSP	35	112	31.25 %
Power consumption		2.056 W	

For the second framework, the image was acquired with a high-resolution webcam in real time, which was done with the help of the OpenCV libraries installed by default in the PYNQ framework. In Figure 5, the top images show the humanoid robot NAO interacting in a virtual environment created by Webots. The bottom images show the object position boxes and the prediction of identified objects in the embedded systems. Figure 6 presents the detection of the same objects created in the virtual setting (i.e., a chair and a person) but in a real controlled environment.

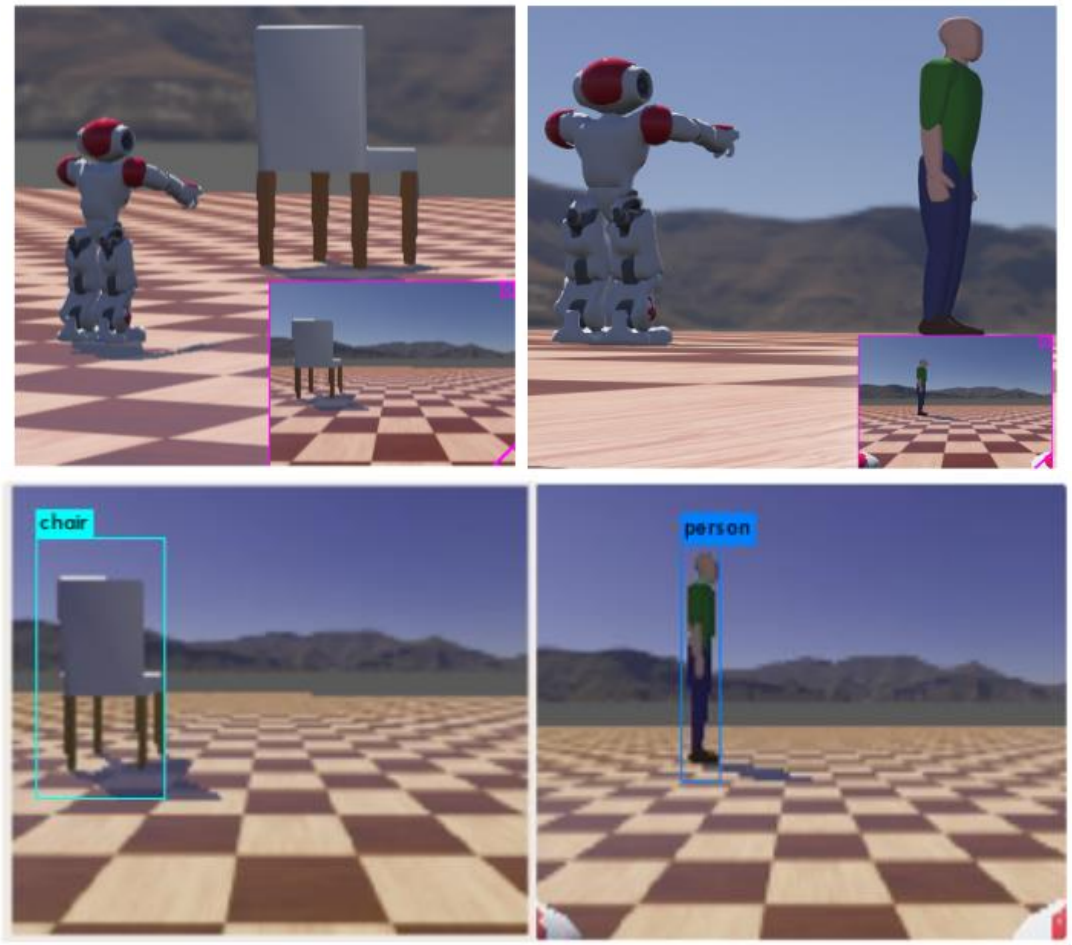

Figure 5. Top: NAO interacting in the virtual environment. Bottom: chair and person detection on embedded systems. Source: Created by the authors.

