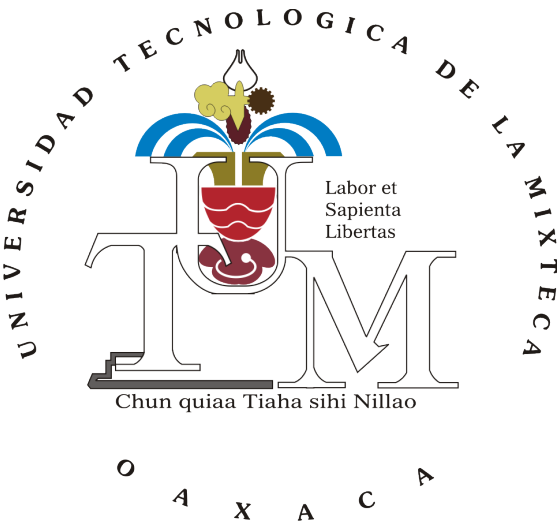


Universidad Tecnológica de la Mixteca

División de Estudios de Posgrado

"Detección simultánea, por medio de CNN, de múltiples robots NAO dentro de un campo de fútbol, con aplicación a RoboCup"

T E S I S

Para obtener el grado en:

MAESTRO EN ROBÓTICA

Presenta:

Ing. Oscar Alberto Zavala Salas

Director:

Dr. Alberto Elías Petrilli Barceló

Codirector:

Dr. José Aníbal Arias Aguilar

Huajuapan de León, Oaxaca, México, Enero de 2020.

Tesis presentada en Enero de 2020
ante los siguientes sinodales:

Dr. Fermín Hugo Ramírez Leyva.
M.T.C.A. Erik Germán Ramos Pérez.
Dr. Juan Humberto Sossa Azuela.

Director de Tesis:
Dr. Alberto Elías Petrilli Barceló

Codirector de Tesis:
Dr. José Aníbal Arias Aguilar.

Dedicado a mis padres, mi mayor inspiración,

porque sin ustedes no hubiera sido posible.

¡Los amo!

Agradecimientos

A mi padre y a mi madre, Oscar y Miriam, por su eterno cariño, apoyo, esfuerzo, dedicación, sacrificio y amor incondicional. Por todas sus enseñanzas, consejos, cuidados y regaños. Porque gracias a todo lo que me han dado, he llegado a convertirme en el hombre que soy el día de hoy. Porque sin ustedes nunca habría llegado tan lejos, gracias.

A mis hermanos y hermanas. Sofia, José, Vicente y Ana, gracias por mi motor. Por motivarme a ser mejor cada día, para volverme un mejor ejemplo para ustedes y por todos los momentos de alegría que hemos compartido. A toda mi familia, por siempre acompañarme, apoyarme y alentarme a ir más allá y lograr todo lo que me proponga.

A mi director de tesis, el Dr. Alberto Petrilli, por su apoyo, tiempo y dedicación para llevar acabo este proyecto. Por sus enseñanzas y todo el aprendizaje que he obtenido de su parte para mi formación académica, tanto en la licenciatura, como en la maestría. Por sus consejos y las experiencias compartidas a lo largo de los últimos años, pero sobre todo, por su amistad.

A mi codirector de tesis, el Dr, Aníbal Arias, por su tiempo, apoyo y orientación durante el desarrollo de este trabajo. Por el conocimiento impartido que ha ayudado a alcanzar el objetivo de esta tesis. A mis sinodales, el Dr. Hugo Leyva, el M.T.C.A. Erik Ramos, el Dr. Humberto Sossa y la M.C. Verónica Rodríguez, cuyas observaciones, comentarios y correcciones han ayudado a alcanzar la versión final de este documento.

A todos y cada uno de mis compañeros de generación y del laboratorio de robótica. Por acompañarme en este viaje de dos años y medio llamado maestría. Por su amistad y todos los buenos momentos compartidos, por todo lo que aprendí de ustedes y todo lo que sufrimos juntos.

A la Universidad Tecnológica de la Mixteca, por la oportunidad de formar parte del programa de Maestría en Robótica, pero especialmente, al instituto de Posgrado. A sus profesores y todo su personal. Al Consejo Nacional de Ciencia y Tecnología (CONACYT), por otorgarme el apoyo económico que me ha permitido el llevar a cabo los estudios de este grado y realizar una estancia en el extranjero como parte de los mismos.

Finalmente, agradezco a Dios, por darme vida para llegar hasta este momento. Por poner en ella a todas las personas a las que aquí agradezco. Por permitirme terminar esta maestría y alcanzar una meta más en mi desarrollo personal. Por iluminar mi camino y acompañarme a cada momento.

¡GRACIAS!

Resumen

Con los excelentes resultados que están siendo entregados en los últimos años por sistemas de detección basados en redes neuronales convolucionales para tareas de detección de objetos en imágenes, estos sistemas están siendo utilizados para incontables aplicaciones de robótica y de inteligencia artificial. Sin embargo, la mayoría de estos métodos tienen a ser, por lo general, computacionalmente costosos para sistemas embebidos de bajo poder de procesamiento, lo que los hace no muy adecuados para su ejecución sobre robots móviles.

Este trabajo propone una arquitectura de red completamente convolucional llamada TYR-NAO, basada en el sistema de detección *tiny* YOLO. El modelo propuesto está pensado para ser implementado sobre la plataforma del robot NAO y está diseñado para mejorar los tiempos de procesamiento presentes en *tiny* YOLO para realizar la detección de los objetivos. Facilitando, al mismo tiempo, su operación sobre la computadora del robot. TYR-NAO está entrenado para la detección exclusiva de robots NAO. Tanto el modelo y su entrenamiento, como la base de datos etiquetada para esta tarea específica, son las contribuciones principales de este proyecto. El enfoque principal en el que está basada esta investigación es en la implementación de la propuesta de modelo para ser utilizada en la categoría SPL de la RoboCup.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La *Robot World Cup*, conocida también simplemente como RoboCup, es una competencia internacional de robótica creada en la década de los 90's. Esta competencia promueve la investigación científica y de ingeniería en las áreas de robótica e inteligencia artificial [10] al proponer un objetivo desafiante a largo plazo para estas áreas de investigación: "Para mediados del siglo 21, un equipo de robots jugadores de *soccer* completamente autónomos deberá ganar un juego de fútbol, cumpliendo las reglas oficiales de la FIFA, contra el más reciente equipo ganador de la Copa del Mundo" [9]. Éste podría parecer un objetivo bastante ambicioso para aquella época, e incluso para la época actual, sin embargo, se ha tomado como referencia algunos de los grandes logros de la humanidad en algunas áreas afines. Por ejemplo, que tomó 50 años desde el primer avión de los hermanos Wright, hasta la misión Apollo donde se envió a un hombre a la luna y se le regresó de manera segura a la tierra; o que también tomó 50 años de la invención de la primera computadora digital, hasta la conocida Deep Blue que venció al mejor ajedrecista del mundo en 1997. Por lo anterior es que se ha propuesto que crear un robot autónomo capaz de jugar fútbol tomaría un tiempo similar, lo que también sería uno de los logros más grandes compartidos por la robótica y la inteligencia artificial.

Dentro de las muchas categorías de las cuales está conformada la RoboCup, existe la liga de plataforma estándar o SPL (por sus siglas en inglés: *Standard Platform League*). Esta categoría recibe su nombre debido que todos los equipos que participan en

ella hacen uso de la misma plataforma (robot) para el desarrollo de los algoritmos de inteligencia artificial. Hoy en día, la plataforma utilizada es el robot NAO.

Aun cuando el juego de fútbol pudiera parecer una actividad sencilla de realizar para la mayoría de las personas, no lo es para un robot. Esto debido a las diversas actividades que se deben realizar de manera simultánea para llevar a cabo un partido. Estas actividades pueden ser el detectar a otros jugadores y clasificarlos en compañeros de equipo y adversarios; buscar, ubicar y hacer un seguimiento de la pelota; detectar las líneas que dividen las secciones del campo y localizarse dentro del mismo; detectar y ubicar las porterías; y mantener una comunicación constante para jugar de manera cooperativa, por mencionar algunas. Es entonces, estrictamente necesario que el robot reciba toda la información relevante, con respecto a lo anterior, para poder desarrollar una lógica de juego que le permita llevar a cabo un partido de la manera en que los humanos lo hacen.

Por lo tanto, para que un robot pueda jugar fútbol necesita, entre otras cosas, un sistema de visión eficiente que le permita detectar e identificar diversos objetos dentro del campo de juego. El problema de la detección de objetos por medio de visión artificial ha sido solucionado a través de diversas técnicas con el paso de los años. Sin embargo, después de que Alex Krizhevsky utilizó técnicas de aprendizaje profundo o *Deep Learning* para la clasificación de objetos en imágenes y ganar la famosa competencia de visión por computadora *ImageNet* en 2012 [6] [19], es que estas técnicas han sido utilizadas para resolver la problemática. Algunas de las técnicas más utilizadas en la actualidad están basadas en el entrenamiento de sistemas conformados por estructuras de redes neuronales artificiales. Éstos últimos son entrenados con bancos de datos que contienen un número grande de imágenes para que el sistema pueda aprender a detectar y/o clasificar el o los objetos de interés. Esto ha sido implementado en algoritmos de inteligencia artificial para que los robots sean capaces de llevar a cabo tareas más elaboradas y precisas.

Este trabajo aborda la detección de múltiples objetivos de manera simultánea. De manera específica, se centra en la detección realizada por un robot jugador de fútbol dentro del campo de juego y que es exclusiva de los jugadores adicionales que se en-

cuentren dentro de su campo de visión. Una vez realizada la detección de cada uno de los robots, el algoritmo de visión podría también permitir la clasificación de los objetivos detectados como jugadores compañeros de equipo o adversarios. Para lograr lo anterior, el robot cuenta con un sistema de visión basado en arquitecturas de redes neuronales convolucionales y es entrenado con bancos de imágenes obtenidos de las diferentes ediciones de la competencia y bases de datos liberadas por otros equipos participantes.

1.1. Planteamiento del Problema

1.1.1. Hipótesis

Este trabajo pretende demostrar si un robot NAO jugador de fútbol es capaz de detectar, de manera simultánea, a todos los robots NAO jugadores que se encuentren dentro de su campo de visión, y de clasificar a cada uno de los robots detectados como compañero de equipo o adversario, a partir de un sistema de detección basado en CNN.

1.1.2. Justificación

Una de las maneras más eficientes de fomentar la investigación en áreas de desarrollo de la robótica es a través de las competencias. Sean nacionales o internacionales, en ellas tanto alumnos de diferentes niveles académicos (desde primaria hasta posgrado), como sus profesores y mentores, desarrollan sistemas robóticos y algoritmos de inteligencia para resolver retos predeterminados. Estos retos dan la pauta para resolver problemas del mundo real y de diferentes ámbitos como el industrial, de seguridad, de medicina, de servicios, entre otras.

El juego de fútbol de robots es una aplicación de sistemas multiagentes y de tecnologías de inteligencia artificial que incluye, entre muchas otras áreas, el procesamiento de imágenes, la visión artificial y el aprendizaje automático. Debido a las características en tiempo real que demanda el juego mismo y a la dinámica del ambiente que se encuentra en constante cambio, es que este reto provee elementos experimentales ex-

tremadamente valiosos para los campos de la robótica. Para poder lograr el objetivo de la competencia, una de las tareas principales es desarrollar un sistema de visión con la capacidad de entregar toda la información del entorno que es necesaria para poder jugar fútbol al nivel de los humanos. Con el paso del tiempo la competencia ha ido evolucionando y presentando, año tras año, nuevos retos para desafiar a los competidores y, que se asemejan más a las condiciones de juego real. Para poder cumplir con estos retos, cada robot debe ser capaz de reconocer los objetos que se encuentran a su alrededor y dentro del campo de juego.

La detección de objetos con robots es una tarea bastante estudiada en las últimas décadas en los campos de la robótica y, por ende, de la inteligencia artificial. El detectar objetos a través de las cámaras que posee un robot es una tarea muy importante. Para que el robot pueda llevar acabo determinadas acciones o tomar decisiones que lo lleven a realizar ciertas actividades, debe recibir la información completa del entorno que lo rodea. Haciendo una pequeña analogía con los sistemas de percepción de los humanos, la vista es el sentido que más información aporta al cerebro. Así mismo, un robot requiere de un sistema de visión óptimo que le proporcione la información necesaria del entorno en el que se encuentra para poder completar las tareas para las que es programado o para desenvolverse de manera autónoma en cualquier escenario.

En los últimos años, el aprendizaje profundo o ha alcanzado altos niveles de precisión en tareas de reconocimiento, detección y clasificación de objetos en imágenes, de tal manera que inclusive ha llegado a superar el rendimiento humano en algunas de éstas. Es debido a esto, que técnicas de aprendizaje profundo como el entrenamiento con redes neuronales artificiales están siendo implementadas para desarrollar sistemas de detección de objetos para robots. Lo anterior ha permitido una mejora sobresaliente en la eficiencia de dichos sistemas ya que se reduce de considerablemente el índice de error en las tareas de clasificación y/o detección, generando mejores resultados. Esto conlleva a que los sistemas complementarios, que trabajan en conjunto con los sistemas de visión, sean más precisos y confiables. Por ello se ha considerado la integración de estas técnicas para resolver las problemáticas de visión artificial de numerosas competencias de robótica.

La importancia de la detección de múltiples objetivos, así como su identificación o clasificación, reside en la posibilidad de implementar este tipo de algoritmos para mejorar sistemas de monitoreo o vigilancia, sistemas de localización y seguimiento, sistemas de evaluación, conteo y clasificación en la industria, sistemas de visión para robots de servicio, así como su implementación en algoritmos de inteligencia para vehículos autónomos como drones y automóviles, entre otros.

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar e implementar, sobre la plataforma del Robot NAO, un algoritmo de visión por computadora para la detección de múltiples robots NAO dentro de un campo de fútbol, a partir del entrenamiento de configuraciones de redes neuronales convolucionales para el sistema de detección.

1.2.2. Objetivos Específicos

- Implementar algoritmos para el procesamiento digital de imágenes para mejorar y resaltar características de las imágenes captadas por el robot.
- Desarrollar un algoritmo para la extracción de características deseadas de las imágenes previamente procesadas.
- Diseñar e implementar estructuras de CNN y, con la aplicación de técnicas de *Transfer Learning*, lograr la detección de los objetivos y su asignación dentro de las clases deseadas.

1.3. Estado del Arte

En primer lugar, y como se ha mencionado antes, una de las principales tareas es la de desarrollar un sistema de percepción que sea óptimo para el desempeño de un juego de

fútbol de robots. En [34] se hace referencia a la importancia de esto, donde se presenta un sistema del control de comportamiento desarrollado por un robot humanoide jugador de fútbol. El control del comportamiento es modelado como un autómata de estado finito, que comprende actividades como la percepción visual, control del comportamiento y las estrategias de juego. De las actividades anteriores, la primera de ellas involucra la localización propia del robot, la localización de la pelota, la estimación del movimiento, entre otras. Los autores aseguran que el sistema de visión es uno de los componentes más importantes del robot ya que prácticamente, todas las tareas más importantes que se tienen que realizar pasan a través de él. Lo que le permite al robot evolucionar de manera eficiente dentro del campo.

En los últimos años, la implementación de redes neuronales ha mostrado excelente desempeño al resolver tareas complejas de reconocimiento de objetos. Como ya ha sido mencionado, es esta la razón principal por la que estas técnicas están siendo aplicadas para resolver numerosas tareas en diversas competencias de robótica. Por ejemplo, en [1] proponen una red recurrente simultánea o SRN (*Simultaneous Recurrent Networks*) basado en un autocodificador para el reconocimiento de objetos que reduce significativamente el número de parámetros entrenables al compartir pesos en capas ocultas. Sus experimentos en la detección de rostros y caracteres muestran que el modelo propuesto ofrece un mejor desempeño en la detección que otras alternativas. Así mismo, demuestran la flexibilidad de incorporar su sistema de detección al implementarlo en la plataforma del robot NAO.

En [2] presentaron un novedoso enfoque para la detección y clasificación de objetos basado en redes neuronales convolucionales o CNN (*Convolutional Neural Networks*). El enfoque está conformado por dos etapas: la segmentación de la imagen para reducir el espacio de búsqueda; y aprendizaje profundo para validación. Donde el uso de CNN para fines de clasificación es importante para obtener una validación de las regiones de imagen extraídas. Como un posible enfoque para las técnicas de aprendizaje profundo para trabajar en los NAO, proponen tener una etapa de preprocesamiento que permita reducir la cantidad de información que será probada por la CNN. Esta reducción es realizada a partir de la segmentación, el escáner de color y la región de extracción. Su

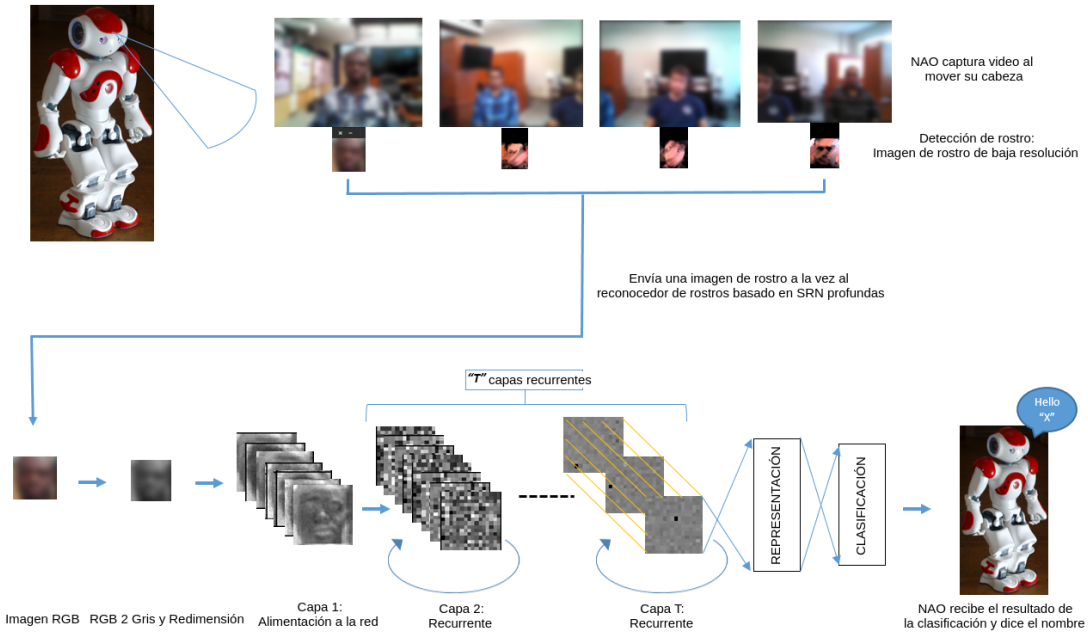


Figura 1.1: Arquitectura de reconocimiento facial profunda basada en SRN utilizando la plataforma robótica humanoide NAO [1].

enfoque adopta un enfoque de detección seguido de una clasificación donde a diferencia de la mayoría de los trabajos de aprendizaje profundo para detección de objetos, su etapa de detección no involucra *Deep Learning*. En su trabajo prueban diferentes configuraciones de CNN, en las que aunque los parámetros cambian, todas las capas convolucionales presentan la misma estructura interior: la etapa convolucional, la etapa de agrupamiento, y la etapa de capas de normalización. Para implementar las redes, utilizaron el marco de código abierto TensorFlow de *Google Brain*, que es un marco de aprendizaje profundo.

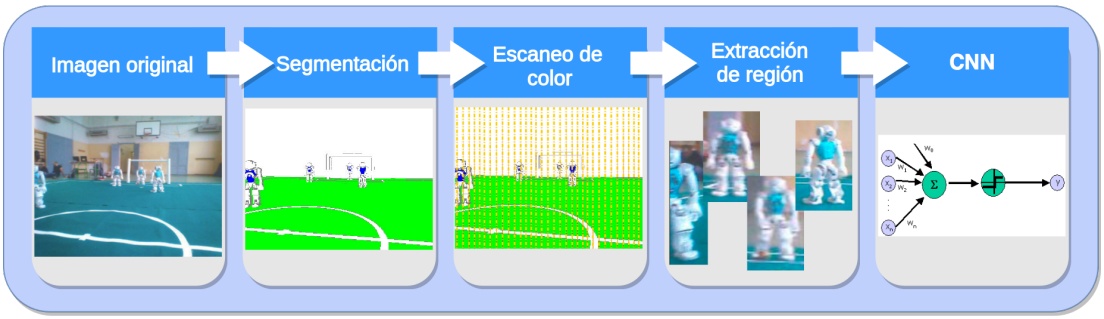


Figura 1.2: Arquitectura del módulo de Aprendizaje Profundo para detección de NAO [2].

Por otro lado, en [15] comparan varios sistemas de detección de dos etapas basados en diferentes arquitecturas de CNN y realzan la comparación de velocidad y precisión. Utilizan y evalúan tres diferentes configuraciones, cada una con diferentes parámetros e iteraciones para crear un sistema de detección de cuerpo robusto para robots humanoides (*LeNet, SqueezeNet, GoogLeNet*). Utilizan distintos hardware para proporcionar una compensación de velocidad y precisión. Su procedimiento para la detección del cuerpo utiliza una segmentación de imagen y CNN.

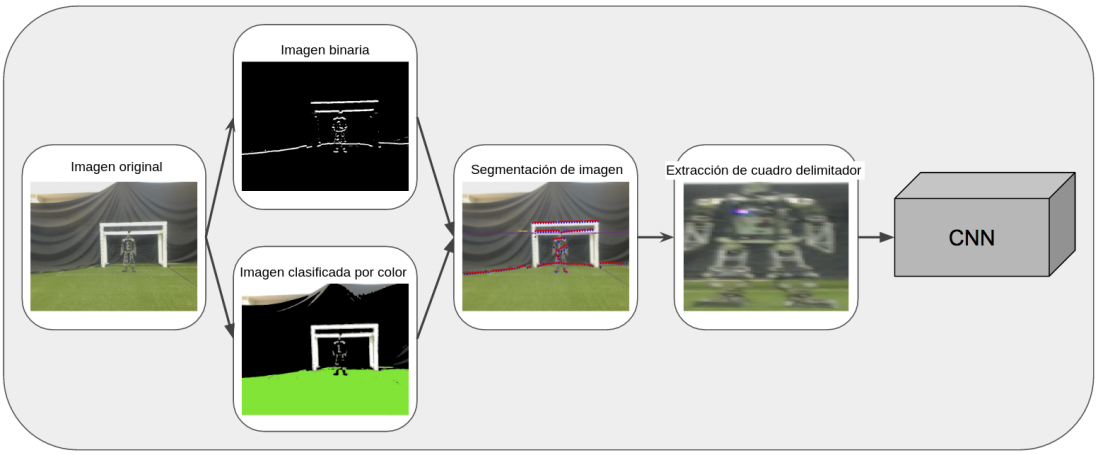


Figura 1.3: Vista general del sistema [15].

