# Evaluación de desempeño de un sistema de reconocimiento de objetos basado en deep learning sobre el robot humanoide NAO implementado con arquitecturas heterogéneas basadas en FPGA o GPU

José Joaquin Guajo Trujillo

Instituto Tecnológico Metropolitano

# EVALUACIÓN DE DESEMPEÑO DE UN SISTEMA DE RECONOCIMIENTO DE OBJETOS BASADO EN DEEP LEARNING SOBRE EL ROBOT HUMANOIDE NAO IMPLEMENTADO CON ARQUITECTURAS HETEROGÉNEAS BASADAS EN FPGA O GPU

**José Joaquin Guajo Trujillo**

Tesis de Maestría presentada como requisito para optar al título de

**Magister**
en Automatización y Control Industrial

*Instituto Tecnólogico Metropolitano*
*Facultad de Ingeniería*
Medellín, 2021

Directores: MSc. L. F. Castaño Londoño
MSc. D. A. Márquez Viloria

Institución Universitaria

# AGRADECIMIENTOS

Agradezco a los directores de esta tesis, el profesor Luis Fernando Castaño y el profesor David Márquez Viloria por sus orientaciones y apoyo constante para la realización de la actual tesis de investigación. Quiero expresar mis agradecimientos al grupo y laboratorio Sistemas de Control y Robótica de Parque i del ITM por su acompañamiento y recursos que permitieron un trabajo continuo en la maestría.

Un agradecimiento a todos los profesores de la maestría que mediante sus clases me orientaron al desarrollo de la misma. Finalmente, agradecerle a mis compañeros de clases por su amistad y apoyo durante este tiempo.

# ÍNDICE GENERAL

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# LISTA DE ACRÓNIMOS

| | |
|---|---|
| **ARM** | Advanced RISC Machine |
| **BRAM** | Block RAM |
| **CCCNN** | Multichannel Convolutional Neural Network |
| **CNN** | Convolutional Neural Network |
| **CNN SDRAM** | Single Data Rate Synchronous Dynamic Random-Access Memory |
| **CPU** | Central Processing Unit |
| **cuBLAS** | Cuda Basic Linear Algebra Subprograms |
| **CUDA** | Compute Unified Device Arquitectur |
| **DPU** | Deep Learning Processing |
| **DMA** | Direct Memory Access |
| **FPGA** | Field Programmable Gate Arrays |
| **Fps** | Frames Per Second |
| **GPU** | Graphics Processing Unit |
| **HLS** | High-Level Synthesis |
| **HPS** | Hard Processor System |
| **HRI** | Human Robot Interaction |
| **IP** | Core Intellectual Property Core |
| **ISP** | Image Signal Processor |
| **Kbps** | Kilobit Por Segundo |
| **LiPo** | Lithium Polymer |
| **LUT** | Look up table |
| **mAP** | Mean Average Precision |
| **MCCNNN** | Multichannel Convolutional Neural Network |
| **NN** | Neural Networks |
| **PACENet** | Programmable Many-Core Accelerator Network |
| **ReRam** | Resistive Random-Access Memory |
| **ResNet** | Residual Network |
| **RTL** | Register-Transfer Level |
| **SDK** | Software Development Kit |
| **VGGNet** | Visual Geometry Group Network |

# INTRODUCCIÓN

Los robots humanoides son utilizados en aplicaciones de tipo asistencial relacionadas con actividades que involucran servicios domésticos, educativos, terapéuticos, entre otros [4]. Este uso surge de las sensaciones de confort generadas en los seres humanos en la interacción con este tipo de robots [2]. Para el desarrollo de estas interacciones cuentan con dispositivos sensóricos que le proporcionan capacidad de percepción y comprensión del entorno. En particular, el sistema de percepción visual de los robots humanoides está compuesto por cámaras integradas en su estructura, las cuales le otorgan un campo de visión [14]. Este campo de visión permite que sean utilizados en aplicaciones orientadas al reconocimiento de objetos en tiempo real [3].

La imagen del objeto a reconocer es capturada por parte del sistema de percepción visual y es tratada a través de un procesador, en el cual, la ejecución se realiza secuencialmente. Este procesamiento secuencial de los píxeles de la imagen producen una latencia y por lo tanto un retraso en el tiempo de ejecución en el sistema de cómputo del robot humanoide, limitando la capacidad de respuesta del robot en la toma de decisiones para labores que involucran reconocimiento de objetos [16]. Esta limitación es mostrada en la implementación de técnicas clásicas de visión por computador por parte de [1, 5].

Sobre el sistema de cómputo secuencial con el que cuenta el robot humanoide NAO, se han introducido algoritmos de aprendizaje profundo. Estos son modelos computacionales que replican la capacidad cognitiva humana y contribuyen al uso práctico de los robots en entornos cotidianos [10]. Dentro de estos algoritmos se hallan las CNNs, las cuales han demostrado ser relevantes en proyectos donde se integra el reconocimiento, la localización y la detección de objetos, logrando ser eficientes en términos de precisión en la elaboración de estas tareas [13]. Diferentes arquitecturas de CNNs han sido desarrolladas, se pueden encontrar la red AlexNet, VGGNet, ResNet, LeNet, entre otras [6–8, 15]. Estas redes fueron entrenadas a partir de millones de imágenes, garantizando un buen desempeño en el reconocimiento de objetos para distintos entornos cotidianos [9].

A pesar de sus ventajas, las CNNs presentan un alto costo computacional al ser implementadas sobre el sistema de cómputo secuencial del robot humanoide. Por esta razón, se han realizado desarrollos sobre sistemas computacionales externos conectados a la arquitectura de cómputo del robot, generalmente a través de una conexión Ethernet [11]. Los trabajos propuestos por [10, 12] concluyeron que las limitaciones en términos de tiempos de ejecución durante el procesamiento de imágenes fueron superadas, sin embargo, la autonomía del robot se vio afectada por la continua conexión, generando dependencia en tareas que requieren un desenvolvimiento libre en su entorno.

Las limitaciones presentadas en cuanto a capacidad de respuesta y autonomía de los robots humanoides con arquitecturas computacionales de operación secuencial, presentan una oportunidad en la implementación de CNNs sobre sistemas embebidos basados en FPGA o GPU. Estos dispositivos presentan alto grado de paralelismo y bajo consumo de potencia en aplicaciones relacionadas con el procesamiento de imágenes y video, lo que podría otorgarle al robot humanoide una mayor capacidad de procesamiento y autonomía en labores que involucran una mayor percepción y comprensión del entorno.

Aunque se han propuesto diferentes implementaciones de CNNs sobre sistemas embebidos basados en FPGA o GPU, su uso en la robótica humanoide presenta un tema abierto de investigación al integrar y evaluar el desempeño de un sistema externo compuesto por una arquitectura heterogénea basada en FPGA o GPU que ejecuta una CNN para aplicaciones relacionadas a la HRI.

## SÍNTESIS DEL PROBLEMA

A pesar de las ventajas que han mostrado las CNNs en aplicaciones relacionadas con el reconocimiento de objetos, su implementación en la robótica humanoide se ve limitada por la capacidad de procesamiento reducida en términos de tiempos de ejecución y autonomía que presenta el sistema de cómputo secuencial de robots humanoides como NAO. Por esta razón, este proyecto se orienta a evaluar el desempeño del hardware en el reconocimiento de objetos del robot humanoide NAO para entornos cotidianos integrando una CNN sobre sistemas embebidos de alto cómputo y autonomía como los FPGA y GPU.

### PREGUNTA DE INVESTIGACIÓN

¿Cuál sería el efecto en el desempeño del hardware en el reconocimiento de objetos del robot humanoide NAO en entornos cotidianos si se le integra arquitecturas heterogéneas basadas en FPGA y GPU para acelerar la ejecución de una red neuronal convolucional?.

### HIPÓTESIS

El sistema de reconocimiento de objetos de los robots humanoides está limitado por la capacidad de procesador secuencial incorporado, el cual no cuenta con una arquitectura de buen desempeño en el procesamiento imágenes y video. Sin embargo, una arquitectura heterogénea basada en FPGA o GPU posee un alto grado de paralelismo que es útil en la aceleración de algoritmos para procesamiento de imágenes y video, por lo que la implementación de una CNN sobre estas arquitecturas mejorará el tiempo de respuesta y autonomía del robot en el reconocimiento de objetos en entornos cotidianos.

### OBJETIVO GENERAL

Evaluar el desempeño alcanzado por el hardware de un sistema de reconocimiento de objetos para un robot NAO implementado con arquitecturas heterogéneas basadas en FPGA o GPU para la aceleración de una CNN.

### OBJETIVOS ESPECÍFICOS

1. Diseñar el esquema de adquisición, comunicación y procesamiento de datos de un sistema de reconocimiento de objetos basado en CNN para el robot NAO usando arquitecturas heterogéneas basadas en FPGA y GPU.

2. Implementar sobre cada arquitectura heterogénea la aceleración de una CNN y adaptar cada arquitectura de manera independiente al sistema de reconocimiento de objetos diseñado.

3. Comparar el desempeño de las implementaciones sobre las arquitecturas heterogéneas propuestas y el sistema de reconocimiento de objetos de NAO usando metodologías de evaluación de desempeño de hardware.

## CONTRIBUCIÓN

En esta tesis se presenta el diseño de un esquema para la integración de un robot NAO con sistemas embebidos basados en FPGA y GPU que ejecutan una CNN, con el fin de lograr el mejoramiento de la percepción visual del humanoide. Este esquema, se basó en el uso de herramientas de software que posteriormente se conectan a los sistemas embebidos a través de una conexión Ethernet. Para cada uno de los sistemas utilizados se realiza la evaluación de desempeño teniendo en cuenta métricas como la precisón de la CNN, tiempo de inferencia y consumo de potencia. A continuación, se detallan los aportes de la actual tesis de investigación.

- Se evalúan diferentes *frameworks* de aceleración de CNN sobre sistemas basados en FPGA o GPU, lo que abre la posibilidad a hacer uso de diferentes tarjetas de desarrollo. En este trabajo fue utilizado Vitis AI, el cual es el framework mas reciente de Xilinx para la aceleración de CNNs.

- Se presenta una solución a través de entornos virtuales para robótica, generando una metodología para aplicaciones enfocadas en el reconocimiento de objetos para robótica humanoide través de herramientas de simulación como Webots y la integración de arquitecturas heterogéneas.

- El sistema presentado aporta un análisis, procedimiento y resultados obtenidos a partir de la integración de diferentes herramientas de software y sistemas computacionales. Lo anterior proporciona una guía u orientación a investigadores que se centran en el desarrollo de aplicaciones enfocadas en el mejoramiento de la percepción visual en robots humanoides.

- La integración de estos sistemas embebidos de alto grado de paralelismo y bajo consumo de potencia resulta ser una solución que presenta un equilibrio entre rendimiento y costo si se compara con el proceso de actualización de software y hardware del robot humanoide NAO.

- Se comparan dos arquitecturas diferentes de aceleración por hardware, FPGA y GPU. Se obtienen medidas de desempeño en términos de tiempo de inferencia y consumo de potencia cuando se ejecutan redes neuronales profundas como las CNN. Lo anterior, permite a investigadores del área o afines tener un punto de referencia de desempeño que puede adaptarse a su aplicación.
- Se presenta la implementación de diferentes arquitecturas de CNN, permitiendo que el robot pueda interactuar en distintos ambientes y reconocer diferentes clases de objetos. De acuerdo a lo anterior, se proporcionan algunas bases para incrementar el desempeño en tareas que involucran una interacción humano-robot a partir del mejoramiento de la percepción visual del robot humanoide NAO.
- El trabajo presenta el análisis y comparación de desempeño de diferentes plataformas de hardware para el procesamiento de datos e imágenes en robótica bípeda, en el cual se obtienen resultados en tiempo real, y en donde se conserva la movilidad, autonomía y mejora su capacidad de percepción visual en entornos cotidianos.
- Se presenta una solución enfocada en educación inicial a través del reconocimiento de juguetes. Mediante un proceso de transfer-learning y haciendo uso del dataset Instre, se crea una aplicación para la detección de objetos infantiles que puede ser aprovechada en tareas HRI en entornos infantiles o tratamiento de niños autistas. Se ha demostrado que el uso de robots humanoides como NAO en estos procedimientos, mejora las habilidades comunicativas de estos niños.
- Se realiza la implementación de diferentes algoritmos clásicos sobre un sistema de cómputo similar al del robot humanoide NAO. Los resultados demostraron la limitación que posee su unidad de cómputo al ejecutar estos algoritmos, los cuales son menos costosos computacionalmente si se comparan con las CNN.

*Aportes al estado del arte*

- Guajo, J., Anzola, C. A., Betancur, D., Castaño-Londoño, L., & Marquez-Viloria, D. (2021, October). Improvement of Visual Perception in Humanoid Robots Using Heterogeneous Architectures for Autonomous Applications. In Workshop on Engineering Applications (pp. 447-458). Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-86702-7_38
- Guajo, J., Anzola, C. A., Betancur, D., Castaño-Londoño, L., & Marquez-Viloria, D. Performance evaluation of convolutional networks on heterogeneous architectures for applications in autonomous robotics (Aceptado - Revista TecnoLógicas).

### ESTRUCTURA GENERAL DE LA TESIS

Este trabajo esta organizado a través de 4 capítulos. El **Capítulo 1** presenta una descripción del estado del arte y marco teórico. En el estado del arte, se detallan las implementaciones de CNNs sobre sistemas embebidos basados en FPGA o GPU, sistemas de percepción visual realizadas sobre el robot humanoide NAO. Respecto al marco teórico se explican los concepto fundamentales que guían la actual de tesis de investigación. En el **Capítulo 2** se presenta el marco experimental en el que se describe la metodología propuesta implementada para la ejecución de CNNs sobre sistemas embebidos basados en FPGA o GPU aplicado a robótica humanoide. En el **Capítulo 3** se describe y discuten los resultados obtenidos. Finalmente, en el **Capítulo 4** se presentan las conclusiones con respecto a los resultados alanzados y los posibles trabajos futuros que pueden seguir la linea de este proyecto.

## REFERENCIAS

[1] David Budden, Shannon Fenn, Josiah Walker, and Alexandre Mendes. A novel approach to ball detection for humanoid robot soccer. In *Australasian Joint Conference on Artificial Intelligence*, pages 827–838. Springer, 2012.

[2] Elizabeth Cha, Maja Matarić, and Terrence Fong. Nonverbal signaling for non-humanoid robots during human-robot collaboration. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, pages 601–602. IEEE Press, 2016.

[3] Sean Fanello, Carlo Ciliberto, Matteo Santoro, Lorenzo Natale, Giorgio Metta, Lorenzo Rosasco, and Francesca Odone. icub world: Friendly robots help building good vision data-sets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 700–705, 2013.

[4] Sean Ryan Fanello, Carlo Ciliberto, Nicoletta Noceti, Giorgio Metta, and Francesca Odone. Visual recognition for humanoid robots. *Robotics and Autonomous Systems*, 91:151–168, 2017.

[5] Alexander Härtl, Ubbo Visser, and Thomas Röfer. Robust and efficient object recognition for a humanoid soccer robot. In *Robot Soccer World Cup*, pages 396–407. Springer, 2013.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[8] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20, 2015.

[9] Hien V Nguyen, Huy Tho Ho, Vishal M Patel, and Rama Chellappa. Dash-n: Joint hierarchical domain adaptation and feature learning. *IEEE Transactions on Image Processing*, 24(12):5479–5491, 2015.

[10] Kuniaki Noda, Hiroaki Arie, Yuki Suga, and Tetsuya Ogata. Multimodal integration learning of robot behavior using deep neural networks. *Robotics and Autonomous Systems*, 62(6):721–736, 2014.

[11] Michal Podpora and Arkadiusz Gardecki. Extending vision understanding capabilities of nao robot by connecting it to a remote computational resource. In *Progress in Applied Electrical Engineering (PAEE)*, pages 1–5. IEEE, 2016.

[12] Michal Puheim, Marek Bundzel, and Ladislav Madarász. Forward control of robotic arm using the information from stereo-vision tracking system. In *Computational Intelligence and Informatics (CINTI), 2013 IEEE 14th International Symposium on*, pages 57–62. IEEE, 2013.

[13] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[14] Syamimi Shamsuddin, Hanafiah Yussof, Luthffi Ismail, Fazah Akhtar Hanapiah, Salina Mohamed, Hanizah Ali Piah, and Nur Ismarrubie Zahari. Initial response of autistic children in human-robot interaction therapy with humanoid robot nao. In *Signal Processing and its Applications (CSPA), 2012 IEEE 8th International Colloquium on*, pages 188–193. IEEE, 2012.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[16] Prahlad Vadakkepat, Ng Buck Sin, Dip Goswami, Rui Xiang Zhang, and Li Yu Tan. Soccer playing humanoid robots: Processing architecture, gait generation and vision system. *Robotics and Autonomous Systems*, 57(8):776–785, 2009.

# 1

# ESTADO DEL ARTE

El siguiente capítulo aborda el estado del arte. Se realiza una descripción de trabajos recientes enfocados en la implementaciones de CNN sobre sistemas embebidos basados en FPGA o GPU. Además, se presentan investigaciones enfocados en la implementaciones de CNN sobre el sistema de cómputo del robot humanoide NAO. Por último, se presentan trabajos centrados en la implementación de algoritmos de reconocimiento de imágenes sobre sistemas computacionales externos que se conectan a robots humanoides como NAO.

## 1.1. ESTADO DEL ARTE

### 1.1.1. IMPLEMENTACIONES DE CNN SOBRE SISTEMAS EMBEBIDOS BASADOS EN GPU

Las GPUs son procesadores de amplio recurso computacional que exigen un alto grado de paralelismo, logrando un buen rendimiento y eficiencia [1]. En los últimos años el rendimiento de la GPU ha ido mejorando a un ritmo mucho más rápido que el desempeño de las CPU a través de plataformas de computación paralela como CUDA que hace que los usuarios desarrollen programas generales de una GPU fácilmente [2]. La arquitectura de hardware de CUDA contiene una memoria global de escritura/lectura general, en el cual, la GPU puede reunir datos desde cualquier ubicación a la memoria global y dispersar datos a cualquier ubicación. Su memoria compartida en el chip puede hacer que los hilos en el mismo procesador múltiple obtengan datos rápidamente, evitando el acceso frecuente a la memoria global y permitir que sus subprocesos en un grupo de subprocesos puedan sincronizarse, permitiendo comunicarse y cooperar para resolver problemas complejos [2].

El procesamiento de imágenes y video para el reconocimiento de objetos en tiempo real, se ha llevado a cabo a través de CNN son aceleradas por unidades GPU [3]. En [4] se propone un método de aceleración basado en el tratamiento de pesos binarios, enfocándose en la optimización del núcleo aritmético en el almacenamiento de los pesos. Un enfoque similar se presenta en [5], en donde la aceleración es realizada mediante una memoria de acceso aleatorio resistiva ReRam. Básicamente se trata de una arquitectura de acelerador adaptada para la convolución bit a bit.

Los estudios anteriores demostraron un alto consumo de potencia. En [6] este consumo es reducido mediante un acelerador programable de muchos núcleos para la arquitectura de una red CNN. El acelerador es denominado PACENet y consiste en la arquitectura de un conjunto de instrucciones específicas del kernel de la red neuronal y en 6 etapas de pipeline para la aceleración de la capa de convolución, las activaciones, la capa maxpool y la capa totalmente conectada. El diseño del acelerador es semejante, sin embargo, dos algoritmos de programación se implementaron en dos fases. El primero, se centraba en la combinación de imágenes para acelerar el proceso feed-fordward de la CNN y el segundo, en un algoritmo de costo ligero de memoria para la aceleración de un modelo CNN arbitrariamente grande para un dispositivo GPU con memoria limitada.

Con el objetivo de proporcionar incremento en el rendimiento del sistema de la GPU, se han realizado aceleradores a través de la arquitectura de cálculo paralelo CUDA en la ejecución de diferentes modelos de CNN. En [7], CUDA es utilizado en la creación de un mecanismo que mejora la red en términos de ejecución, el cual consta de la integración de los datos de los nodos de la red y del ajuste dinámico del factor de suavizado de la

función de la base. Sobre CUDA [8], implementaron una librería en C++ para acelerar el entrenamiento y proceso de clasificación de una CNN y librerías CUBLAS de NVIDIA con el fin de intercambiar el vector matemático y las operaciones funcionales.