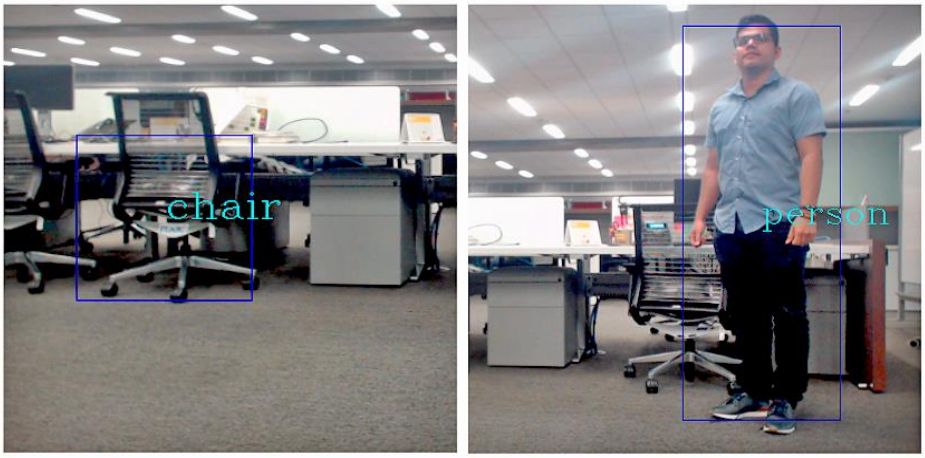

Figure 6. Detection of objects in a real environment. Source: Created by the authors.

As a result, on the Ultra96, we obtained an execution time of 83 ms, which is equivalent to 12 frames per second and still below real-time execution. In comparison, in [19], 5 fps were obtained by implementing Tiny-YOLO on the Jetson Tx2, while, in [20], 21 fps were achieved by running the CNN on a VC707 FPGA. The results show that implementations of this same

CNN on different platforms present higher fps on the Tiny-YOLO executed on the Ultra96 than on the Jetson Tx2, but a worse performance than another FPGA implementation.

The amount of resources used in the Ultra96 is the same for any implementation since the MO architecture is topology-independent. It can be seen in Table 4 that the BRAM is at more than 90 % of the available resources on the board, but the utilization of the rest of the resources does not exceed 50 %. This is expected since FPGAs have little internal memory storage capacity.

Table 4. Resources used in the implementation of the QNN framework on the Ultra96. Source: Created by the authors.

FINN Framework			
Resources	Used	Available	Utilization percentage
LUT	29754	70560	42.17 %
Register	36050	141120	25.55 %
Block RAM	200	216	92.59 %
DSP	56	360	15.56 %
Power consumption		3.1 W	

In the application explored here, i.e., classification of toys for early childhood education, the modified implementation of the Inception-V3 network achieved a 98 % classification rate and a real-time performance of 27 fps. This architecture was implemented at a clock frequency of 370 MHz using the DPU. The SGD optimizer was used with a learning-rate of 0.0001 and a momentum value of 0.9; category cross-entropy was employed as the cost function. For the training with transfer-learning, the first 172 convolutional layers of the network are frozen, and the output layer of Inception-V3 is a softmax activation with 1000 classes. In this study, this layer is removed and replaced with a 10-class softmax layer, whereby this layer is trained together with the remaining dense layers for this application, which is focused on the classification of 10 types of toys. The training result obtained in Keras was converted to the DNNDK framework using the DECENT and DNNC tools, in which the elf file containing all the CNN information (weights and architecture) was created. During the inference, this file was read by the ARM of the FPGA SoC, which was in charge of sending the tasks to the DPU to process and the returned result. The resources used in the Ultra96 for the QNN framework, and the IP DPU are shown in Table 5.

Table 5. QNN framework and IP DPU Ultra96 FPGA resources. Source: Created by the authors.