En [5] analizan el problema general de usar CNN en robots con capacidades computacionales limitadas y proponen pautas generales para su uso. Adicionalmente, proponen dos detectores de robots NAO basados en CNN que son capaces de correr en tiempo real mientras se juega al fútbol. Uno de los detectores está basado en la red XNOR-NET y el otro en la red *SqueezeNet*. Cada detector puede procesar una propuesta de objeto de robot en aproximadamente 1 ms, con un promedio de 1.5 propuestas por fotograma obtenido por la cámara superior de la NAO. Ambas redes son usadas para la detección del robot NAO en el contexto del fútbol de robots, obteniendo buenos resultados mientras se consumen pocos recursos computacionales. Con este trabajo se demuestra que utilizar aprendizaje profundo en robots NAO es factible y que es posible alcanzar una detección óptima en el juego de fútbol. También que, arquitecturas de redes neuronales similares a las propuestas pueden ser utilizadas para otras tareas de detección, como para la pelota o los postes de las porterías. Además, ya que la metodología presentada en

su trabajo para alcanzar capacidades en tiempo real es genérica, es posible implementar las mismas estrategias en aplicaciones con restricciones de hardware similares.

A su vez, en [8] presentan un nuevo canal para la detección, el seguimiento y la identificación basados en la visión en línea de robots con un aspecto conocido e idéntico en tiempo real. A diferencia de trabajos previos que abordan el rastreo e identificación de robots, utilizan un enfoque orientado a los datos, basado en redes neuronales recurrentes para aprender las relaciones entre entradas y salidas secuenciales. Para la detección de los robots han diseñado e implementado una plataforma que trabaja bajo varias configuraciones del robot y distintas condiciones de iluminación basado en Histogramas de gradientes Orientados (HOG). Para el sistema de seguimiento e identificación de los objetivos utilizan redes neuronales recurrentes o RNN (*Recurrent Neural Networks*), específicamente la estructura de las redes LSTM (*Long Short-Term Memory*) que son muy poderosas para capturar dependencias espaciales y temporales en secuencias de información de entrada y salida. Utilizan una red de dos capas con 500 unidades para asociación de la información en forma de una sola red que es usada múltiples veces para procesar múltiples detecciones.

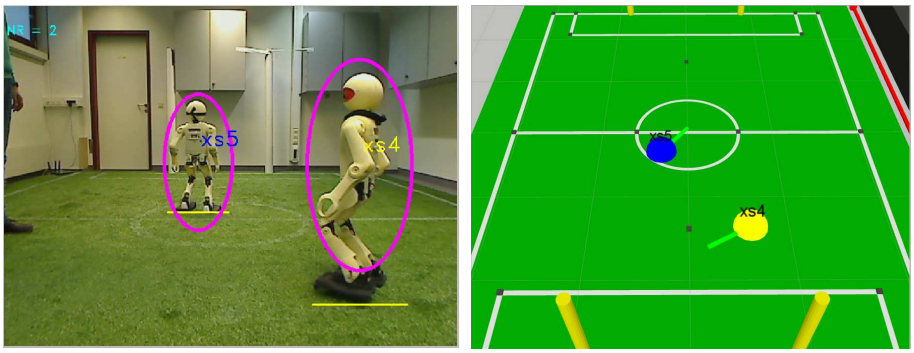


Figura 1.4: Resultados de detección, seguimiento e identificación obtenidos por el sistema en una sola secuencia [8].

Recientemente, se han presentado más trabajos, cuyos resultados están siendo utilizados, específicamente, en la categoría SPL de la RoboCup, los cuales involucran el uso de las CNN para la detección de objetos. En [31] remarcan que a pesar de que las redes neuronales convolucionales (CNNs) son el método más usado para tareas de visión por computadora en el estado del arte, su despliegue en plataformas móviles o embebidas

es todo un reto debido a los excesivos requerimientos computacionales de las mismas. Proponen aquí un método para la detección de objetos en tiempo real de extremo a extremo utilizando redes neuronales profundas para la comprensión de escena para el fútbol de robots. Se componen dos redes neuronales clave: la primera es una red neuronal profunda entrenada para llevar a cabo la segmentación semántica sobre la imagen de la cámara del robot, y una segunda red, más pequeña, entrenada para propagar las etiquetas de clase de una imagen previa a la siguiente. Estas redes se combinan para alcanzar alta precisión a una velocidad razonable. Las redes son entrenadas con bases de datos sintéticas y son afinadas con un conjunto conformado por imágenes reales obtenidas de un robot NAO. El sistema es capaz de localizar cuatro clases de primer plano: pelota, robot, postes de portería y líneas del campo. Además, investigan y evalúan algunos métodos prácticos para el incremento de la eficiencia y desempeño de las redes. Presentan al final, RoboDNN, una biblioteca ligera de red neuronal en C++. Esta es de propagación hacia adelante solamente, y está diseñada para una rápida inferencia en los robots NAO. La biblioteca no tiene dependencias y facilita la compilación para el NAO. La implementación también ofrece compatibilidad con el marco de trabajo Pytorch DNN, con la posibilidad de importar redes neuronales entrenadas utilizando Pytorch.

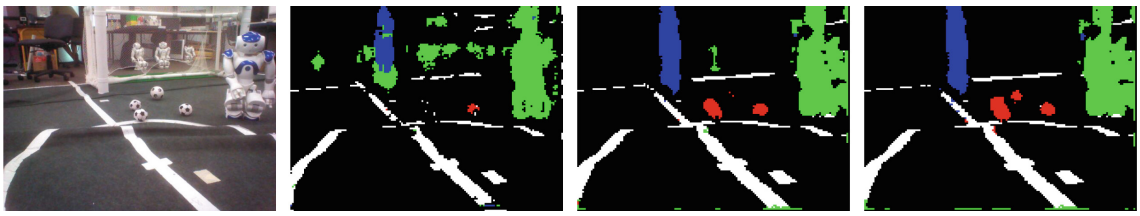


Figura 1.5: Capacidad de autocorrección de la red de propagación de etiquetas [31].

Un año más tarde, los mismos autores del trabajo anterior, proponen en [32] una arquitectura eficiente de redes neuronales para el problema de la detección de objetos relevantes en los ambientes del fútbol de robots, llamada ROBO. Aquí se resalta que la detección de objetos es una de las tareas fundamentales para la visión por computadora, ya que una localización y clasificación precisa de objetos relevantes es un componente necesario para la comprensión de una escena.

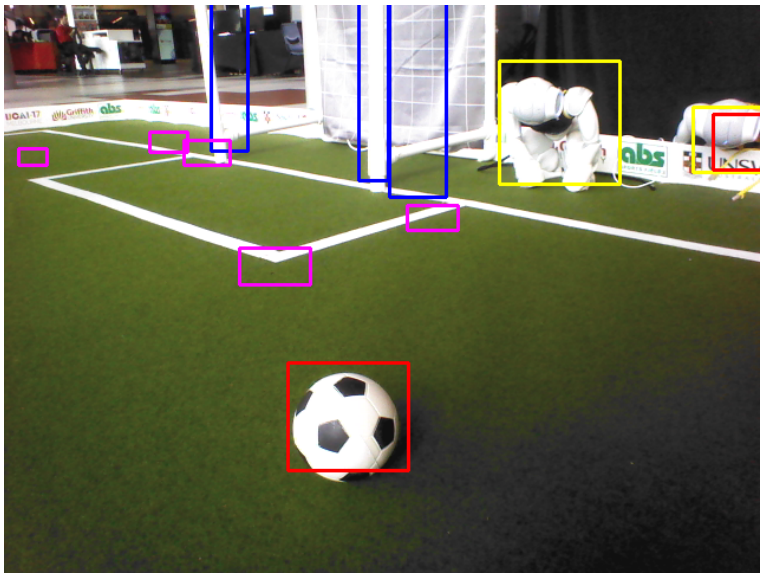


Figura 1.6: Ejemplo de resultados de detección en conjunto de datos real [32].

Al mismo tiempo toma en consideración que mientras los métodos basados en aprendizaje profundo proporcionan un buen desempeño al alcanzar un éxito considerable últimamente en diferentes áreas, y que han dominado el campo de visión en los últimos años, sus altos requerimientos computacionales limitan su uso en sistemas embebidos de baja potencia. Este artículo presenta un sistema de detección de objetos completamente neuronal para la plataforma del robot NAO es su versión 6, capaz de detectar simultáneamente cuatro clases de objetos relevantes para el fútbol de robots (pelota, cruces de líneas, postes de portería y robots). Este modelo explota la idiosincrasia del ambiente de fútbol de robots para proporcionar un descenso de aproximadamente de 35 veces en tiempo de ejecución y logra una precisión superior, en comparación con el modelo *tiny* de YOLOv3, modelo en el que está basado la arquitectura presentada. Las redes de esta publicación son preentrenadas en una base de datos sintética, y subsecuentemente son afinadas y evaluadas en una base de datos realista y de menor tamaño. Para la afinación, propone un nuevo método llamado transferencia de aprendizaje sintética, que permite una mejor convergencia al solo reentrenar algunas primeras capas de la red. Las redes son reducidas usando una regularización L1 para una mejora en la velocidad. Adicionalmente, son evaluadas 3 diferentes versiones de la arquitectura ROBO en la base de datos para demostrar la velocidad y precisión superior con respecto

al del modelo de YOLO. Y demuestran los efectos de la reducción y de la transferencia de aprendizaje sintética.

Finalmente, inspirados también por la importancia significativa que tiene la detección de objetos en el entorno de la mayoría de aplicaciones de robots autónomos y que enfoques sofisticados como las redes neuronales profundas, que han presentado un alto desempeño en muchas tareas de detección, son difíciles de aplicar debido a los limitados recursos computacionales de estos sistemas, [24] presenta la red JET-Net. Su nombre viene del inglés *Just Enough Time*, que quiere decir "Solo Tiempo Suficiente", este es un modelo para detección de objetos eficiente basado en redes neuronales convolucionales. Es capaz de realizar detección de robots en tiempo real sobre la plataforma NAO v5 en un entorno de fútbol para robots. Los experimentos muestran que el sistema es capaz de detectar de manera confiable otros robots en diferente situaciones. También presenta una técnica que reutiliza las características aprendidas para obtener más información de los objetos detectados. Debido a que la información adicional es aprendida enteramente de los datos de simulación, es llamada Transferencia de Aprendizaje de Simulación.

1.4. Organización de la Tesis

El contenido de los capítulos restantes de este documento está organizado de la siguiente manera: El capítulo 2, conformado por el marco teórico, presenta información teórica sobre la plataforma del robot NAO, el aprendizaje profundo y las redes neuronales convolucionales, así como técnicas de transferencia de aprendizaje. Información que es importante para la comprensión de los capítulos subsecuentes. En el capítulo 3 se detalla, paso a paso, la metodología seguida para el desarrollo de este trabajo. El capítulo 4, por su parte, presenta la redacción de los resultados experimentales obtenidos en el proceso. Finalmente, el capítulo 5 engloba las conclusiones generadas a partir de los resultados y, presenta una pequeña propuesta para darle una posible continuidad al proyecto en cuestión.

Capítulo 2

Marco Teórico

2.1. Plataforma NAO

NAO es un robot de tipo humanoide de aproximadamente 58 centímetros de alto y es el primer humanoide de la compañía *SoftBank Robotics*, el cual ha estado evolucionando desde su primera aparición en el año 2006. Actualmente se encuentra en su sexta versión del modelo. Su diseño es el fruto de una combinación única de ingeniería mecánica y software, está compuesto por una multitud de sensores, motores y software pilotado por un sistema operativo hecho a la medida: NAOqi OS [30].

NAOqi es un sistema operativo basado en Linux y es el nombre del software principal que se ejecuta en el robot y que lo controla. El NAOqi *Framework* es el marco de programación que se utiliza para programar al NAO. Éste responde a las necesidades comunes de la robótica, incluyendo: paralelismo, sincronización, eventos, etc. Este marco permite una comunicación entre los diferentes módulos (movimiento, audio, video), así como programación homogénea y el intercambio homogéneo de información.

El robot NAO cuenta con 25 grados de libertad y una forma humanoide que le permiten moverse y adaptarse al mundo que le rodea. Su unidad inercial le permite mantener el equilibrio y saber si está de pie o acostado. Los numerosos sensores en su cabeza manos y pies, así como sus sonares, le permiten percibir el entorno y orientarse. Con sus 4 micrófonos direccionales y sus altavoces, NAO interactúa con humanos de una manera completamente natural, al escuchar y hablar. Está equipado con dos cámaras

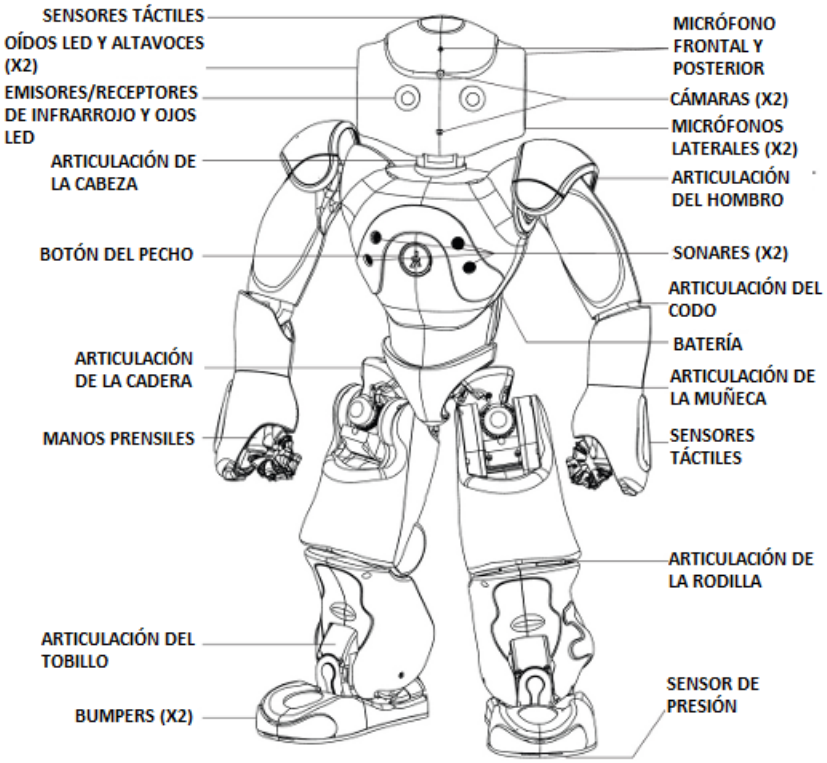


Figura 2.1: Robot humanoide NAO [1].

que graban su entorno en alta resolución, ayudándolo a reconocer figuras y objetos. Es por todo esto que a tan sólo un año después de su lanzamiento, el robot fue seleccionado para suplir al AIBO de SONY en la RoboCup y desde entonces ha sido la plataforma utilizada en la competencia SPL *Soccer* de dicho evento.

2.2. Aprendizaje Profundo

El aprendizaje profundo del inglés *Deep Learning* es un tipo específico de aprendizaje automático o *Machine Learning*. Un algoritmo de aprendizaje automático es un algoritmo capaz de *aprender* a partir de datos. Se dice que un programa de computadora aprende de una experiencia E con respecto a alguna clase de tareas T y una medida de rendimiento P, si su rendimiento en las tareas T, medido por P, mejora con la experiencia E [12].

Las tareas en el aprendizaje automático son usualmente descritas en términos de cómo los sistemas de aprendizaje automático deberían procesar una muestra. Una mues-

tra es una colección de características que han sido medidas cuantitativamente de algún objeto o evento que se quiere que el sistema de aprendizaje procese. Típicamente una muestra se representa como un vector $x \in \mathbb{R}^n$ donde cada entrada x_i del vector es otra característica. Por ejemplo las características de una imagen son usualmente los valores de los píxeles en la imagen. Uno de los tipos de tareas resueltos a partir de aprendizaje automático más comunes es la **clasificación.** En esta tarea, se le pide al programa de computadora que especifique a cuál de k número de categorías pertenecen algunas entradas. Un ejemplo de tarea de clasificación es la de **reconocimiento de objetos**, donde la entrada es una imagen y la salida es un código numérico que identifica al objeto en la imagen. Para evaluar las habilidades del algoritmo de aprendizaje automático, se debe diseñar una medida cuantitativa del desempeño que usualmente es específica a la tarea que lleva a cabo el sistema. Para tareas como clasificación, la mayoría de las veces se mide la exactitud del modelo. La **exactitud** es la proporción de ejemplos o muestras para las cuales el modelo produce una salida correcta. También se puede obtener información equivalente al medir la taza de error, que es la proporción de ejemplos para los cuales el modelo produce una salida incorrecta.

Los algoritmos de aprendizaje automático pueden ser categorizados, en general, como supervisados o no supervisados a partir del tipo de experiencia que les será permitido tener durante el proceso de aprendizaje. Los algoritmos de **aprendizaje supervisado** experimentan un conjunto de datos que contienen características, pero cada ejemplo también está asociado con una etiqueta o un objetivo. Dicho de otra manera, el aprendizaje supervisado involucra observar varios ejemplos de un vector x aleatorio y un valor asociado del vector y, y aprender a predecir y a partir de x, usualmente al estimar $p(x|y)$. El término aprendizaje supervisado se origina a partir del punto de vista de que el objetivo está siendo proporcionado por un instructor o maestro que muestra al sistema de aprendizaje automático qué hacer.

El desarrollo del aprendizaje profundo fue motivado en parte por el fallo de los algoritmos tradicionales para generalizar bien en tales tareas de IA. Esto debido a que generalizar nuevos ejemplos o muestras se vuelve exponencialmente más difícil cuando se trabaja con datos de **alta dimensión**. Prácticamente, el **aprendizaje profundo** es

entonces, un conjunto de algoritmos o métodos de aprendizaje automático que intenta modelar abstracciones de alta dimensión de datos usando arquitecturas compuestas de transformaciones no lineales múltiples. El aprendizaje profundo moderno provee un marco de trabajo muy poderoso para el aprendizaje supervisado. Al agregar más capas y más unidades dentro de una capa, una red profunda puede representar funciones de complejidad creciente. La mayoría de las tareas que consisten en mapear un vector de entrada a un vector de salida, y aquellas que son fáciles de realizar por las personas rápidamente, se pueden lograr a través del aprendizaje profundo si se cuenta con los modelos adecuados y con los conjuntos de datos suficientemente grandes para realizar el entrenamiento.

2.3. Redes Convolucionales

Las redes convolucionales, también conocidas como redes neuronales convolucionales o CNN (del inglés *Convolutional Neural Networks*), son un tipo especializado de redes neuronales para el procesamiento de datos que tienen una topología similar a una cuadrícula conocida. Tal es el caso de las imágenes, que pueden ser vistas como una cuadrícula 2D de píxeles [12]. Las redes convolucionales han sido muy exitosas en aplicaciones prácticas en los últimos años. Su nombre indica que emplean un operador matemático llamado convolución, el cual es un tipo especializado de operación lineal. Las CNN son simples redes neuronales que usan convolución en lugar de la multiplicación general de matrices, en al menos una de sus capas.

Las redes neuronales son un modelo computacional basado en un gran conjunto de unidades neuronales simples o *neuronas artificiales* de forma aproximadamente análoga al comportamiento observado en los axones de las neuronas en los cerebros biológicos. Las redes neuronales artificiales (RNA) son sistemas adaptativos complejos que pueden cambiar su estructura interna en función de la información que pasa por ella [13].

Su funcionamiento, en resumen, es sencillo de explicar. Con las entradas recibidas realiza una serie de operaciones, a esto se le aplican otros cálculos mediante una fórmula que transforma el valor recibido y produce un resultado numérico. Compu-

tacionalmente no son algorítmicas, no tienen algoritmos preestablecidos que les hagan seguir una secuencia de instrucciones, pero constan de funciones tan complicadas con tantos parámetros que finalmente son una solución matemática a un problema. En otras palabras, estos sistemas aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita [20].

La clase de problemas que mejor se resuelven con las redes neuronales son los mismos que el ser humano resuelve mejor: asociación, evaluación y reconocimiento de patrones. El objetivo de la red neuronal es resolver los problemas de la misma manera que el cerebro humano. Los proyectos de redes neuronales modernos suelen trabajar desde unos miles a unos pocos millones de unidades neuronales y, a la vez, millones de conexiones. Sin embargo, estos números siguen siendo de una magnitud menos compleja que la de la actividad cerebral humana.

2.3.1. La operación Convolución

En su forma más general, la convolución es una operación en dos funciones de un argumento de valor real. Para facilitar la definición, *Goodfellow* (2016) describe en [12] el siguiente ejemplo:

Suponiendo que se está rastreando la ubicación de una nave espacial con un sensor láser. El sensor entrega una única salida $x(t)$, que es la posición de la nave en el tiempo t. Tanto x, como t son valores reales, y se puede obtener un valor diferente leído por el sensor en cualquier instante de tiempo. Se supone ahora, que el láser es de alguna manera ruidoso. Para obtener una estimación menos ruidosa de la posición de la nave, sería conveniente promediar varias mediciones. Por su puesto, las mediciones más recientes son más relevantes, por lo que se quiere un promedio ponderado que otorgue más peso a las mediciones recientes. Esto es posible con una función de ponderación $w(a)$, donde a es la época de una medición. Si se aplica dicha operación de promedio ponderado en todo momento, se obtiene una nueva función s que entrega una estimación suavizada de la posición de la nave:

$$s(t) = \int x(a)\, w(t-a)\, da \tag{2.1}$$

Esta operación es llamada convolución. La operación de convolución normalmente es denotada por un asterisco:

$$s(t) = (x * w)(t) \tag{2.2}$$

En el ejemplo, w necesita ser una función de densidad de probabilidad válida, o la salida no será un promedio ponderado válido. También, w necesita ser cero para todos los argumentos negativos, o se estará viendo en el futuro, lo que presuntamente no es posible. Esta limitación es particular para el ejemplo. En general, la convolución es definida para cualquier función para la cual se define la integral anterior, y puede ser usada para otros propósitos además de tomar promedios ponderados.