### 1.1.2. IMPLEMENTACIONES DE CNN SOBRE SISTEMAS EMBEBIDOS BASADOS EN FPGA

Los FPGA son dispositivos que contienen un conjunto de bloques lógicos programables que pueden configurarse para tener el mismo comportamiento que cualquier circuito arbitrario [9]. Cada bloque lógico tiene una o más LUT y varios bits de memoria. Como resultado, los bloques lógicos pueden implementar funciones lógicas arbitrarias [10]. Son altamente personalizables, lo cual permite que desarrolladores puedan dirigir directamente la infraestructura de hardware de módulo a módulo y equilibrar los recursos y el rendimiento, eligiendo el nivel adecuado de paralelismo al momento de implementar aceleradores de algoritmos [11].

Dentro de los lenguajes de programación utilizados en el desarrollo de sistemas electrónicos basados en FPGA encontramos códigos de nivel RTL. Las implementaciones de algoritmos realizados en RTL son complicados por su relación con el lenguaje ensamblador, sin embargo, existen herramientas de diseño de hardware y síntesis de alto nivel que involucran lenguajes de programación como C y C++ y que son más amigables a los desarrolladores [12]. Dentro de estas herramientas encontramos Vivado HLS, en la cual, es posible obtener código RTL a partir de estos lenguajes de programación. Vivado HLS proporciona una metodología para implementar bloques de procesamiento de video e imágenes y técnicas de maximización de paralelismo, dentro de las cuales encontramos Loop Pipelining y Loop Unrolling. La técnica Loop Pipelining mejora el rendimiento del sistema al suponer la ejecución de operaciones con diferentes iteraciones de bucle y la técnica Loop Unroolling aumenta la utilización de recursos de computación masiva del FPGA.

Los FPGA, al ser un hardware reconfigurable con bajo consumo de potencia, permite a los investigadores tener control de los recursos del hardware en la ejecución de operaciones matemáticas de una CNN. En [13] se explora paralelizaciones a nivel de peso y nodos sobre los cálculos de las capas convolucionales. El sistema utiliza los recursos máximos mediante la reutilización, concatenación o descomposición de los datos. La descomposición de los datos de entrada en los cálculos convolucionales es un enfoque distinto mostrado por [14]. En el trabajo cada iteración reutiliza las unidades de cálculo aritmético para procesar en diferentes fragmentos los datos divididos. La trasmisión de los datos entre el FPGA y la memoria se realiza a través de la unidad DMA e interfaces de transmisión Axi-Stream. Esta transmisión en [15] incluye técnicas de partición e incrustamiento de los pesos en la memoria local y el uso de técnicas de paralelización. Estas paralelizaciones y el método basado en lotes redujeron el ancho de banda de memoria requerido en multiplicaciones matriciales de la CNN [16]. Caso contrario ocurre en la implementación del algoritmo de Winograd propuesto por [17], en donde se utilizan menos recursos informáticos pero se ejerce mayor presión sobre el ancho de banda de la memoria.

Por otro lado, la utilización de entornos de desarrollo por parte de los investigadores, ha reducido la complejidad en cuanto al flujo de diseño, control del uso de memorias y uso de técnicas de paralelización sobre estos dispositivos reconfigurables. En [18, 19] la información de una red CNN entrenada es sintetizada en el hardware a través de la herramienta de síntesis de alto nivel Vivado HLS. Esta herramienta define como se debe generar el IP Core que contendrá la información de las imágenes a clasificar. Esto es generado mediante Wrappers de alto nivel desarrollados internamente para facilitar la comunicación con el acceso directo a la memoria DMA. [16] presentan un trabajo similar, sin embargo, en su desarrollo propusieron un modelo de diseño analítico denominado Roofline. El modelo permite analizar cuantitativamente el rendimiento informático y el ancho de banda de memoria requerido para cualquier solución de un diseño de una CNN.

Los desarrollos de aceleradores de CNN sobre sistemas embebidos basados en FPGA o GPU descritos en los párrafos anteriores demostraron alta capacidad de procesamiento y bajo consumo de potencia. Sin embargo, el uso de estos sistemas presentan un tema abierto de investigación alrededor de la evaluación del desempeño al aplicarsen en robots humanoides con procesadores secuenciales integrados en su estructura. Estos procesadores presentan limitaciones en su capacidad de procesamiento al ejecutar algoritmos sofisticados de técnicas de aprendizaje profundo, restringiendo al robot en el desarrollo de tareas relacionadas al reconocimiento de objetos en tiempo real.

Con el fin de mejorar las limitaciones en la capacidad de procesamiento del sistema convencional de los robots al ejecutar algoritmos de CNN, diferentes enfoques se han presentado alrededor de grupos de investigación interesados en el área. En [20] se realiza un pre-procesamiento que permite reducir la cantidad de infor-

mación que la CNN debe clasificar a través de la reducción del espacio de búsqueda, en el cual, esta reducción se lleva a cabo a través de segmentaciones y extracción de la región de interés. El trabajo funciona bien bajo un punto de precisión, sin embargo, surgen problemas de aplicabilidad en tiempo real cuando se considera el tiempo computacional, es decir, no es viable si el objetivo es alcanzar una tasa de fotogramas alta.

Los trabajos orientados a mejorar el rendimiento computacional en robots con procesadores secuenciales presentaron una restricción en el tiempo de ejecución en el reconocimiento de objetos en tiempo real. En [21], con el fin de superar esta restricción se implementaron diferentes técnicas para el reconocimiento de objetos sobre arquitecturas computacionales externas de mayor capacidad de procesamiento. Los tiempos de respuesta aumentan, sin embargo, estas conexiones externas crean una continua dependencia en el robot humanoide, restringiéndolo a un mayor desenvolvimiento libre en su entorno. Este problema de autonomía es tratado por [22] a través de la integración de una arquitectura de menor tamaño a la estructura física del robot, sin embargo, el dispositivo no cuenta con los requerimientos de hardware necesarios para la ejecución de algoritmos que requieren alto nivel de paralelismo como las técnicas de aprendizaje profundo.

### 1.1.3. IMPLEMENTACIONES DE CNN SOBRE EL SISTEMA DE CÓMPUTO DEL ROBOT HUMANOIDE NAO

Las CNN son NN de varias capas especializadas en reconocer patrones visuales directamente desde los píxeles de la imagen. Se utilizaron para la detección y reconocimiento de objetos, incluidas las caras, con una precisión récord y un rendimiento en tiempo real [23]. Un trabajo relacionado con el reconocimiento de las expresiones emocionales se muestra en [24], donde se implementa un tipo de CNN llamado CCCNN. El CCCNNN está dividido en dos canales, el primero es el encargado de aprender a extraer información en función del contorno, la forma y la textura del rostro. El segundo canal codifica la información sobre la orientación, dirección y velocidad de los cambios dentro de las secuencias de cada uno. Una vez extraídas las características, se realiza la clasificación, donde los canales cruzados se conectan a una capa oculta totalmente conectada. La capa oculta está conectada a una capa de salida que implementa un clasificador Softmax. Este clasificador como capa final es utilizado en [20] para el reconocimiento de objetos en la plataforma RoboCup. Se realiza una etapa de pre-procesamiento basada en funciones de segmentación y extracción con el objetivo de reducir el coste computacional. A continuación, se implementa una CNN compuesta por una capa convolucional, pooling, normalización, conexión completa y softmax para la validación y reconocimiento de las áreas de imagen extraídas en la etapa de segmentación. Con esto, logran reducir la cantidad de datos a procesar, sin embargo, pese a ser una clasificación binaria (robots NAO o no NAO) los resultados teniendo en cuenta los Fps fueron muy bajos para una aplicación en tiempo real. Por otro lado, la CNN permite la extracción de características temporales y espaciales con respecto a una secuencia de datos de entrada. Esta característica en [25] permite realizar una aplicación relacionada con el reconocimiento de gestos en tiempo real a través de un tipo de CNN llamado MCCNNN. Esta arquitectura identifica y presenta la extracción de un fondo de la escena, que contiene la representación del movimiento generado por el gesto. Así, la MCCNNN recibe una imagen que contiene esta representación y aplica una serie de operadores convolucionales y de max pooling, que al final generarán un vector de características. Una aplicación similar se presenta en [26] para el reconocimiento facial por parte de un robot humanoide NAO. Se realiza un *transfer-learning* sobre AlexNet y VGG-face utilizando solo un ejemplo de entrenamiento por persona. Los resultados muestran que VGG-face logra mejor desempeño a una resolución baja y alta comparado con AlexNet, sin embargo, el tiempo de inferencia es de 0.24 s a comparación de los 0.03 s de AlexNet, lo cual, se encuentra muy por debajo de lo esperado para una aplicación en tiempo real. El tiempo de inferencia obtenido con AlexNet supera los 30 Fps, no obstante, con una sola imagen de entrenamiento por persona no se aprovechan las capacidades que pueden otorgar las CNN al realizar el entrenamiento con una mayor cantidad de imágenes ni se le da la capacidad al robot de reconocer el rostro en diferentes ángulos.

Se han realizado diferentes implementaciones de CNN sobre el sistema de cómputo de NAO para el reconocimiento de objetos en el campo de juego durante la competencia RoboCup. En [27] son ejecutadas las CNN XNOR-Net y SqueezeNet, en donde, los tiempos de respuesta fueron en tiempo real y se mantuvo la precisión en la detección, sin embargo, esto se produjo por la reducción de la resolución de la imagen a 24 x 24, la cual, al modificase a 640 x 480 (resolución de la cámara del robot) puede aumentar los tiempos de respuesta. Por otro lado, este tiempo de respuesta podría crecer si en lugar de ser una aplicación de clasificación binaria se implementa una clasificación multiclase. En [28] se implementa una CNN para detectar balones en lugar de

robots NAO. Una CNN liviana llamada Bynari-8 es implementada sobre la CPU de NAO logrando una precisión del 97.13% a 140 Fps. Los resultados demuestran una buena precisión y procesamiento de la imagen en tiempo real, sin embargo, al momento de agregarle mas clases a la red, la arquitectura no tiene la capacidad de reducir parámetros. Esta limitante crece el tamaño de la red y puede aumentar los tiempos de respuesta.

Diferentes trabajos se han centrado en el uso de diferentes CNN de alto desempeño como Yolo, VGG-16, VGG19 y Inception-V3 para la detección y clasificación de objetos por parte del robot humanoide NAO. Respecto a Yolo, en [29] se crea un sistema de detección y *tracking* de objetos en un ambiente cotidiano. Se toma un modelo pre-entrenado de la red Tiny-Yolo y se modifica la última capa para la detección de tres objetos, obteniendo un mAP de 44.3% con un tamaño de imagen de entrada de 448 x 448. A pesar de obtener resultados aceptables en términos de tiempos de inferencia, obtienen un promedio de confianza de predicción no mayor a 0.5, lo cual fue causado por falta de datos que contengan imágenes de diferentes tipos de ambiente. La versión Yolo-V3 es implementada en [30] en conjunto con un sistema distribuido para operar múltiples robots NAO en un motor de inferencia central que se conecta vía LAN al robot, pese a que se obtienen predicciones en tiempo real, al no estar el sistema de cómputo integrado a la estructura del robot, y al ser una conexión Ethernet, se restringe al robot desplazarse con mayor facilidad en un ambiente cotidiano. Finalmente, en [31] se implementan las redes VGG-16, VGG-19 y Inception V3 sobre la CPU de NAO para una tarea centrada en la clasificación de espacios de la casa como cocina, dormitorios y baños. Obtienen un accuracy del 95%, sin embargo, durante las pruebas, la clasificación fue correcta cuando se ubicaba dentro de la escena un único objeto de interés, lo cual no es suficiente para el desempeño de robots sociales.

### 1.1.4. IMPLEMENTACIÓN DE ALGORITMOS DE RECONOCIMIENTO DE IMÁGENES SOBRE SISTEMAS COMPUTACIONALES EXTERNOS APLICADO A ROBOTS HUMANOIDES

Las arquitecturas computacionales permiten la ejecución de las diferentes técnicas propuestas en un sistema de percepción visual de un robot humanoide. Estos procesadores son arquitecturas computacionales compuestas de un solo núcleo, que aunque su procesamiento es limitado frente a algoritmos robustos como las CNN, ha presentado resultados aceptables al ejecutar algoritmos de menor costo computacional como los modelos clásicos de aprendizaje automático. Por otro lado, se han integrado procesadores multi-core con el objetivo de procesar algoritmos más robustos. La siguiente sección describe los trabajos realizados que buscaron mejorar la percepción visual de robots humanoides al integrarle sistemas computacionales externos. La actual tesis de investigación busca mejorar la percepción visual del robot NAO al integrar un FPGA y GPU que ejecuta una CNN, por lo que los trabajos mencionados a continuación, respecto a sus limitaciones, fundamenta el presente trabajo.

*Single-core*

Diferentes enfoques se han centrado en el uso del procesador convencional que se incorpora en los robots humanoides. A pesar de sus limitaciones computacionales frente a algoritmos robustos, se han ejecutado técnicas de procesamiento de imágenes para aplicaciones relacionadas con el reconocimiento de gestos, emociones, objetos, objetivos en el RoboCup, entre otros. En el caso del robot humanoide NAO, se han utilizado diferentes versiones de este robot humanoide en diferentes aplicaciones. Un NAO con un procesador de CPU ATOM se utiliza para el reconocimiento de objetivos en RoboCup Standard [20],[32],[33]. En el mismo campo de aplicación, un procesador Pentium iV integrado al robot ejecuta los algoritmos propuestos para el reconocimiento de los rivales y el campo de juego [34],[35], mientras que [36] un procesador x86 AMD Geode sólo para el reconocimiento de robots humanoides NAO presentes en el juego. Otras aplicaciones relacionadas con el reconocimiento de objetos en diferentes entornos se presentan en [37],[38] utilizando una CPU ATOM. En el caso de las diferentes versiones del robot humanoide Armar, se han integrado en su sistema circuitos integrados controlados por un procesador de secuencia externo. En [39] un circuito integrado tridimensional (CI 3D), que se integra en el robot humanoide, que es controlado por un PC externo. Además, un PC externo en [40] está conectado a un módulo controlador universal (UcoM) integrado en el robot. Un procesador CPU externo de 3 GHz que ejecuta directamente los algoritmos de procesamiento de imágenes se utiliza para aplicaciones relacionadas con el agarre robótico [41],[42],[43] y sólo el reconocimiento de objetos [44]. Finalmente, los sistemas de percepción visual del robot humanoide iCub han sido implementados en una CPU Intel XEON de 2.4 GHz en tareas de reconocimiento de expresiones emocionales [24] y en un PC104 para una aplicación enfocada al reconocimiento de objetos [45].

Un enfoque diferente se presenta en [46] y [47]. En el primer trabajo se implementa un sistema de cómputo distribuido externo que se conecta vía Wifi al robot humanoide para la ejecución de algoritmos de visión por computador. Respecto al segundo trabajo, proponen un sistema micro cloud que entrena los modelos y realiza la inferencia a partir de la imagen captada por la cámara del robot. Los enfoques anteriores hacen uso de una conexión Wifi, en donde, en este tipo de conexión al no tener una conexión directa, los tiempos de respuesta del robot no serán los adecuados. En el caso de [47] solo se obtuvieron 2 fps, lo cual, esta muy por debajo de los intervalos para una aplicación en tiempo real.

*Multi-core*

El uso de procesadores multi-core para el procesamiento de imágenes ha crecido debido a su capacidad de cálculo. Las implementaciones sobre este tipo de procesadores se han enfocado en robots humanoide como NAO y iCub, debido a limitaciones de desempeño de su CPU al ejecutar algoritmos de mayor costo computacional. Generalmente, estos procesadores multi-core se conectan al robot humanoide a través de una conexión Ethernet. Un sistema de monitorización basado en el robot humanoide NAO se realiza mediante la ejecución de imágenes de un multiprocesador Intel i5 a 3,20 GHz [48]. Aquí, la implementación del sistema multi-core se centró en una sola aplicación, sin embargo, diferentes trabajos han hecho que la integración de estos procesadores deje abierta su aplicación de uso. Como en el caso de [21] y [22] mediante la integración de la tarjeta Odroid XU4. Por otro lado, el robot humanide iCub no dispone de un procesador integrado, lo que lleva a la integración de un sistema informático externo. [49] integra una GPU para el entrenamiento y ejecución de algoritmos de aprendizaje de máquina con el fin de reconocer objetos localizados en la mano del robot. En [50] el entrenamiento y ejecución de algoritmos fue realizado por un Intel i7 a 3.4 GHz. En este caso, los núcleos se dividieron para dos tareas diferentes. Cuatro núcleos realizaron la captura de los cuadros proporcionados por la cámara y un núcleo para los algoritmos de ejecución.

Una implementación de una CNN sobre sistemas computacionales externos se presenta en [51], en donde, el desarrollo se basa en la ejecución de una CNN SSD mobilenet DNN en los sistemas de cómputo Intel CPUcentered IntelR NUC7i7BNH (NUC) y la Jetson TX2 para una aplicación centrada en la detección de peatones sobre el robot humanoide NUgus. La implementación de la DNN sobre la NUC fue realizada sobre la CPU debido a la incompatibilidad entre Tensorflow y OpenCL, lo que impidió a los investigadores el desarrollo del lado de la GPU. Respecto al desarrollo sobre la Jetson TX2, a través de CUDA, realizaron la implementación de la DNN en la GPU. Los resultados mostraron que la CPU de la NUC es mas rápida que la GPU de la Jetson TX2 al ejecutar la DNN, en la NUC se obtiene 0.17s y en la Jetson TX2 0.57s. En cuanto al consumo de potencia, la CPU de la NUC consume 40.52 W y la Jetson TX2 alrededor 9.8 W. Los resultados demostraron un alto consumo de potencia y tiempos de inferencia por debajo de lo establecido para una aplicación en tiempo real. De acuerdo a [52], una aplicación relacionada a la detección de objetos se ejecuta en tiempo real cuando alcanza una velocidad de procesamiento de 30 Fps o mayor, sin embargo, para esta aplicación, se toma 25 Fps o más para concluir que la aplicación se ejecuta en tiempo real.

## REFERENCIAS

[1] J. Nickolls and W. J. Dally, "The gpu computing era," *IEEE micro*, vol. 30, no. 2, 2010.

[2] Z. Yang, Y. Zhu, and Y. Pu, "Parallel image processing based on cuda," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 3. IEEE, 2008, pp. 198–201.

[3] S. Li, Y. Dou, X. Niu, Q. Lv, and Q. Wang, "A fast and memory saved gpu acceleration algorithm of convolutional neural networks for target detection," *Neurocomputing*, vol. 230, pp. 48–59, 2017.

[4] R. Andri, L. Cavigelli, D. Rossi, and 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)*. IEEE, 2016, pp. 236–241.

[5] L. Ni, Z. Liu, H. Yu, and R. V. Joshi, "An energy-efficient digital reram-crossbar-based cnn with bitwise parallelism," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 3, pp. 37–46, 2017.

[6] A. Kulkarni, T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin, "Pacenet: Energy efficient acceleration for convolutional network on embedded platform," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[7] T. Gong, T. Fan, J. Guo, and Z. Cai, "Gpu-based parallel optimization of immune convolutional neural network and embedded system," *Engineering Applications of Artificial Intelligence*, vol. 62, pp. 384–395, 2017.

[8] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010, pp. 317–324.

[9] M. Halvorsen, "Hardware acceleration of convolutional neural networks," Master's thesis, NTNU, 2015.

[10] B. A. Draper, J. R. Beveridge, A. W. Bohm, C. Ross, and M. Chawathe, "Accelerated image processing on fpgas," *IEEE transactions on image processing*, vol. 12, no. 12, pp. 1543–1551, 2003.

[11] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with gpus and fpgas," in *Application Specific Processors, 2008. SASP 2008. Symposium on*. IEEE, 2008, pp. 101–107.

[12] J. Hiraiwa and H. Amano, "An fpga implementation of reconfigurable real-time vision architecture," in *2013 27th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2013, pp. 150–155.

[13] A. Dundar, J. Jin, B. Martini, and E. Culurciello, "Embedded streaming deep neural networks accelerator with applications," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 7, pp. 1572–1583, 2017.

[14] Q. Yu, C. Wang, X. Ma, X. Li, and X. Zhou, "A deep learning prediction process accelerator based fpga," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2015, pp. 1159–1162.

[15] H. Park, C. Lee, H. Lee, Y. Yoo, Y. Park, I. Kim, and K. Yi, "Work-in-progress: optimizing dcnn fpga accelerator design for handwritten hangul character recognition," in *2017 International Conference on Compilers, Architectures and Synthesis For Embedded Systems (CASES)*. IEEE, 2017, pp. 1–2.

[16] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and 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.

[17] Q. Xiao, Y. Liang, L. Lu, S. Yan, and 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.

[18] E. Del Sozzo, A. Solazzo, A. Miele, and 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)*. IEEE, 2016, pp. 217–224.