Resources	QNN	DPU	Available
LUT	29754	37055	70560
Register	36050	72850	141120
Block RAM	200	161,5	216
DSP	56	290	360
Power consumption	3.1 W	-	-

Regarding the implementation of classical machine learning algorithms for early childhood education applications, a DELL computer with a 1.6-GHZ ATOM processor and 1 GB of RAM was used for the inference stage of each of the classical classifiers implemented here. Table 6 presents the results of these classifiers in terms of accuracy and training time in seconds. For this application, each pixel is used as a feature; thus, each classification model

has 200x200x3 features. Table 6 indicates that, by training each of the classical classifiers, an accuracy between 70 % and 85 % was obtained in the validation set.

Table 6. Accuracy and training time of classical classifiers. Source: Created by the authors.

Model	Accuracy (%)	Training time (s)
Naive Bayes	0.54	0.02
Logit	0.84	4.96
Decision Tree	0.76	0.59
Random Forest	0.81	2.97

Table 7 compares the performance of CNN-FPGA/GPU and classical algorithms run on the sequential system of the NAO humanoid robot. For the inference tests, five random images were taken from the test database. It can be seen that running the Inception-V1 and Inception-V3 transfer-learning architectures yields real-time results. The opposite case occurs with the remaining implementations, where the frames per second were well below those established for a real-time application. In the case of the Tinier-YOLO CNN executed on the FPGA and GPU, despite the quantification of its weights, lower fps was obtained, but the FPGA was able to process more images per second than the GPU for this convolutional neural network architecture. On the other hand, the sequential system of the NAO humanoid robot has processing limitations when classical computer vision techniques are executed.

Despite the fact that they are much lighter models, the maximum that the NAO CPU could process was 3.95 fps with Decision Tree. This limitation arises because each pixel is used as a feature by each classification model, which has 200x200x3 features (input image size). Finally, since CNNs are larger architectures, the NAO processing system would be restricted in computing each of the layers and weights contained in a CNN.

Table 7 shows that, when it performs the inference, the embedded system takes 0.037 seconds, which is equivalent to 27 fps.

Table 7. Comparison of execution times and fps of a CNN (on two FPGAs and one GPU) vs. classical techniques (on an Intel® Atom Z530 CPU). Source: Created by the authors.

Hardware type	Development board	Architecture	Model	Execution time (s)	Frame rate (fps)
FPGA	Cyclone V-SoC	CNN	AlexNet	0.200	4.87
	Ultra96		Tinier-YOLO	0.083	12.04
			Inception-V1	0.033	30.30
			Inception-V3 transfer learning	0.037	27.02
GPU	Jetson TX2		Tinier-YOLO	0.100	9.86
			Inception-V1	0.035	28.50
			AlexNet	0.035	28.50
CPU	Intel Atom Z530	Classical techniques	Decision Tree	0.253	3.95
			Logit	0.504	1.98
			Naive Bayes	9	0.0001
			Random Forest	0.316	3.16

4. CONCLUSIONS

In this paper, we presented a system to improve the visual perception in humanoid robots in autonomous applications by integrating heterogeneous architectures based on a GPU or a FPGA running a CNN. Three CNNs (Alexnet, Inception-V1, and Tinier-YOLO) were used to select the architecture in the final implementation. We evaluated the performance of the Xilinx Ultra96-V2 FPGA, Intel Cyclone V SoC FPGA, and Nvidia Jetson TX2 GPU platforms when running these CNN models. The Ultra96 achieved 12 fps when running Tinier-YOLO at 3.1 W, while the Jetson TX2 achieved 9.86 fps and a power consumption of 7 W; their input was an image with a size of 640x480 pixels, which is the standard resolution of the NAO humanoid robot's camera. In our application on the Inception-V1 CNN, real-time results were obtained with heterogeneous architectures on two FPGAs (i.e., Intel Cyclone V SoC and Ultra96-V2), while the expected behavior was obtained when AlexNet was run on the Jetson TX2. In terms of accuracy, the Inception V1 network presents the best performance (88.9 %) and low resource consumption when implemented on the Ultra96. On the other hand, in Table 7, it can be observed that the available resources are sufficient when CNN is implemented, opening the possibility of implementing larger network architectures. Considering the above and the results in terms of execution time and power consumption, the Ultra96 FPGA was selected for the design of the backpack for NAO.