En la terminología de las CNN, el primer argumento, para la convolución, es referido como la entrada y el segundo como *kernel* (o filtro). La salida a veces es referida como *mapa de características*.

Usualmente, cuando se trabaja con datos en una computadora, el tiempo es discretizado por lo que para el ejemplo, el sensor entregaría información en intervalos de tiempo regulares. El índice de tiempo t puede entonces tomar solo valores enteros. Si ahora se asume que x y w están definidos en el entero t, se puede definir la *convolución discreta*:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\, w(t-a) \tag{2.3}$$

En las aplicaciones de aprendizaje automático, la entrada es normalmente un arreglo multidimensional de datos y el kernel es normalmente un arreglo multidimensional de parámetros que son adaptados por el algoritmo de aprendizaje. Se referirá a estos arreglos multidimensionales como *tensores* debido a que cada elemento de la entrada y del kernel deben ser almacenados explícitamente por separado, normalmente se asume que estas funciones son cero en todas partes, excepto en el conjunto finito de puntos

para los cuales se almacenaron los valores. Esto significa que en la práctica se puede implementar la suma infinita como una suma sobre un número finito de elementos de un arreglo.

Finalmente, en general se usa la convolución sobre más de un eje al mismo tiempo. Por ejemplo, para una imagen bidimensional I como entrada, se querrá usar también, probablemente, un kernel bidimensional K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n) \tag{2.4}$$

La convolución tiene propiedades conmutativas, por lo que se puede escribir de manera equivalente:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \tag{2.5}$$

Por lo general, la última expresión es más sencilla de implementar en las bibliotecas de aprendizaje automático debido a que existe menos variación en el rango de valores válidos para m y n.

La propiedad conmutativa de la convolución surge porque se ha intercambiado el kernel en relación a la entrada, en el sentido de que a medida que m incrementa, el índice de la entrada también lo hace, pero el índice en el kernel decrece. La única razón para cambiar el kernel es para obtener la propiedad de conmutatividad. Mientras que esta propiedad es útil para escribir pruebas, normalmente no es una propiedad importante en la implementación de una red neuronal. En su lugar, muchas bibliotecas de redes neuronales implementan una función relacionada llamada *correlación cruzada*, que es igual a que la convolución pero sin cambiar el kernel:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.6}$$

Algunas bibliotecas de aprendizaje automático implementan la correlación cruzada pero la llaman convolución. Se le puede llamar a ambas operaciones convolución siempre y cuando se especifique cuando se haga un intercambio del kernel o no, en el contexto

donde este cambio sea relevante. En el contexto de aprendizaje automático, el algoritmo de aprendizaje aprenderá los valores apropiados del kernel en el lugar apropiado, así un algoritmo basado en convolución con el kernel intercambiado aprenderá un kernel que está *volteado* en relación con el kernel aprendido por un algoritmo sin el intercambio. También es poco común que se use la convolución sola para aprendizaje automático; en su lugar la convolución es usada de manera simultánea con otras funciones y la combinación de éstas no conmuta sin importar si la operación de convolución voltea su kernel o no.

La convolución discreta puede ser vista como una multiplicación por una matriz. Sin embargo, para convolución discreta univariada, cada renglón de la matriz está limitado a ser igual al renglón anterior desplazado por un elemento. Esto es conocido como una *matriz Toeplitz*. En dos dimensiones, una *matriz de doble circuito de bloqueo* corresponde a la convolución. Adicionalmente a estas limitaciones de que varios elementos deben ser iguales a otros, la convolución generalmente corresponde a una matriz muy dispersa (matriz cuyas entradas son en su mayoría igual a cero). Esto es porque el kernel es normalmente mucho más pequeño que la imagen de entrada.

Cualquier algoritmo de red neuronal que trabaje con multiplicación de matrices, y no dependa de especificar las propiedades de la estructura de la matriz, debería trabajar con convolución sin requerir ningún cambio a la red. Las CNN comunes sí hacen uso de otras especializaciones para lidiar de manera eficiente con entradas grandes, pero no es estrictamente necesario desde el punto de vista teórico.

2.3.2. Estructura

De manera específica y detallada, lo que una hace una CNN al procesar una imagen se resume a que toma dicha imagen, la pasa por una serie de capas convolucionales, no lineales, de reducción y completamente conectadas, y obtiene una salida. Esta salida puede ser una sola clase o una probabilidad de clases que mejor describa la imagen o el contenido de la misma. La parte más importante es comprender qué hace cada una de estas capas.

2.3.2.1. Capa convolucional

La primera capa en una CNN es siempre una capa convolucional. Considerando como entrada, por ejemplo, un arreglo de valores de píxeles de 32x32x3, la mejor manera de explicar la capa de convolución es imaginando una linterna que está alumbrando sobre la parte superior izquierda de la imagen y suponiendo que el brillo de la linterna cubre un área de 5x5 (Figura2.2). En términos de aprendizaje automático la linterna es el llamado *filtro* o *kernel* y la región a la que está alumbrando es llamada *campo receptivo*. Ahora, este filtro es también un arreglo de números, llamados pesos o parámetros. La profundidad de este filtro debe ser la misma que la de la imagen de entrada para que las operaciones matemáticas funcionen, por lo que la dimensión del filtro es de 5x5x3. Tomando la primera posición en la que se encuentra el filtro (esquina superior izquierda), a medida que éste se desliza, o convoluciona, alrededor de la imagen de entrada, está multiplicando los valores en el filtro con los valores de los píxeles originales de la imagen (multiplicación elemento por elemento). Estas multiplicaciones se suman, resultando en un solo número. Este proceso se repite para cada posición en el volumen de entrada, moviendo el filtro a la derecha una unidad a la vez. Cada posición única en el volumen de entrada produce un número. Después de deslizar el filtro sobre todas las posiciones se obtiene un arreglo de números de 28x28x1 el cual que se conoce como *mapa de activación* o *mapa de características*. Usando dos filtros de 5x5x3 en lugar de uno, el volumen de salida sería de 28x28x2. Mientras más filtros se utilizan, es posible preservar mejor la dimensión espacial [6].

Cada uno de estos filtros puede ser visto como *identificadores de características*, siendo estas características bordes, colores, curvas, etc., pensando en las características más simples que todas las imágenes tienen en común. Suponiendo que un primer filtro es de 7x7x3 y es un detector de curvas, dejando de lado que la profundidad del filtro es de 3 y solo considerando la primera unidad de profundidad tanto del filtro como de la imagen, por simplicidad. Como detector de curvas el filtro debe tener una estructura de píxeles en donde habrá valores numéricos más altos en el área donde se forma una curva, tal como se muestra en la Figura 2.3.

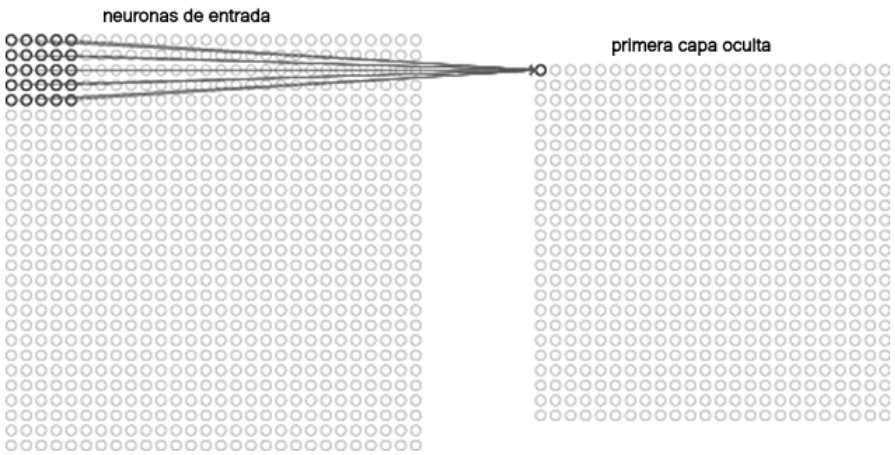


Figura 2.2: Visualización de filtro 5x5 convolucionando alrededor de un volumen de entrada de 32x32 y produciendo un mapa de activación de 28x28.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

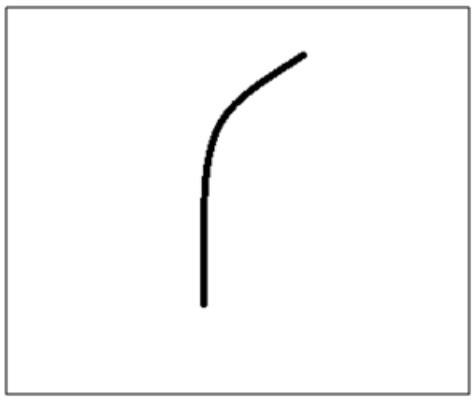


Figura 2.3: Representación en píxeles de un filtro detector de curvas [6].

Visualizando esto matemáticamente, cuando se tiene el filtro en la esquina superior izquierda del volumen de entrada (Figura 2.4), se están calculando multiplicaciones entre el filtro y los valores de los píxeles en esa región.

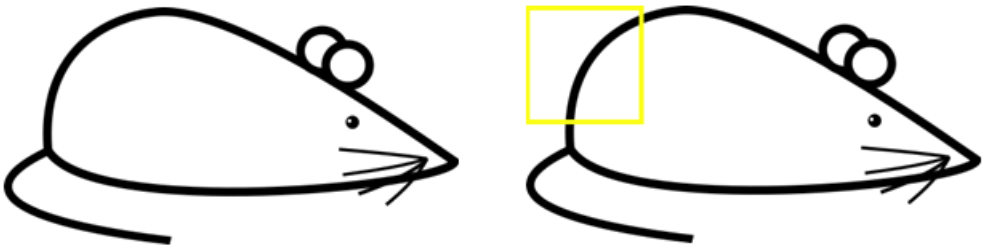


Figura 2.4: Imagen de entrada y la visualización del filtro sobre la imagen [6].

Hay que recordar que lo que se tiene que hacer es multiplicar los valores en el filtro con los valores de los píxeles originales de la imagen, Figura 2.5. Básicamente, en la

imagen de entrada, si existe una forma que generalmente se asemeje a la curva que representa el filtro, entonces todas las multiplicaciones sumadas juntas resultaran en un valor alto, tal como muestra la ecuación 2.7. En el caso de que no haya nada en la sección de la imagen que responda al filtro detector de curvas, el valor será mucho menor. Un alto valor significa que es probable que en esa posición haya algún tipo de curva en el volumen de entrada que provoca que el filtro se active. Se pueden tener más filtros para detectar más líneas como curvas hacia la izquierda o bordes rectos, etc. Mientras más filtros, mayor es la profundidad del mapa de activación y más información se tiene del volumen de entrada.

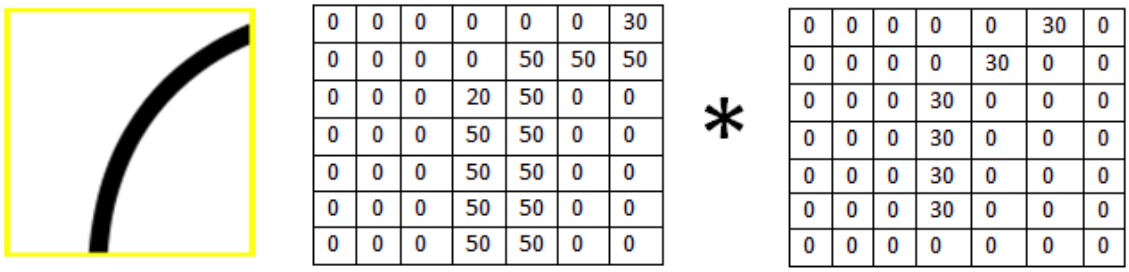


Figura 2.5: Multiplicación de las representaciones en píxeles del campo receptivo y filtro [6].

$$R = (50 * 30) + (50 * 30) + (50 * 30) + (20 * 30) + (50 * 30) = 6600 \qquad (2.7)$$

2.3.2.2. Paso y Relleno

Existen dos parámetros que se pueden cambiar para modificar el comportamiento de cada capa convolucional. Después de escoger el tamaño del filtro o kernel, también se tiene que escoger el paso (o *stride*) y el relleno (o *padding*).

El *paso* controla cómo el filtro convoluciona alrededor de el volumen de entrada y es la cantidad para la cual el filtro se mueve. El caso más común es que el filtro convolucione alrededor del volumen de entrada moviéndose una unidad a la vez. Normalmente, el paso se fija en una manera en que el volumen de salida sea un entero y no una fracción. Por ejemplo, imaginando un volumen de entrada de 7x7, un filtro de 3x3, y un paso de 1, la convolución se realizaría como se presenta en la Figura 2.6. En caso de que el paso aumentara su valor a 2, el volumen de salida sería como el de la Figura 2.7.

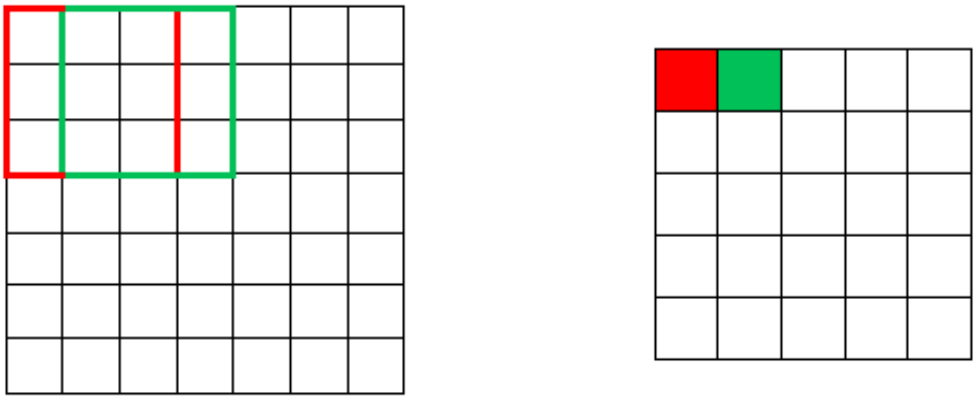


Figura 2.6: Ejemplo del movimiento del filtro en la convolución con paso 1: Los colores muestran el píxel resultante de la operación sobre el volumen de entrada.

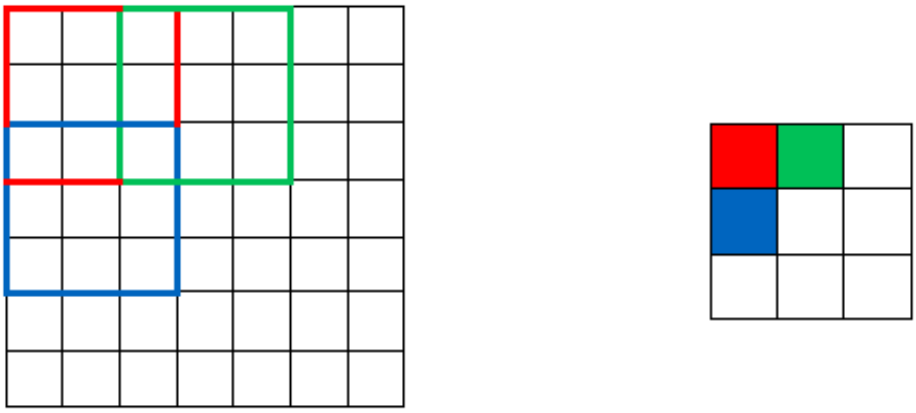


Figura 2.7: Ejemplo del movimiento del filtro en la convolución con paso 2: Los colores muestran el píxel resultante de la operación sobre el volumen de entrada.

Como se observa, el campo receptivo se mueve cada 2 unidades y el volumen de salida se encoje. Nótese que si se trata de fijar el paso en 3, se tendrían problemas con el espaciado y al tratar de asegurarse que los campos receptivos se ajusten en el volumen de entrada. Normalmente, es posible programar un incremento del paso si se desea campos receptivos que se superpongan menos y si se quieren dimensiones espaciales más pequeñas.

Con respecto al *relleno*, si se aplican tres filtros de 5x5x3 a un volumen de entrada de 32x32x3, el volumen de salida sería de 28x28x3. Nótese que la dimensión espacial se redujo. Conforme se sigan aplicando capas convolucionales, el tamaño del volumen decrecerá más rápido de lo que se quisiera. En las primeras capas de la red, lo que se quiere es preservar tanta información acerca del volumen de entrada original para

poder extraer las características de niveles inferiores. Si lo que se quiere es aplicar la misma capa convolucional y que el volumen de salida permanezca en 32x32x32, se puede aplicar un relleno de ceros de tamaño 2 a esa capa. El relleno cero rellena el volumen de entrada con ceros alrededor de sus bordes. Entonces esto resultaría en un volumen de entrada de 36x36x3 como muestra en la Figura 2.8.

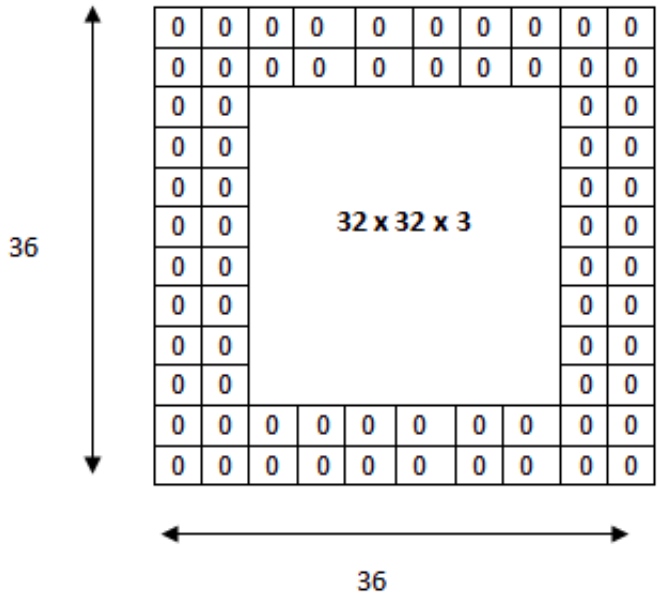


Figura 2.8: El volumen de entrada es de 32x32x3. Si se imaginan dos bordes de ceros alrededor del volumen, esto se vuelve un volumen de 36x36x3. Después cuando se aplique la convolución con los tres filtros de 5x5x3, se tendrá un volumen de salida de 32x32x3.

Si se tiene un paso fijo en 1 y fijando el relleno de la manera en que se muestra en la ecuación 2.8:

$$ZeroPadding = \frac{(K-1)}{2} \tag{2.8}$$

Donde K es el tamaño del filtro, entonces el volumen de la entrada y de la salida siempre tendrán las mismas dimensiones espaciales. La fórmula para calcular el tamaño de la salida para cualquier capa convolucional dada es:

$$O = \frac{(W-K+2P)}{S} + 1 \tag{2.9}$$

Donde O es la longitud de la salida, W es la longitud de entrada, K es el tamaño del filtro, P es el relleno, y S es el paso.

2.3.2.3. Hiperparámetros

Responder a las preguntas acerca de cómo saber cuántas capas usar, cuántas de ellas convolucionales, cuál es el tamaño apropiado del filtro o cuáles los valores para el paso y el relleno que se deben usar, no es sencillo. Realmente no está definido un estándar que sea usado por todos los investigadores. Esto es debido a que la red dependerá en grandes rasgos del tipo de información que se tenga. Ésta puede variar en tamaño, complejidad de la imagen, tipo de tarea de procesamiento de imagen, entre otros. Una manera de pensar cómo elegir los hiperparámetros, considerando el conjunto de datos que se maneja, es tratando de encontrar la combinación correcta para crear abstracciones de la imagen a una escala adecuada. En una arquitectura tradicional de redes neuronales convolucionales existen otras capas que están entremezcladas entre las capas convolucionales. En un sentido general estas capas le otorgan a la red no linealidades y preservación de la dimensión que ayuda a mejorar la robustez de la red y a controlar el sobreajuste [6]. Una arquitectura clásica de CNN estaría compuesta de la siguiente manera:

$$Entrada \rightarrow Conv \rightarrow ReLU \rightarrow Conv \rightarrow ReLU \rightarrow Pool \rightarrow C.Conectada$$

2.3.2.4. Capas ReLU

Seguido de una cada capa convolucional, se aplica una capa de activación no lineal inmediatamente después. El propósito de esta capa es introducir la no linealidad a un sistema que básicamente solo ha estado haciendo operaciones lineales durante las capas convolucionales, estas operaciones son multiplicación de elemento por elemento y sumatorias. En el pasado funciones no lineales como tanh o la sigmoide eran usadas, pero investigadores han encontrado que las capas ReLU (Unidades Lineales Rectificadas) funcionan mucho mejor debido a que la red es capaz de entrenar mucho más rápido (por la eficiencia computacional) sin causar una diferencia significativa en la precisión. También ayuda a aliviar el problema del desvanecimiento del gradiente, que es el problema donde las capas más bajas de la red se entrenan muy lento debido a que el gradiente decrece de manera exponencial a través de las capas.

La capa ReLU aplica la siguiente función:

$$f(x) = \text{máx}(0, x) \tag{2.10}$$

Esta función es aplicada a todos los valores del volumen de entrada. En términos básicos, esta capa solo cambia todas las activaciones negativas a cero. La capa incrementa las propiedades no lineales del modelo y de la red en general, sin afectar los campos receptivos de la capa convolucional.

2.3.2.5. Capa de reducción

Una típica capa de una red convolucional consiste de tres etapas. En la primera, la capa realiza varias operaciones de convolución en paralelo para producir un conjunto de activaciones lineares. En la segunda etapa, cada activación linear se pasa por una función de activación no lineal, como la función de activación lineal rectificada. Esta etapa es llamada algunas veces la etapa *detector*. En la tercera etapa, se usa una función de reducción o *pooling* para modificar la salida de la capa más lejana.

Después de algunas capas ReLU, se suele aplicar una capa de reducción, también referida como una capa de submuestreo. En esta categoría hay varias opciones de capas, siendo la capa *maxpooling* la más popular de todas. Ésta básicamente toma un filtro (normalmente de tamaño 2x2) y un paso de la misma longitud, luego lo aplica al volumen de entrada y entrega a la salida el número máximo en cada subregión en los que el filtro convoluciona alrededor.