[19] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[20] D. Albani, A. Youssef, V. Suriani, D. Nardi, and D. D. Bloisi, "A deep learning approach for object recognition with nao soccer robots," in *Robot World Cup*. Springer, 2016, pp. 392–403.

[21] M. Podpora and A. Gardecki, "Extending vision understanding capabilities of nao robot by connecting it to a remote computational resource," in *Progress in Applied Electrical Engineering (PAEE)*. IEEE, 2016, pp. 1–5.

[22] M. Mattamala, G. Olave, C. González, N. Hasbún, and J. Ruiz-del Solar, "The nao backpack: An open-hardware add-on for fast software development with the nao robot," *arXiv preprint arXiv:1706.06696*, 2017.

[23] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," in *Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 342–347.

[24] P. Barros, C. Weber, and S. Wermter, "Emotional expression recognition with a cross-channel convolutional neural network for human-robot interaction," in *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*. IEEE, 2015, pp. 582–587.

[25] P. Barros, G. I. Parisi, D. Jirak, and S. Wermter, "Real-time gesture recognition using a humanoid robot with a deep neural architecture," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*. IEEE, 2014, pp. 646–651.

[26] D. Bussey, A. Glandon, L. Vidyaratne, M. Alam, and K. M. Iftekharuddin, "Convolutional neural network transfer learning for robust face recognition in nao humanoid robot," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2017, pp. 1–7.

[27] N. Cruz, K. Lobos-Tsunekawa, and J. Ruiz-del Solar, "Using convolutional neural networks in robots with limited computational resources: detecting nao robots while playing soccer," in *Robot World Cup*. Springer, 2017, pp. 19–30.

[28] Q. Yan, S. Li, C. Liu, and Q. Chen, "Real-time lightweight cnn in robots with very limited computational resources: Detecting ball in nao," in *International Conference on Computer Vision Systems*. Springer, 2019, pp. 24–34.

[29] J. Zhou, L. Feng, R. Chellali, and H. Zhu, "Detecting and tracking objects in hri: Yolo networks for the nao "i see you" function," in *2018 27th IEEE international symposium on robot and human interactive communication (RO-MAN)*. IEEE, 2018, pp. 479–482.

[30] S. Chatterjee, F. H. Zunjani, and G. C. Nandi, "Real-time object detection and recognition on low-compute humanoid robots using deep learning," in *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2020, pp. 202–208.

[31] K. M. Othman and A. B. Rad, "An indoor room classification system for social robots via integration of cnn and ecoc," *Applied Sciences*, vol. 9, no. 3, p. 470, 2019.

[32] A. Härtl, U. Visser, and T. Röfer, "Robust and efficient object recognition for a humanoid soccer robot," in *Robot Soccer World Cup*. Springer, 2013, pp. 396–407.

[33] U. Kaufmann, G. Mayer, G. Kraetzschmar, and G. Palm, "Visual robot detection in robocup using neural networks," in *Robot Soccer World Cup*. Springer, 2004, pp. 262–273.

[34] J. M. Cañas Plaza, E. Perdices García, T. González Sánchez, and D. Puig Valls, "Recognition of standard platform robocup goals," 2010.

[35] J. M. Canas, D. Puig, E. Perdices, and T. González, "Visual goal detection for the robocup standard platform league," in *X Workshop on Physical Agents, WAF*, 2009, pp. 121–128.

[36] J. F. Garcıa, F. J. Rodrıguez, F. Martın, and V. Matellán, "Using visual attention in a nao humanoid to face the robocup any-ball challenge," in *5th Workshop on Humanoids Soccer Robots. Nashville, TN, USA*, 2010.

[37] R. Boukezzoula, D. Coquin, T.-L. Nguyen, and S. Perrin, "Multi-sensor information fusion: Combination of fuzzy systems and evidence theory approaches in color recognition for the nao humanoid robot," *Robotics and Autonomous Systems*, vol. 100, pp. 302–316, 2018.

[38] S. Nefti-Meziani, U. Manzoor, S. Davis, and S. K. Pupala, "3d perception from binocular vision for a low cost humanoid robot nao," *Robotics and autonomous systems*, vol. 68, pp. 129–139, 2015.

[39] A. Andreopoulos, S. Hasler, H. Wersing, H. Janssen, J. K. Tsotsos, and E. Korner, "Active 3d object localization using a humanoid robot," *IEEE Transactions on Robotics*, vol. 27, no. 1, pp. 47–64, 2011.

[40] T. Asfour, K. Regenstein, P. Azad, J. Schroder, A. Bierbaum, N. Vahrenkamp, and R. Dillmann, "Armar-iii: An integrated humanoid platform for sensory-motor control," in *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*. IEEE, 2006, pp. 169–175.

[41] P. Azad, T. Asfour, and R. Dillmann, "Stereo-based 6d object localization for grasping with humanoid robot systems," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE, 2007, pp. 919–924.

[42] J. Bohg, K. Welke, B. León, M. Do, D. Song, W. Wohlkinger, M. Madry, A. Aldóma, M. Przybylski, T. Asfour *et al.*, "Task-based grasp adaptation on a humanoid robot," *IFAC Proceedings Volumes*, vol. 45, no. 22, pp. 779–786, 2012.

[43] A. Morales, T. Asfour, P. Azad, S. Knoop, and R. Dillmann, "Integrated grasp planning and visual object localization for a humanoid robot with five-fingered hands," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 5663–5668.

[44] D. Gonzalez-Aguirre, T. Asfour, and R. Dillmann, "Towards stratified model-based environmental visual perception for humanoid robots," *Pattern Recognition Letters*, vol. 32, no. 16, pp. 2254–2260, 2011.

[45] S. R. Fanello, C. Ciliberto, N. Noceti, G. Metta, and F. Odone, "Visual recognition for humanoid robots," *Robotics and Autonomous Systems*, vol. 91, pp. 151–168, 2017.

[46] F. Badeig, Q. Pelorson, S. Arias, V. Drouard, I. Gebru, X. Li, G. Evangelidis, and R. Horaud, "A distributed architecture for interacting with nao," in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, 2015, pp. 385–386.

[47] Q. Liu, C. Zhang, Y. Song, and B. Pang, "Real-time object recognition based on nao humanoid robot," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 644–650.

[48] Y. Hu, K. Sirlantzis, G. Howells, N. Ragot, and P. Rodríguez, "An online background subtraction algorithm deployed on a nao humanoid robot based monitoring system," *Robotics and Autonomous Systems*, vol. 85, pp. 37–47, 2016.

[49] B. Browatzki, V. Tikhanoff, G. Metta, H. H. Bülthoff, and C. Wallraven, "Active in-hand object recognition on a humanoid robot," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1260–1269, 2014.

[50] A. Holzbach and G. Cheng, "A fast and scalable system for visual attention, object based attention and object recognition for humanoid robots," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*. IEEE, 2014, pp. 316–321.

[51] A. Biddulph, T. Houliston, A. Mendes, and S. K. Chalup, "Comparing computing platforms for deep learning on a humanoid robot," in *International Conference on Neural Information Processing*. Springer, 2018, pp. 120–131.

[52] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

# 2

# METODOLOGÍA

El montaje experimental del sistema de reconocimiento de objetos fue realizado en el laboratorio de sistemas de control y robótica de Parque i del Instituto Tecnológico Metropolitano ITM. Allí se encontraron los elementos de hardware y de software necesarios para la investigación y desarrollo de cada uno de los objetivos planteados. El trabajo, presenta el diseño de un sistema de adquisición de la imagen mediante el uso de herramientas de software que posteriormente se conectan a los sistemas embebidos a través de una conexión Ethernet. A través de simuladores como Webots y interfaces de programación como Choreographe, se generan ambientes virtuales que incorporan objetos y un robot humanoide NAO. El diseño presentado se centra en el prototipo de un sistema de adquisición y comunicación dirigido a NAO, sin embargo, puede aplicarse a otros robots humanoides como NUgus, iCub, entre otros. Este diseño de adquisición y comunicación es utilizado en el sistema de reconocimiento de objetos que integran un FPGA o GPU y una CNN. Para efectos comparativos, se realiza una implementación enfocada en la ejecución de modelos clásicos de aprendizaje de máquina sobre el sistema de cómputo del robot humanoide NAO, para esto, fue utilizado un procesador que contiene las mismas características de la CPU de NAO. Este hardware consta de un Miniportatil Dell Inspiron Mini que contiene un procesador Intel Atom Z530 a 1.6 GHz. El uso de un procesador con las mismas características de la CPU de NAO, facilita las pruebas sin la presencia o encendido del robot humanoide.

Los subsistemas que conforman el sistema de reconocimiento de objetos son: adquisición del video, comunicación (transmisión, recepción del video entre NAO y cada sistema embebido) y procesamiento del video por parte de cada arquitectura heterogénea. En adelante, se introduce respecto al funcionamiento general del sistema y la función que cumple cada bloque. Dado a que el presente texto es introductorio y no pretende abordar a profundidad cada subsistema, en los apartados siguientes se brindará información detallada del desarrollo, referencias y características de cada etapa. La imagen 2.1 muestra el sistema de reconocimiento propuesto.

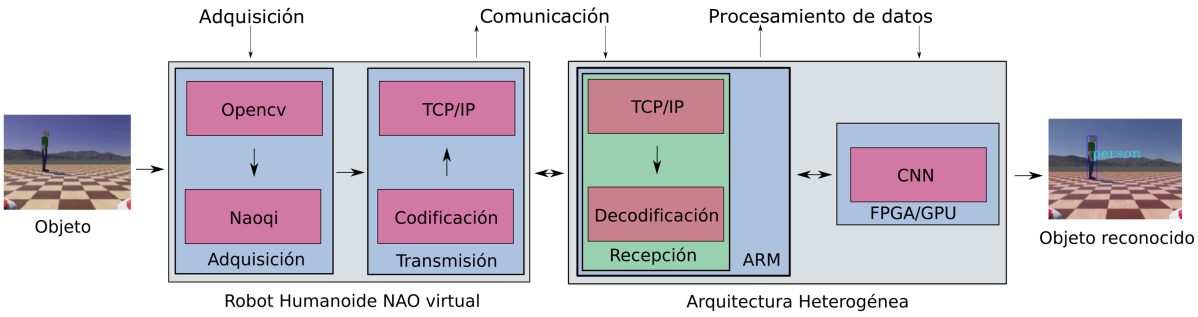


**Figura** 2.1: Esquema del sistema de reconocimiento propuesto

El sistema de adquisición de la imagen está basado en el uso del robot humanoide NAO, el cual, a través de dos cámaras ubicadas en su cabeza obtienen la imagen del objeto. Por medio de su unidad de procesamiento

secuencial (ATOM Z530), el cual contiene un sistema operativo basado en Linux, se adquiere el video por parte de las dos cámaras. En el caso de la aplicación, la imagen es adquirida, procesada y enviada a través de una conexión Ethernet a cada sistema embebido.

La imagen adquirida es transmitida mediante el uso del protocolo de control de transmisión TCP y socket. Una vez la imagen es enviada al sistema embebido (FPGA o GPU), es decodificada utilizando librerías de Python para transmisión de paquetes por socket, asignación de dirección IP y puerto de red. Teniendo la imagen del objeto vista por el robot humanoide en cada sistema embebido, se realiza la integración del video a cada framework de aceleración de las CNN's pertenecientes a cada tarjeta de desarrollo. En el caso de la GPU, se utilizan como framework *Darknet* y *TensorRT*. En el caso del FPGA, son usados los frameworks *Dnndk-Pynq*, *Vitis-Dpu*, *Finn* y *PipeCnn*.

Inicialmente, se realizó la simulación de un robot humanoide NAO virtual mediante la herramienta de simulación Webots. Una vez obtenida la imagen virtual proporcionada por Webots, es enviada al software de programación Choregraphe. En este software, se crearon y editaron movimientos interactivos para el robot humanoide, en donde, cada movimiento dado se reflejaba en el movimiento de la cámara. Finalmente, la imagen virtual generada por Webots y controlada por Choregraphe, es enviada a cada sistema embebido a través de una dirección IP e integrada a la CNN para realizar el reconocimiento del objeto.

Lo anterior, se resume en cuatro secciones. La sección 2.1 muestra los elementos y conceptos claves en los que se fundamenta la actual tesis de investigación. Se describen los elementos de hardware y software empleados, los conjuntos de imágenes y cada una de las técnicas de reconocimiento de objetos. La sección 2.2, se basa en la adquisición y comunicación de la imagen, en esta, se diseñó el esquema de adquisición, comunicación y procesamiento de datos teniendo en cuenta el protocolo de comunicación entre las arquitecturas heterogéneas y la CPU del robot humanoide. La sección 2.3 se basa en el procesamiento de datos, en esta etapa, se realizó la selección de diferentes CNN con base en trabajos encontrados en el estado del arte, teniendo en cuenta su desempeño en el reconocimiento de objetos y las características de las arquitecturas heterogéneas. Para el caso de la red Inception V3, se realizó un transfer-learning a partir del conjunto de datos INSTRE, en el caso de las redes Tinier-Yolo y Inception V1, se tomaron estos modelos preentrenados a partir del conjunto de imágenes proporcionadas por Pascal Voc2012 y ImageNet respectivamente. La implementación de las CNN sobre el FPGA se desarrolló a través de los frameworks de código abierto *Dnndk-Pynq*, *Vitis-Dpu*, *Finn* y *PipeCnn*, los cuales permitieron realizar la estructura de bloques de memorias y comunicación de datos en el hardware, y la paralelización de datos a partir de directivas de optimización. En cuanto a la implementación sobre la GPU, la aceleración se llevó a cabo a partir *Darknet* y *TensorRT*. Las implementaciones se efectuaron de manera independiente, los sistemas embebidos a utilizar fueron la *Ultra96 FPGA* , *Intel Cyclone V SoC FPGA* y la *Jetson TX2 GPU*. Por otro lado, diferentes técnicas clásicas son ejecutadas sobre un sistema de computo similar al del robot humanoide NAO. La implementación de estas técnicas se fundamenta en el hecho de demostrar que el mejoramiento de la percepción visual en robots humanoides como NAO, no solo se basa en la mejora en cuanto al tiempo de respuesta que podría otorgarle un FPGA o GPU. Otro punto crucial se basa en la cantidad de objetos que el robot podría reconocer en un ambiente cotidiano, en este punto, las CNN han demostrado un mejor desempeño en términos de precisión respecto a las técnicas clásicas cuando el número de clases aumenta y la cantidad de objetos a reconocer en una misma escena crece. Finalmente, en la sección 2.4, se muestra la evaluación de desempeño de la implementación al realizar la comparativa frente a los principales trabajos encontrados en el estado del arte.

## 2.1. ELEMENTOS Y CONCEPTOS FUNDAMENTALES

Se detallan los componentes de software y hardware. Del lado del hardware, se describen cada uno de los sistemas embebidos utilizados para la ejecución de las CNN. También, se detalla al robot humanoide NAO, el cual, fue usado como caso de estudio para evaluar la mejora en la percepción visual de robots humanoides al integrar una FPGA y GPU que ejecutan una CNN. Respecto a los componentes de software, se detallan las herramientas de simulación, las cuales permitieron crear un ambiente virtual de un sistema de reconocimiento objetos orientado a robots humanoides como NAO, los cuales tienen un sistema de cómputo limitado. De igual forma, se presentan las CNN implementadas sobre los sistemas embebidos.

### 2.1.1. HARDWARE

Los sistemas embebidos utilizados tienen arquitecturas heterogéneas que integran un procesador y un acelerador con alto grado de paralelismo GPU o FPGA. Uno de los dispositivos es la Jetson TX2, que es un SoC que integra un procesador ARM de cuatro núcleos con un GPU fabricado por NVIDIA. Por su parte, la Ultra96 contiene un procesador de cuatro núcleos ARM con un MPSoC fabricado por Xilinx que incorpora un FPGA. Respecto a Cyclone V SoC FPGA de Intel proporciona un *Hard Processor System (HPS)* dual core ARM* Cortex*-A9, periféricos integrados, controladores de memoria multipuerto, transceptores serie y puertos PCI Express* (PCIe*). Estos dispositivos cuentan con características que incluyen alto grado de paralelismo y bajo consumo de potencia. El ARM les permite a estos dispositivos la instalación de sistemas operativos que faciliten la interfaz con el usuario y la conexión con otros dispositivos, además de la instalación de frameworks y librerías para aplicaciones relacionadas a la inteligencia artificial. La Tabla 2.1 muestra las características principales de la Jetson TX2, Ultra96 FPGA y Cyclone V SoC FPGA. A continuación, se detallan cada uno de los sistemas embebidos utilizados.

#### JETSON TX2 NVIDIA-GPU

La Jetson TX2 de Nvidia, es una plataforma de alta eficiencia computacional y energética, dirigida al desarrollo de soluciones de Inteligencia Artificial. Se trata de una placa basada en una GPU con arquitectura pascal de 256 núcleos de Nvidia junto con una CPU ARMv8 hexa core de 64 bits. Contiene con una memoria de 8 gigabytes y una interfaz de 59,7 GB/s de 128 bits de capacidad de transferencia de datos de memoria. Cuenta con dos multiprocesadores (SM), cada SM provee 128 núcleos de 1.3 GHz, compartiendo memoria caché L2 de 512 KB. Cada núcleo de la CPU y GPU comparten 8 GB de memoria DRAM a 1,866 GHz [1]. Incluye una serie de interfaces de equipo estándar como pantalla, cámara, GPIO, etc, lo que permite la creación rápida de prototipos. Este sistema embebido, acelera arquitecturas de DNN mediante las librerías NVIDIA cuDNN y TensorRT. Cuenta con soporte para redes neuronales recurrentes (RNN) y aprendizaje por refuerzo en línea [2]. Posee un ISP de 1400 megapíxeles por segundo. El SDK Jetpack, se utiliza para automatizar las instalaciones básicas en la Jetson TX2, que incluye los *Board Support Packages*, bibliotecas especialmente para el aprendizaje profundo y la visión por ordenador.

#### ULTRA96-V2 XILINX-FPGA

La Ultra96-V2 es una placa de desarrollo creada por Xilinx para aplicaciones que requieren un alto grado de paralelismo y bajo consumo de potencia. Este sistema embebido posee dos CPU de doble núcleo, un ARM Cortex A-53 y un Cortex-R5 que trabajan a 1.5 GHz y hasta 600 MHz respectivamente. Se conectan con un Xilinx Zynq UltraScale + MPSoC, el cual contiene 70.560 LUTs, 360 DSP *slices* y 7.5 MB de BRAM [3].

#### CYCLONE V-SOC INTEL-FPGA

La Cyclone V-SoC de Altera es una red de compuertas programables FPGA con un procesador ARM integrado. Está compuesto de dos partes distintas, un HPS ARM Cortex-A9 de uno o dos núcleos y una FPGA. La arquitectura HPS integra un amplio conjunto de periféricos que reducen el tamaño de la placa y aumentan el rendimiento del sistema. En general, la comunicación entre el HPS y el FPGA en este dispositivo se puede realizar de las siguientes maneras: puente de HPS a FPGA, puente ligero de HPS a FPGA, puente de FPGA a HPS y interfaz SDRAM de FPGA a HPS [4].

#### ROBOT HUMANOIDE NAO

NAO es un robot humanoide compuesto por dispositivos sensóricos y motores, los cuales, son controlados por el sistema operativo denominado *NAOqi OS*. Estos dispositivos electrónicos permiten que en él se generen sentidos para distintos tipos de interacción natural. Su sistema de visión está equipado de dos cámaras situadas en la cabeza, centradas horizontalmente con un desplazamiento vertical. Una cámara está posicionada para mirar directamente hacia adelante, mientras que la otra apunta hacia el suelo y pies del robot. Cuenta con protocolos de comunicación Ethernet y WiFi que le permiten comunicarse con arquitecturas computacionales externas para la ejecución de una tarea determinada. El procesamiento de las operaciones asignadas al procesador integrado al robot humanoide, se ve afectado cuando se ejecutan algoritmos que incluyen gran cantidad de datos, produciendo el efecto denominado cuello de botella “Bottleneck”, en el cual, el procesamiento de los datos se realiza lentamente, limitando su rendimiento.