Despite the implementations of deep learning models in the conventional computational system of humanoid robots reviewed in the introduction, processing times during object recognition are affected by the high computational cost required by deep learning models such as CNNs. The integration of a CNN and a heterogeneous FPGA- or GPU-based architecture in humanoid robots can provide these automatons with real-time visual perception enhancement that can be exploited in human-robot interaction applications. In this study, we solved three problems: (1) reducing the execution time and improving the accuracy of object detection in an everyday environment while maintaining the autonomy of the robot; (2) creating a system that can be replicable to humanoid robots such as Pepper and Robotis OP3; and (3) producing a development that can be used to integrate heterogeneous architectures with a high degree of parallelism, low power consumption and small size that execute a CNN and can be easily integrated into the structure of any humanoid robot using a backpack.

In this study, simulation tools were used to prototype a virtual environment—that includes a humanoid NAO robot and objects around it—and then send the image to each of the embedded systems to perform object classification or detection. One point to consider is that virtual environments do not consider all the possible factors that in real environments may affect the performance of the system. However, in this application, other tests were carried out in real environments, obtaining a good performance. Considering the above, in future work, we will implement the system developed here in real conditions considering other external factors and a real-life application. We also aim to examine the quantification of convolutional neural networks in greater depth. Since CNNs are computationally expensive, quantization is an open research topic because the size of the network is expected to be reduced considerably without losing accuracy. With this reduction in size, CNNs are expected to be more easily implemented in embedded systems that can be integrated into humanoid robots such as NAO. Furthermore, NAO's depth sensor could be used to calculate the actual distance of the object once it has been detected by the CNN. This could extend the robot's capabilities in human–robot interaction tasks.

5. ACKNOWLEDGMENTS

This study was supported by the Automática, Electrónica y Ciencias Computacionales (AE&CC) research group (Minciencias code: COL0053581) at the Control Systems and Robotics Laboratory of the Instituto Tecnológico Metropolitano (ITM) in Medellín. This article is part of the project called Improving visual perception in humanoid robots for object recognition in natural environments using Deep Learning, which was funded by ITM in association with Universidad de Antioquia (project code: P17224).

CONFLICTS OF INTEREST

The authors declare that there is no conflict of interest.

AUTHOR CONTRIBUTIONS

Joaquin Guajo: conceptualization, methodology, validation, formal analysis, investigation, original draft preparation, review, and editing

Cristian Alzate-Anzola: methodology, validation, original draft preparation

Luis Castaño-Londoño: conceptualization, methodology, validation, investigation, review and editing, funding acquisition,

David Márquez-Viloria: conceptualization, methodology, validation, investigation, review and editing, funding acquisition.

6. REFERENCES

[1] S. R. Fanello; C. Ciliberto; N. Noceti, G. Metta; F. Odone, "Visual recognition for humanoid robots", *Rob. Auton Syst.*, vol. 91, pp. 151–168, May 2017. https://doi.org/10.1016/j.robot.2016.10.001

[2] E. Cha; M. Mataric; T. Fong, "Nonverbal signaling for non-humanoid robots during human-robot collaboration", in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2016, pp. 601–602. https://doi.org/10.1109/HRI.2016.7451876

[3] S. Shamsuddin *et al.*, "Initial response of autistic children in human-robot interaction therapy with humanoid robot NAO", in *2012 IEEE 8th International Colloquium on Signal Processing and its Applications*, 2012, pp. 188–193. https://doi.org/10.1109/CSPA.2012.6194716

[4] J. G. Hoyos-Gutiérrez; C. A. Peña-Solórzano; C. L. Garzón-Castro; F. A. Prieto-Ortiz; J. G. Ayala-Garzón, "Hacia el manejo de una herramienta por un robot NAO usando programación por demostración", *TecnoLógicas*, vol. 17, no. 33, pp. 65-76, Aug. 2014. https://doi.org/10.22430/22565337.555