Otras opciones para las capas de reducción son la reducción promedio (*average pooling*) y la reducción *L2-norm*. El razonamiento intuitivo detrás de esta capa es que una vez que se sabe que una característica específica se encuentra en el volumen de entrada original (donde habrá un valor de activación alto), su ubicación exacta no es tan importante como su ubicación relativa para las otras características. Esta capa reduce drásticamente la dimensión espacial (sin modificar la profundidad) del volumen de entrada. Esto sirve para dos propósitos principales. El primero es que la cantidad de parámetros o pesos es reducida al 75 %, disminuyendo así el costo computacional. El

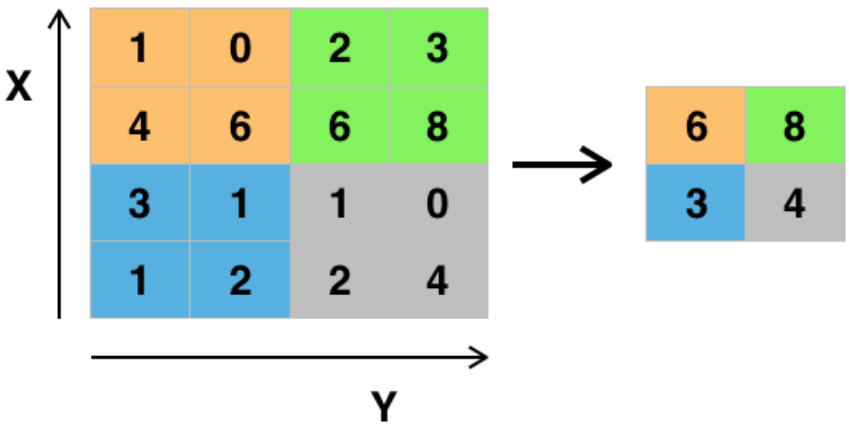


Figura 2.9: Ejemplo de *maxpooling* con un filtro 2x2 y un paso de 2.

segundo es que controlará el *sobreajuste*. Este término se refiere a cuando un modelo es tan ajustado a los ejemplos del entrenamiento que no es capaz de generalizar bien para los conjuntos de validación o de prueba. Un ejemplo del sobreajuste es cuando se tiene un modelo que obtiene un 100 % o 99 % de efectividad en el conjunto de entrenamiento pero solo el 50 % con los datos de prueba.

2.3.2.6. Capas dropout

Las capas *dropout* tienen una función muy específica en las redes neuronales. En la sección anterior se discutió el problema del sobreajuste, donde después del entrenamiento, los pesos de la red están tan ajustados a los ejemplos de entrenamiento que le fueron dados que la red no se desempeña bien cuando se le dan nuevos ejemplos. La idea del *dropout* es simple por naturaleza. Estas capas *omiten* un conjunto aleatorio de activaciones en esa capa al definirlas en cero. En cierta manera, obliga a la red a ser redundante. Eso significa que la red debe ser capaz de entregar la clasificación o salida correcta para un ejemplo específico aun cuando algunas activaciones son omitidas. Lo que asegura que la red no se vuelva muy ajustada a los datos de entrenamiento y ayuda a evitar el problema del sobreajuste. Cabe mencionar que esta capa sólo se usa durante el entrenamiento y no durante las pruebas.

2.3.2.7. Capa completamente conectada

Ya que se pueden detectar características de alto nivel con arreglos de capas anteriores, lo siguiente es unir una capa completamente conectada al final de la red. Esta capa básicamente toma un volumen de entrada y, sin importar de qué otra capa provenga, entrega a la salida un vector N-dimensional donde N es el número de clases de las que el programa tiene que escoger. Por ejemplo, para un programa clasificador de dígitos, N sería 10 ya que existen 10 dígitos. Cada número en este vector N-dimensional representa la probabilidad de una cierta clase. Por ejemplo, si el vector resultante del programa anterior es $\begin{bmatrix} 0 & 0,1 & 0,1 & 0,75 & 0 & 0 & 0 & 0 & 0 & 0,05 \end{bmatrix}$, esto representa 10 % de probabilidad de que la imagen sea un 1 o un 2, 75 % de probabilidad de que la imagen sea un 3 y 5 % de probabilidad de que sea un 9.

Existen algunas otras maneras de representar la salida, la anterior es del enfoque conocido como *softmax*. La manera en que esta capa completamente conectada funciona es que mira en la salida de la capa anterior y determina qué características se correlacionan más con una clase particular. Básicamente, una capa completamente conectada detecta qué características de alto nivel correlaciona más fuertemente con una clase particular y tiene pesos particulares, así cuando se calculen los productos entre los pesos y las capas anteriores, se obtendrán las probabilidades correctas para las diferentes clases.

Finalmente, la Figura 2.10 muestra una representación de un ejemplo de red neuronal convolucional.

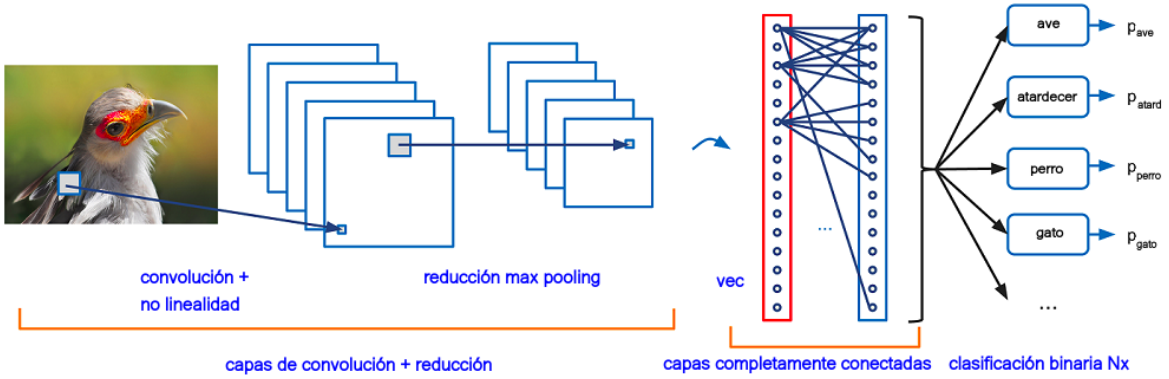


Figura 2.10: Ejemplo de estructura de una CNN [6].

2.3.3. Entrenamiento

El entrenamiento es uno de los más importantes aspectos de una red neuronal. La manera en la que la computadora es capaz de ajustar los valores de los filtros, o los pesos, es a través del proceso de entrenamiento llamado *backpropagation* o propagación hacia atrás.

Antes de empezar, los pesos o valores del filtro, son inicializados de manera aleatoria. Los filtros, en cualquiera de las capas, no saben "buscar" por ninguna característica en particular. El proceso de entrenamiento por el que pasan las CNN consiste en la idea de darle a la red una imagen y una etiqueta para aprender. Se tiene un conjunto de entrenamiento que contiene miles de imágenes con objetos y cada una de estas imágenes tiene una etiqueta de qué objeto está en esa imagen. El proceso de la propagación hacia atrás puede estar separado en 4 secciones distintas: 1) el paso hacia adelante; 2) la función de costo; 3) el paso hacia atrás y; 4) la actualización de los pesos.

Durante el paso hacia adelante, se toma una imagen de entrenamiento y se pasa por toda la red. Ya que los pesos o valores del filtro fueron inicializados aleatoriamente, la salida obtenida con el primer ejemplo de entrenamiento probablemente será una salida que no da preferencia a ninguna clase en particular. La red con sus valores actuales no es capaz de buscar esas características de bajo nivel, por lo que tampoco es capaz de hacer ninguna conclusión razonable acerca de qué clase debería ser. Esto lleva a la parte de la *función de costo*. Una función de costo puede ser definida de muchas maneras pero una muy común es el error cuadrado promedio o MSE (del inglés *Mean Squared Error*):

$$E_{total} = \sum \frac{1}{2} \left(valor_{deseado} - valor_{predicho} \right)^2 \tag{2.11}$$

Considerando que la variable L es igual a este valor. Como se puede imaginar, el error será extremadamente alto para las primeras parejas de imágenes de entrenamiento. Pensando de manera intuitiva, se quiere llegar a un punto donde la etiqueta predicha (salida de la red convolucional) sea la misma que la etiqueta de entrenamiento. Para llegar a ello, se necesita minimizar la cantidad de error que se tiene. Visualizando esto como un problema de optimización en cálculo, se quiere encontrar qué puntos (pesos

en este caso) contribuyen más directamente al error en la red.

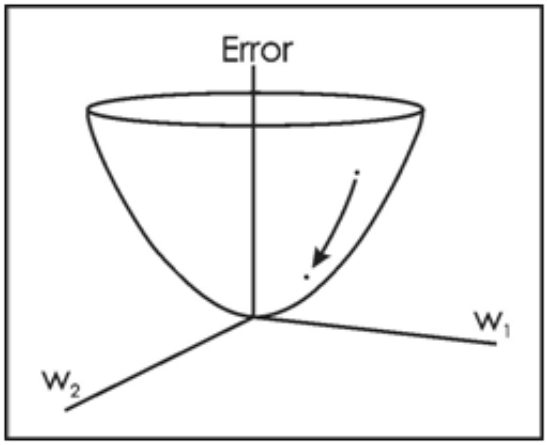


Figura 2.11: Una manera de visualizar la idea de minimizar el error es considerando una gráfica 3D donde los pesos de la red neuronal son las variables independientes y las variables dependientes son el error. La tarea de minimizar el error involucra el tratar de ajustar los pesos para que el error decrezca. En términos visuales, se quiere alcanzar el punto más bajo en el objeto con forma de tazón [23].

Para lograr lo descrito en la Figura 2.11 se tiene que tomar la derivada del error con respecto a los pesos. Esto es matemáticamente equivalente a dL/dW, donde W son los pesos en una capa particular. Ahora, lo que se quiere es llevar a cabo un *paso hacia atrás* a través de la red, lo cual es determinando qué pesos contribuyen más al error y encontrar la manera de ajustarlos para que el error se reduzca. Una vez calculada esta derivada, se pasa al último paso que es la *actualización de los pesos*. Aquí se toman todos los pesos de los filtros y se actualizan para que cambien en la dirección opuesta al gradiente.

$$W = W_i - \eta \frac{dL}{dW} \tag{2.12}$$

Donde W es el nuevo peso, W_i es el peso inicial y η es el factor de aprendizaje. El *factor de aprendizaje* es un parámetro que es escogido por el programador. Un alto valor significa pasos más grandes tomados en la actualización de los pesos y así puede tomar menos tiempo para el modelo en converger en un conjunto de pesos óptimos. Sin embargo, un factor de aprendizaje que es muy alto puede resultar en saltos muy largos y no suficientemente precisos para alcanzar el punto óptimo.

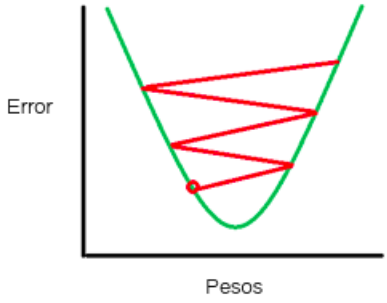


Figura 2.12: Consecuencia de un valor muy alto para el factor de aprendizaje donde los saltos son muy largos y no se es capaz de minimizar el error [23].

El proceso de las cuatro etapas anteriormente descritas, es una *iteración*. El programa repetirá este proceso por un número fijo de iteraciones por cada conjunto de imágenes de entrenamiento, también llamado lote o *batch*. Una vez que finaliza la actualización de parámetros en el último ejemplo de entrenamiento, la red deberá estar entrenada lo suficientemente bien si es que los pesos fueron ajustados correctamente.

2.3.4. Pruebas

Finalmente, para comprobar el funcionamiento de la red, se toma un conjunto diferente de imágenes y etiquetas, las cuales no hayan sido presentadas para el entrenamiento, y se pasan estas imágenes a través de la red neuronal convolucional. Se comparan las salidas obtenidas con los valores de las etiquetas y se calcula la precisión de la red.

2.4. Transferencia de aprendizaje

La Transferencia de Aprendizaje, del inglés *Transfer Learning*, es conocido como la mejora del aprendizaje en una tarea nueva, a través de la transferencia del conocimiento adquirido de una tarea similar que ya ha sido aprendida. Mientras la mayoría de los algoritmos de aprendizaje automático están diseñados para abordar tareas individuales, el desarrollo de algoritmos que facilitan la transferencia de aprendizaje es un tema de interés continuo en la comunidad del aprendizaje automático [33].

2.4.1. Analogía con el aprendizaje humano

El aprendizaje humano parece tener maneras inherentes de transferir conocimiento entre tareas. Es decir, reconocer y aplicar conocimiento relevante de experiencias de aprendizaje anteriores al encontrarse con nuevas tareas. Mientras más relacionada esté la nueva tarea con las experiencias previas, más fácilmente se puede completar. Los algoritmos de aprendizaje comunes, en contraste, normalmente abarcan tareas aisladas. La transferencia de aprendizaje procura cambiar esto al desarrollar métodos para transferir conocimiento aprendido en una o más *tareas fuente* y usarlo para mejorar el aprendizaje en una *tarea objetivo* relacionada. Las técnicas que permiten la transferencia de conocimiento representan un progreso en el camino de hacer el aprendizaje automático tan eficiente como el aprendizaje humano.

2.4.2. Objetivo

El objetivo de la transferencia de aprendizaje es mejorar el aprendizaje en la tarea objetivo aprovechando el conocimiento de una tarea fuente. Existen tres medidas comunes por las cuales la transferencia puede mejorar el aprendizaje. Primero, está el rendimiento inicial que se puede lograr en la tarea objetivo utilizando solo el conocimiento transferido, antes de que se realice cualquier aprendizaje adicional, en comparación con el rendimiento inicial de un agente ignorante. En segundo lugar, está la cantidad de tiempo que toma el aprender completamente la tarea objetivo dado el conocimiento transferido, comparado con la cantidad de tiempo para aprenderlo desde cero. Por último, está el nivel de desempeño final que se puede lograr en la tarea objetivo, comparado con el nivel final sin transferencia [33].

Si el método de transferencia en realidad disminuye el rendimiento, entonces ha ocurrido una transferencia negativa. Uno de los mayores retos en el desarrollo de métodos de transferencia es el producir transferencias positivas entre tareas relacionadas apropiadamente, mientras se evita la transferencia negativa entre tareas que están menos relacionadas.

2.4.3. Transferencia de aprendizaje en la práctica

En la práctica, muy poca gente entrena una red convolucional completa desde cero porque es relativamente *raro*, por no decir muy complicado, contar con un conjunto de datos de tamaño suficiente. En lugar de ello, es común *preentrenar* una red convolucional con un conjunto de datos muy grande (*ImageNet*, por ejemplo), y después utilizar esa red ya sea como una inicialización o como un extractor de características fijo para la tarea de interés. Los tres mayores escenarios para la transferencia de aprendizaje son: a) Red convolucional como extractor de características fijo, b) Hacer una afinación a la red convolucional, y c) Modelos preentrenados.

Red convolucional como extractor de características fijo

Consiste en tomar una red neuronal preentrenada con el conjunto de datos *ImageNet* (siguiendo el ejemplo), remover la última capa completamente conectada, cuyas salidas son los 1000 puntajes de clases para una diferente tarea como *ImageNet*, entonces se trata al resto de la red como un extractor de características fijo para el nuevo conjunto de datos. Finalmente se entrena un clasificador lineal como un enfoque SVM o *softmax*, para la nueva información.

Afinación de la red convolucional

La segunda estrategia es no solo remplazar y reentrenar el clasificador en la cima de la red con un nuevo conjunto de datos, sino que también afinar los pesos de la red preentrenada al continuar con la propagación hacia atrás. Es posible afinar todas las capas de la red convolucional, o es posible mantener algunas de las capas iniciales fijas (por preocupaciones de sobreajuste) y solo afinar alguna porción del nivel más alto de la red. Esto está motivado por la observación de que las primeras capas de la red contienen más características genéricas (detectores de bordes, o de color, por ejemplo) que pueden ser muy útiles para muchas tareas, pero las capas finales de la red se convierten progresivamente más específicas a los detalles del contenido de las clases del conjunto de datos original.

Modelos preentrenados

Ya que las redes convolucionales modernas toman de 2 a 3 semanas para entrenarse a través de múltiples GPUs con *ImageNet*, es muy común ver que algunas personas liberan los puntos de control finales de sus red convolucional para el beneficio de otros que pueden usar estas redes para afinarlas.

2.4.4. Consideraciones para realizar la afinación

Decidir qué tipo de transferencia de aprendizaje utilizar en un nuevo conjunto de datos depende de muchos factores, pero los dos más importantes son el tamaño del nuevo conjunto de datos, que puede ser grande o pequeño, y su similitud con el conjunto de datos original. Hay que tener en cuenta que las características de la red convolucional son más genéricas en las primeras capas y más específicas, con respecto al conjunto de datos original, en las últimas. Las siguientes son algunas reglas comunes para navegar por los 4 principales escenarios posibles:

El nuevo conjunto de datos es chico y similar que el original

Ya que el tamaño de los datos es menor, no es buena idea afinar la red por preocupación al sobreajuste. Debido a que la información es similar a la original, se espera que las características de alto nivel en la red sean relevantes al nuevo conjunto de datos también. Por lo tanto, la mejor idea podría ser entrenar un clasificador lineal en los códigos de la red convolucional.

El nuevo conjunto de datos es grande y similar al original

Ahora que se tiene más datos, se puede tener más confianza de que no se caerá en sobreajuste si se trata de afinar a través de toda la red.

El nuevo conjunto de datos es chico pero muy diferente del original

Ya que el conjunto de datos es pequeño es más probable que sea mejor entrenar un clasificador lineal. El que sea diferente al original, hace que posiblemente no sea lo mejor

entrenar el clasificador desde la cima de la red, donde se encuentran las características específicas. En su lugar lo mejor sería entrenar el clasificador lineal en las activaciones iniciales en la red.

El nuevo conjunto de datos es grande pero muy diferente del original

Ya que el conjunto de datos es grande, se puede esperar que se pueda entrenar la red convolucional desde cero. Sin embargo, en la práctica a menudo es mejor inicializar con los pesos de un modelo preentrenado. En este caso, se tendría datos suficientes y la confianza para hacer la afinación a través de toda la red.

2.4.5. Consejos prácticos

Existen algunos detalles adicionales que se deben considerar al llevar a cabo la transferencia de aprendizaje:

- *Restricciones por parte de los modelos preentrenados.* Tomando en cuenta que si se desea utilizar una red preentrenada, se puede estar ligeramente limitado en términos de la arquitectura que se puede usar para un nuevo conjunto de datos. Por ejemplo, no se podría remover arbitrariamente copas convolucionales de la red preentrenada. Sin embargo, algunos cambios son posibles: Debido a que se comparten parámetros, se puede correr fácilmente una red preentrenada en imágenes de diferentes dimensiones espaciales.

- *Factores de aprendizaje.* Es común usar un valor pequeño para el factor de aprendizaje de los pesos de la red convolucional que están siendo afinados, en comparación con los pesos (inicializados aleatoriamente) para el nuevo clasificador lineal que calcula los puntajes de clase del nuevo conjunto de datos. Esto es porque se espera que los pesos de la red sean relativamente buenos, así que no se desea distorsionarlos muy rápido ni mucho, especialmente mientras el nuevo clasificador lineal que se encuentra sobre ellos está siendo entrenado con inicialización aleatoria.

Capítulo 3

Desarrollo

La metodología seguida para el desarrollo de este proyecto es la mostrada en la Figura 3.1 y se describe a continuación:

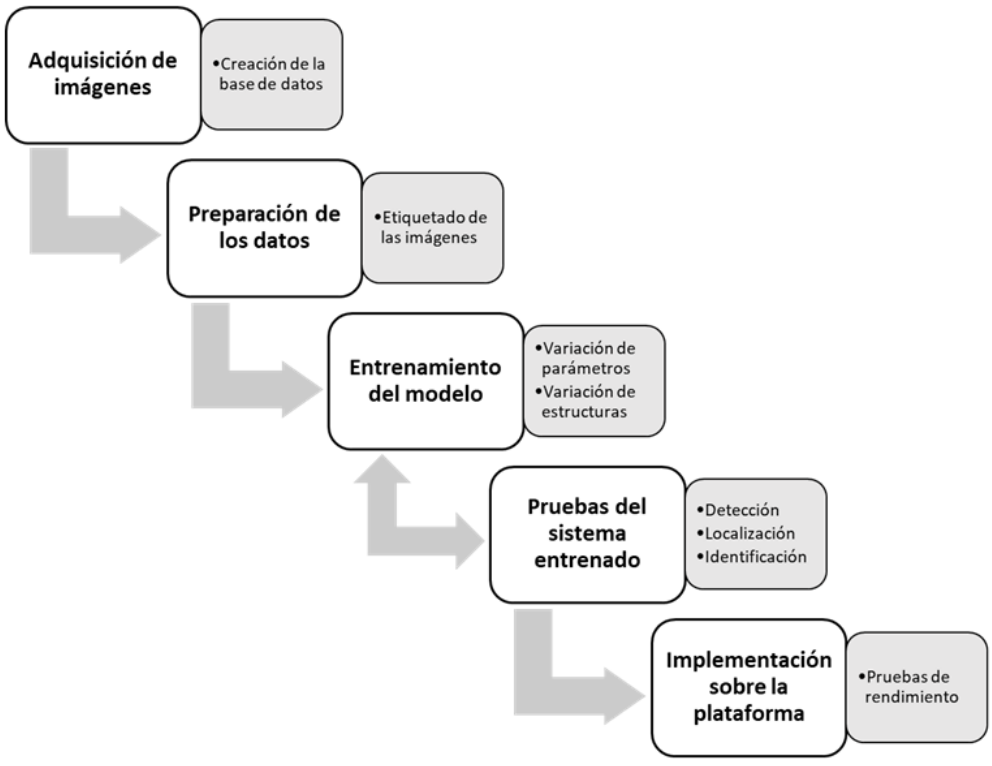


Figura 3.1: Metodología para el desarrollo del proyecto.