**Tabla** 2.1: Características de los sistemas embebidos basados en FPGA y GPU

| Hardware | Ultra96 FPGA | Jetson TX2 GPU | CycloneV SoC FPGA |
|---|---|---|---|
| SoC | Xilinx Zynq UltraScale + MPSoC ZU3EG A484 | 256-core NVIDIA Pascal™ GPU | SX SoC—5CSXFC6D6F31C6N |
| CPU | Quad-core ARM® Cortex®-A53 | Quad-Core ARM® Cortex®-A57 MPCore | ARM Cortex-A9® MPCore |
| RAM | Micron 2 GB (512M x32) LPDDR4 | 8GB 128-bit LPDDR4 Memory | 1 GB DDR3 SDRAM |
| Almacenamiento | Delkin 16 GB microSD card + adaptador | 32GB eMMC 5.1 | 4 GB Micro-SD |
| **Sistema Operativo** | PetaLinux | Ubuntu 18.04 | Linux LXDE Desktop |

### 2.1.2. SOFTWARE

#### SIMULADOR WEBOTS

Webots es un programa informático que permite la simulación de aplicaciones de robótica móvil mediante la creación de prototipos para el modelado, programación y simulación de robots móviles [5]. Consta de cuatro etapas de desarrollo al momento de realizar una simulación, empezando por un modelado del robot hasta la transferencia del programa creado al robot real. Dentro de los lenguajes de programación soportados se encuentran C++, C, Java, Matlab, Python. Esta plataforma de código abierto tiene un amplio uso en la industria, educación y investigación, en donde podemos simular aplicaciones sobre robots humanoides como NAO, Tiago y NUgus.

#### HERRAMIENTA DE PROGRAMACIÓN CHOREOGRAPHE

Choregraphe es un entorno gráfico desarrollado por Aldebaran Robotics para la programación de robot humanoide NAO. Este software, permite realizar conexiones de movimientos con un alto nivel de complejidad, otorgando un software profundo para este robot humanoide, el cual contiene 25 grados de libertad [6]. Además, ofrece la posibilidad de realizar ajustes de movimientos conjuntos o cartesianos complejos. Cuando se desarrolla un programa para una tarea específica, los bloques que contienen los movimientos o acciones específicas, se llevan al flujo de programación, conectándose con lineas que denotan el flujo de datos y control del programa [7]. Para este trabajo, a través de Choregraphe se realizó el control del robot humanoide virtual generado por Webots y el envío de la imagen al sistema embebido.

### 2.1.3. TÉCNICAS DE CLASIFICACIÓN Y DETECCIÓN DE OBJETOS

#### REDES NEURONALES CONVOLUCIONALES (CNN)

La evaluación de las redes neuronales convolucionales sobre los sistemas embebidos se basó en el uso de los modelos Tinier-Yolo (versión modificada de Yolo), InceptionV1, AlexNet. Se tomaron los modelos pre-entrenados a partir de estas arquitecturas de CNN teniendo cuenta su desempeño en aplicaciones relacionadas a la clasificación y detección de objetos. Se presenta una arquitectura implementada a partir de un *transfer-learning* sobre la red InceptionV3 para una tarea centrada en la clasificación de juguetes. Esta topología consiste en 310 capas convolucionales y capas completamente conectadas. Para el entrenamiento con *transfer-learning* se congelan las primeras 172 capas convolucionales de la red. El entrenamiento para la clasificación de juguetes es desarrollada mediante un *transfer-learning*, el cual, es un método que consiste en reutilizar el entrenamiento anterior de una red congelando algunas capas y seleccionando una tasa de aprendizaje menor para adaptar los pesos al nuevo conjunto de entrenamiento. El *transfer-learning* permite disminuir el tiempo de entrenamiento y utilizar bases de datos pequeñas debido a que la mayoría de características como bordes, curvas y colores ya vienen definidas por los pesos entrenados por las primeras capas. En este trabajo, se utiliza *transfer-learning* con la red convolucional Inception-V3. A continuación, se detallan cada una de estas arquitecturas de CNN.

**AlexNet**

**Tabla** 2.2: Arquitectura AlexNet

| Capa | Mapa de características | Tamaño | Tamaño Kernel | Activación |
|---|---|---|---|---|
| Imagen | 1 | 227x227x3 | - | - |
| Convolución | 96 | 55x55x96 | 11x11 | relu |
| Agrupación maxima | 96 | 27x27x96 | 3x3 | relu |
| Convolución | 256 | 27x27x256 | 5x5 | relu |
| Agrupación maxima | 256 | 13x13x256 | 3x3 | relu |
| Convolución | 384 | 13x13x384 | 3x3 | relu |
| Convolución | 384 | 13x13x384 | 3x3 | relu |
| Convolución | 256 | 13x13x256 | 3x3 | relu |
| Agrupación maxima | 256 | 6x6x256 | 3x3 | relu |
| FC | - | 9216 | - | relu |
| FC | - | 4096 | - | relu |
| FC | - | 4096 | - | relu |
| FC | - | 1000 | - | softmax |

**Tabla** 2.3: Arquitectura Inception V1

| | *Tamaño filtro/Cantidad de pasos* | *Tamaño entrada* |
|---|---|---|
| Conv | 3 x 3 / 2 | 222 x 229 |
| Conv | 3 x 3 / 1 | 149 x 149 x 32 |
| Conv padded | 3 x 3 / 1 | 147 x 147 x 32 |
| Ppool | 3 x 3 / 2 | 147 x 147 x 64 |
| Conv | 3 x 3 / 1 | 73 x 73 x 64 |
| Conv | 3 x 3 / 2 | 71 x 71 x 80 |

Esta arquitectura de CNN fue entrenada para clasificar 1000 clases conformadas por 1.2 millones de imágenes de alta resolución del conjunto de datos ImageNet LSVRC-2010. Tiene tasas de error del 37,5% y del 17,0% en el top-1 y el top-5 respectivamente durante la fase de validación. Su arquitectura cuenta con 60 millones de parámetros y alrededor de 650.000 neuronas. Consta de 5 capas convolucionales, algunas conectadas a capas de agrupación máxima y 3 capas totalmente conectadas con una softmax de 1000 clases. Con el objetivo de disminuir el sobreajuste, en las capas *fully connected*, se emplea el método de regularización *dropout*, obteniendo una alta precisión [8]. La Tabla2.2 muestra la arquitectura de la CNN AlexNet.

**Inception v1**

La red Inception v1, también conocida como GoogLeNet fue entrenada con la base de datos ImageNet. Esta red posee en total 22 capas ocultas y contiene una precisión del 88.9%. Esta red utiliza módulos inception, que permiten a la red elegir entre múltiples tamaños de filtro convolucional en cada bloque. Una red Inception apila estos módulos uno encima del otro, con capas *maxpooling* con pasos de dos para reducir a la mitad la resolución de la grilla. La Tabla 2.3 muestra la arquitectura de la CNN Inception V1.

**Inception v3**

Inception v3 es un modelo de reconocimiento de imágenes muy utilizado, el cual alcanza una exactitud superior al 78.1% en el conjunto de datos de ImageNet. El modelo está formado por bloques de construcción simétricos y asimétricos que incluyen convoluciones, reducción promedio, reducción máxima, concatenaciones, retirados y capas completamente conectadas. Esta arquitectura obtuvo tasas de error del 21,2% en el top-1 y 5,6% en el top-5 para evaluación al tener un costo computacional de 5.000 millones de multiplicaciones en inferencia [9]. La normalización por lotes se usa con frecuencia en todo el modelo y se aplica a las entradas de activación. Finalmente, las pérdidas se calculan a través de Softmax. La Tabla 2.4 muestra la arquitectura de la CNN Inception v3.

**Tabla** 2.4: Arquitectura Inception V3

| | ***Tamaño filtro/Cantidad de pasos*** | ***Tamaño entrada*** |
|---|---|---|
| Conv | 3 x 3 / 2 | 224 x 224 x 3 |
| Conv | 3 x 3 / 1 | 111 x 111 / 32 |
| Conv padded | 3 x 3 / 1 | 109 x 109 x 32 |
| Pool | 3 x 3 / 2 | 109 x 109 x 64 |
| Conv | 3 x 3 / 1 | 54 x 54 x 64 |
| Conv | 3 x 3 / 2 | 52 x 52 x 80 |
| Conv | 3 x 3 / 1 | 25 x 25 x 192 |
| Inception A x3 | - | 25 x 25 x 288 |
| Inception B x5 | - | 12 x 12 x 768 |
| Inception C x2 | - | 5 x 5 x 1280 |
| Fc | 51200 x 1024 | 5 x 5 x 2048 |
| Fc | 1024 x 1024 | 1024 |
| Fc | 1024 x 4 | 1024 |
| Softmax | Clasificador | 4 |

**Tinier-Yolo**

Tinier-Yolo hace parte de una de las derivaciones de *Yolo v3*, una CNN para detección de objetos creada por *Joseph Redmon*. Esta red, básicamente, se trata de un modelo de menor tamaño pero para entornos restringidos. A pesar de no contener variedad de entornos es ideal, por su reducido tamaño, para sistemas embebidos con poca capacidad de almacenamiento. Esta redes entrenada con la base de datos PASCAL VOC, pero con 1 bit para los pesos y 3 para las activaciones. Tiene 50.1 % con respecto al 57.1 % de mAP que tiene Tiniy-Yolo, y puede aplicarse en tareas que involucran clasificación y segmentación para el reconocimiento de 20 clases que contienen personas, animales, vehículos y elementos domésticos. La Tabla 2.5 muestra la arquitectura de esta CNN, en donde *ci* son el número de canales de entrada, *k* el tamaño del kernel, *co* el número de canales de salida y *parameters* el número de parámetros.

**Tabla** 2.5: Arquitectura Tinier-Yolo

| | ***ci*** | ***k*** | ***co*** | ***parameters*** |
|---|---|---|---|---|
| Conv1 | 3 | 3 | 16 | 184 |
| Conv2 | 16 | 3 | 32 | 740 |
| Conv3 | 32 | 3 | 64 | 2888 |
| Conv4 | 64 | 3 | 128 | 11408 |
| Conv5 | 128 | 3 | 256 | 45344 |
| Conv6 | 256 | 3 | 512 | 180800 |
| Conv7 | 512 | 3 | 1024 | 722048 |
| Conv8 | 1024 | 1 | 256 | 74016 |
| Conv9 | 256 | 3 | 512 | 53536 |
| Conv10 | 384 | 3 | 256 | 53536 |

### CONJUNTO DE IMÁGENES

Para la comparación entre los sistemas embebidos basados en FPGA y GPU, y la CPU del robot NAO, se utilizaron modelos preentrenados, los cuales hicieron uso de la base de datos Pascal Voc2012 y ImagNet. Un problema de clasificación de juguetes mediante un transfer-learning de InceptionV3 es evaluado sobre estos sistemas con el objetivo de realizar un mejoramiento de la HRI a nivel pedagógico, para esta aplicación fue utilizado el conjunto de datos de INSTRE [10]. Las subsecciones siguientes detallan cada dataset.

**PASCAL Visual Object Classes Challenge (Pascal VOC2012)**

Para un problema centrado en la detección de objetos a partir de la CNN Tinier-Yolo, fue utilizado Pascal Voc 2012 como dataset. Pascal Voc2012 tiene como principal objetivo reconocer objetos a partir de un determinado número de clases en ambientes reales. Es utilizado básicamente en algoritmos de aprendizaje supervisado y es posible aplicarlo en tres tipos de reconocimiento de objetos, que son: clasificación, detección y segmentación.

*Clasificación*: Para cada una de las veinte clases, predice la presencia - ausencia de un ejemplo de esa clase.

*Detección*: Predice el *bounding box* y la etiqueta de cada objeto a partir de las veinte clases.

*Segmentación*: Genera la segmentación de píxeles que da la clase del objeto visible en cada píxel o el *background*.

La Figura 2.2 muestra cada uno de los objetos son su respectiva etiqueta, en este caso, para un problema de detección de objetos.


**Figura** 2.2: Objetos del dataset Pascal Voc2012 con su respectiva etiqueta

**ImageNet Large-Scale Visual Recogni-tion Challenge (ILSVRC)**

Para esta aplicación, se toman los modelos preentrenados InceptionV1 y AlexNet a través del conjunto de imágenes proporcionado por ImageNet. Este dataset contiene variedad de clases permitiendo que sea utilizada en aplicaciones que contienen diversas escenas del mundo real. Posee más de 15 millones de imágenes de alta resolución etiquetadas que pertenecen a unas 22.000 categorías. Abarca alrededor de 1000 clases y 5 imágenes por cada clase. Las imágenes se recogieron de la web y fueron etiquetadas por personas que utilizaron la herramienta de crowdsourcing Mechanical Turk de Amazon. A partir de 2010, como parte del Pascal Visual Object Pascal, un concurso anual llamado ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). Experimentos en [11] demostraron que para 80 clases incluidas en este dataset se logró una precisión del 99%. Al igual que Pascal VOC2012, Imagenet posee imágenes para entrenamiento aplicables a problemas de clasificación, detección y segmentación. La clasificación a nivel de imagen indican ausencia o presencia de una clase de una imagen. La detección proporcionan un cuadro delimitador alrededor del objeto reconocido y la segmentación obtiene el *background* del texto reconocido.

**INSTRE: for INSTance-level object REtrieval and REcognition**

El dataset INSTRE además de la recuperación y el reconocimiento de objetos a nivel de instancia, sirve para

muchos otros algoritmos de visión por computadora, como la detección, características invariantes y coincidencia de características. Este conjunto de datos se divide en tres subconjuntos separados INSTRE-S1 (para el caso de un solo objeto 1), INSTRE-S2 (para el caso de un solo objeto 2) e INSTRE-M(para el caso de varios objetos). INSTRE-S1 e INSTRE-S2 se recopilan para medir el caso de un solo objeto, y ambos tienen 100 clases de objeto. INSTRE-S1 contiene 11011 imágenes e INSTRE-S2 contiene 12059 imágenes. Agrupamos los 100 objetos de INSTRE-S1 en 50 dos tuplas. Finalmente, INSTRE-M contiene 5473 imágenes distribuidas en 50 clases de dos tuplas. De este dataset se realiza una selección para construir una base de datos de 10 juguetes. Para cada una de las clases seleccionadas, se definió un conjunto de entrenamiento que consta de entre 70 a 100 imágenes y un conjunto de validación conformado por entre 30 y 40 imágenes. El dataset completo consta de 810 imágenes de entrenamiento y 356 imágenes de validación. El dataset de 10 objetos se muestra en la Figura 2.3.

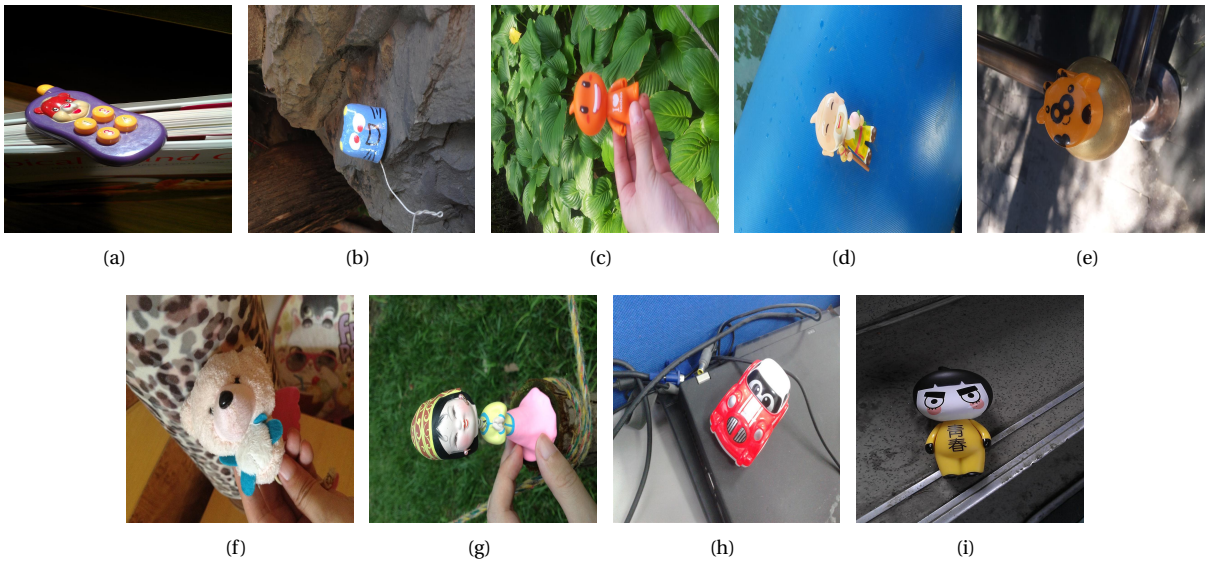


**Figura** 2.3: Clases de la base de datos utilizada para aplicar *transfer-learning*: (a) Toy_phone, (b) Toy_cat, (c) Toy_taobao, (d) Toy_oldman, (e) Toy_cow, (f) Toy_bear, (g) Toy_woman, (h) Toy_car, (i) Toy_young, (j) Toy_young.

## 2.2. ADQUISICIÓN Y COMUNICACIÓN

El sistema de adquisición de imágenes se basó en el uso de un entorno virtual que contenía objetos que debían ser reconocidos por el robot humanoide NAO, el cual, se encontraba también en la escena recreada. Este entorno virtual se creó a partir del simulador Webots, donde la imagen de la escena recreada se envía a Choreographe para editar los movimientos interactivos del robot e iniciar la transmisión de la imagen a cada sistema integrado de manera independiente.

### 2.2.1. ADQUISICIÓN

La adquisición de la imagen del objeto vista por el robot humanoide NAO se creó a partir de un ambiente virtual que contuvo objetos a reconocer. Para esto, fueron utilizadas las herramientas de software Webots y Choreographe. En webots fue generado el entorno virtual, en donde la imagen del objeto vista por el robot humanoide fue transmitida a Choreographe. Una vez la imagen es obtenida en Choreographe, se editan movimientos para el robot humanoide NAO generando distintos ángulos de la imagen del objeto. Como herramienta de integración entre Webots y Choregraphe se añadió al simulador el paquete *naoqisim*, el cual, permitió realizar el control del robot humanoide NAO creado por Webots desde Choregraphe a través de la interfaz de programación *Naoqi*, un API de Python que permite desarrollar aplicaciones sobre el robot humanoide NAO, y que contiene librerías de adquisición de imagen y transmisión asignando una dirección IP.

El script desarrollado, inicialmente importa las librerías *opencv* y *vision_definitions* de *naoqi* para la adquisi-

ción de la imagen. Posteriormente, la dirección IP y puerto de red del robot es asignado. A partir de la función *ALProxy* el video transmitido por Webots es obtenido desde Choregraphe. Esta imagen, es enviada al video-monitor utilizando la función *suscribeCamera*, en la cual, se agrega un buffer del formato de imagen solicitado a la lista de buffers. Finalmente, la imagen obtenida es visualizada en el video monitor de Choregraphe mediante el método *getImageRemote*. El pseudocódigo de la adquisición del video se muestra en el Algoritmo 1.

**Algorithm 1:** Obtener imagen de Webots en Choregraphe

```
ip_robot = "longitud de strings";
port_robot = "longitud de int";
videoDevice = "ALProxy('imagen_webots', ip_robot, port_robot)";
captureDevice = "videoDevice.suscribeCamera()";
width = "ancho de la imagen";
height = "alto de la imagen";
while True do
    result = videoDevice.getImageRemote(captureDevice);
    for i -> height do
        for j -> width do
            Agregar imagen result en widht x height
        end
    end
    ;
    Result: Mostrar imagen en Choregraphe result
end
```

### 2.2.2. TRANSMISIÓN DEL VIDEO

En el desarrollo de esta etapa se hizo uso de TCP como protocolo de transmisión de paquetes. Mediante librerías de Python se codifica la imagen y posteriormente se envía a través de una conexión Ethernet asignando una dirección IP y un puerto de red. A través de la librería *Base 64* se codificó y decodificó la imagen teniendo en cuenta el estándar *RFC 3548*. Con *Base 64* se tomaron los algoritmos en *Base 16*, *Base 32* y *Base 64* para codificar y decodificar cadenas arbitarias en cadenas de texto que pudieron ser enviadas por la red.

Con el objetivo de aumentar la velocidad en la transmisión de paquetes, fue utilizada la librería *Zmq*, en donde su capacidad de mensajería asíncrona, proporcionó un alto rendimiento en la ejecución y transmisión de paquetes. Su uso, básicamente se fundamentó en las características que otorga en cuanto a velocidad de ejecución, permitiendo reducir la cantidad de paquetes perdidos para obtener resultados correctos en tiempo real. Finalmente, la imagen visualizada en Choregraphe es transmitida haciendo mediante las librerías anteriormente mencionadas. Para esto, se utilizó el Algoritmo 1 y se adaptaron las lineas de código para la codificación - decodificación de la imagen y la transmisión a los sistemas embebidos. El Pseudocódigo es mostrado en el Algoritmo 2.