[5] P. Vadakkepat; N. B. Sin; D. Goswami; R. X. Zhang; L. Y. Tan, "Soccer playing humanoid robots: Processing architecture, gait generation and vision system", *Rob. Auton. Syst.*, vol. 57, no. 8, pp. 776–785, Jul. 2009. https://doi.org/10.1016/j.robot.2009.03.012

[6] A. Härtl; U. Visser; T. Röfer, "Robust and Efficient Object Recognition for a Humanoid Soccer Robot", Springer, Berlin, Heidelberg, 2014, pp. 396–407. https://doi.org/10.1007/978-3-662-44468-9_35

[7] D. Budden; S. Fenn; J. Walker; A. Mendes, "A Novel Approach to Ball Detection for Humanoid Robot Soccer", Springer, Berlín, Heidelberg, 2012, pp. 827–838. https://doi.org/10.1007/978-3-642-35101-3_70

[8] P. Sermanet; D. Eigen; X. Zhang; M. Mathieu; R. Fergus; Y. Le Cun, "Integrated recognition, localization and detection using convolutional networks", 2013. https://arxiv.org/abs/1312.6229

[9] A. Krizhevsk; I. Sutskever; G. E. Hinton, "ImageNet classification with deep convolutional neural networks", *Commun. ACM*, vol. 60, no. 6, pp. 84–90, Jun 2017. https://doi.org/10.1145/3065386

[10] K. Simonyan; A. Zisserman, "Very deep convolutional networks for large-scale image recognition", arXiv preprint, Sep.2015. https://arxiv.org/abs/1409.1556

[11] K. He; X. Zhang; S. Ren; J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. https://doi.org/10.1109/CVPR.2016.90

[12] H. V. Nguyen; H. T. Ho; V. M. Patel; R. Chellappa, "DASH-N: Joint Hierarchical Domain Adaptation and Feature Learning," *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 5479–5491, Dec. 2015. https://doi.org/10.1109/TIP.2015.2479405

[13] M. Podpora; A. Gardecki, "Extending vision understanding capabilities of NAO robot by connecting it to a remote computational resource," in *2016 Progress in Applied Electrical Engineering (PAEE)*, 2016, pp. 1–5. https://doi.org/10.1109/PAEE.2016.7605119

[14] M. Puheim; M. Bundzel; L. Madarasz, "Forward control of robotic arm using the information from stereo-vision tracking system", in *2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI)*, 2013, pp. 57–62. https://doi.org/10.1109/CINTI.2013.6705259

[15] K. Noda; H. Arie; Y. Suga; T. Ogata, "Multimodal integration learning of robot behavior using deep neural networks", *Rob. Auton. Syst.*, vol. 62, no. 6, pp. 721–736, Jun. 2014. https://doi.org/10.1016/j.robot.2014.03.003

[16] A. Biddulph; T. Houliston; A. Mendes; S. K. Chalup, "Comparing Computing Platforms for Deep Learning on a Humanoid Robot", Springer, Cham, 2018. https://doi.org/10.1007/978-3-030-04239-4_11

[17] A. Dundar; J. Jin; B. Martini; E. Culurciello, "Embedded Streaming Deep Neural Networks Accelerator with Applications," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 28, no. 7, pp. 1572–1583, Jul. 2017. https://doi.org/10.1109/TNNLS.2016.2545298

[18] H. Park *et al.*, "Optimizing DCNN FPGA accelerator design for handwritten hangul character recognition", in *Proceedings of the 2017 International Conference on Compilers, Architectures and Synthesis for Embedded Systems Companion*, 2017, pp. 1–2. https://doi.org/10.1145/3125501.3125522

[19] C. Zhang; P. Li; G. Sun; Y. Guan; B. Xiao; J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks", in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 161–170. https://doi.org/10.1145/2684746.2689060