3.1. Adquisición de imágenes

La adquisición de todas las imágenes utilizadas para el entrenamiento y para las pruebas del sistema de detección se ha llevado a cabo por la plataforma del robot NAO, más específicamente, por la cámara superior de la cabeza del robot. Estas imágenes, con resolución de 640x480 píxeles, han sido recolectadas y agrupadas en una base de datos que consta de dos conjuntos de imágenes. El primer conjunto, llamado *conjunto de entrenamiento*, contiene el mayor número de imágenes debido a que son estos datos con los que el sistema aprenderá a detectar los objetos de interés. En la literatura se recomienda que el conjunto de entrenamiento esté formado por entre el 70 % y el 90 % del total de las imágenes de la base de datos, elegido aleatoriamente [22]. Por otro lado, las imágenes restantes corresponden al *conjunto de prueba*. Este conjunto de imágenes debe de ser completamente diferente al conjunto de entrenamiento, es decir, debe ser información nueva y que no haya sido presentada anteriormente al sistema. Con este grupo de imágenes es posible realizar una validación del entrenamiento mientras es llevado a cabo, sin embargo, su función principal es la de evaluar el resultado final del entrenamiento calculando el porcentaje de precisión del sistema en la detección, localización e identificación de los objetos de interés.

3.2. Preparación de los datos

Como toda tarea de aprendizaje profundo, la preparación de los datos es lo primero más importante a realizar. Ya con la base de datos separada en dos conjuntos de imágenes, lo siguiente es el etiquetar la información. El etiquetado de los datos es indispensable pues el entrenamiento realizado es para un aprendizaje supervisado, por lo que el sistema necesita que se le indique, en el caso de las imágenes, en dónde se encuentra el o los objetos de interés y, de haber más de una clase de objetos, que se le indique a cuál pertenece cada objeto en la imagen. El formato para el etiquetado de imágenes puede variar de un enfoque de entrenamiento a otro, ya que dependiendo de los diferentes algoritmos y plataformas que hay en la actualidad, el sistema recibe diversos parámetros dentro de cada archivo de etiqueta. El enfoque seleccionado para

el desarrollo de los entrenamientos a realizar, ha sido el proporcionado por *Darknet*.

3.2.1. *Darknet*

Darknet es un marco de trabajo de código abierto para el aprendizaje profundo y redes neuronales escrito en C. Es rápido y de instalación sencilla, permite la computación tanto para CPU, como para GPU. Este enfoque permite trabajar con varios tipos o formatos de imágenes al soportar la compilación con la biblioteca de visión artificial OpenCV. Toda la información necesaria para la instalación y construcción del entorno *Darknet* se encuentra disponible en [25].

3.2.2. Etiquetado de las imágenes

El etiquetado de los datos es, junto al entrenamiento en sí, uno de los pasos más demandantes en cuanto a tiempo se refiere ya que este proceso se realiza de manera manual. Para el enfoque seleccionado requiere que por cada una de las imágenes de los conjuntos antes mencionados, exista un archivo de texto (de mismo nombre y con extensión *.txt*) que contenga la información de los objetos de interés en la respectiva imagen que entra al modelo. Cada renglón en este archivo de etiqueta representa un *bounding box* o *cuadro delimitador* en la imagen, éste es un cuadro que encierra o envuelve, en su totalidad o parcialidad, al objeto que se encuentra en la imagen. La información de cada cuadro delimitador de la etiqueta se estructura de la siguiente manera:

$$< class - object - id >< center - x >< center - y >< width >< height >$$

El primer campo *class-object-id* corresponde a un número entero que representa un identificador de la clase a la que pertenece el objeto. El rango de este valor es $\{0, n-1\}$, siendo n el número de clases totales. En el caso del entrenamiento para una sola clase, este campo estará fijado siempre en cero. El segundo y tercer campo, *center-x* y *center-y*, corresponden respectivamente a las coordenadas en x y y del centro del cuadro delimitador, este valor esta normalizado por el ancho de la imagen, en píxeles,

para el valor de x y por el valor del alto de la imagen, para el de y. El cuarto y quinto campo, *width* y *height*, son el ancho y alto, respectivamente, del cuadro delimitador, de nuevo normalizados con respecto al ancho y alto de la imagen. La normalización de los últimos cuatro campos indica que estos valores corresponden a un número que se encuentra entre 0 y 1.

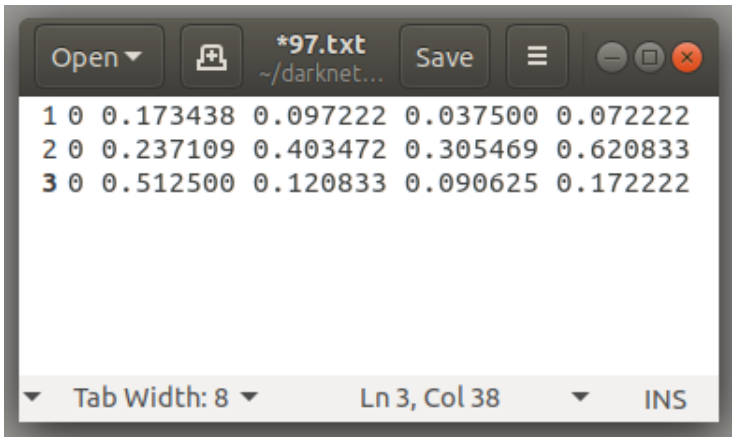


Figura 3.2: Ejemplo de archivo de etiqueta para tres objetos de la misma clase.

Considerando los siguientes datos:

x - Coordenada en x (en píxeles) del centro del cuadro delimitador
y - Coordenada en y (en píxeles) del centro del cuadro delimitador
w - Valor del ancho (en píxeles) del cuadro delimitador
h - Valor del alto (en píxeles) del cuadro delimitador
W - Valor del ancho (en píxeles) de toda la imagen
H - Valor del alto (en píxeles) de toda la imagen

Los valores de los campos correspondientes para el archivo de etiqueta son calculados de la siguiente manera:

$$center_x = \frac{x}{W} \tag{3.1}$$

$$center_y = \frac{y}{H} \tag{3.2}$$

$$width = \frac{w}{W} \tag{3.3}$$

$$height = \frac{h}{H} \tag{3.4}$$

Existen algunas *interfaces gráficas de usuario* o GUI, por sus siglas en inglés, que permiten realizar el etiquetado de manera gráfica y con el formato requerido por los diferentes enfoques de entrenamiento. Para el formato necesario para trabajar con el enfoque *Darknet*, se encuentra disponible el software *Yolomark*, el cual se presenta en [4]. Un ejemplo del uso de esta herramienta se presenta en las Figura 3.3. Hay que recordar que se debe realizar el etiquetado manual de todas y cada una de las imágenes contenidas en la o las bases de datos con las que se desea entrenar al sistema.

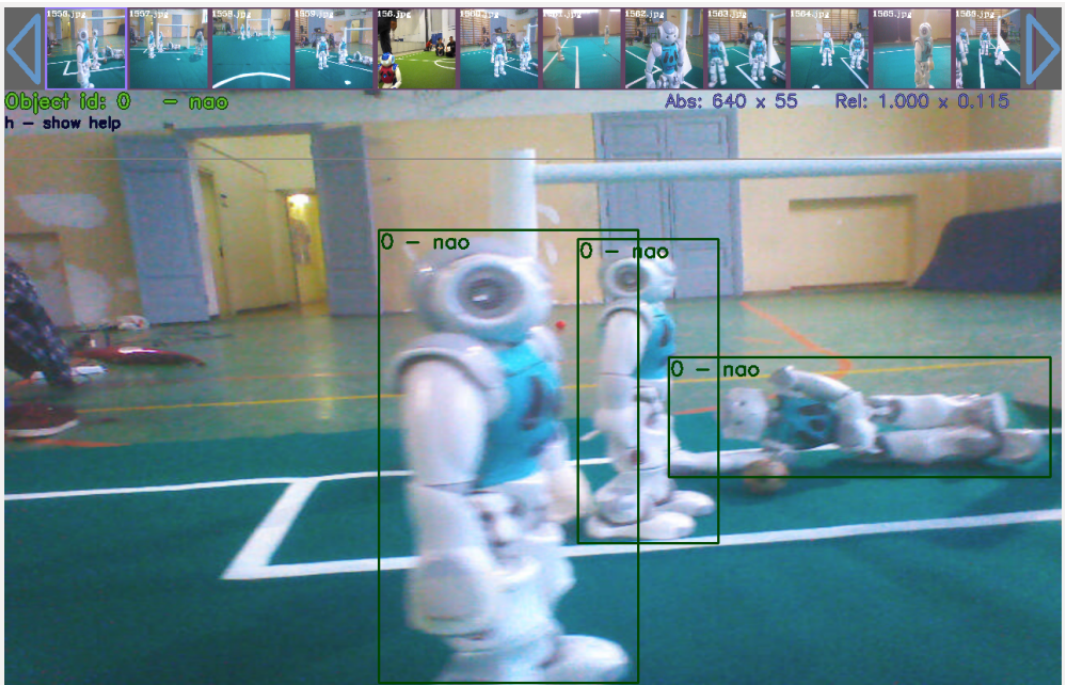


Figura 3.3: Ejemplo de etiquetado de imágenes con la herramienta *Yolomark* para el enfoque de entrenamiento *Darknet*.

3.3. Modelos

En la actualidad la mayoría de los proyectos para la detección de objetos en imágenes involucran el aprendizaje profundo, en donde, la mayoría de las veces, se toma el modelo de un sistema y se vuelve a entrenar para una tarea en particular. Dentro del estado del arte de sistemas de detección se puede encontrar una gran variedad de modelos que entregan excelentes resultados, uno de estos es el conocido como YOLO. En esta sección se presentan su funcionamiento y las arquitecturas que han sido propuestas, basándose en este sistema, para llevar a cabo la detección del objeto de interés.

3.3.1. YOLO

Del inglés *You Only Look Once*, este sistema se distingue de los demás por ser extremadamente rápido y preciso. Como su nombre lo indica, solo requiere de ver la imagen una sola vez, lo que lo hace considerablemente más rápido en comparación con otros sistemas de detección (Figura 3.4). Además, permite intercambiar entre velocidad y precisión simplemente cambiando el tamaño del modelo, sin necesidad de un nuevo entrenamiento [28].

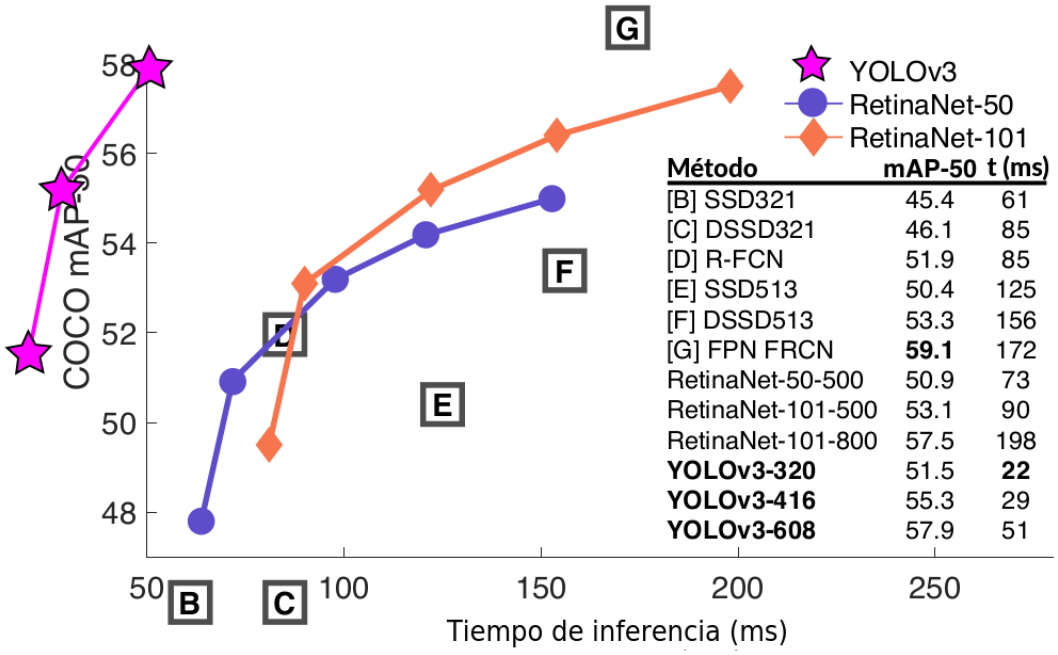


Figura 3.4: Gráfico comparativo de YOLO con respecto a otros sistemas de detección [28].

3.3.1.1. Estructura de la red

El modelo original es implementado como una red neuronal convolucional. Las capas convolucionales iniciales de la red extraen características de la imagen mientras que las capas completamente conectadas predicen las probabilidades de salida y las coordenadas. La arquitectura de la red está inspirada en el modelo GoogleLeNet para clasificación de imágenes [26]. En su versión más reciente (YOLOv3) cuenta con un extractor de características, el cual es un un enfoque híbrido entre la red utilizada en YOLOv2, Darknet-19 [27] y elementos de la red residual. Utiliza sucesivas capas convolucionales de $3x3$ y $1x1$, pero también tiene algunas conexiones de acceso directo. Es significativamente más grande que la versión anterior pues cuenta con 53 capas convolucionales y es llamado Darknet-53 (Figura 3.5) [28]. Para la tarea de detección, están apiladas otras 53 capas convolucionales, lo que proporciona la arquitectura subyacente completamente convolucional de 106 capas [18].

	Tipo	Filtros	Tamaño	Salida
	Convolucional	32	3x3	256x256
	Convolucional	64	3x3 / 2	128x128
1x	Convolucional	32	1x1	
	Convolucional	64	3x3	
	Residual			128x128
	Convolucional	128	3x3 / 2	64x64
2x	Convolucional	64	1x1	
	Convolucional	128	3x3	
	Residual			64x64
	Convolucional	256	3x3 / 2	32x32
8x	Convolucional	128	1x1	
	Convolucional	256	3x3	
	Residual			32x32
	Convolucional	512	3x3 / 2	16x16
8x	Convolucional	256	1x1	
	Convolucional	512	3x3	
	Residual			16x16
	Convolucional	1024	3x3 / 2	8x8
4x	Convolucional	512	1x1	
	Convolucional	1024	3x3	
	Residual			8x8
	AvgPool		Global	
	Conectada		1000	
	Softmax			

Figura 3.5: Extractor de características Darknet-53 [28].

El sistema de detección de YOLO también cuenta con una versión reducida del modelo. Conocido como *tiny* YOLO, está diseñado para presionar sobre límites de

la detección rápida de objetos y pensado para entornos con recursos computacionales limitados. Esta versión está conformada por una arquitectura similar a la original pero con tan solo 23 capas (Figura 3.6).

Capa	Tipo	Filtros	Tamaño / Paso	Entrada	Salida
0	Convolucional	16	3x3 / 1	416x416x3	416x416x16
1	Maxpool		2x2 / 2	416x416x16	208x208x16
2	Convolucional	32	3x3 / 1	208x208x16	208x208x32
3	Maxpool		2x2 / 2	208x208x32	104x104x32
4	Convolucional	64	3x3 / 1	104x104x32	104x104x64
5	Maxpool		2x2 / 2	104x104x64	52x52x64
6	Convolucional	128	3x3 / 1	52x52x64	52x52x128
7	Maxpool		2x2 / 2	52x52x128	26x26x128
8	Convolucional	256	3x3 / 1	26x26x128	26x26x256
9	Maxpool		2x2 / 2	26x26x256	13x13x256
10	Convolucional	512	3x3 / 1	13x13x256	13x13x512
11	Maxpool		2x2 / 1	13x13x512	13x13x512
12	Convolucional	1024	3x3 / 1	13x13x512	13x13x1024
13	Convolucional	256	1x1 / 1	13x13x1024	13x13x256
14	Convolucional	512	3x3 / 1	13x13x256	13x13x512
15	Convolucional	255	1x1 / 1	13x13x512	13x13x255
16	YOLO				
17	Ruta a capa 13				
18	Convolucional	128	1x1 / 1	13x13x256	13x13x128
19	Upsampling		2x2 / 1	13x13x128	26x26x128
20	Ruta a capas 19 y 8				
21	Convolucional	256	3x3 / 1	26x26x384	26x26x256
22	Convolucional	255	1x1 / 1	26x26x256	26x126x255
23	YOLO				

Figura 3.6: Estructura del modelo *tiny* YOLO.

3.3.1.2. Funcionamiento

Sistemas de detección anteriores reutilizan clasificadores o localizadores para realizarla detección, aplicando el modelo a la imagen en múltiples ubicaciones y escalas, donde las regiones de alta puntuación de la imagen se consideran como detecciones. YOLO utiliza un enfoque totalmente diferente, de manera resumida, aplica una sola red neuronal a la imagen completa. Esta red divide la imagen en regiones o celdas, a partir de una cuadrícula de SxS (Figura 3.7 (a)).

En cada una de las celdas predice N posibles cuadros delimitadores y calcula la incertidumbre o probabilidad de cada uno de ellos (Figura 3.7 (b)), calculando entonces $SxSxN$ cuadros delimitadores, los cuales están ponderados por las probabilidades predichas (Figura 3.7 (c)).

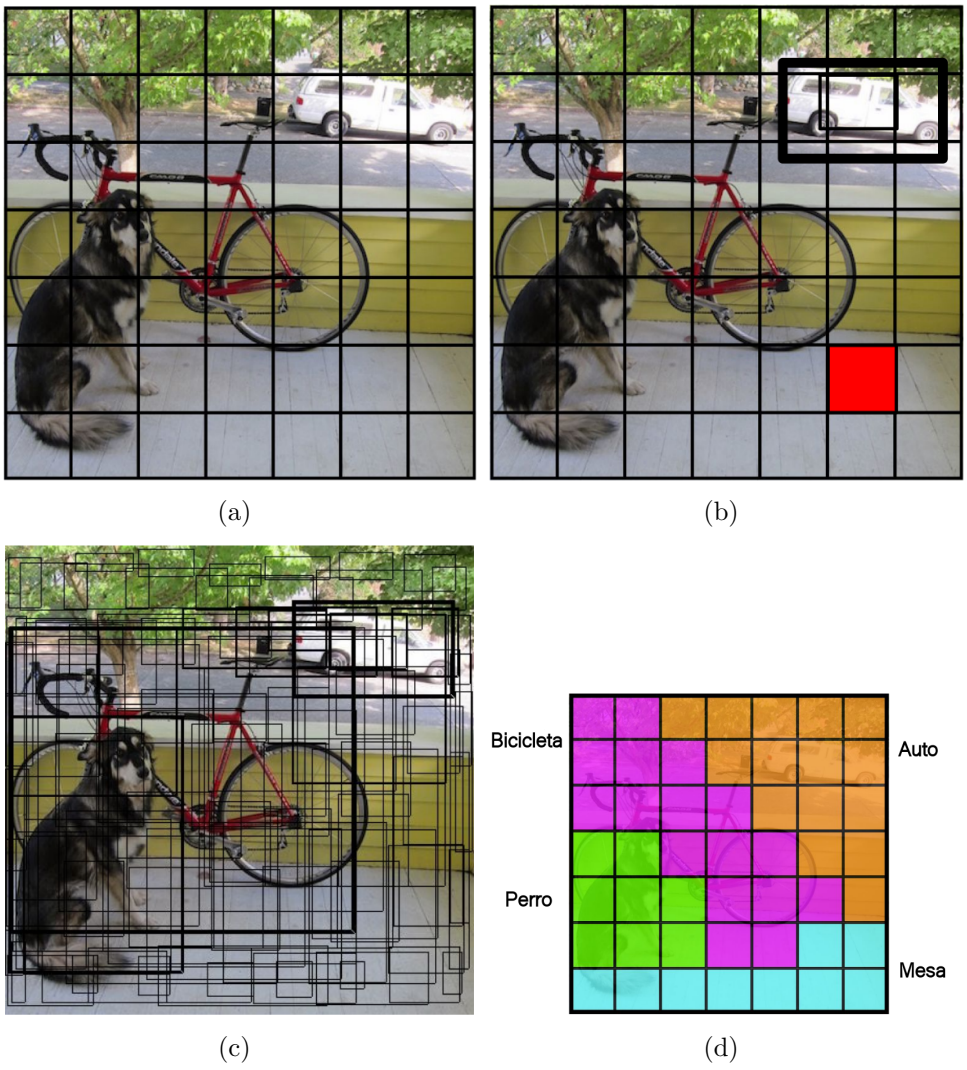
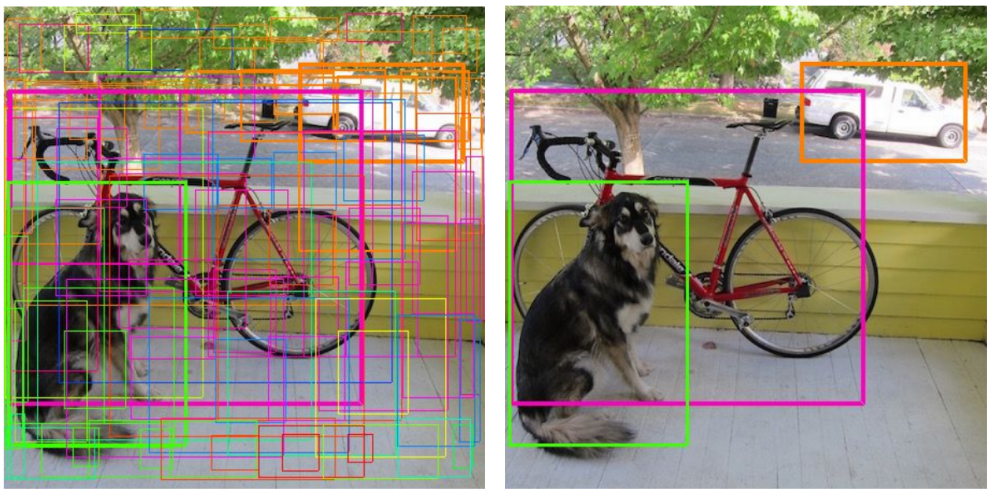


Figura 3.7: Funcionamiento de sistema YOLO. (a) División de imagen en celdas. (b) Predicción de cuadros delimitadores y cálculo de probabilidad por celda. (c) Total de cuadros delimitadores calculados. (d) Probabilidades de clase condicionadas a objetos de imagen [26].