### 2.2.3. RECEPCIÓN DEL VIDEO

Para la recepción de la imagen virtual proporcionada por Webots en cada sistema, se utilizaron las librerías descritas en la sección anterior. Inicialmente el puerto de red asignado en la transmisión es configurado. A través de la función *setsockopt_string* son manipulados los datos recibidos y posteriormente convertidos a una serie de caracteres *string*. Adicional, esta serie de caracteres es decodificada y convertida al sistema de numeración posicional *Base 64*. Finalmente la imagen decodificada es leída a partir del buffer almacenado en la memoria utilizando la función de opencv *cv2.imdecode*. El Pseudocódigo es mostrado en el Algoritmo 3.

**Algorithm 2:** Transmisión del video a cada sistema embebido

```
ip_robot = "longitud de strings";
port_robot = "longitud de int";
videoDevice = "ALProxy('imagen_webots', ip_robot, port_robot)";
captureDevice = "videoDevice.suscribeCamera()";
widht = "ancho de la imagen";
height = "alto de la imagen";
while True do
    result = videoDevice.getImageRemote(captureDevice);
    for i -> height do
        for j -> width do
            Agregar imagen result en widht x height
        end
    end
    ;
    result = encoded(result);
    envío por socket (result);
    Result: Transmisión del video result
end
```

**Algorithm 3:** Recepción del video en cada sistema embebido

```
ip_robot = "longitud de strings";
port_robot = "longitud de int";
socket.bind(escuchar por ip_robot y port_robot);
setsockpt(imagen transmitida);
while True do
    result = decodificación base 64;
    result = lectura del bufer por opencv;
    Result: Visualización del video result
end
```

**Tabla** 2.6: Sistemas embebidos, frameworks de aceleración y CNN implementadas

| Tipo de hardware | Tarjeta de desarrollo | Framework aceleración | CNN |
|---|---|---|---|
| **FPGA** | Cyclone V-SoC | PipeCNN | AlexNet |
| | Ultra96 | Finn | Tinier-Yolo |
| | | Vitis-DPU | Inception-V1 |
| | | Dnndk | Inception-V3 transfer-learning |
| **GPU** | Jetson TX2 | Darknet | Tinier-Yolo |
| | | TensorRT | Inception-V1 |
| | | | AlexNet |

## 2.3. PROCESAMIENTO DE DATOS

Se realiza la implementación y evaluación de desempeño para un sistema de reconocimiento de objetos basado en una FPGA o una GPU al ejecutar una CNN. Inicialmente, se realizó la selección de diferentes CNN con base a trabajos encontrados en el estado del arte, teniendo en cuenta su desempeño en el reconocimiento de objetos y la posibilidad que dan los frameworks de aceleración de ejecutar estas CNN en los sistemas embebidos. Para el caso de la red Inception V3 implementada en Ultra96 FPGA, se realizó un transfer-learning a partir del conjunto de datos INSTRE, en el caso de las redes Tinier-Yolo, Inception V1 y AlexNet, se tomaron estos modelos preentrenados a partir de los dataset Pascal Voc2012 y ImageNet. Estas CNN fueron implementadas mediante frameworks de aceleración sobre cada sistema embebido, en donde, el objetivo fue evaluar y comparar el rendimiento de cada arquitectura heterogénea.

Por otro lado, se implementaron algoritmos clásicos de aprendizaje automático sobre un sistema de cómputo secuencial que contiene las mismas características de la CPU del robot NAO. Durante este implementación, se generaron resultados comparativos con el sistema propuesto integrado por una CNN y una FPGA o GPU. Esta comparación se realizó con el objetivo de demostrar las capacidades que se le pueden otorgar a un robot humanoide al integrarle una arquitectura heterogénea y una CNN. Como una aplicación del sistema, se llevó a cabo la implementación de una CNN que permita la clasificación de juguetes en tiempo real con el objetivo de mejorar la interacción del humanoide en un ambiente pedagógico. El desarrollo de esta aplicación se basa en un re-entrenamiento de las últimas capas convolucionales de la red Inception-V3 para la identificación de juguetes. Para tener una referencia del desempeño de la red, este fue comparado con los resultados obtenidos al utilizar técnicas clásicas de clasificación y los otros modelos de CNN. Las siguientes subsecciones detallan la implementación de cada una de las CNN sobre cada sistema embebido haciendo uso de frameworks de aceleración, la implementación de algoritmos clásicos sobre un sistema de còmputo similar al del robot NAO y la evaluación de desempeño a través de una comparación de resultados con trabajos recientes del estado del arte.

### 2.3.1. IMPLEMENTACIÓN DE LAS CNN EN LOS SISTEMAS EMBEBIDOS

La aceleración de las redes neuronales convolucionales se basó en el uso de frameworks de aceleración de las CNN para cada uno de los sistemas embebidos. En el caso del sistema embebido basado en GPU se utilizaron dos frameworks: *Tensor RT* y *Darknet*. Del lado de TensorRT se tomaron los modelos preentrenados AlexNet y InceptionV1, y, mediante el uso de CUDA se realizó la transferencia y procesamiento de datos entre la CPU (host) y la GPU (device). En cuanto a Darknet, un framework de ejecución de redes neuronales de código abierto, se implementó Tinier-Yolo, en donde, el procesamiento de datos fue realizado a través de C y CUDA para el cómputo entre la CPU y GPU. Por otro lado, sobre el FPGA, fueron empleados los frameworks *VITIS-Pynq*, *DNNDK-Pynq*, *FINN* y *PipeCNN*. Al igual que en los frameworks de aceleración de CNN para GPU, en estos, se optimiza el manejo de memoria entre la CPU y el FPGA con la diferencia que utiliza Python para la ejecución en host y C++ en device. La Tabla 2.6 muestra cada uno de los sistemas embebidos utilizados, junto con el framework de aceleración y CNN ejecutada. En este trabajo, las CNN seleccionadas son ejecutadas en los dos tipos de hardware (FPGA y GPU) con el objetivo de comparar los resultados en términos de precisión, tiempo de inferencia y consumo de potencia. Adicional, la aceleración de las CNN sobre estos sistemas embebidos se orienta a aplicaciones relacionadas a educación inicial y clasificación de objetos en ambientes cotidianos.

### IMPLEMENTACIÓN DE LAS CNN SOBRE LA ARQUITECTURA HETEROGÉNEA BASADA EN FPGA

La implementación se realizó para dos marcos de aceleración diferentes. Para el desarrollo de este trabajo se adaptó el framework Pipe CNN para el sistema de desarrollo Cyclone V-SoC en la plataforma INTEL. El dispositivo Cyclone V-SoC cuenta con un procesador ARM que actúa como host y una FPGA que funciona como acelerador mediante la ejecución de un kernel implementado con OpenCL. PipeCNN utiliza dos parámetros para controlar el coste de los recursos hardware y mejorar el tiempo de ejecución. Estos parámetros son el tamaño de la vectorización de los datos y el número de unidades de computación en paralelo. Este framework también utiliza metodologías de alto nivel, pero en este caso están basadas en código OpenCL, lo que permite disponer de kernels altamente eficientes y configurables para ser adaptados a una gran variedad de modelos de CNN. PipeCNN es un acelerador de CNN en FPGA desarrollado en OpenCL. Soporta Caffenet (AlexNet), VGG-16 y ResNeT-50. El framework es reconfigurable y esto lo hace fácilmente adaptable a diferentes placas y al estar escrito en OpenCL permite una fácil implementación entre diferentes plataformas como CPU y GPU.

La arquitectura convolucional definida en el marco de PipeCNN es en cascada, en la que cada capa se ejecuta una vez que la anterior ha terminado. Otra optimización utilizada por PipeCNN es el uso de la representación aritmética de punto fijo en lugar de punto flotante. Esto reduce considerablemente el consumo de recursos en las FPGAs, aunque también se reduce la precisión de las CNN. Esta arquitectura se muestra en la Figura 2.4 tomada de [12]. Del lado izquierdo se pueden observar los kernels en cascada implementados en el FPGA, en donde estos kernels, estan diseñados para acelerar los cálculos intensivos de las CNN. Del lado derecho se puede ver la vista interna de lo que contiene cada kernel.

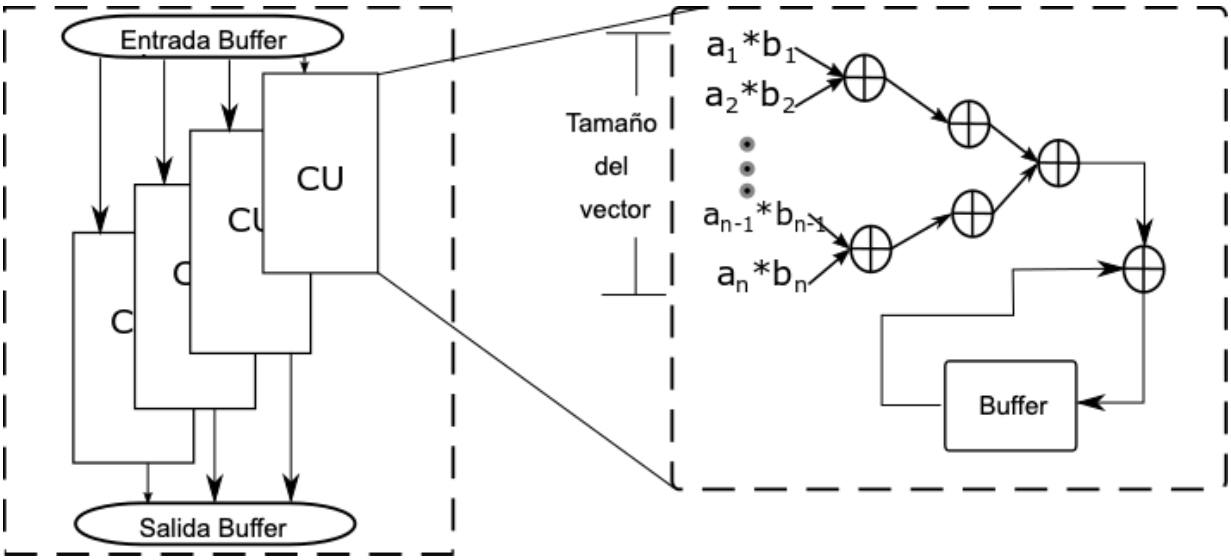


**Figura** 2.4: Arquitectura de hardware del marco PipeCNN

Para la implementación de PipeCNN, se descargó e instaló el software Intel® FPGA SDK for OpenCL desde el sitio web de Intel. Una vez instalado, se sigue la guía para crear la SD de arranque para el SoC Cyclone V, que cargará las librerías OpenCL en el ARM para que pueda comunicarse con la FPGA. Después de que el SoC tenga las librerías OpenCL instaladas procedemos a seguir la guía en el GitHub de los creadores de PipeCNN. Teniendo en cuenta que las librerías serán compiladas para ARM esto se hace cambiando en el makefile la variable PLATFORM = x86 por PLATFORM = arm32. Esta compilación generará dos ficheros, uno terminado en *.aocx que es el binario de la FPGA y un ejecutable que se encargará de cargar la imagen y enviarla a la FPGA. Estos dos archivos se pasan a la tarjeta y con los comandos de carga de Bitstream que ya están configurados en el arranque de Linux, se procede a su ejecución.

Además, se adaptó una aplicación de software libre llamada Quantized Neural Network en una plataforma XILINX utilizando una arquitectura heterogénea basada en FPGA. Esta aplicación se basa en el framework FINN presentado por [13]. Este framework permite implementar redes convolucionales en dispositivos Xilinx con una arquitectura predefinida y con una alta eficiencia para centrarse más en la implementación. Este framework está

disponible libremente para su implementación y se puede encontrar en FINN. La implementación se realiza en una placa SoC Ultra96 utilizando el framework PYNQ. El Ultra96 es una placa de desarrollo construida en torno a un dispositivo Xilinx Zynq UltraScale+ MPSoC, que incluye ARM A53 de cuatro núcleos, ARM R5 de dos núcleos, 2GB de RAM y tecnología Ultrascale+ de 16nm. El marco PYNQ se utiliza para el desarrollo rápido de código en el host. Este marco permite que las aplicaciones de alta velocidad se ejecuten lado a lado en el hardware con aplicaciones de software basadas en Python.

En el marco de FINN existen dos tipos de arquitecturas de aceleración de CNN implementadas en FPGA, estas arquitecturas se muestran en la Figura. 2.5. Una es la arquitectura DF y la otra es la arquitectura MO. La principal diferencia entre ellas es que la arquitectura DF se construye para una sola topología de CNN, pesos y activaciones ya definidos. Sin embargo, esto es ineficiente en las FPGAs porque son arquitecturas en las que se puede aprovechar el cómputo reconfigurable. Esta ventaja se aprovecha en la arquitectura MO porque, a diferencia de la arquitectura DF, no depende de la topología de la red porque el propio bloque se retroalimenta a sí mismo y se reconfigura en función de la topología de la red. La arquitectura DF tiene una ventaja sobre la arquitectura MO porque es una arquitectura dedicada y mejorada sólo para una topología concreta. Como los pesos están en el mismo sistema, el tiempo de procesamiento es mucho menor, al contrario que las MO que se configuran en cada capa de la CNN. En la arquitectura MO se pueden implementar topologías de CNN que no se pueden implementar en la arquitectura DF porque consumen más recursos de los que dispone la placa. La arquitectura MO carga cada una de estas capas secuencialmente en la FPGA y así no ocupa todo el espacio de recursos, especialmente la BRAM.

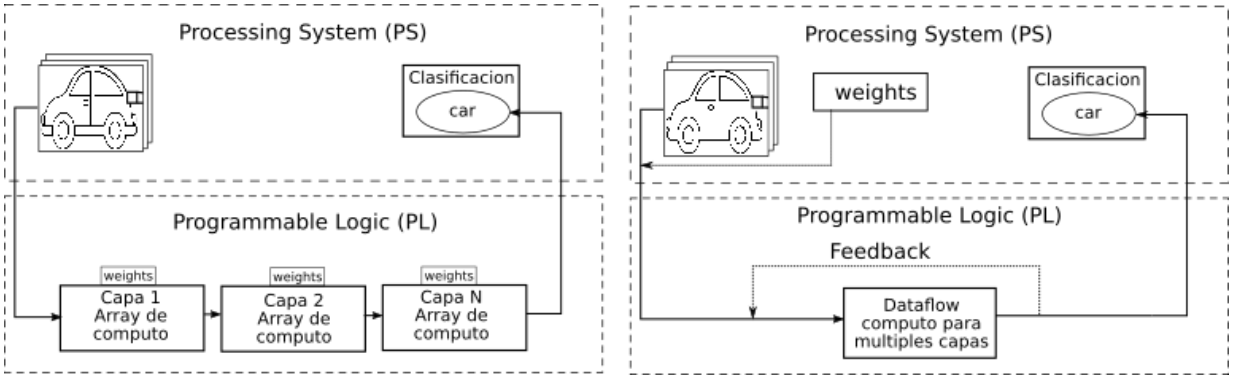


**Figura** 2.5: Tipos de arquitecturas del marco FINN (a) Arquitectura DF con capas y pesos definidos. (b) Arquitectura MO con capas y pesos para diferentes precisiones y tamaños.

Para la implementación del Tinier-Yolo en el ultra96, debes tener ya instalado el framework Pynq v2.5, es decir, tener la imagen de Linux en la SD. El tutorial para esta instalación está completamente documentado para el ultra96 en el GitHub de PYNQ. Con el framework PynqV2.5 instalado se procede a seguir la guía en el GitHub QNN-PYNQ. Esto instala las bibliotecas de Python para abrir un archivo de cuaderno llamado tiny-yolo-image.ipynb. Este archivo ejecutará Tinier Yolo que es una variante de tiny-Yolo.

La Figura. 2.6 tomada de `https://www.programmersought.com/article/18245807398/`, muestra la arquitectura de la red Tinier-Yolo que se implementa en el SoC Ultra96. Las capas en son las que se ejecutan en hardware y las otras capas se ejecutan en Python. Las capas consisten en operaciones comunes como convoluciones y max pooling. El framework sólo soporta capas cuantizadas, lo que significa que los pesos y las activaciones se representan de 1 a 3 bits. La Tinier-Yolo es una versión modificada del sistema de detección de objetos Tiny Yolo. La CNN Tinier-Yolo también está entrenada con la base de datos PASCAL VOC, pero con un 1 bit para los pesos y 3 bit para las activaciones. Tinier-yolo tiene como mAP un 50.1% con respecto al 57.1% de mAP que tiene Tiny-yolo.

Por otro lado, se utilizó la Vitis-DPU de Xilinx para la ejecución de la CNN Inception-V1 en la FPGA Ultra96-V2. La DPU es un motor programable dedicado a la ejecución de cada una de las capas convolucionales presentes en una CNN a través de un módulo de configuración de registros, un módulo controlador de datos y un módulo de cálculo de convolución. Este módulo DPU está integrado como una unidad lógica programable (PL) que se conecta al sistema de procesamiento (PS). Al igual que el marco FINN, junto con la DPU, PYNQ se utiliza

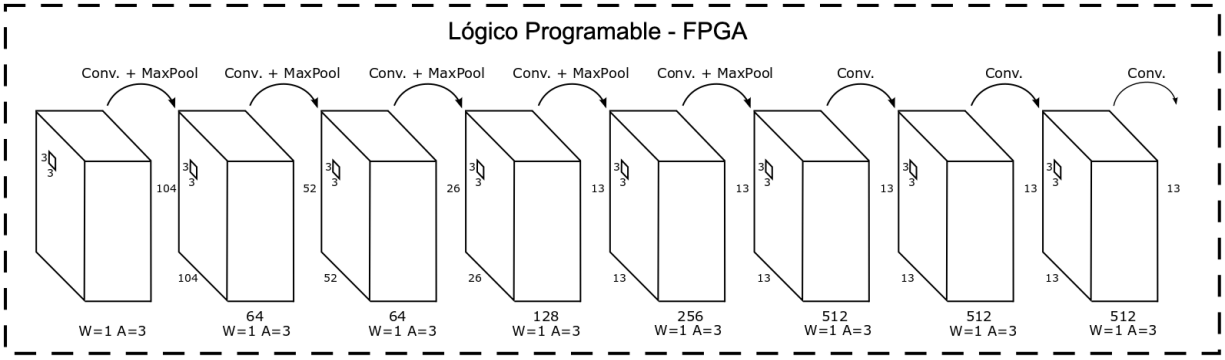


**Figura** 2.6: Topología de red Tinier-yolo implementada con el marco QNN

para el desarrollo de aplicaciones del lado del host a través de una interfaz basada en AX14. Vitis IA se utiliza para convertir el modelo entrenado en Tensorflow a formato *.elf, este formato contiene la información de los pesos, que será leída por el ARM de la FPGA SoC y desde allí se encarga de enviar las tareas a la DPU para que las procese y envíe el resultado de vuelta. Puedes encontrar archivos *.elf para modelos entrenados como Mnist, Resnet50 y YoloV3, sin embargo, para comparar con la Jetson TX2 hemos seleccionado Inception-V1 como red de prueba.

Por último, se presenta un sistema para el mejoramiento de la percepción visual del robot NAO que permita su uso en actividades pedagógicas para educación inicial. En este trabajo se propone una primera aproximación mediante el reconocimiento de juguetes por parte del robot, permitiendo una interacción amigable con los niños en una actividad pedagógica. Para esto, se seleccionó la red Inception-V3, la cual puede clasificar hasta 1000 clases, lo que le permite al robot una mayor comprensión de su entorno en un ambiente de interacción. Esta red no detecta los objetos en la imagen como Tinier-Yolo, sin embargo, en esta aplicación consideramos un factor clave el número de clases, y aunque algunas redes podrían tener mejor desempeño en este aspecto, necesitan mas recursos de hardware que los existentes en la Ultra96.

Para la clasificación de juguetes, se construye un dataset de 10 clases a partir de la base datos INSTRE [14]. En cada una de las clases seleccionadas, se definió un conjunto de entrenamiento que consta de 70 a 100 imágenes y un conjunto de validación entre 30 y 40. El dataset completo consta de 810 imágenes de entrenamiento y 356 imágenes de validación.