[20] Q. Xiao; Y. Liang; L. Lu; S. Yan; Y.-W. Tai, "Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs", in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6. https://doi.org/10.1145/3061639.3062244

[21] E. Del Sozzo; A. Solazzo; A. Miele; M. D. Santambrogio, "On the Automation of High Level Synthesis of Convolutional Neural Networks", in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 217–224. https://doi.org/10.1109/IPDPSW.2016.153

[22] C. Zhang; G. Sun; Z. Fang; P. Zhou; P. Pan; J. Cong, "Caffeine: Toward Uniformed Representation and Acceleration for Deep Convolutional Neural Networks", in *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 38, no. 11, pp. 2072–2085, Nov. 2019. https://doi.org/10.1109/TCAD.2017.2785257

[23] R. Andri; L. Cavigelli; D. Rossi; L. Benini, "YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights", in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 236–241. https://doi.org/10.1109/ISVLSI.2016.111

[24] L. Ni; Z. Liu; H. Yu; R. V. Joshi, "An Energy-Efficient Digital ReRAM-Crossbar-Based CNN With Bitwise Parallelism", *IEEE J. Explor. Solid-State Comput. Devices Circuits*, vol. 3, pp. 37–46, Dec. 2017. https://doi.org/10.1109/JXCDC.2017.2697910

[25] A. Kulkarni; T. Abtahi; C. Shea; A. Kulkarni; T. Mohsenin, "PACENet: Energy efficient acceleration for convolutional network on embedded platform", in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4. https://doi.org/10.1109/ISCAS.2017.8050342

[26] T. Gong; T. Fan; J. Guo; Z. Cai, "GPU-based parallel optimization of immune convolutional neural network and embedded system", *Eng. Appl. Artif. Intell.*, vol. 62, pp. 384–395, Jun. 2017. https://doi.org/10.1016/j.engappai.2016.08.019

[27] D. Strigl; K. Kofler; S. Podlipnig, "Performance and Scalability of GPU-Based Convolutional Neural Networks", in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010, pp. 317–324. https://doi.org/10.1109/PDP.2010.43

[28] O. Michel, "Cyberbotics Ltd. Webots ™: Professional Mobile Robot Simulation", *Int. J. Adv. Robot. Syst.*, vol. 1, no. 1, p. 39-42, Mar. 2004. https://doi.org/10.5772/5618

[29] E. Pot; J. Monceaux; R. Gelin; B. Maisonnier, "Choregraphe: a graphical tool for humanoid robot programming", in *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, 2009, pp. 46–51. https://doi.org/10.1109/ROMAN.2009.5326209

[30] M. Blott *et al.*, "FINN- R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks", *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–23, Sep. 2018. https://doi.org/10.1145/3242897

[31] S. Wang; S. Jiang, "INSTRE: A New Benchmark for Instance-Level Object Retrieval and Recognition", *ACM Trans. Multimed. Comput. Commun. Appl.*, vol. 11, no. 3, pp. 1–21, Feb. 2015.

https://doi.org/10.1145/2700292

[32] M. Mattamala; G. Olave; C. González; N. Hasbún; J. Ruiz-del-Solar, "The NAO Backpack: An Open-Hardware Add-on for Fast Software Development with the NAO Robot", 2018, pp. 302–311. https://doi.org/10.1007/978-3-030-00308-1_25

[33] D. Wang; K. Xu; D. Jiang, "PipeCNN: An OpenCL-based open-source FPGA accelerator for convolution neural networks", in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 279–282. https://doi.org/10.1109/FPT.2017.8280160

[34] S. Xu; A. Savvaris; S. He; H. Shin; A. Tsourdos, "Real-time Implementation of YOLO+JPDA for Small Scale UAV Multiple Object Tracking," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 1336–1341. https://doi.org/10.1109/ICUAS.2018.8453398

[35] J. Ma; L. Chen; Z. Gao, "Hardware Implementation and Optimization of Tiny-YOLO Network", Springer, Singapur, 2018, pp. 224–234, https://doi.org/10.1007/978-981-10-8108-8_21