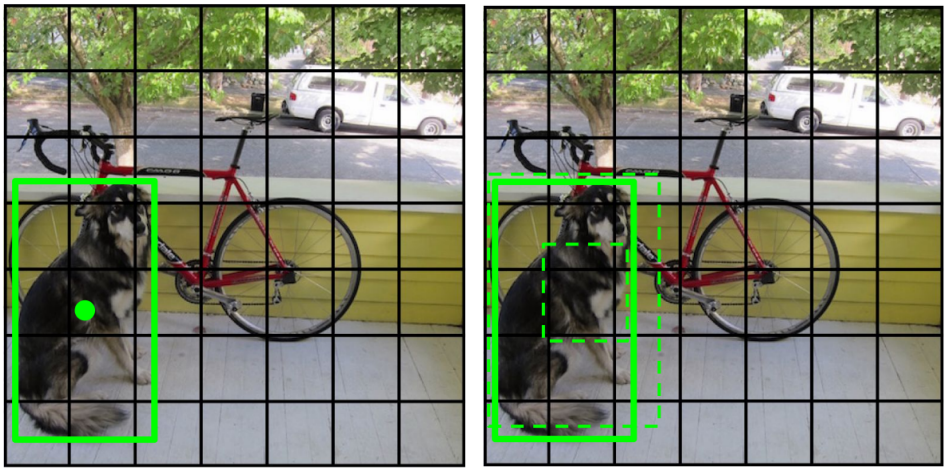
Cada celda predice también probabilidades de clase condicionadas a cada objeto (Figura 3.7 (d)). Se combinan entonces la predicción de cuadros delimitadores y la predicción de probabilidades de clase (figura 3.8 (a)). Después de obtener las predicciones, lo siguiente es eliminar los cuadros delimitadores menos convenientes, es decir, aquellos que contengan un valor de probabilidad por debajo de un umbral fijado. A los cuadros que sobreviven a este procedimiento se les aplica una supresión no máxima (*non-max suppression*) para eliminar las posibles detecciones duplicadas de los objetos y dejar únicamente la más acertada (Figura 3.8 (b)).



(a) (b)

Figura 3.8: Funcionamiento de sistema YOLO. (a) Combinación de los cuadros delimitadores y las probabilidades de clase. (b) Detección de los objetos [26].

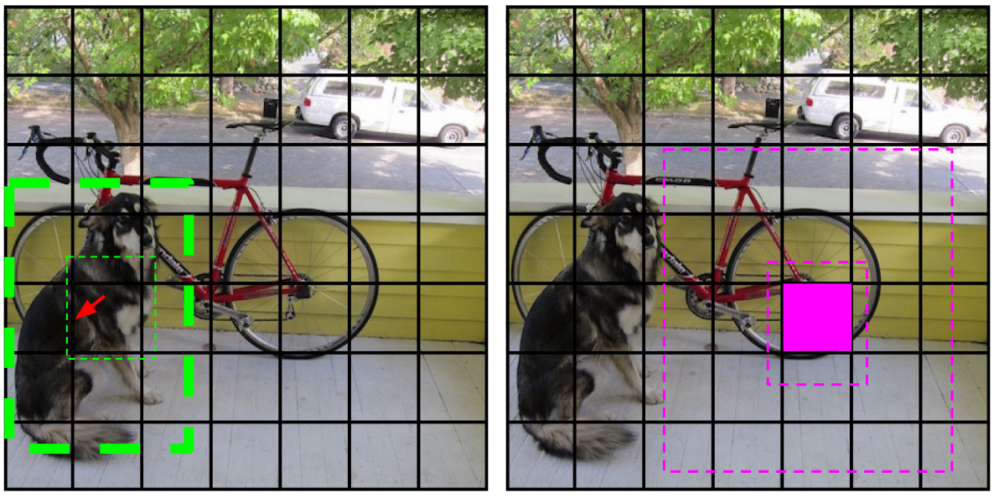
Durante el entrenamiento el sistema empareja la imagen de ejemplo, es decir la imagen que entra a la red, con la celda que contiene el centro del objeto y ajusta la predicción de clase de dicha región (Figura 3.9 (a)). Posteriormente se busca entre las predicciones de los cuadros delimitadores de esa celda, el mejor de ellos de acuerdo al del etiquetado (Figura 3.9 (b)) y lo ajusta, incrementando su valor de incertidumbre y reduciendo el de los cuadros delimitadores restantes (Figura 3.10 (a)).



(a) (b)

Figura 3.9: Entrenamiento de sistema YOLO. (a) Definición de la celda con el centro del objeto y ajuste de predicción. (b) Selección del mejor cuadro delimitador predicho [26].

Finalmente, para las regiones de la imagen que no contienen objetos, se reducen los valores de probabilidad de los cuadros delimitadores predichos por cada celda y se dejan sin ajustar sus probabilidades de clase (Figura 3.10 (b)).



(a) (b)

Figura 3.10: Entrenamiento de sistema YOLO. (a) Incremento y reducción de incertidumbre de los cuadros delimitadores. (b) Reducción de probabilidad de cuadros delimitadores de las celdas que no contienen objetos [26].

3.3.2. Arquitecturas propuestas

Como se muestra anteriormente, YOLO es un sistema muy prometedor que entrega resultados en precisión y tiempo destacables, lo que lo ha hecho posicionarse por encima de otros sistemas y que sea utilizado para diversas aplicaciones. Centrándose, específicamente, en el objetivo principal de este proyecto se podría considerar el entrenar este sistema de detección en su versión *tiny* con la base de datos construida para la detección del robot NAO. La razón de considerar utilizar este modelo que se centra en las limitaciones de ciertos sistemas computacionales es, precisamente, debido a las limitaciones en procesamiento de la plataforma del robot sobre la que se desea ejecutar el sistema de detección. Por la "simplicidad" que conllevaba un sistema de detección para una sola clase y los recursos computacionales disponibles para efectuar los entrenamientos, podría suponerse el más adecuado. Sin embargo, la restricción que presenta la plataforma del robot NAO provoca que esta configuración de la red pueda ser considerada

como no tan ligera o compacta en algunos casos. Debido a esta razón, se proponen tres modelos para procurar alcanzar el objetivo que expone este trabajo. Estas propuestas están conformadas por dos reducciones de la estructura original *tiny*, así como por una nueva estructura de red, la cual está basada también en el funcionamiento del sistema YOLO. Estos modelos son propuestos con la finalidad de facilitar la ejecución de los mismos sobre la plataforma del robot y reducir el tiempo de detección, manteniendo el más alto índice de precisión posible para el sistema de detección.

3.3.2.1. Reducción de *tiny* YOLO, modelo R1

Las primeras dos arquitecturas de los modelos propuestos son, como ha sido especificado previamente, reducciones de la arquitectura original, por lo que son muy similares a ésta. Para el modelo denominado R1, se ha buscado mantener una concordancia entre las dimensiones de los volúmenes de entrada y salida de cada capa, con respecto a los de la arquitectura original. Aquí, la modificación principal se concentra en una pequeña reducción del número de capas totales y en el número de filtros contenidos en cada una de las capas. Teóricamente, una reducción en el número de filtros representa una menor recopilación de información sobre el volumen de entrada a la red, por lo que se espera, sin duda, una disminución en los índices de precisión. Por lo anterior es que el desempeño esperado para los modelos al hacer estas reducciones se supone no tan eficiente como el del modelo original. Sin embargo, la disminución del número de capas y filtros equivale también a disminuir el número de operaciones totales que requiere el sistema, por lo que también es de esperarse que se reduzcan tanto los tiempos de entrenamiento, como los de detección. La estructura de esta propuesta de modelo se muestra en la Figura 3.11, aquí se puede observar que no existe una gran diferencia entre ésta y la estructura original. La verdadera disparidad entre ambas ha de ser esperada en los resultados que arrojen los entrenamientos de cada una de ellas. Es importante notar que este modelo termina trabajando con volúmenes de un tamaño dos veces más grande que con los que finaliza el modelo *tiny*, por lo que cabe la posibilidad que incluso con las reducción que presenta R1, esta red puede no ser procesada más rápido que la arquitectura que presenta YOLO como se espera.

Capa	Tipo	Filtros	Tamaño / Paso	Entrada	Salida
0	Convolucional	32	3x3 / 1	416x416x3	416x416x32
1	Maxpool		2x2 / 2	416x416x32	208x208x32
2	Convolucional	64	3x3 / 1	208x208x32	208x208x64
3	Maxpool		2x2 / 2	208x208x64	104x104x64
4	Convolucional	128	3x3 / 1	104x104x64	104x104x128
5	Maxpool		2x2 / 2	104x104x128	52x52x128
6	Convolucional	256	3x3 / 1	52x52x128	52x52x256
7	Maxpool		2x2 / 2	52x52x256	26x26x256
8	Convolucional	512	3x3 / 1	26x26x256	26x26x512
9	Convolucional	128	1x1 / 1	26x26x512	26x26x128
10	Convolucional	256	3x3 / 1	26x26x128	26x263x256
11	Convolucional	18	1x1 / 1	26x26x256	26x26x18
12	YOLO				
13	Ruta a capa 9				
14	Convolucional	64	1x1 / 1	26x26x128	26x26x64
15	Upsampling		2x2 / 1	26x26x64	52x52x64
16	Ruta a capas 15 y 6				
17	Convolucional	128	3x3 / 1	52x52x320	52x52x128
18	Convolucional	18	1x1 / 1	52x52x128	52x52x18
19	YOLO				

Figura 3.11: Estructura de modelo R1 propuesta para reducción de *tiny* YOLO.

3.3.2.2. Reducción de *tiny* YOLO, modelo R2

Debido a que existe la posibilidad de que la propuesta anterior se comporte de una manera no tan eficaz como se espera, se propone una segunda reducción más significativa que su predecesora que ha sido denominada R2. Esta arquitectura, ilustrada por la Figura 3.12, presenta una estructura similar a tanto a R1 como a *tiny*, sin embargo, la reducción que se presenta aquí ha sido mucho mayor que en el caso anterior. Además, parte de esta propuesta es la de trabajar con una resolución de imagen que corresponde a la mitad del valor con el que trabaja YOLO de manera predeterminada. Este cambio se hace principalmente, para reducir el tamaño de los volúmenes con los que R2 tendría que trabajar, tamaño que finaliza siendo muy cercano al manejado por la versión original. Este modelo entonces trabaja con volúmenes reducidos y hace un gran sacrificio de capas y de filtros por capa, lo que la postulan a ejecutar un procesamiento más veloz que los modelos hasta ahora descritos. A pesar de esto, y de acuerdo con la teoría, estas modificaciones producen una menor extracción de características de los datos entrantes

al sistema, por lo que, incluso si se alcanza el objetivo de ser procesado más rápido, la efectividad de su desempeño puede verse afectada de manera considerable.

Capa	Tipo	Filtros	Tamaño / Paso	Entrada	Salida
0	Convolucional	32	3x3 / 1	224x224x3	224x224x32
1	Maxpool		2x2 / 2	224x224x32	112x112x32
2	Convolucional	64	3x3 / 1	112x112x32	112x112x64
3	Maxpool		2x2 / 2	112x112x64	56x56x64
4	Convolucional	128	3x3 / 1	56x56x64	56x56x128
5	Maxpool		2x2 / 2	56x56x128	28x28x128
6	Convolucional	256	3x3 / 1	28x28x128	28x28x256
7	Convolucional	18	1x1 / 1	28x28x256	28x28x18
8	YOLO				
9	Ruta a capa 6				
10	Convolucional	64	1x1 / 1	28x28x128	28x28x64
11	Ruta a capas 10 y 6				
12	Convolucional	128	3x3 / 1	28x28x320	28x28x128
13	Convolucional	18	1x1 / 1	28x28x128	28x28x18
14	YOLO				

Figura 3.12: Estructura de modelo R2 propuesta para reducción de *tiny* YOLO.

3.3.2.3. Modelo TYR-NAO

La estructura *tiny* de YOLO ha sido diseñada y pensada para ser implementada en la detección de múltiples clases de objetos. Esto la hace una arquitectura de red un tanto compleja cuando se trata de un escenario, o aplicación, en donde se desea detectar una sola clase de objeto. Es por eso que para una tarea más elemental, como la que se pretende alcanzar con este proyecto, e incluso con las modificaciones que se proponen previamente, los resultados en relación a tiempo de procesamiento y precisión que ofrece este modelo y las reducciones propuestas, no son los más deseables para la aplicación requerida.

Teniendo en cuenta lo anterior, e inspirado en los resultados que se presentan recientemente en [32], se presenta la tercera de las arquitecturas propuestas. Tal y como se observa en la Figura 3.13, el modelo TYR-NAO está estructurado de una manera diferente a los modelos presentados con anterioridad pero, sin dejar de estar basado en la idea principal de *tiny* YOLO. En primer lugar, este modelo elimina las capas de reducción *maxpool* que se presentan en YOLO. El reducir los volúmenes de esta manera

representa una pérdida de información entre una capa y otra, pues la operación que hace esta capa solo conserva el valor mayor del arreglo en cuestión. En su lugar, la reducción se realiza utilizando un valor de 2 para el parámetro *stride* de algunas de las capas convolucionales. Así, la estructura está compuesta de, prácticamente, nada más que capas convolucionales lo que igualmente permite que, a pesar de que cada capa contiene un número no muy alto de filtros, la cantidad total de filtros que se encargan de la extracción de características es mayor que la presente en R2, esperando con esto un mejor desempeño al ejecutarse. Esta propuesta también trabaja sobre una relación en la resolución que es diferente a las utilizadas en los entrenamientos anteriores. Por configuración predeterminada, *tiny* YOLO utiliza una relación de resolución de imagen cuadrada. Con este cambio se busca mantener la relación 4 : 3 de las imágenes obtenidas con la cámara del robot pero, al mismo tiempo, se baja la resolución predeterminada de $640x480$ a un tamaño de $512x384$ para alcanzar las dimensiones de volúmenes que presenta este modelo. Con estas dimensiones, que son significativamente más pequeñas que la de los modelos anteriores, es de esperarse una reducción considerablemente aceptable en los tiempos de procesamiento tanto para detección.

Capa	Tipo	Filtros	Tamaño / Paso	Entrada	Salida
0	Convolucional	16	3x3 / 2	512x384x3	256x192x16
1	Convolucional	32	3x3 / 2	256x192x16	128x96x32
2	Convolucional	64	3x3 / 2	128x96x32	64x48x64
3	Convolucional	64	3x3 / 1	64x48x64	64x48x64
4	Convolucional	128	3x3 / 2	64x48x64	32x24x128
5	Convolucional	128	3x3 / 1	32x24x128	32x24x128
6	Convolucional	256	3x3 / 2	32x24x128	16x12x256
7	Convolucional	256	3x3 / 1	16x12x256	16x12x256
8	Convolucional	18	1x1 / 1	16x12x256	16x12x18
9	YOLO				
10	Ruta a capa 7				
11	Convolucional	64	1x1 / 2	16x12x256	8x6x64
12	Ruta a capa 11				
13	Convolucional	128	3x3 / 1	8x6x64	8x6x128
14	Convolucional	18	1x1 / 1	8x6x128	8x6x18
15	YOLO				

Figura 3.13: Estructura de modelo TYR-NAO propuesta para detección.

3.4. Entrenamiento

El entrenamiento de un modelo depende, en gran parte, del enfoque utilizado para entrenar y de la arquitectura del mismo. Para el entrenamiento de YOLO con *Darknet*, enfoque en el que fue desarrollado, es necesario contar con ciertos archivos fundamentales para indicar las configuraciones y los parámetros de entrenamiento. Dichos archivos son listados a continuación.

3.4.1. Modelo preentrenado

Cuando se entrena un detector de objetos propio, es una buena idea aprovechar modelos existentes que ya han sido entrenados con una base de datos muy grande, aun cuando esta base de datos no contenga el objeto que se desea detectar. Este proceso es llamado *transferencia de aprendizaje* o *transfer learning* [22]. En lugar de aprender desde cero, se utiliza un modelo preentrenado que contiene los pesos convolucionales ya entrenados y se reentrena con alguna otra base de datos. Usando estos pesos como los pesos iniciales, la red puede aprender de manera más rápida. Estos pesos se encuentran normalmente en archivos *.weights* y que, en su mayoría, son puestos a disposición junto con los modelos originales. El enfoque de entrenamiento *Darknet* permite hacer la sustracción de los pesos de los diferentes modelos del sistema de detección YOLO [3], donde es posible elegir hasta que punto específico de la estructura se desea mantener los pesos preentrenados del modelo.

3.4.2. Archivo de datos

YOLO necesita también de una archivo *.data* que contiene información acerca de las especificaciones del detector de objetos deseado y algunos directorios relevantes.

El parámetro *classes* se refiere al número de clases totales que se entrenaran. Es necesario indicar los directorios absolutos de los archivos *train.txt* y *test.txt*. Estos archivos contienen un listado de las imágenes que serán utilizadas para el entrenamiento y para la validación, respectivamente. El campo *names* hace referencia al directorio absoluto en la máquina del archivo *.names* que contiene los nombres de todas las clases

```
classes = 1
train = /path/to/file/train.txt
valid = /path/to/file/test.txt
names = /path/to/file/classes.names
backup = /path/to/file/weights/
```

Figura 3.14: Estructura del archivo de datos.

a aprender. Finalmente, en el parámetro *backup* se indica el directorio existente donde se desea que se almacenen los archivos de los pesos intermedios que se van guardando conforme el entrenamiento progresa.

3.4.3. Archivo de configuración del modelo

Junto con el archivo *.data* y el archivo *.names*, YOLO necesita también un archivo de configuración con extensión *.cfg*. En este archivo están almacenados todos los parámetros de entrenamiento y también se describe en él, la estructura del modelo de la red. Para entrenar un detector de objetos propio, a partir de los modelos aquí presentados, es posible hacer una copia de alguno de los archivos de configuración y modificarlo de acuerdo a las necesidades de cada detector.

3.4.3.1. Parámetros de configuración

Batch

Este parámetro indica el tamaño del lote que se usará durante el entrenamiento. Aun cuando el conjunto de datos propio puede contener algunos cientos o miles de imágenes, los casos más comunes de entrenamiento utilizan millones de imágenes. El proceso de entrenamiento implica la *actualización iterativa* de los pesos de la red neuronal basados en la cantidad de errores que está cometiendo en el conjunto de datos de entrenamiento. Sin embargo, no es práctico usar todas las imágenes del conjunto de entrenamiento al mismo tiempo para la actualización de los pesos. Por lo que un subconjunto de imágenes es usado en una iteración, este subconjunto es llamado *batch size*. Cuando este parámetro está definido en 64, significa que 64 imágenes son usadas en una iteración para actualizar los pesos de la red.

```
[net]
# Testing
# batch=1
# subdivisions=1
# Training
batch=64
subdivisions=16
```

Figura 3.15: Parámetros de configuración *batch* y *subdivisions*

Subdivisions

A pesar de que se quiera utilizar un subconjunto de tamaño 64 para entrenar la red, es posible que no se tenga un GPU con suficiente memoria para usar un tamaño como ese. Afortunadamente, *Darknet* permite especificar una variable llamada *subdivisions* que permite procesar una fracción del subconjunto a la vez en el GPU. Se puede hacer pruebas con valores de múltiplos de 2 (2, 4, 8, 16) para este parámetro hasta que el entrenamiento proceda de manera exitosa. El GPU procesará el número de imágenes correspondiente al resultado de dividir el valor del parámetro *batch* entre el de *subdivisions* a la vez. Sin embargo, toda la iteración estará completa solo hasta después de que todas las 64 imágenes del subconjunto sean procesadas. Durante las pruebas del modelo, ambos valores deben ser fijados a 1 (Figura 3.15).

Width*, *Height* y *Channels

Estos parámetros de configuración especifican el tamaño de la imagen de entrada y su número de canales.

```
width=416
height=416
channels=3
```

Figura 3.16: Parámetros *width*, *height* y *channels*.

La imágenes de entrada para el entrenamiento son primero redimensionadas al tamaño *widthxheight* antes de entrenar. El parámetro *channels=3* indica que las imágenes de entrada que serán procesadas son imágenes de 3 canales, en este caso imágenes RGB.

Momemtum y *Decay*

Existen algunos parámetros que que controlan qué tan rápido se realiza la actualización de los pesos durante el entrenamiento.

```
momentum=0.9
decay=0.0005
```

Figura 3.17: Paŕametros *momemtum* y *decay*.

Debido a que los pesos de la red son actualizados en base a los pequeños subconjuntos de imágenes y no a la base de datos completa, las actualizaciones de los pesos tienden a oscilar un poco. El parámetro *momentum* es usado para penalizar cambios grandes de los pesos entre las iteraciones.

Una típica red neuronal tiene miles de pesos y por esa razón éstos pueden fácilmente sobreajustar cualquier información de entrenamiento. El *sobreajuste* u *overfitting* se refiere a que la red hará un buen trabajo con los datos de entrenamiento, pero no con con los datos de prueba. En otras palabras, es como si la red memorizara las respuestas a todas las imágenes del conjunto de entrenamiento sin aprender el concepto fundamental para aplicarlo con las imágenes de prueba. Una de las maneras de mitigar este problema es penalizando valores muy grandes de los pesos. El parámetro *decay* controla la penalización de estos valores.

Learning rate, *Steps*, *Scales* y *Burn in*

El parámetro *learning rate* controla qué tan agresivamente se debe aprender basado en el subconjunto de datos de la iteración actual. Típicamente, este valor es un número entre 0,01 y 0,0001. Al principio del proceso de entrenamiento, el sistema empieza con cero información, por lo que el valor de este parámetro, llamado *factor de aprendizaje*, necesita ser alto. Pero conforme la red neuronal ve más información, los pesos necesitan actualizarse de manera menos agresiva. Dicho de otra manera, el factor de aprendizaje debe de disminuir con el tiempo mientas avanza el entrenamiento.

En el archivo de configuración, esta disminución es lograda, en primer lugar, especificando que la política de disminución del parámetro es a pasos o *steps*. De acuerdo a la

```
learning_rate=0.001
policy=steps
steps=3800
scales=.1
burn_in=400
```

Figura 3.18: Parámetros *Learning rate*, *Steps*, *Scales* y *Burn in*.

Figura 3.18, el factor de aprendizaje empieza en 0,001 y se mantiene constante durante 3800 iteraciones. Después de esto, multiplicará al parámetro *scales* para obtener un nuevo factor de aprendizaje.