El entrenamiento para la clasificación de juguetes es desarrollada mediante un *transfer-learning*, el cual, es un método que consiste en reutilizar el entrenamiento anterior de una red congelando algunas capas y seleccionando una tasa de aprendizaje menor para adaptar los pesos al nuevo conjunto de entrenamiento. El *transfer-learning* permite disminuir el tiempo de entrenamiento y utilizar bases de datos pequeñas debido a que la mayoría de características como bordes, curvas y colores ya vienen definidas por los pesos entrenados por las primeras capas. En este trabajo, se utiliza *transfer-learning* con la red convolucional Inception-V3, la cual fue originalmente entrenada con el dataset Imagenet para reconocer 1000 tipos de clases. Esta topología consiste en 310 capas convolucionales y capas completamente conectadas. Para el entrenamiento con *transfer-learning* se congelan las primeras 172 capas convolucionales de la red, la capa de salida de Inception-V3 es una activación softmax con 1000 clases. En este trabajo, se elimina esta capa y se reemplaza con una capa softmax de 10 clases, por lo cual, esta capa es entrenada junto con las capas densas restantes para la aplicación centrada en la clasificación de 10 tipos de juguetes.

Esta red fue implementada sobre la Ultra96 utilizando el framework DNNDK, el cual implementa un módulo DPU que permite una alta computación y fácil acceso para programación heterogénea. DNNDK consiste de varias herramientas como lo son el DECENT y el DNNC. El DECENT es la herramienta que se encarga de reducir el tamaño de las CNN en términos de datos, es decir, un modelo que se guarda en punto flotante de 32 bits es cuantizado a 8 bits para cada peso. Al reducir esta información, se reduce el tamaño del modelo, permitiendo una optimización en términos de eficiencia computacional, eficiencia energética y menos memoria para el sistema, especialmente en el ancho de banda durante la transmisión de datos desde el host al FPGA. El DNNC es el encargado de mejorar los recursos utilizados por la DPU optimizando el ancho de banda y el con-

sumo de potencia. El DNNC utiliza el modelo depurado del DECENT y le aplica técnicas de transformación y compilación optimizada como lo son la fusión de nodos de computación, planeación de instrucciones eficiente y reuso de las características y los pesos de la memoria On-Chips.

#### IMPLEMENTACIÓN DE LAS CNN SOBRE LA ARQUITECTURA HETEROGÉNEA BASADA EN GPU

Se utilizaron dos marcos para la aceleración de las CNN en la Jetson TX2. En el primero, TensorRT, se implementaron las redes Inception-V1 y Alexnet. TensorRT consta de dos etapas: etapa de entrenamiento y etapa de inferencia. La etapa de entrenamiento se basa principalmente en el uso de Digits, donde es posible realizar el entrenamiento, la gestión de datos y la evaluación del modelo ejecutándolo en la nube o en un localhost. Una vez realizado el entrenamiento, se descargan los pesos entrenados y se realiza la inferencia. Esta etapa no se realizó, aquí se obtuvo un modelo previamente entrenado y se realizó la optimización entre el host y el dispositivo para la etapa de inferencia.

Para la etapa de inferencia se utilizaron TensorRT, Cuda y Cudnn. TensorRT es un optimizador de un modelo entrenado con Cuda y Cudnn, consiguiendo una baja latencia, ideal para aplicaciones en tiempo real. Este marco proporciona una operación de cuantificación para el motor de inferencia de la GPU. La latencia computacional se acorta debido a las operaciones aritméticas flotantes y, para no reducir el MAP del modelo, los pesos se cuantificaron a 16 bits en la etapa de inferencia.

TensorRT modifica el tamaño de las imágenes antes y después del proceso de inferencia. Dado que esta modificación es computacionalmente costosa en la CPU, el marco utiliza *multithreading* en la CPU para acelerar el proceso. TensorRT genera dos hilos en cada núcleo de la CPU y cada hilo procesa un lote de datos. El Jetson TX2 tiene 6 núcleos de CPU por lo que TensorRT crea 12 hilos. La inferencia en la GPU sólo puede funcionar en un único hilo; por lo tanto, el marco toma la inferencia como un proceso mutuo y los diferentes hilos deben competir por la GPU. Se realizan pruebas de detección y se obtienen datos satisfactorios en tiempo real en términos de fps. Al hacer uso de este marco, se obtiene una alta precisión y un bajo consumo de energía. Por último, para la implementación de la red Tinier-Yolo se utilizó Darknet, que es un framework de código abierto para la ejecución de redes neuronales convolucionales, donde el procesamiento de los datos se realiza a través de C y CUDA para el cálculo entre la CPU y la GPU.

### 2.3.2. IMPLEMENTACIÓN DE ALGORITMOS CLÁSICOS DE APRENDIZAJE AUTOMÁTICO PARA APLICACIONES EN EDUCACIÓN INICIAL

Se realiza la implementación de clasificadores clásicos con la misma base de datos INSTRE [14]. Los clasificadores fueron implementados utilizando Scikit-Learn en lugar de frameworks populares como Keras y TensorFlow, esto, ya que la arquitectura de 32 Bits del sistema de cómputo de NAO no permite la instalación y configuración de estas herramientas. Esta restricción limita la ejecución de algoritmos como las CNN sobre el sistema secuencial del robot NAO, reduciendo su aplicabilidad en actividades que requieren una mayor comprensión del entorno. Se utiliza cada píxel como una característica por, lo cual para los modelos de clasificación se contaban con 200x200x3 características. Dentro de los cinco modelos clásicos implementados se encuentra la Regresión logística, Nayve Bayes, Árbol de desición y Random forest. Estos modelos han sido utilizados en aplicaciones relacionadas a clasificación binaria y multiclase de objetos. En imágenes, estos modelos presentan un buen desempeño cuando se realiza la optimización de hiperparámetros, sin embargo, esta labor es compleja y depende de las características del dataset. Las CNN, a través de sus capas ocultas tienen la capacidad de seleccionar las características mas importantes sin dificultad, por lo que se obtienen mejores resultados en términos de precisión cuando el dataset es considerable.

## 2.4. EVALUACIÓN DE DESEMPEÑO

Esta sección presenta la descripción de las métricas utilizadas para la evaluación del desempeño de cada sistema embebido al realizar la ejecución de las CNN. Se presenta la precisión para evaluar el rendimiento de la arquitectura de la red, y el tiempo de ejecución y consumo de potencia para la evaluación de las arquitecturas heterogéneas durante la etapa de inferencia. La selección de la precisión como métrica de evaluación, se basó en su amplio uso en aplicaciones de aprendizaje automático relacionadas con el reconocimiento de objetos. Respecto al tiempo de ejecución y consumo de potencia, han sido ampliamente utilizadas en la evaluación de aplicaciones a nivel de hardware, en las que se busca desempeño en tiempo real con bajo consumo de energía.

En esta tesis, el mejoramiento de la percepción visual del robot humanoide NAO sen basó en tres puntos: (I) mayor reconocimiento de objetos en un ambiente cotidiano o pedagógico, (II) capacidad de reconocimiento de los objetos en tiempo real y (III) reconocimiento de objetos en tiempo real manteniendo la autonomía a través de un bajo consumo de potencia. Teniendo en cuenta lo anterior, la evaluación de desempeño y las comparaciones con el estado del arte se basaron en cada una de estas tres métricas. La aplicación de estas métricas en conjunto proporciona una base para el desarrollo de aplicaciones enfocadas al mejoramiento de la percepción visual en robots humanoides a través de la integración de sistemas computacionales externos que ejecutan algoritmos para el reconocimiento de objetos.

### 2.4.1. PRECISIÓN

En el trabajo actual se realiza la implementación de diferentes CNN para la clasificación o detección de objetos. Se evalúan las arquitecturas Inception-V1, Inception-V3 y Alexnet para tareas de clasificación de objetos. Por otra parte, se realiza la implementación de la CNN Tinier-Yolo para la detección de objetos. El objetivo de los tipos de aplicación mencionados anteriormente se basa en la identificación de objetos, sin embargo, tienen diferencias en cuanto a lo que identifican de la escena y en el como se evalúa su rendimiento. Durante la clasificación de imágenes se identifica si un objeto está presente en la imagen y la clase del objeto. Respecto a la detección de objetos se realiza una tarea de clasificación y localización, la cual predice las coordenadas del cuadro delimitador alrededor del objeto cuando está presente en la imagen. Con relación a la evaluación de la CNN para la detección de objetos es utilizada la precisión media (mAP), ya que para este tipo de aplicaciones se debe localizar y clasificar un número variable de objetos en una imagen que indica que en la salida del objeto la detección puede cambiar de una imagen a otra. Finalmente, para esta aplicación, se toman modelos preentrenados, los cuales fueron evaluados a partir de la precisión o mAP para clasificación o detección de objetos respectivamente.

### 2.4.2. TIEMPO DE INFERENCIA

El tiempo de inferencia consiste en la duración que tarda un sistema de cómputo al procesar una imagen de entrada y obtener una salida. Para el caso de la aplicación, está asociado al tiempo necesario para el reconocimiento o ubicación de un objeto en un espacio dado. La métrica utilizada refleja la ejecución completa de la red convolucional, partiendo desde la entrada de la imagen e incluyendo el tiempo necesario para transmitir datos desde la CPU hacia el FPGA o GPU de cada sistema embebido. Las medidas fueron tomadas mediante el uso de librerías de Python que se ejecutaron dentro de la CPU de cada arquitectura heterogénea.

### 2.4.3. CONSUMO DE POTENCIA

En arquitecturas basadas en FPGA y GPU, el consumo de potencia depende de la cantidad de elementos lógicos o núcleos utilizados durante la aplicación. Para el caso del FPGA, el consumo de potencia se obtiene con la estimación proporcionada por los entornos de desarrollo Vivado y Quartus. Respecto a la medición de potencia del lado del GPU, al tomar modelos ya ejecutados en tarjetas como la Jetson TX2, se toman los resultados proporcionados por los autores de estas implementaciones.

## REFERENCIAS

[1] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, "Gpu scheduling on the nvidia tx2: Hidden details revealed," in *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2017, pp. 104–115.

[2] D. Franklin, "Nvidia jetson tx2 delivers twice the intelligence to the edge (2017)," *URL: https://devblogs. nvidia. com/jetson-tx2-delivers-twiceintelligence-edge/(visited on 02/11/2019)*, 2018.

[3] A. Kalantar, Z. Zimmerman, and P. Brisk, "Fa-lamp: Fpga-accelerated learned approximate matrix profile for time series similarity prediction," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 40–49.

[4] V. Cyclone, "Hard processor system technical reference manual," 2015.

[5] O. Michel, "Cyberbotics ltd. webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[6] E. Pot, J. Monceaux, R. Gelin, and 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*. IEEE, 2009, pp. 46–51.

[7] C. Li, E. Imeokparia, M. Ketzner, and T. Tsahai, "Teaching the nao robot to play a human-robot interactive game," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2019, pp. 712–715.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[10] S. Wang and S. Jiang, "INSTRE: a new benchmark for instance-level object retrieval and recognition," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 11, no. 3, p. 37, 2015.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[12] D. Wang, K. Xu, and D. Jiang, "Pipecnn: An opencl-based open-source fpga accelerator for convolution neural networks," in *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 279–282.

[13] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[14] S. Wang and S. Jiang, "INSTRE: a new benchmark for instance-level object retrieval and recognition," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 11, no. 3, p. 37, 2015.

# 3

# RESULTADOS

La siguiente sección muestra los resultados obtenidos en cada fase del trabajo de investigación. Para la fase de adquisición y comunicación entre el robot NAO y los sistemas embebidos se toman métricas como FPS y Kbps. En la fase de aceleración de la CNN se muestran los resultados teniendo en cuenta el tipo de aplicación (detección o clasificación de objetos) implementado en cada arquitectura heterogénea. Por otro lado, se muestran los resultados de la implementación de algoritmos clásicos sobre la CPU del robot NAO con el objetivo de hacer una comparación entre estos resultados y los obtenidos en la aceleración de las CNN sobre sistemas embebidos basados en FPGA o GPU. Adicional, a través de una tabla se realiza la comparación de los resultados obtenidos versus el estado del arte descrito anteriormente. Finalmente, como resultado, se presenta el diseño de un *backpack* para NAO, el cual contiene una arquitectura heterogénea y una batería que la alimenta, esto, con el objetivo de habilitar un sistema computacional externo que le brinde a NAO una buena precisión mientras mantiene su autonomía y movilidad.

## 3.1. ADQUISICIÓN Y COMUNICACIÓN

Se realizaron tres pruebas distintas variando la resolución de la imagen adquirida y el factor de calidad de la codificación y decodificación. En cada prueba se toman datos del número de frames por segundo (fps) y la velocidad de transmisión medida en kbps (kbits por segundo). La Tabla 3.1 muestra cada uno de los resultados para cada resolución. Se puede observar que a medida que crece la resolución de la imagen de entrada, los cuadros por segundo disminuyen. Pese a esto, se obtienen fps en tiempo real para las resoluciones 640x480 (resolución estándar de la cámara de NAO) y 224x224 (ImageNet). Al centrarse el trabajo de investigación en el robot humanoide NAO, se toma esta resolución como punto de entrada al sistema de procesamiento de datos dado para cada arquitectura heterogénea al ejecutar la CNN.

La Figura 3.1 muestra la escena virtual proporcionada por la herramienta de simulación Webots y la Figura 3.2 muestra la misma escena en Choreographe, la cual, posteriormente fue transmitida a cada sistema embebido.

La Figura 3.3 se toma como ejemplo para mostrar que la misma imagen generada por Webots es recibida por cada sistema embebido, en este caso, se presenta la detección de una persona la plataforma Jetson TX2.

**Tabla** 3.1: FPS y Kbps en la transmisión y recepción de la imagen

| **Resolución imagen** | **Transmisión** | | **Recepción** | |
|---|---|---|---|---|
| | 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 |

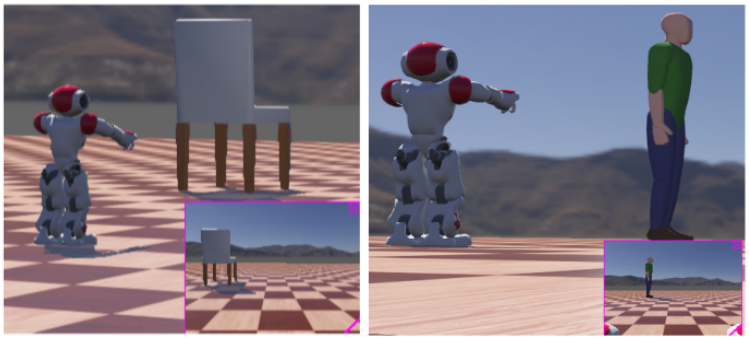

**Figura** 3.1: Ambiente virtual proporcionado por Webots

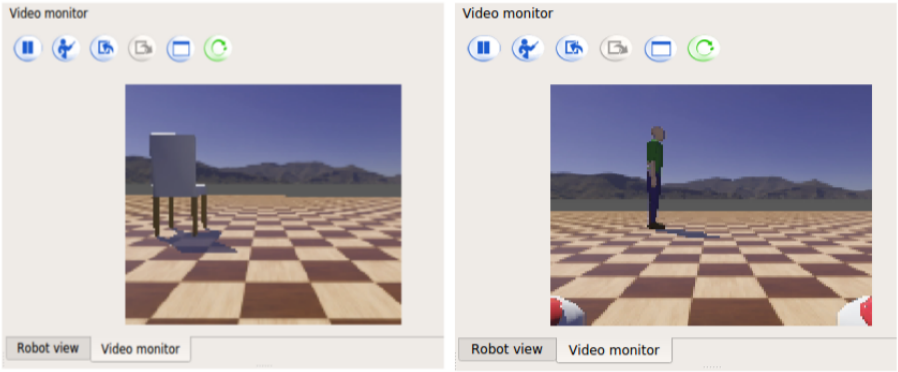


**Figura** 3.2: Visualización de imagen virtual en Choregraphe

## 3.2. PROCESAMIENTO DE DATOS

Los resultados de esta sección se muestran teniendo en cuenta la implementación de cada red neuronal convolucional en cada sistema embebido. Se toman métricas como cuadros por segundo, tiempo de inferencia y consumo de potencia al ejecutar las CNN en cada sistema embebido. Estas métricas son también tomadas al momento de implementar algoritmos clásicos de visión por computador en la CPU del robot NAO, en donde, como se mencionó anteriormente, el objetivo de estas implementaciones fue comparar los resultados al integrar una CNN-FPGA/GPU a NAO versus técnicas clásicas-CPU NAO.

### 3.2.1. IMPLEMENTACIÓN DE LAS CNN TINIER-YOLO, INCEPTION-V1 Y ALEXNET SOBRE EL FPGA PARA DETECCIÓN Y CLASIFICACIÓN DE OBJETOS

Los resultados en términos de tiempos de ejecución y cuadros por segundo al ejecutar las dos arquitecturas de CNN en la Ultra96 se muestran en la Tabla 3.2. Estos resultados muestran que la CNN Tinier-Yolo obtuvo un tiempo de ejecución de 0.083 s y 12.04 fps, lo que indica que la Ultra96 FPGA solo puede procesar para esta red 12 imágenes por segundo. De lado de la CNN Inception-V1 se obtuvieron fps que se encuentran dentro de los intervalos que definen una aplicación en tiempo real que son entre 20 y 30 cuadros por segundo, indicando que es posible integrar al robot humanoide NAO una arquitectura heterogénea basada en FPGA con una CNN como la red Inception-V1 para aplicaciones relacionadas a la HRI que requieren una mayor comprensión del entorno y respuestas rápidas. A pesar de que las tareas de cada red son diferentes, según los resultados, se puede ver que para arquitecturas de CNN centradas en clasificación, la Ultra96 tiene un mejor desempeño en tiempo real. Caso contrario ocurre al ejecutar una CNN para un problema centrado en la detección de objetos, donde tan solo se obtienen 12.04 fps al ejecutar Tinier-Yolo.

La Figura 3.4 muestra la detección en el sistema embebido a partir de la escena virtual proporcionada por Webots y la Figura 3.5 la detección en tiempo real.

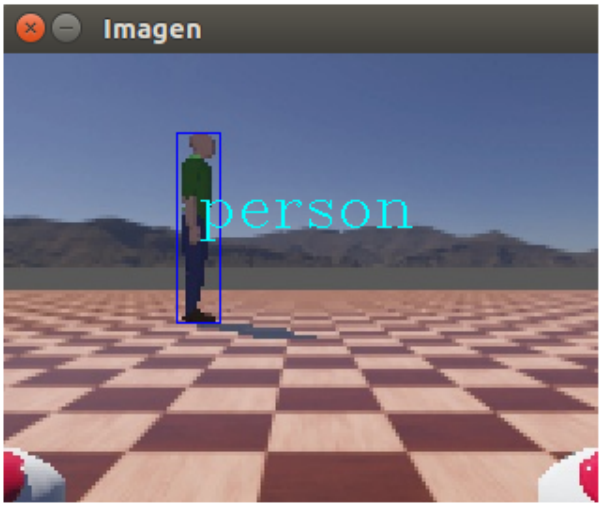


**Figura** 3.3: Visualización de imagen virtual en el sistema embebido

**Tabla** 3.2: Tiempos de ejecución y cuadros por segundo al ejecutar Tinier-Yolo y Inception-V2 en la Ultra96 FPGA

| Modelo | Tiempo de ejecución (s) | Cuadros por segundo (fps) |
|---|---|---|
| Inception-V1 | 0.033 | 30.3 |
| Tinier-Yolo | 0.083 | 12.04 |

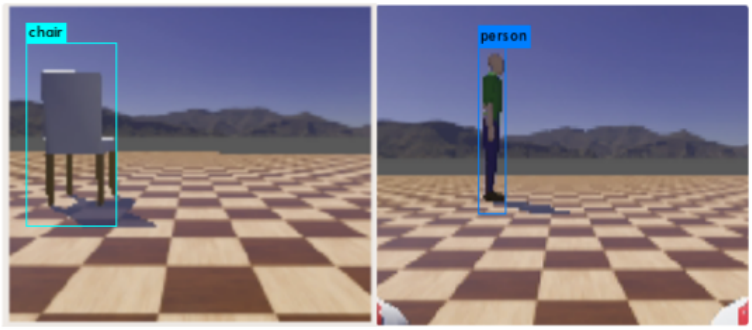


**Figura** 3.4: Detección de objetos: Ultra96 y Tinier-Yolo integrado a visión del robot humanoide NAO proporcionado por Webots

Respecto a la ejecución de AlexNet sobre la tarjeta de desarrollo Cyclone V-SoC, se obtuvo un tiempo de inferencia de 0.205 s por imagen, por lo que la FPGA pudo procesar 4.87 imágenes por segundo. Se puede observar que en esta implementación, no se obtuvieron fps en tiempo real, ya que se encuentra por debajo de los 30 fps. En la Tabla 3.3, se muestra la comparación en términos de fps de diferentes implementaciones de AlexNet en diferentes plataformas versus los resultados obtenidos en el presente trabajo.