Mientras que la declaración de que el factor de aprendizaje necesita ser alto al inicio es verdadera, ha sido encontrado de manera empírica [22] que la velocidad del entrenamiento tiende a incrementar si se tiene un bajo valor del factor de aprendizaje por un corto periodo de tiempo justo al inicio del entrenamiento. Esto es controlado por el parámetro *burn in*. Algunas veces este periodo también es llamado *periodo de calentamiento* o de *warm up*.

Parámetros de aumento de datos

Se sabe que la recolección de datos puede tomar mucho tiempo. El proceso de "cocinar" nueva información a partir de la ya recolectada es llamado *aumento de datos* o *data augmentation*. Por ejemplo, una imagen de un objeto rotada ciertos grados sigue siendo una imagen del mismo objeto. El parámetro *angle* permite rotar, de manera aleatoria, la imagen dada. Esta rotación puede realizarse en ambos sentidos y está dada por el valor del parámetro.

```
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
```

Figura 3.19: Parámetros de aumento de datos.

De manera similar, si se transforman los colores de toda la imagen usando los parámetros *saturation*, *exposure*, y *hue*, sigue siendo una imagen del mismo objeto. Estos parámetros cambian los valores de la *saturación*, *exposición* y la *tonalidad* de los

colores en la imagen, respectivamente.

Número de iteraciones

```
max_batches=5200
```

Figura 3.20: Parámetro *max batches*.

Finalmente, se requiere de un parámetro que especifique cuántas iteraciones debe correr el proceso de entrenamiento. Para detectores de múltiples clases de objetos el valor del parámetro *max batches* debe ser alto. De manera general, para un detector de objetos con n clases se recomienda correr el entrenamiento por al menos $2000 * n$ iteraciones [22]. Por lo tanto, para un detector de una sola clase de objetos, 5200 iteraciones (Figura 3.20) deben ser más que suficientes. Sin embargo, una vez que se inicia el entrenamiento es importante monitorizar el comportamiento del mismo. Cada vez que se completa una iteración de entrenamiento, se actualiza el valor del error promedio que se va validando con los pesos actualizados del sistema y el conjunto de prueba. El objetivo del aprendizaje es que este valor de error disminuya conforme el proceso avanza y que se acerque lo más posible a cero.

3.5. Pruebas del sistema entrenado

Con el sistema entrenado, lo que se desea es que el sistema realice las siguientes tres tareas: *Detección*, *Localización* e *Identificación*.

- **Detección:** De manera general, este proceso toma una imagen de entrada y obtiene como salida un número de clase o etiqueta de un conjunto de categorías. Este número debe indicar si dentro de la imagen se encuentra el objeto de interés o no.
- **Localización:** Además de obtener una etiqueta, y en caso de que haya sido detectado, se debe dibujar un cuadro delimitador que describa dónde se encuentra el objeto en la imagen.

- **Identificación:** Aquí, la localización debe hacerse en todos los objetos de la imagen que pertenezcan a una clase en cuestión. Obteniendo, de esta manera, múltiples cuadros delimitadores y varias etiquetas de clase.

El enfoque de *Darknet* [25] no solo permite realizar el entrenamiento de nuevos modelos sino que también permite probar los modelos entrenados dentro de su mismo repositorio [28]. *Darknet* imprime en la consola los objetos que detecta, con su respectiva etiqueta, su valor de confianza o probabilidad de pertenencia de clase y cuánto tiempo le toma localizarlos a todos (Figura 3.21). Una vez terminado, la imagen con los objetos detectados es almacenada en el repositorio bajo el nombre de *prediction.jpg*.

Figura 3.21: Ejemplo de detección de robot NAO con YOLO, implementado sobre enfoque *Darknet*.

Debido a que uno de los objetivos de los detectores de objetos en tiempo real es que sean rápidos, es prudente buscar alternativas para las implementaciones del sistema que aumenten la velocidad de detección de éste. Una de las encontradas está basada en la biblioteca de visión artificial *OpenCV* y se puede encontrar en [21]. Así como la anterior, esta implementación también imprime la etiqueta del objeto detectado, su valor de probabilidad de pertenencia de clase y el tiempo que tarda en hacer la detección, sin embargo, esta impresión se realiza sobre la imagen misma (Figura 3.22), la cual también es guardada en el directorio terminado el proceso.

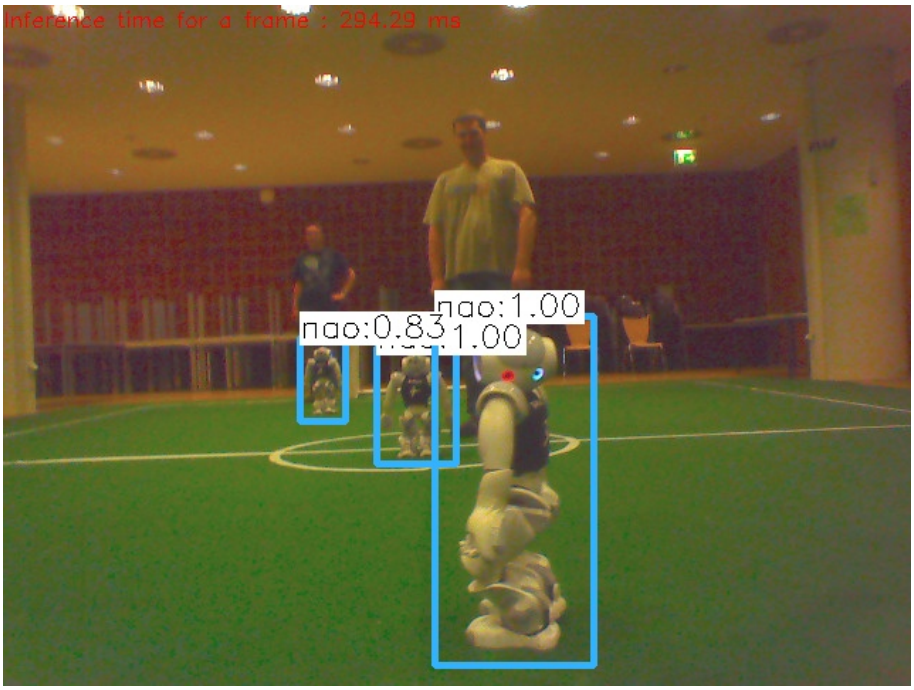


Figura 3.22: Ejemplo de detección de robot NAO con YOLO, implementado con OpenCV.

Utilizando la misma imagen para ambas implementaciones y haciendo una pequeña comparación entre ellas, se puede observar que en la Figura 3.22 el tiempo total requerido para la detección de los todos los objetos en la imagen es de aproximadamente 300 milisegundos, mientras que en la implementación de la Figura 3.21 toma poco menos de 3 segundos. Esto indica que la implementación con OpenCV es, en primer instancia, 10 veces más rápida que con el enfoque *Darknet*, lo que la hace más eficiente.

3.6. Clasificación de objetivos

Una vez que la detección de los objetivos en la imagen ha sido realizada satisfactoriamente, la siguiente tarea que se necesita resolver para el sistema de visión, es la de poder diferenciar, de todos los robots detectados, aquellos que pertenecen al equipo rival de aquellos que no lo hacen.

La manera de hacer lo descrito anteriormente es aprovechando la información que arroja el sistema de detección. Éste dibuja los cuadros delimitadores en la imagen, por lo que se puede utilizar los datos de los cuadros delimitadores para aislar cada uno de ellos y tratarlo como una imagen independiente. Una vez hecho esto, se lleva acabo una segmentación por color de las imágenes obtenidas de la original, es decir, de los cuadros delimitadores que pertenecen a los objetivos detectados. La razón de optar por una segmentación por color es debido a que, por simple sentido común, uno de los colores de los uniformes de los equipos es conocido. Sabiendo esto, solo se necesita ajustar los valores de umbral para poder detectar este color a través de una segmentación y asociar los resultados de ésta para identificar a todos los jugadores pertenecientes al mismo equipo y diferenciarlos de los que pertenecen al equipo contrario.

3.6.1. Segmentación por color

Para realizar una segmentación por color de una imagen, siguiendo el proceso del procesamiento digital de imágenes [11], lo más recomendable es hacer una transformación de espacio o modelo de color, posteriormente se definen los umbrales de los valores de color que se quiere segmentar, lo que entregará una imagen binarizada a la que se le hacen algunas mejoras por medio de morfología matemática.

3.6.1.1. Espacios de color

Un espacio de color tiene el propósito de facilitar la especificación de los colores en una manera estándar. En esencia, un modelo de color es una especificación de un sistema de coordenadas y un subespacio dentro de ese sistema donde cada color es representado por un punto.

El sensor de la cámara del robot NAO entrega una imagen RGB. En el espacio de color RGB (por sus siglas en inglés, *Red Green Blue*), cada color aparece en su componentes espectrales primarios de rojo, verde y azul [11]. Este modelo está basado en un sistema coordenado Cartesiano. El subespacio de color de interés es el mostrado en la Figura 3.23 donde los valores RGB se encuentran en tres esquinas; los colores cían, magenta y amarillo están en otras 3 esquinas; el color negro se encuentra en el origen; y el color blanco se encuentra en el punto más lejano al origen.

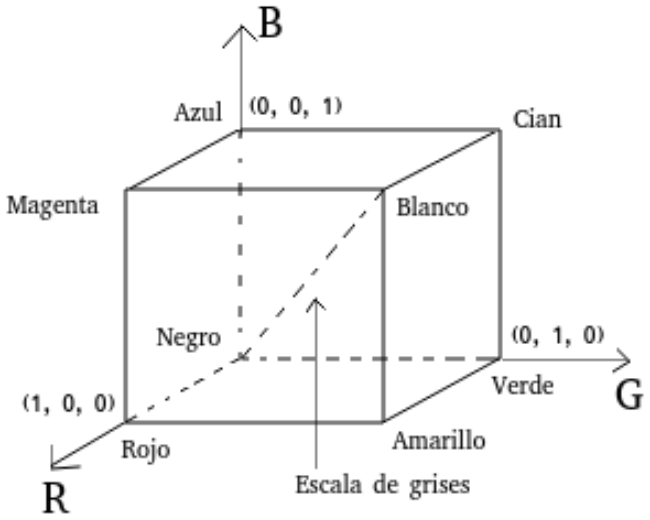


Figura 3.23: Subespacio de interés del modelo de color RGB [11].

Aquí, la escala de grises se extiende sobre la línea que une los puntos de los colores negro y blanco. Los diferentes colores en este modelo son puntos dentro o fuera del cubo, y están definidos por vectores que se extienden desde el origen.

A pesar de que este modelo es ideal para aplicaciones de hardware y que adicionalmente, se empareja perfecto al hecho de que el ojo humano es fuertemente perceptivo a los primarios de rojo, verde y azul, no es el mejor para describir los colores en términos prácticos para la interpretación humana. El modelo RGB es ideal para la generación de imágenes a color, como la de una imagen capturada por una cámara, pero su uso para la descripción de color es muy limitado.

Por otro lado, cuando un humano ve un objeto de color, lo describe por su tonalidad, saturación y brillo. Donde la *tonalidad* o *matiz* es el atributo que describe la pureza del

color, mientras la *saturación* entrega una medida del grado en el que un color puro es diluido por la luz blanca. El brillo, sin embargo, no puede ser medido, pues se trata de un descriptor subjetivo pero, el término está embebido con el acrónimo de *intensidad* que es uno de los factores clave para describir la sensación de color [11]. El modelo de color HSI (por las siglas en inglés *Hue Saturation Intensity*), desacopla el componente intensidad de la información de color en una imagen. Como resultado, se obtiene una herramienta ideal para el procesamiento de imágenes basado en descriptores de color. El subespacio de color HSI está representado por un eje vertical de intensidad y el lugar geométrico de los puntos de color que se encuentran en los planos perpendiculares a este eje (Figura 3.24).

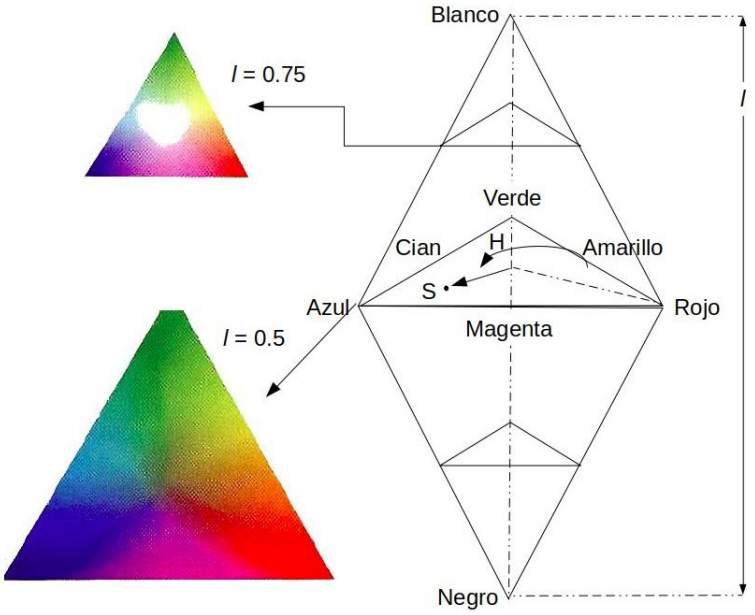


Figura 3.24: Subespacio de color del modelo HSI [11].

Conforme los planos se mueven hacia arriba y hacia abajo sobre el eje de intensidad, los límites definidos por la intersección de cada plano con las caras del cubo (Figura 3.23) tienen una forma, ya sea triangular o hexagonal.

Esto puede apreciarse mejor si se mira hacia abajo al cubo RGB sobre el eje de la escala de grises. En este plano se puede observar que tanto los colores primarios, como los secundarios, están separados por 120° entre sí (Figura 3.25).

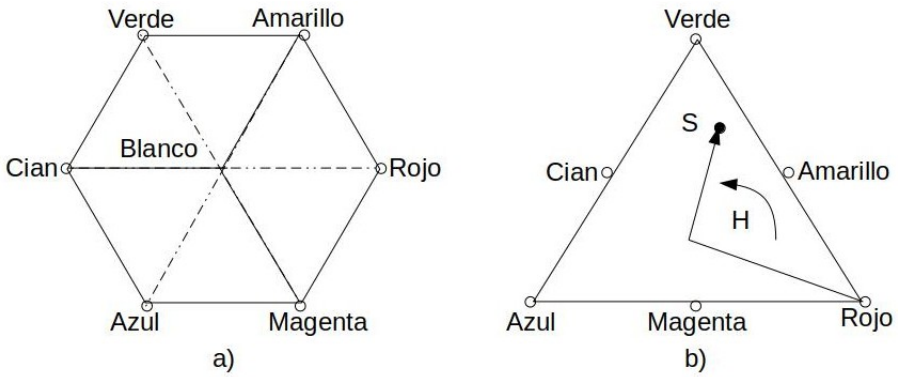


Figura 3.25: Perspectiva del cubo de color RGB sobre el eje de la escala de grises. a) Geometría hexagonal, se observa la separación de los colores primarios y secundarios, 120° entre unos y otros, y 60° entre cada uno. b) Geometía triangular, se ilustra como se mide el parámetro de la tonalidad y la representación del de la saturación.

La tonalidad de un punto es determinado por un ángulo desde un punto de referencia. Usualmente, un ángulo de 0° desde el eje del color rojo designa un valor de tonalidad de cero, y aumenta en sentido contrario al de las manecillas del reloj a partir de ese punto. La saturación, que es la distancia del punto al eje vertical, es la longitud del vector desde el origen al punto donde el origen está definido por la intersección del plano de color y el eje vertical de intensidad.

3.6.1.2. Transformación de color de RGB a HSI

Dada una imagen a color con el formato RGB, el componente H de cada píxel RGB se obtiene a partir de la ecuación:

$$H = \begin{cases} \theta & \text{if B} \leq \text{G} \\ 360 - \theta & \text{if B} > \text{G} \end{cases} \tag{3.5}$$

con

$$\theta = \cos^{-1} \frac{\frac{1}{2}[(R-G)+(R-B)]}{\left[(R-G)^2+(R-B)(G-B)\right]^{\frac{1}{2}}} \tag{3.6}$$

El componente de saturación está dado por:

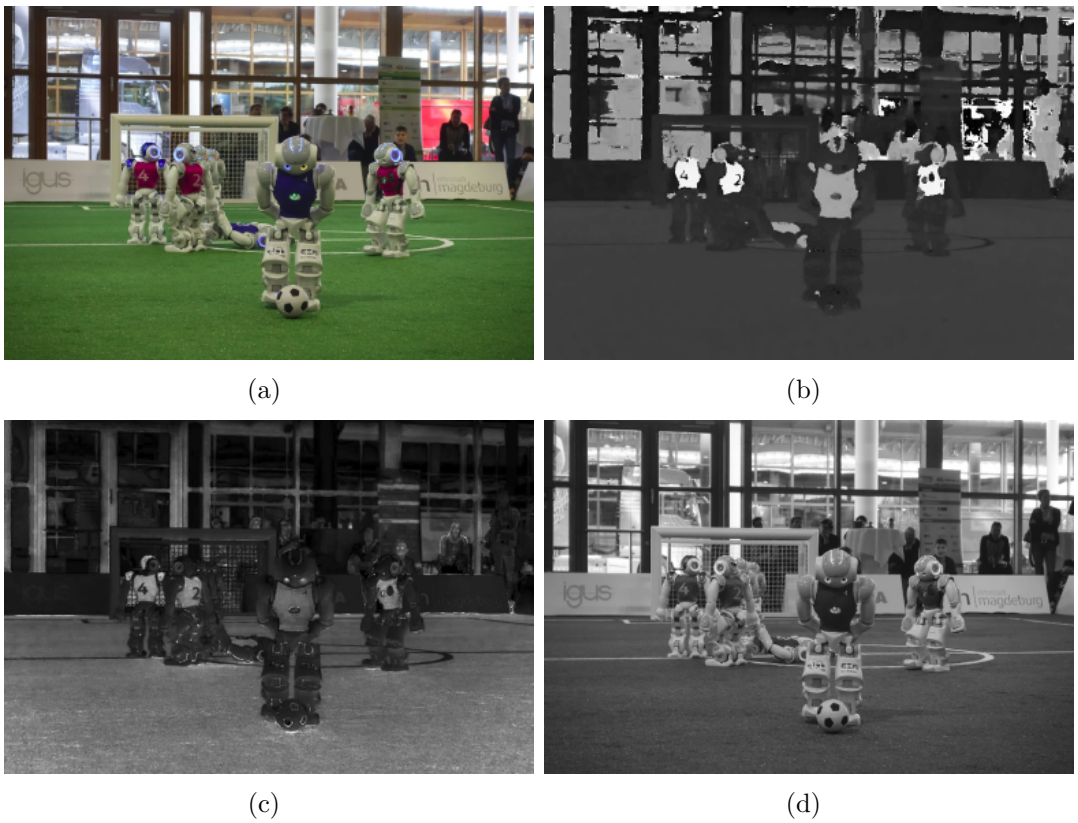
$$S = 1 - \frac{3}{(R+G+B)}[min\,(R,G,B)] \tag{3.7}$$

Finalmente, el componente de intensidad se calcula a partir de:

$$I = \frac{1}{3}(R + G + B) \tag{3.8}$$

Se asume que los valores RGB de la imagen han sido normalizados al rango $[0, 1]$ y que el ángulo θ es medido con respecto al eje del color rojo en el espacio HSI. La tonalidad puede ser normalizada en el rango $[0, 1]$ dividiendo entre 360° todos los valores resultantes de la ecuación 3.5. Los otros 2 componentes ya se encuentran en este rango si los valores de RGB dados lo estaban también [11].

Con la transformación de color se pueden obtener entonces los valores de los parámetros HSI de cada píxel de la imagen (Figura 3.26), lo que permite definir umbrales para los parámetros que en conjunto generan el espacio de color que representa un color en específico de los uniformes de los robots.



(a) (b)

(c) (d)

Figura 3.26: (a) Imagen original en formato RGB, descompuesta en los elementos del formato HSI: (b) Componente H de tonalidad, (c) Componente S de saturación, (d) Componente I de intensidad.

Al realizar un barrido de la imagen con estos valores en cuenta, es posible obtener una imagen en escala de grises, donde se le asigna un valor específico a las regiones encontradas que representan los colores que se desean aislar de la imagen completa (Figura 3.27). Algo importante a considerar, es que el resultado del procedimiento anterior puede resultar en que los objetos aislados presenten cierta deformación. Esto es debido a que durante la segmentación se puede perder un poco de información de los objetos que se quieren aislar. Para corregir esto existen ciertas técnicas de procesamiento de imagen como la ya mencionada morfología matemática.

(a) (b)

Figura 3.27: (a) Imagen original. (b) Imagen segmentada de acuerdo a los valores HSI de los colores presentes en los uniformes de los robots.

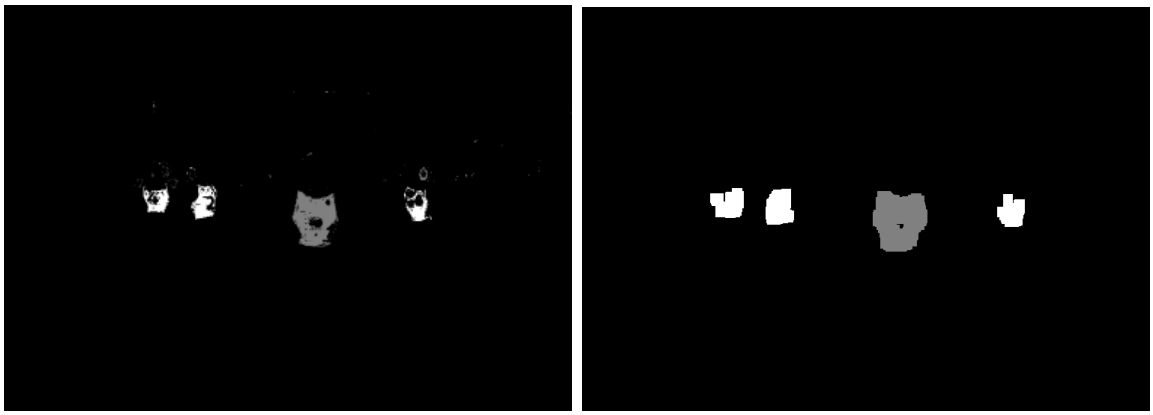
3.6.1.3. Morfología matemática

La morfología matemática resulta ser una herramienta para la extracción de componentes de una imagen que pueden ser útiles para la representación y descripción de la figura de una región. Las operaciones básicas de las que están basados la mayoría de los algoritmos de morfología matemática son conocidas como *dilatación* y *erosión* [11]. De manera general, la operación dilatación, como su nombre lo indica, se encarga de expandir el objeto o los objetos que se encuentran en la imagen. Mientras que la erosión, en el caso contrario, se encarga de la reducción de los mismos.