La potencia consumida por la Cyclone V-SoC fue estimada con el software Intel Quartus®Prime Power Analyzer. El número máximo de unidades de cómputo que pueden implementarse en el Cyclone V-SoC es de cuatro. En la Tabla 3.4, se puede observar que los recursos utilizados están por debajo de la cantidad total contenida en la placa. Esto se debe a que la placa sólo contiene 4192 lab, los cuales, se han agotado en esta implementación. Por esta razón, no es posible aumentar el rendimiento en el Cyclone V-SoC, a diferencia de la implementación realizada por [1] en la plataforma DE5-net, en donde, se logra una clasificación en tiempo real, ya que la plataforma es mucho mayor en términos de recursos.


**Figura** 3.5: Detección de objetos: Ultra96 - Tinier-Yolo

**Tabla** 3.3: Comparación de FPS al implementar AlexNet sobre diferentes plataformas

| | **PipeCNN** | PipeCNN [1] | CUDA [2] | AlexNet [3] | Ncsdk [2] |
|---|---|---|---|---|---|
| Plataforma | **Cyclone V-SoC** | DE5-net | Jetson TX2 | Stratix-V | Movidius |
| FPS | **4.87** | 66 | 70 | 864.7 | 11 |

### 3.2.2. IMPLEMENTACIÓN DE LA RED INCEPTION-V3 SOBRE EL FPGA PARA APLICACIONES EN EDUCACIÓN INICIAL

La implementación modificada de la red Inception-V3 logró un porcentaje de clasificación del 98% y un desempeño en tiempo real de 27 fps. Esta arquitectura fue implementada a una frecuencia de reloj de 370 MHz usando el DPU. A nivel de entrenamiento, la Figura 3.6 muestra la disminución de la función de costo obtenida a medida que incrementan las épocas y la exactitud de la red con respecto a las épocas. Fue utilizado el optimizador SGD con un *learning-rate* de 0.0001 y *momentum* de 0.9; como función de costo fue usado *categorical_crossentropy*. Para el entrenamiento con *transfer-learning* se congelan las primeras 172 capas convolucionales de la red, la capa de salida de Inception-V3 es una activación softmax con 1000 clases. En este trabajo, se elimina esta capa y se reemplaza con una capa softmax de 10 clases, por lo cual, esta capa es entrenada junto con las capas densas restantes para la aplicación centrada en la clasificación de 10 tipos de juguetes. En ambas gráficas, el azul corresponde a los datos de entrenamiento y en rojo los de validación. El resultado del entrenamiento realizado en Keras fue convertido al framework DNNDK utilizando las herramientas DECENT y DNNC, en el cual, se creó el archivo elf que contiene toda la información de la CNN (pesos y arquitectura). Durante la inferencia, este archivo fue leído por el ARM del SoC FPGA, el cual, se encargó de enviar las tareas al DPU para que este procesara y enviara el resultado devuelta.

Los recursos utilizados en la Ultra96 para el framework QNN y el IP DPU se muestran en la Tabla 3.5. La Tabla 3.8 muestra que al realizar la etapa de inferencia, el sistema embebido tarda 0.037 segundos, lo que equivale a 27 fps.

### 3.2.3. IMPLEMENTACIÓN DE LA RED TINIER-YOLO, INCEPTION-V1 Y ALEXNET SOBRE LA GPU PARA DETECCIÓN DE OBJETOS

Se tomaron dos imágenes de prueba para las clases persona y silla y se obtuvieron métricas de desempeño en cuanto a cuadros por segundo, tiempo de inferencia y consumo de potencia. La Tabla 3.6 muestra los resultados obtenidos. Se puede ver que se toman métricas de desempeño cuando la resolución de la imagen de entrada es de 640x480 VGA, que es la resolución estándar de la cámara del robot NAO. Los resultados en cuanto a cuadros por segundo evidencian que la Jetson TX2 al ejecutar la CNN Tinier-Yolo, tan solo procesa 9.98 cuadros por

**Tabla** 3.4: Recursos consumidos en la Cyclone V-SoC al utilizar el framework PipeCNN

| PipeCNN | | | |
|---|---|---|---|
| Recurso | Usados | Disponible | Porcentaje de uso |
| LUT | 48173 | 110000 | 43.79% |
| FF | 66830 | 219144 | 30.50% |
| RAM | 285 | 514 | 55.45% |
| DSP | 35 | 112 | 31.25% |
| Consumo de potencia | | 2.056W | |

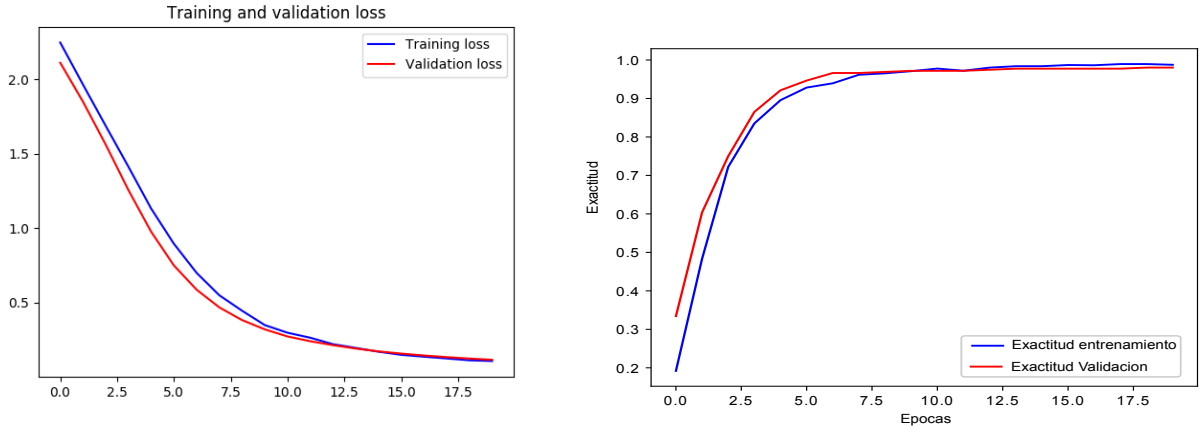


**Figura** 3.6: a) Función de pérdida a través de las épocas en el entrenamiento. b) Exactitud a través de las épocas en el entrenamiento.

segundo, lo cual, se encuentra por debajo dentro de los intervalos esperados para una aplicación en tiempo real (mayor a 20 fps). Respecto a las CNN Inception-V1 y AlexNet, se obtienen fps satisfactorios, lo que permite que estas redes sean utilizadas para problemas de clasificación de imágenes en tiempo real aplicado a robots humanoides.

La Figura 3.7 muestra la detección en el sistema embebido a partir de la escena virtual proporcionada por Webots y la Figura 3.8 la detección en tiempo real.

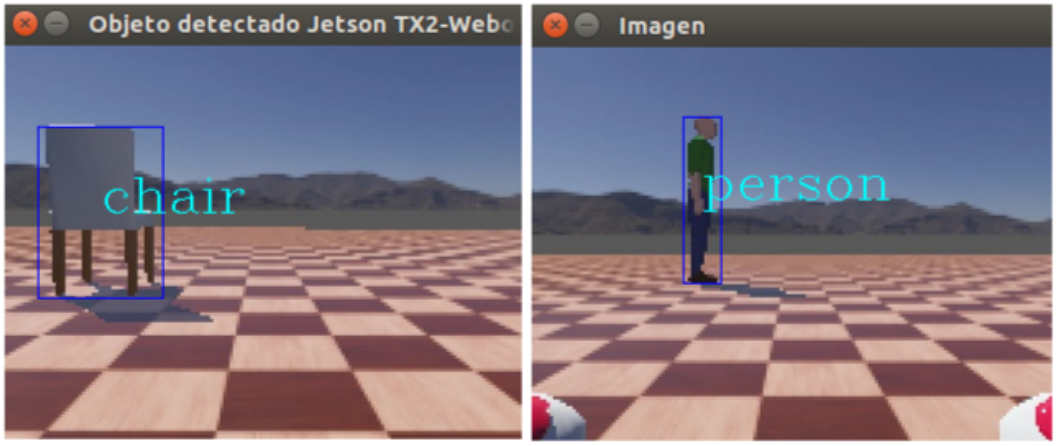


**Figura** 3.7: Detección de objetos: Jetson TX2 - Tinier-Yolo integrado a visión del robot humanoide NAO proporcionado por Webots

**Tabla** 3.5: Recursos del framework QNN y el IP DPU Ultra96 FPGA

| Recursos | QNN | DPU | Disponibles |
|---|---|---|---|
| LUT | 29754 | 37055 | 70560 |
| Register | 36050 | 72850 | 141120 |
| Block RAM | 200 | 161.5 | 216 |
| DSP | 56 | 290 | 360 |
| Consumo de potencia | 3.1 W | - | - |

**Tabla** 3.6: Desempeño Jetson TX2

| Métrica | Tinier-Yolo | Inception-V1 | AlexNet |
|---|---|---|---|
| Resolución | | 640x480 VGA | |
| Tiempo de ejecución (s) | 0.1014 | 0.035 | 0.035 |
| Cuadros por segundo (fps) | 9.86 | 28.5 | 28.5 |
| Consumo de potencia (W) | 0.84 | 4.8 [4] | 5.6 [4] |


**Figura** 3.8: Detección de objetos: Jetson TX2 - Tinier-Yolo

### 3.2.4. IMPLEMENTACIÓN DE ALGORITMOS CLÁSICOS DE APRENDIZAJE AUTOMÁTICO PARA APLICACIONES EN EDUCACIÓN INICIAL

Un computador DELL con procesador ATOM 1.6 GHZ y 1 GB de RAM fue utilizado para la etapa de inferencia de cada uno de los clasificadores clásicos implementados. En la Tabla 3.7 se presentan los resultados de estos clasificadores en términos de precisión y tiempo de entrenamiento en segundos. Para esta aplicación, se utiliza cada pixel como una característica, por lo cual, cada modelo de clasificación cuenta con 200x200x3 características. Se evidencia que al entrenar cada uno de los clasificadores clásicos se obtuvo una precisión entre el 70% y el 85% en el conjunto de validación.

### 3.2.5. COMPARACIÓN EJECUCIÓN CNN-FPGA/GPU VERSUS TÉCNICAS CLÁSICAS-CPU NAO

Para las pruebas de inferencia son tomadas cinco imágenes aleatorias de la base de datos de prueba. La Tabla 3.8 muestra la comparación al integrar una CNN-FPGA/GPU y al ejecutar algoritmos clásicos sobre el sistema secuencial del robot humanoide NAO. Se puede observar que al ejecutar las arquitecturas Inception-V1 y Inception-V3 *transfer-learning* se obtienen resultados en tiempo real. Caso contrario ocurre con las implementaciones

**Tabla** 3.7: Precisión y tiempo de entrenamiento de clasificadores clásicos

| Modelo | Precisión | Tiempo de entrenamiento (s) |
|---|---|---|
| Naive_Bayes | 0.54 | 0.02 |
| Logit | 0.84 | 4.96 |
| Desicion tree | 0.76 | 0.59 |
| Random forest | 0.81 | 2.97 |

**Tabla** 3.8: Comparación tiempos de ejecución y cuadros por segundo CNN-FPGA/GPU vs Técnicas clásicas-ATOM Z530

| Hardware | Tarjeta de desarrollo | Arquitectura | Modelo | Tiempo de ejecución (s) | Cuadros por segundo (fps) |
|---|---|---|---|---|---|
| **FPGA** | Cyclone V-SoC | CNN | AlexNet | 0.20 | 4.87 |
| | Ultra96 | | Tinier-Yolo | 0.083 | 12.04 |
| | | | Inception-V1 | 0.033 | 30.3 |
| | | | Inception-V3 transfer-learning | 0.037 | 27.02 |
| **GPU** | Jetson TX2 | | Tinier-Yolo | 0.10 | 9.86 |
| | | | Inception-V1 | 0.035 | 28.5 |
| | | | AlexNet | 0.035 | 28.5 |
| **CPU** | Intel Atom Z530 | Técnicas clásicas | Desicion Tree | 0.253 | 3.95 |
| | | | Logit | 0.504 | 1.98 |
| | | | Nayve Bayes | 9 | 0.0001 |
| | | | Random Forest | 0.316 | 3.16 |

restantes, en donde los cuadros por segundo se ubicaron muy por debajo de los establecidos para una aplicación en tiempo real. En el caso de la CNN Tinier-Yolo ejecutada en la FPGA y GPU, a pesar la cuantificación de sus pesos se obtuvo FPS menores, en donde, el FPGA fue capaz de procesar mas imágenes por segundo que la GPU para esta arquitectura de red neuronal convolucional. Por otro lado, se puede ver que el sistema secuencial del robot humanoide NAO tiene limitantes de procesamiento al ejecutar técnicas clásicas de visión por computador. A pesar de que son modelos mucho mas livianos, lo máximo que pudo procesar la CPU de NAO fue 3.95 FPS para *Desicion Tree*. Esta limitante, se presenta ya que cada píxel se utiliza como una característica para cada modelo de clasificación, el cual, cuenta con 200x200x3 características (tamaño imagen de entrada). Finalmente, las CNN son arquitecturas de mayor tamaño, por esta razón, el sistema de procesamiento de NAO estaría restringido al computar cada una de las capas y pesos que contiene una CNN.

### 3.2.6. COMPARACIÓN EJECUCIÓN CNN-FPGA/GPU VERSUS ESTADO DEL ARTE

La siguiente sección compara los resultados de la tesis actual versus los trabajos del estado del arte recientes, los cuales se enfocan en la implementación de diferentes CNN para el mejoramiento de la percepción visual de NAO. La comparación se basó en tres métricas fundamentales: tiempo de inferencia del sistema de cómputo al ejecutar la CNN, precisión de la CNN y autonomía del robot humanoide. La Tabla 3.9 muestra los resultados de los trabajos recientes encontrados en el estado del arte y la tesis actual, en donde, se observan los resultados obtenidos en términos de cuadros por segundo.

### 3.2.7. LIMITACIONES EN TIEMPO DE INFERENCIA

Los trabajos propuestos han integrado una CNN con el objetivo de mejorar la capacidad de percepción visual del robot humanoide, sin embargo, se han encontrado limitaciones que restringe al robot a utilizarse en aplicaciones de tiempo real. En [5] se implementó una CNN compuesta por una capa convolucional, pooling, normalización, conexión completa y softmax para la validación y reconocimiento de la imagen extraída en la etapa de segmentación. Los autores, a pesar de haber realizado una etapa de preprocesamiento basada en funciones de segmentación y extracción con el objetivo de reducir el costo computacional, obtienen un desempeño en

**Tabla** 3.9: Comparación cuadros por segundo CNN-FPGA/GPU versus estado del arte

| Hardware | Modelo | Cuadros por segundo (fps) | Autores |
|---|---|---|---|
| FPGA | AlexNet | 4.87 | Propuesto |
| | Tinier-Yolo | 12.04 | |
| | Inception-V1 | 30.3 | |
| | Inception-V3transfer-learning | 27.02 | |
| GPU | Tinier-Yolo | 9.86 | |
| | Inception-V1 | 28.5 | |
| | AlexNet | 28.5 | |
| CPU NAO | Propuesto por autores | 11-19 | [5] |
| | AlexNet transfer-learning | 33.3 | [6] |
| | VGG-Face | 4.16 | |
| | XNOR-Net | 4395 | [7] |
| | SqueezeNet | 14.6 | |
| | Bynari-8 | 140 | [8] |
| Sistema distribuido externo | Yolo-V3 | 58 | [9] |
| | Técnicas clásicas | 25 | [10] |
| | SSD_MobileNet_VI_COCO | 2 | [11] |

términos de cuadros por segundo máximo 19 fps, el cual, se encuentra por debajo de lo establecido para una aplicación en tiempo real. Este tiempo de inferencia fue obtenido al realizarse una clasificación binaria (robots NAO o no NAO), el cual, puede aumentar si el número de clases y imágenes de entrenamiento crece. Un enfoque de procesamiento distinto se presentó en [11], en donde propusieron un sistema micro cloud que entrena los modelos y realiza la inferencia a partir de la imagen captada por la cámara del robot. Se obtuvieron tan solo 2 fps al transmitir los datos a través de una conexión Wifi. Fueron realizadas pruebas mediante una conexión Ethernet con el objetivo de aumentar esta tasa, sin embargo, el valor máximo obtenido fue de 7 fps.

### 3.2.8. LIMITACIONES DE TRABAJOS PREVIOS USANDO UNA CNN

Distintos trabajos han obtenido resultados satisfactorios en términos de tiempo de inferencia para una aplicación en tiempo real, sin embargo, sus resultados no fueron mayores en cuanto a precisión de la red y capacidad de reconocimiento extendida con base en la cantidad de objetos a reconocer. Los autores [6] realizaron un *transfer-learning* sobre AlexNet y VGG-face utilizando solo un ejemplo de entrenamiento por persona para una aplicación relacionada al reconocimiento facial. Los resultados mostraron que VGG-face logra mejor desempeño a una resolución baja y alta comparado con AlexNet, sin embargo, el tiempo de inferencia es de 0.24 s a comparación de los 0.03 s de AlexNet, lo cual, se encuentra muy por debajo de lo esperado para una aplicación en tiempo real. El tiempo de inferencia obtenido con AlexNet superó los 30 fps, no obstante, con una sola imagen de entrenamiento por persona no se aprovechan las capacidades que pueden otorgar las CNN al realizar el entrenamiento con una mayor cantidad de imágenes, ni se le da la capacidad al robot de reconocer el rostro en diferentes ángulos. Un problema de reconocimiento en diferentes regiones de interés se presentó en [12], aquí, se ejecutaron las redes VGG-16, VGG-19 y Inception V3 sobre la CPU de NAO para una tarea centrada en la clasificación de espacios de la casa como cocina, dormitorios y baños. Obtuvieron una precisión del 95%, sin embargo, durante las pruebas, la clasificación fue correcta cuando se ubicaba dentro de la escena un único objeto de interés, lo cual no es suficiente para el desempeño de robots sociales.

Por otro lado, la clasificación multiclase de una CNN en un sistema de percepción visual permite identificar diferentes objetos o escenas presentes en un ambiente natural enfocado a robots humanoides. Los autores [7] y [8] presentaron una clasificación binaria utilizando imágenes de entrada de tamaño reducido. En el primero, las CNN XNOR-Net y SqueezeNet fueron implementadas sobre el sistema de cómputo del robot NAO para la detección de NAOs sobre el campo de juego en la RoboCup. En este trabajo el tiempo de respuesta fue rápido para la XNOR-Net manteniendo la precisión en la detección, sin embargo, esto se produjo por la reducción de la resolución de la imagen a 24 x 24, la cual, al modificase a 640 x 480 (resolución de la cámara del robot) puede aumentar los tiempos de respuesta y perder información (características). Por otro lado, este tiempo de

respuesta podría crecer si en lugar de ser una aplicación de clasificación binaria se implementa una clasificación multiclase. El segundo trabajo implementó una CNN liviana llamada Bynari-8 es sobre la CPU de NAO logrando una precisión del 97.13% a 140 fps. Los resultados demostraron una buena precisión con tiempos de respuesta rápidos, sin embargo, al momento de agregarle mas clases a la red, la arquitectura no tiene la capacidad de reducir parámetros. Esta limitante crece el tamaño de la red y puede aumentar los tiempos de respuesta.

Finalmente, un problema de precisión de la CNN se presentó en [13], en donde se tomó un modelo preentrenado de la red tiny-yolo y se modificó la última capa para la detección de tres objetos, obteniendo un mAP de 44.3% con un tamaño de imagen de entrada de 448 x 448. A pesar de lograr resultados aceptables en términos de tiempos de inferencia, obtienen un promedio de confianza de predicción no mayor a 0.5, el cual es bajo y fue causado por falta de datos que contengan imágenes de diferentes tipos de ambiente.

### 3.2.9. LIMITACIONES EN AUTONOMÍA DE TRABAJO

La capacidad de autonomía de un robot humanoide le permite al robot desplazarse libremente en un ambiente cotidiano. A pesar de los resultados considerables que presenta un sistema externo en términos de tiempos de inferencia, su tamaño y continua conexión LAN, restringen la movilidad del humanoide. Este caso se presentó en [9], en donde se implementó Yolo-V3 en conjunto con un sistema distribuido para operar múltiples robots NAO en un motor de inferencia central que se conectó vía LAN al robot, pese a que se obtienen predicciones en tiempo real, la conexión vía Ethernet restringe al robot desplazarse con mayor facilidad en un ambiente cotidiano. Un caso similar se muestra en el trabajo de [10], en el cual, se implementó un sistema de cómputo distribuido externo que se conecta vía Wifi al robot humanoide para la ejecución de algoritmos de visión por computador. Este enfoque hizo uso de una conexión Wifi, en donde, en este tipo de conexión al no tener una conexión directa, los tiempos de respuesta del robot no serán los adecuados.

Los trabajos mencionados anteriormente presentaron limitaciones en sus resultados en cuanto a tiempos de inferencia, precisión de la red, autonomía, cantidad de clases a reconocer. La Tabla 3.9 muestra que el trabajo propuesto logra tiempos de inferencia en tiempo real al ejecutar las CNN Inception-V1 y Inception-V3 *transfer-learning* sobre la Ultra96 FPGA. Para la primera red se obtiene 30.3 fps y para la segunda 27.02 fps. Por otro lado, al ser Ethernet la conexión entre el sistema embebido y la CPU de NAO, la pérdida de datos en la transmisión y recepción de las imágenes es menor a comparación de una conexión WiFi. Ya que en el procesamiento de las CNN se presenta un tiempo de cómputo, al no perdersen suficientes datos en la transmisión, los tiempos de inferencia no se reducieron significativamente.

En este trabajo, se presenta la integración de un sistema embebido en la estructura del robot humanoide NAO. Pese a ser una conexión externa, no presenta limitantes en cuanto a movilidad del robot en un ambiente cotidiano, ya que todo el todo sistema se encuentra ubicado en un mismo lugar. Esto permite que no se restrinjan los movimientos de NAO y pueda ser utilizado en aplicaciones relacionadas a la interacción humano-robot y localización de objetos.

Finalmente, en la actual tesis de investigación fueron utilizadas arquitecturas de CNN que presentan una alta precisión y buen desempeño en tareas de clasificación y detección de objetos. La CNN Inception-V1 posee una precisón del 88.9% y en la implementación *transfer-learning* sobre la Inception-V3 se obtiene una precisión del 98%. Peso a ello, lo mas importante es la cantidad de clases con las cuales se entrenaron los pesos de la red. Al ser mayor el número de objetos a reconocer, le permite al robot humanoide NAO un mayor entendimiento de su entorno. Teniendo en cuenta las limitantes de los trabajos ya mencionados y los resultados obtenidos en el presente trabajo, se puede observar que la ejecución de las CNN sobre sistemas embebidos con alto grado de paralelismo y bajo consumo de potencia, puede otorgarle al robot humanoide un mejoramiento en la clasificación y detección de objetos para distintas aplicaciones que requieren un alto desempeño en tiempo real y mayor autonomía.

## 3.3. DISEÑO *Backpack* PARA NAO

Se presenta el diseño de un *backpack* para NAO que se integra en la espalda del robot humanoide. Esta extensión de NAO contiene una Ultra96-V2 FPGA que ejecuta algoritmos de alto costo computacional como las CNN. La selección del sistema embebido a utilizar para la creación del diseño del *backpack* se basó en los resultados obtenidos en cuanto a consumo de potencia, tiempo de inferencia y mayor precisión en la clasificación de objetos. La Ultra96 FPGA al ejecutar el *transfer-learning* de Inception V3 obtuvo como consumo de potencia 3.1 W,

27.02 fps y una precisión del 98%. Además de estos resultados frente a los obtenidos por la Jetson TX2, se suma un componente principal y es la dimensión de cada sistema embebido. La Ultra96 tiene un dimensión de 8.5cm x 5.4cm mientras que la Jetson TX2 posee un tamaño de 17cm x 17 cm.

Como referencia frente a la implementación de un *backpack* para NAO se toman los resultados presentados por [14]. Aquí se realiza un *backpack* que contiene un ODROID XU4 que ejecuta algoritmos de clasificación de objetos. Al ejecutar ORB-SLAM2 obtienen 12 fps, lo cual esta por debajo de lo establecido para una aplicación en tiempo real. En la ejecución de algoritmos de deep learning presentan problemas de memoria, esto se da ya que el hardware utilizado es un microprocesador Cortex™-A7 Octa core, el cual es secuencial y está limitado en sus capacidades frente a la ejecución de algoritmos de alto costo computacional como las CNN.

Dado lo anterior, el esquema propuesto presenta una mejora en la integración de un sistema computacional externo que le brinda un mejoramiento en la percepción visual del robot humanoide NAO en tareas que requieren un alto desempeño, autonomía y movilidad. Las siguientes subsecciones presentan los detalles del diseño realizado.

### 3.3.1. CÁLCULO BATERÍA QUE ALIMENTA AL SISTEMA EMBEBIDO

El cálculo de la batería se basó en alimentación que requiere la Ultra96 FPGA, la potencia consumida al ejecutar una CNN y la autonomía con relación a la cantidad de tiempo que el robot humanoide estará activo. Frente a este último factor, se tomó como referencia lo mencionado por [14], en donde se específica que a pesar de los 60 minutos que ofrece la batería incluida en NAO, se ha notado que en la competencia de la Robo Cup Soccer la autonomía del robot es de solo 30 minutos. Respecto a esto, el tiempo de funcionamiento de la Ultra96 FPGA fue estimado a 50 minutos de actividad.

El voltaje de entrada de la Ultra96 se encuentra en el intervalo de 8V a 18V con un máximo de corriente de 3 A. Pese a que el FPGA solo consume 3.1 W en la etapa de inferencia, se realiza una sobre estimación de la potencia consumida a 6 W. Teniendo en cuenta estos datos, a partir de la fórmula de calculo de autonomía de una batería se procede a calcular la potencia entregada por parte de la batería a utilizar.

$$T = \frac{Wb}{Wc} \tag{3.1}$$

donde T es el tiempo de autonomía en horas, Wb la potencia entregada por la batería y Wc la potencia consumida. La autonomía estimada de 50 minutos en horas equivale a 0.833 h. Teniendo en cuenta lo anterior la potencia que debe entregar la batería equivale a:

$$Wb = 4.998W \tag{3.2}$$

El paso siguiente se basó en la selección de la batería a partir del cálculo de la potencia entregada y autonomía deseada. Para esto, se optó por el uso de una batería de polímero de litio LiPo, las cuales, son baterías recargables que se componen de células secundarias idénticas en paralelo para aumentar la capacidad de la corriente de descarga. Poseen una alta velocidad de descarga, peso ligero y tamaño reducido. Estas características se adecuaron para el esquema propuesto, el cual también tiene como objetivo brindar autonomía al robot sin que su capacidad de desplazamiento se vea afectado por un *backpack* ubicado en la parte trasera de su estructura.

Una vez seleccionado el tipo de batería a utilizar, se realizó una búsqueda de las diferentes baterías LiPo presentes en el mercado. Teniendo como base que la potencia entregada por la batería debe ser de 4.998 W, se elige una batería LiPo comercial que entregue esta capacidad y que su voltaje de salida se encuentre dentro de los rangos de voltaje de entrada para la Ultra96 FGPA (8v - 18v). Se toma como referencia la tabla Lipo Battery Pack Size Chart, en la cual, se detallan las especificaciones eléctricas de cada una de las baterías LiPo presentes en el mercado. Dado que la potencia entregada debe ser de 4.998 W y sus dimensiones se adecuan a la aplicación, se seleccionó una batería Lipo de capacidad 440 mAh y voltaje 11.1, como se muestra en la Tabla 3.10.

La selección de la batería se fundamenta en la ecuación de potencia, que es:

$$P = V * I \tag{3.3}$$

donde P es la potencia, V el voltaje y I la corriente de la batería. Partiendo de que el voltaje de la batería es 11.1 V y la corriente es 0.45 A, se obtiene una potencia de 4.99 W.

**Tabla** 3.10: Especificaciones eléctricas de la batería LiPo seleccionada

| Capacidad (A) | Voltaje (V) | Altura (m) | Longitud (m) | Ancho (m) | Peso (Kg) |
|---|---|---|---|---|---|
| 0,45 | 11,1 | 0,015 | 0,056 | 0,031 | 0,045 |

Teniendo las especificaciones eléctricas de la batería que cumple con las necesidades del diseño, se realiza una búsqueda de la referencia comercial de la batería LiPo que posee las necesidades de consumo eléctrico. Dado que las especificaciones eléctricas y dimensionales de las baterías LiPo son comunes entre los distintos fabricantes, durante la búsqueda inicial se selecciona la batería de referencia Gens Ace 450mAh-11.1V de 3 celdas.

### 3.3.2. DISEÑO 3D *backpack*

Se presenta el diseño 3D de la mochila, teniendo en cuenta los resultados obtenidos por cada placa de desarrollo. Para nuestra aplicación, la FPGA Ultra96 presentó resultados prometedores en términos de tiempo de ejecución y consumo de energía. Además, este sistema embebido presenta un tamaño y un peso más reducidos, que no interfieren con el movimiento libre del robot humanoide. Esta mochila de fácil construcción fue hecha especialmente para esta placa a partir de material PLA, teniendo un fácil montaje, desmontaje y un sencillo ajuste al robot. La figura 3.9 muestra de forma independiente la mochila y cada una de las posiciones que se integran en el robot humanoide NAO.

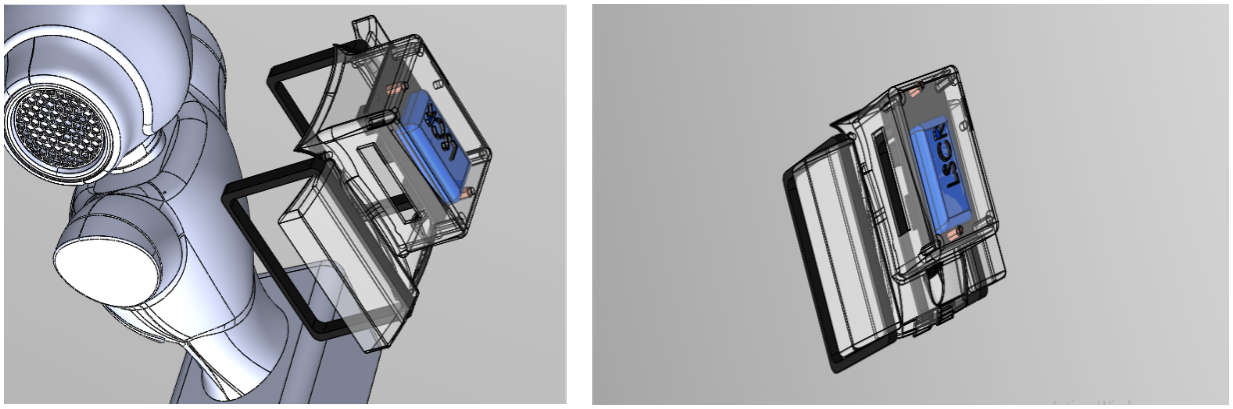

**Figura** 3.9: Diseño de *backpack* 3D para el robot humanoide Nao

## REFERENCIAS

[1] D. Wang, K. Xu, and D. Jiang, "Pipecnn: An opencl-based open-source fpga accelerator for convolution neural networks," in *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 279–282.

[2] M. Modasshir, A. Q. Li, and I. Rekleitis, "Deep neural networks: a comparison on different computing platforms," in *2018 15th Conference on Computer and Robot Vision (CRV)*. IEEE, 2018, pp. 383–389.

[3] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[4] D. Franklin, "Nvidia jetson tx2 delivers twice the intelligence to the edge," *NVIDIA Accelerated Computing—Parallel Forall*, vol. 6, 2017.

[5] D. Albani, A. Youssef, V. Suriani, D. Nardi, and D. D. Bloisi, "A deep learning approach for object recognition with nao soccer robots," in *Robot World Cup*. Springer, 2016, pp. 392–403.

[6] D. Bussey, A. Glandon, L. Vidyaratne, M. Alam, and K. M. Iftekharuddin, "Convolutional neural network transfer learning for robust face recognition in nao humanoid robot," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2017, pp. 1–7.

[7] N. Cruz, K. Lobos-Tsunekawa, and J. Ruiz-del Solar, "Using convolutional neural networks in robots with limited computational resources: detecting nao robots while playing soccer," in *Robot World Cup*. Springer, 2017, pp. 19–30.

[8] Q. Yan, S. Li, C. Liu, and Q. Chen, "Real-time lightweight cnn in robots with very limited computational resources: Detecting ball in nao," in *International Conference on Computer Vision Systems*. Springer, 2019, pp. 24–34.

[9] S. Chatterjee, F. H. Zunjani, and G. C. Nandi, "Real-time object detection and recognition on low-compute humanoid robots using deep learning," in *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2020, pp. 202–208.

[10] F. Badeig, Q. Pelorson, S. Arias, V. Drouard, I. Gebru, X. Li, G. Evangelidis, and R. Horaud, "A distributed architecture for interacting with nao," in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, 2015, pp. 385–386.

[11] Q. Liu, C. Zhang, Y. Song, and B. Pang, "Real-time object recognition based on nao humanoid robot," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 644–650.

[12] K. M. Othman and A. B. Rad, "An indoor room classification system for social robots via integration of cnn and ecoc," *Applied Sciences*, vol. 9, no. 3, p. 470, 2019.

[13] J. Zhou, L. Feng, R. Chellali, and H. Zhu, "Detecting and tracking objects in hri: Yolo networks for the nao "i see you" function," in *2018 27th IEEE international symposium on robot and human interactive communication (RO-MAN)*. IEEE, 2018, pp. 479–482.

[14] M. Mattamala, G. Olave, C. González, N. Hasbún, and J. Ruiz-del Solar, "The nao backpack: An open-hardware add-on for fast software development with the nao robot," in *Robot World Cup*. Springer, 2017, pp. 302–311.

# 4

# CONCLUSIONES Y FUTURO TRABAJO

En este trabajo, se presenta la evaluación de desempeño de arquitecturas heterogéneas basadas en FPGA y GPU al implementar redes neuronales convolucionales para aplicaciones autónomas. Una etapa inicial de adquisición y comunicación de la imagen fue propuesta, en donde, se presenta la creación de un ambiente virtual orientado al robot humanoide NAO en la etapa de adquisición. En la fase de comunicación se opta por una transmisión Ethernet, en donde, para la resolución estándar de la cámara de NAO (640x480 VGA), se obtienen resultados en tiempo real (65 fps). Estos resultados se encuentran por encima de lo establecido para una aplicación en tiempo real (30 fps), lo que permite que el sistema de adquisición y comunicación propuesto sea implementado en robots humanoides que contengan como mínimo una cámara VGA y una conexión Ethernet.

Respecto a la implementación de las CNN, las arquitecturas YOLO-V3, Tinier-YOLO, Inception-V1 y Inception-V3 *transfer-learning*, fueron usadas como referencia para la evaluación de desempeño de cada sistema embebido durante la etapa de inferencia. Se realiza la implementación de algoritmos clásicos de visión por computador sobre un sistema de cómputo similiar al del robot NAO, con el objetivo de comparar los resultados versus con los obtenidos al acelerar una CNN sobre arquitecturas heterogéneas. Los resultados demuestran que al ejecutar una CNN sobre estos sistemas embebidos, es posible obtener métricas para una aplicación en tiempo real. En el presente trabajo, se lograron métricas para este tipo de aplicaciones al ejecutar las redes Inception-V1 y Inception-V3 sobre la Ultra96, que fue 30.3 fps y 27.02 fps respectivamente. Caso contrario ocurre con las técnicas clásicas al ejecutarsen en el sistema secuencial de NAO, en donde, para este trabajo el máximo valor de cuadros por segundo fue de 3.95 fps para el modelo *Desicion Tree*.

A pesar de las distintas implementaciones de CNN sobre el sistema secuencial de NAO o en dispositivos computacionales externos conectados al robot, se encuentran limitantes que fueron abordadas en la actual tesis de investigación. Realizando una primera comparación en cuanto a cuadros por segundo, en los trabajos realizados por [1, 3–5] el máximo valor obtenido fue de 19 fps y el mínimo de 2 fps. En el trabajo propuesto, se alcanzan valores en tiempo real al ejecutar las CNN Inception-V1 y Inception-V3*transfer-learning* sobre el sistema embebido basado en FPGA (Ultra96). A pesar del alto grado de paralelismo de las FPGA, estos resultados fueron apoyados por el tipo de conexión entre el hardware externo y la CPU de NAO. Al ser una conexión Ethernet, la pérdida de datos en la transmisión y recepción de las imágenes es menor a comparación de una conexión WiFi. Ya que en el procesamiento de las CNN se presenta un tiempo de cómputo, al no perdersen suficientes datos en la transmisión, los tiempos de inferencia no se redujeron significativamente.

Teniendo en cuenta los resultados mencionados anteriormente, la actual tesis de investigación presenta unos resultados significativos frente a los alcanzados por [2], en donde, realizan la implementación de una DNN sobre sistemas que contienen una CPU y una GPU aplicado a un robot humanoide. Respecto a la implementación sobre la Jetson TX2, se obtienen resultados similares en consumo de potencia pero con mejores tiempos de respuesta, en donde, se están ejecutando diferentes CNN que fueron entrenadas con un mayor número de clases. Adicional, en el trabajo propuesto, se realiza la implementación de distintas CNN sobre un sistema de cómputo basado en FPGA, el cual, presenta un alto grado de paralelismo, menor tamaño, bajo consumo de potencia y un costo menor. A partir de los resultados descritos en términos de consumo de potencia y cuadros

por segundo se mejora la capacidad de respuesta que requiere el robot humanoide NAO para ser empleado en aplicaciones que requieren tiempos de procesamiento mayores y mayor autonomía.

Por otro lado, en este trabajo, se presenta el diseño un *backpack* para NAO que contiene una tarjeta de desarrollo y una batería que la alimenta, Pese a ser una conexión externa, no presenta limitantes en cuanto al desenvolvimiento del robot en un ambiente cotidiano, ya que todo el sistema se encuentra ubicado en un mismo lugar. Esto permite que no se restrinjan los movimientos de NAO y que pueda ser utilizado en aplicaciones relacionadas a la interacción humano-robot y localización de objetos.

Finalmente, fueron utilizadas arquitecturas de CNN que presentan una alta precisión y buen desempeño en tareas de clasificación y detección de objetos. La CNN Inception-V1 posee una precisón del 88.9%, y en la implementación del *transfer-learning* sobre la Inception-V3, se alcanza una precisión del 98%. Peso a ello, lo mas importante es la cantidad de clases con las cuales se entrenaron los pesos de la red. Al ser mayor el número de objetos a reconocer, le permite al robot humanoide NAO un mayor entendimiento de su entorno. Teniendo en cuenta las limitantes de los trabajos mencionados y los resultados obtenidos en el presente trabajo, se puede observar que la integración de una CNN y arquitecturas heterogéneas basadas en FPGA o GPU sobre los robots humanoides, pueden otorgarle al autómata un mejoramiento de la percepción visual en tiempo real. Esta capacidad puede ser aprovechada en aplicaciones que requieren una mayor comprensión del entorno, tiempo de respuesta rápidos y mayor autonomía.

Como trabajo futuro, se espera abordar con mayor profundidad la cuantificación de redes neuronales convolucionales. Al ser las CNN computacionalmente costosas, la cuantificación será un tema abierto de investigación, en el cual, se espera reducir considerablemente el tamaño de la red sin perder la precisión. Con esta reducción de tamaño se espera que las CNN puedan ser implementadas con mayor facilidad en sistema embebidos que puedan integrarse a robots humanoides como NAO. Por otro lado, se podría hacer uso del sensor de profundidad que posee el robot NAO, como elemento que permita calcular la distancia real del objeto una vez sea detectado por la CNN. De esta manera, se podría ampliar las capacidades del robot en tareas que involucran una interacción HRI.

## REFERENCIAS

[1] Dario Albani, Ali Youssef, Vincenzo Suriani, Daniele Nardi, and Domenico Daniele Bloisi. A deep learning approach for object recognition with nao soccer robots. In *Robot World Cup*, pages 392–403. Springer, 2016.

[2] Alexander Biddulph, Trent Houliston, Alexandre Mendes, and Stephan K Chalup. Comparing computing platforms for deep learning on a humanoid robot. In *International Conference on Neural Information Processing*, pages 120–131. Springer, 2018.

[3] Daniel Bussey, Alexander Glandon, Lasitha Vidyaratne, Mahbubul Alam, and Khan M Iftekharuddin. Convolutional neural network transfer learning for robust face recognition in nao humanoid robot. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE, 2017.

[4] Nicolás Cruz, Kenzo Lobos-Tsunekawa, and Javier Ruiz-del Solar. Using convolutional neural networks in robots with limited computational resources: detecting nao robots while playing soccer. In *Robot World Cup*, pages 19–30. Springer, 2017.

[5] Qianyuan Liu, Chenjin Zhang, Yong Song, and Bao Pang. Real-time object recognition based on nao humanoid robot. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 644–650. IEEE, 2018.