Otras dos operaciones importantes son la *apertura* y el *cierre*. La apertura generalmente hace que el contorno de un objeto se rompa, rompe franjas estrechas y elimina

protuberancias delgadas. El cierre, por su parte, también tiende a suavizar las secciones de los contornos, pero a diferencia de la apertura, generalmente fusiona las roturas estrechas y los largos y finos abismos elimina los pequeños orificios y llena los huecos en el contorno [11].

Combinaciones de estas operaciones son realizadas para alcanzar una representación adecuada, o lo más cercana posible al resultados deseado. En la Figura 3.28 se puede apreciar el resultado de aplicar estos métodos para mejorar el resultado de la segmentación por color.



(a) (b)

Figura 3.28: (a) Imagen resultado después de segmentación. (b) Imagen resultado después de aplicar morfología.

Capítulo 4

Resultados Experimentales

Este apartado presenta los resultados obtenidos, en primer lugar, de la segmentación por color de los uniformes para los robots detectados en la imagen, la cual se realiza para poder clasificar dichos robots entre compañeros de equipo y rivales. En segundo lugar, y de manera más extensa, se presentan los todos los resultados de los entrenamientos realizados sobre los diferentes modelos utilizados para desempeñar la detección del robot NAO.

4.1. Resultados de clasificación

La clasificación de los objetivos detectados en las imágenes consiste en una combinación de técnicas de procesamiento digital de imágenes para identificar, por medio de segmentación por color, a todos los robots detectados que pertenecen al mismo equipo. De esta manera es posible saber cuáles de los objetivos encontrados pueden ser considerados compañeros de equipo o rivales. De acuerdo a lo descrito en la sección anterior, se utilizan los datos del sistema de detección para aislar cada uno de los robots detectados y a partir del color de uniforme conocido se lleva a cabo la segmentación para realizar la clasificación. Considerando entonces, por ejemplo, que se define la información del color magenta que presentan los uniformes de la Figura 4.1 (a). Después de la detección realizada por el sistema (Figura 4.1 (b)), se procede a realizar la segmentación únicamente sobre los objetos detectados, lo que entrega la imagen que se presenta en

la Figura 4.1 (c). Finalmente, se utiliza la información de la segmentación para separar los robots detectados entre compañeros de equipo (verde) o rivales (rojo), que da como resultado lo que se muestra en la Figura 4.1 (d).

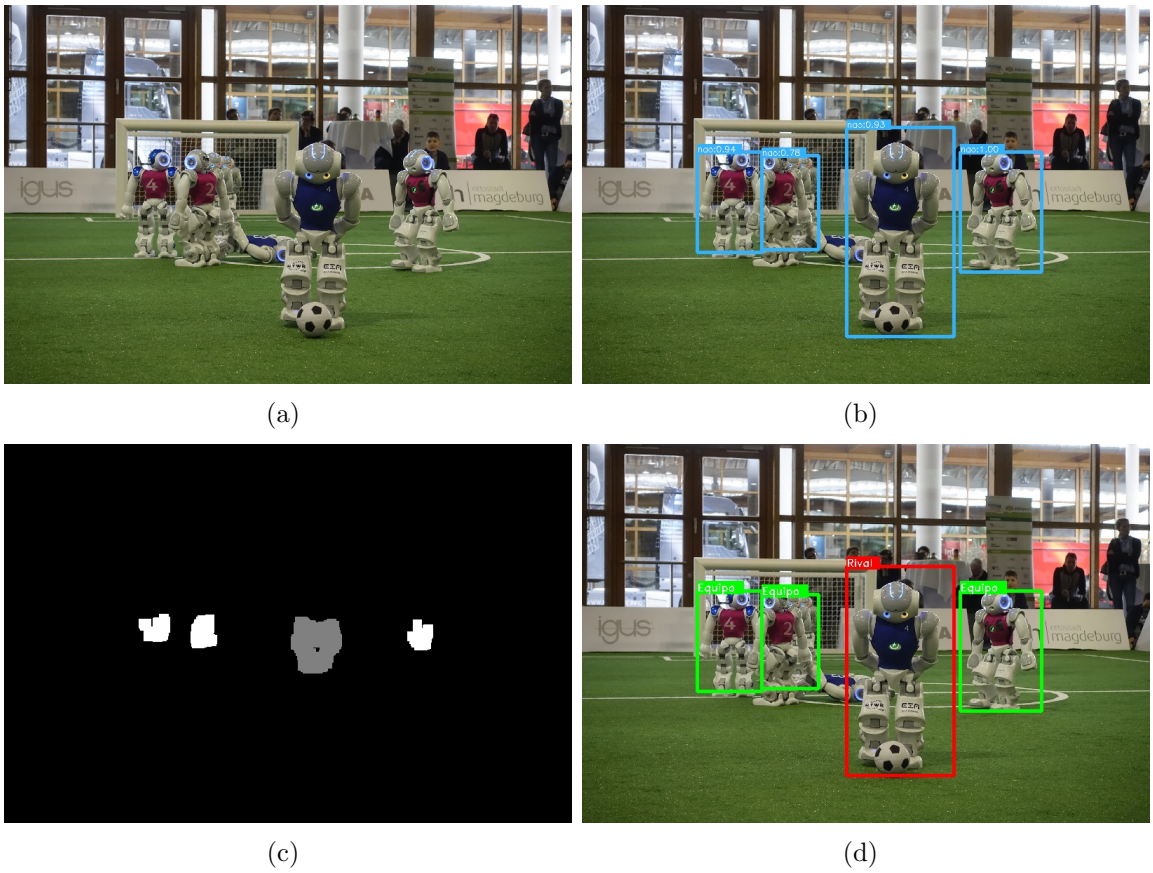


Figura 4.1: Resultados de clasificación de los objetos detectados.

4.2. Entrenamientos

Como se menciona en el capítulo anterior, con el enfoque *Darknet* es posible realizar tanto el entrenamiento, así como la inferencia (detección) y la evaluación del modelo entrenado. Sin embargo, existen diversas implementaciones de este enfoque que también proveen soporte para realizar las mismas tareas. Qué tipo de implementación es mejor dependerá en grandes rasgos de los propósitos y objetivos deseados, y de la accesibilidad que se tiene en cuanto a software y hardware se refiere.

4.2.1. Tipos de entrenamientos

Diferentes implementaciones se han utilizado para el entrenamiento de los modelos utilizados para la detección de los robots NAO de este proyecto. Una de ellas es la ya incluida en el enfoque *Darknet* [25], y la otra es una implementación en *PyTorch* del sistema de detección YOLOv3 [16]. *PyTorch* es una biblioteca de aprendizaje automático, gratuita y de código abierto, basada en la biblioteca Torch y es usada para aplicaciones de aprendizaje automático y procesamiento del lenguaje natural. La razón por la que la implementación de *PyTorch* también ha sido considerada se debe a que está basada en el enfoque (*Darknet*), por lo que requiere exactamente de los mismos archivos de configuración y características de etiquetado de la base de datos para llevar a cabo el entrenamiento. Lo anterior es debido a que la implementación sobre la plataforma se realiza por medio del mismo enfoque (*Darknet*) por lo que todos los entrenamientos realizados deben ser compatibles con éste.

Normalmente 2000 iteraciones por cada clase de objeto es suficiente, pero no menos de 4000 iteraciones en total [3]. Algo importante a considerar, como ya fue mencionado con anterioridad, es que el valor de error vaya disminuyendo hasta acercarse a cero. El valor de error representa la perdida promedio (*average loss*) y lo deseado es que este valor sea tan pequeño como sea posible. Durante el entrenamiento se puede observar la variación de los indicadores de error, sin embargo, es posible que el valor de error deje de disminuir en algún momento después de un número considerable de iteraciones, y es ahí cuando debería ser detenido. Los valores promedio para el valor de error pueden ser de 0,05 para modelos de red pequeños y conjunto de datos sencillos y de 3,0 para modelos mayores y con conjuntos de datos mas complicados [3].

Al realizar un monitoreo del entrenamiento, es posible tomar la decisión de detener el mismo en un punto en el que el valor del error alcanza un umbral deseado o que simplemente ya no disminuye, lo que también permite ahorrar tiempo que puede ser usado para otros procesos. En la figura 4.2 se puede observar un ejemplo de la convergencia del valor de error para un entrenamiento. En este caso se observa que el valor del error se mantiene relativamente constante a partir, aproximadamente, de la iteración número 500 y hasta después de la iteración 2000. Esto podría dar una idea de que ya no es ne-

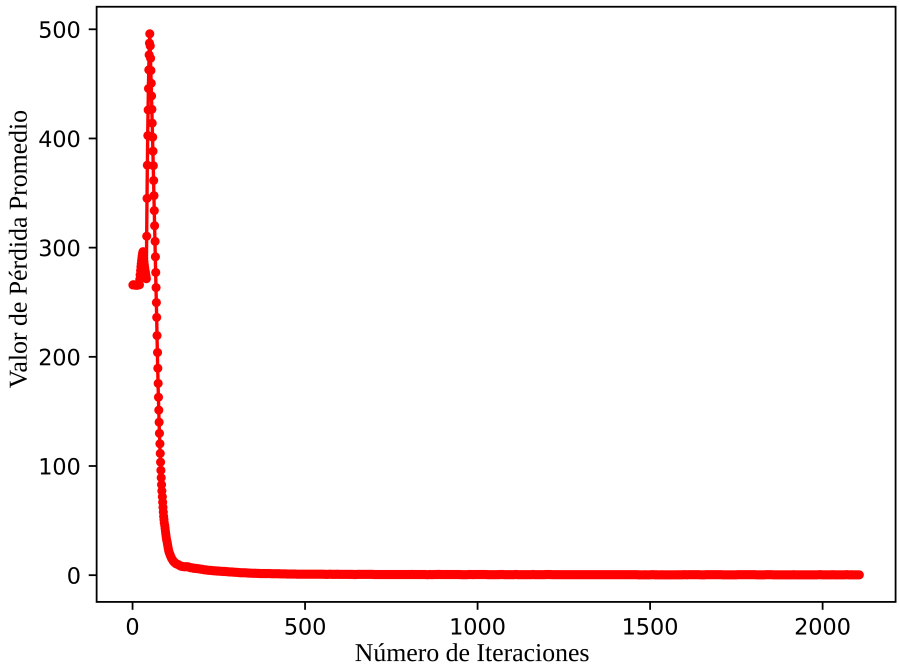


Figura 4.2: Ejemplo de un gráfico de los valores de error obtenidos durante un entrenamiento.

cesario continuar el entrenamiento por más iteraciones pues el valor de error parece no disminuir más y, como se observa en el gráfico, dicho valor se encuentra cercano al cero. Sin embargo, la única manera de estar seguro de la eficacia de detener el entrenamiento en ese punto es con la evaluación del archivo de pesos que resulta del entrenamiento.

4.2.2. Archivo de pesos

De acuerdo a los parámetros de entrenamiento deseados es posible indicar en qué momento, durante el proceso de entrenamiento, se desea almacenar los valores de los pesos obtenidos después de cada iteración de la red. Esto se lleva a cabo, entre otras razones, debido a que es posible que se presenten cambios significativos en el resultado de la detección al utilizar un archivo de pesos de una iteración previa a la última realizada, por ejemplo, si el entrenamiento es detenido después de 2000 iteraciones, es posible que el mejor archivo de pesos haya sido salvado en iteraciones anteriores (1800, 1600, etc). Esto puede pasar debido al sobreajuste, lo que sucede cuando es posible detectar objetos en las imágenes del conjunto de entrenamiento, pero no es posible hacer la detección de los objetos en imágenes diferentes a las de este conjunto.

Una diferencia entre las dos implementaciones utilizadas para realizar el entrenamiento del modelo es que los archivos de pesos obtenidos del entrenamiento con *Darknet*, tienen un formato de tipo *.weights*, mientras que los archivos entregados con la implementación en *PyTorch* son de tipo *.pt*. Esta diferencia resulta considerable al momento de implementar los archivos del entrenamiento para llevar a cabo la detección, pues las implementaciones son muy susceptibles al formato de los archivos de pesos. Por otro lado, algo que tienen en común ambas implementaciones es la accesibilidad a hardware del tipo unidades de procesamiento gráfico o GPUs (*Graphics Proccesing Unit*). Esto se refiere, de manera específica, a que los entrenamientos realizados tanto con la implementación de *Darknet*, como los entrenamientos soportados por la implementación de *PyTorch* han sido realizados con el apoyo de GPUs. Esta accesibilidad puede depender de problemas de compatibilidad con algunas versiones de software entre las implementaciones y los equipos donde se llevan a cabo los entrenamientos. El uso de GPUs desemboca en la existencia de una diferencia muy considerable en cuanto a tiempos de entrenamiento, ya que en casos en los que no se utilizaron la duración del entrenamiento se extendía hasta 3 semanas, mientras que con GPU se trataba de cuestión de horas. A pesar de esto, los resultados obtenidos por ambas implementaciones, ya sea con el uso de GPUs o sin ellas, son muy similares y cumplen con los requerimientos necesarios. Además, tomando en cuenta que los tiempos de interés para este caso particular corresponden a los tiempos de detección en tiempo real sobre la plataforma, los tiempos de entrenamiento pueden no ser muy relevantes, mientras los resultados sean prometedores.

4.3. Evaluación

Una vez que es ha sido detenido o que ha finalizado el entrenamiento se obtiene el archivo de pesos de la última iteración realizada, sin embargo, este no siempre es el mejor archivo de pesos para realizar la inferencia. Por esta razón es que se guardan diferentes archivos de pesos durante el entrenamiento y, se tiene que encontrar entre éstos el archivo más adecuado para realizar la detección. Existe más de una manera

de decidir cuál es el archivo más adecuado para un sistema de detección. Una de ellas podría ser simplemente probando dicho sistema con cada uno de los archivos de pesos y, de acuerdo a los resultados recibidos de la detección, elegir el archivo que presentó mejor desempeño. Esta podría ser la manera más adecuada de decidir, pues al final, lo que que se quiere es un resultado visual óptimo y nos interesa saber qué archivo puede proporcionarlo. Sin embargo, es posible que la cantidad de archivos de pesos almacenados sea muy grande para ser probados, de uno por uno, en la base de datos. Una alternativa, y al mismo tiempo, complemento del procedimiento anterior es la de evaluar el modelo con los diferentes archivos de pesos y encontrar cuál de ellos entrega el mejor resultado para la detección. Así, evaluar el modelo puede ser la tarea más importante ya que esto indica qué tan buenas son las predicciones realizadas. Como se sabe, mientras más cerca a cero ha sido reducido el valor de error durante el entrenamiento, mejor será el resultado en la detección. Por esto, en primer lugar, hay que considerar llevar a cabo la evaluación con los archivos de pesos que fueron guardados durante las iteraciones que entregaron un valor más cercano a cero. Una vez que se tienen seleccionados los archivos, se procede a realizar la evaluación del modelo. Esta evaluación, en la mayoría de los casos es soportada por la misma implementación donde se realiza el entrenamiento, inclusive es posible que se vaya realizando a la par del mismo. Las métricas para medir éstos valores pueden variar de una implementación a otra y normalmente están inspiradas en algunas competencias. Para seleccionar el mejor de los archivos de pesos se debe realizar, entonces, una comparación de los términos conocidos como *Intersección sobre la Unión* o **IoU** (*Intersect over Union*) y, la *media del Promedio de Precisión* o **mAP** (*mean Average Precision*).

El IoU se refiere a la intersección sobre la unión promedio de los objetos y las detecciones para un cierto valor de umbral [3]. Mide la superposición entre dos delimitaciones y se usa para medir que tanto se superpone la predicción con la delimitación del objeto real (Figura 4.3). Mientras que el término mAP, es el valor de la media de las precisiones promedio para cada clase, donde el promedio de precisión es el valor promedio de 11 puntos en la curva de precisión-recuperación, o curva PR (*Precision-Recall*), para cada uno de los posibles umbrales (probabilidad de detección) de la misma clase. El mAP es

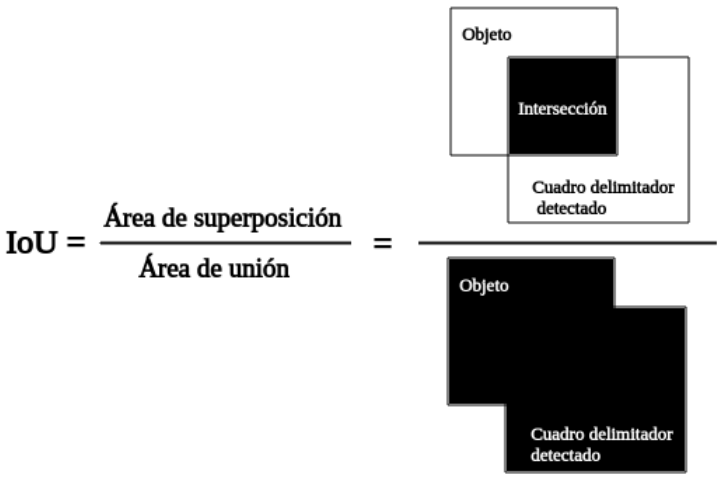


Figura 4.3: Representación gráfica de la intersección sobre la unión [14].

la métrica de precisión definida en la competencia *PascalVOC* [7].

4.3.1. mAP para detección de objetos

El Promedio de Precisión o AP (*Average Precision*) es una métrica popular para medir la precisión de algunos detectores de objetos. Este calcula el valor de la precisión promedio para el valor de recuperación en el rango de 0 a 1. Para poder calcular los términos antes mencionados, entran en juego cuatro parámetros importantes:

- **Verdaderos Positivos o TP (*True Positives*):** Se refiere a los valores positivos predichos correctamente, lo que significa que el modelo predice correctamente la clase positiva.
- **Verdaderos Negativos o TN (*True Negatives*):** Este resultado indica, de manera análoga, los valores negativos predichos de manera correcta. Es decir, cuando el modelo predice correctamente la clase negativa.
- **Falsos Positivos o FP (*False Positives*):** Valores positivos predichos incorrectamente, lo que indica que la predicción de modelo para la clase positiva es incorrecta.
- **Falsos Negativos o FN (*False Negatives*):** Valores obtenidos cuando el modelo predice de manera incorrecta la clase negativa.

4.3.1.1. Exactitud, Presión, Recuperación y Puntaje F1

La exactitud es la medida de desempeño más intuitiva, de manera simplificada, es la relación entre la predicciones realizadas correctamente y el total de predicciones realizadas. Por lo anterior se puede pensar que mientras más alto el nivel de exactitud se tenga, mejor es el modelo, sin embargo, la exactitud es una excelente métrica solo para conjuntos de datos simétricos, donde los valores de falsos positivos y falsos negativos son casi iguales [17]. Debido a esto, hay que considerar los siguientes términos para evaluar el desempeño del modelo. Su definición matemática está dada por:

$$Exactitud = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.1}$$

La precisión es la relación entre las predicciones positivas realizadas correctamente y el total de predicciones positivas realizadas. En otras palabras mide el porcentaje de las predicciones positivas que son correctas. Por su parte, la recuperación, o exhaustividad, es la relación entre las predicciones positivas realizadas correctamente y todas las predicciones realizadas para la clase actual, es decir, mide qué porcentaje de los positivos verdaderos se predijo de manera correcta [14]. Ambos términos son ilustrados en la Figura 4.4.

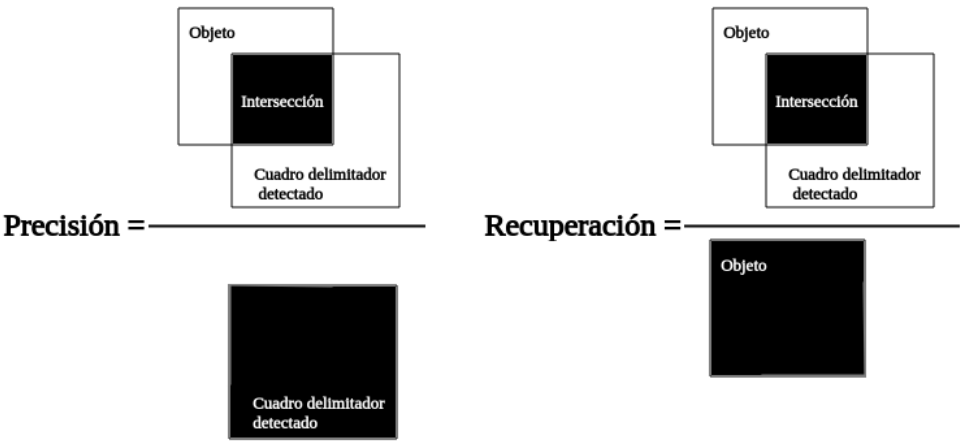


Figura 4.4: Representación gráfica de precisión y recuperación [14].

El puntaje o puntuación F1 es el promedio ponderado de precisión y recuperación. Por lo tanto, este valor toma tanto falsos positivos, como falsos negativos en el cálculo. Podría no ser tan intuitiva la comprensión de este término, sin embargo, normalmente

es más útil que la exactitud [17]. De acuerdo con [7] las definiciones matemáticas para los últimos tres términos son las siguientes:

$$Precisión = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recuperación = \frac{TP}{TP + FN} \tag{4.3}$$

$$F1 = 2 \cdot \frac{precisión \cdot recuperación}{precisión + recuperación} \tag{4.4}$$

4.3.2. Cálculo de AP

En [14] se encuentra disponible un ejemplo para el cálculo del promedio de precisión, el cual se resume a continuación. Se tiene la siguiente gráfica PR (Figura 4.5) donde se observa que los valores de recuperación incrementan a medida que los de precisión descienden. La precisión presenta un patrón en zigzag que va hacia abajo con falsos positivos y vuelve a subir con verdaderos positivos. La definición general para el AP es encontrar el área bajo la curva PR:

$$AP = \int_0^1 p(r)dr \tag{4.5}$$

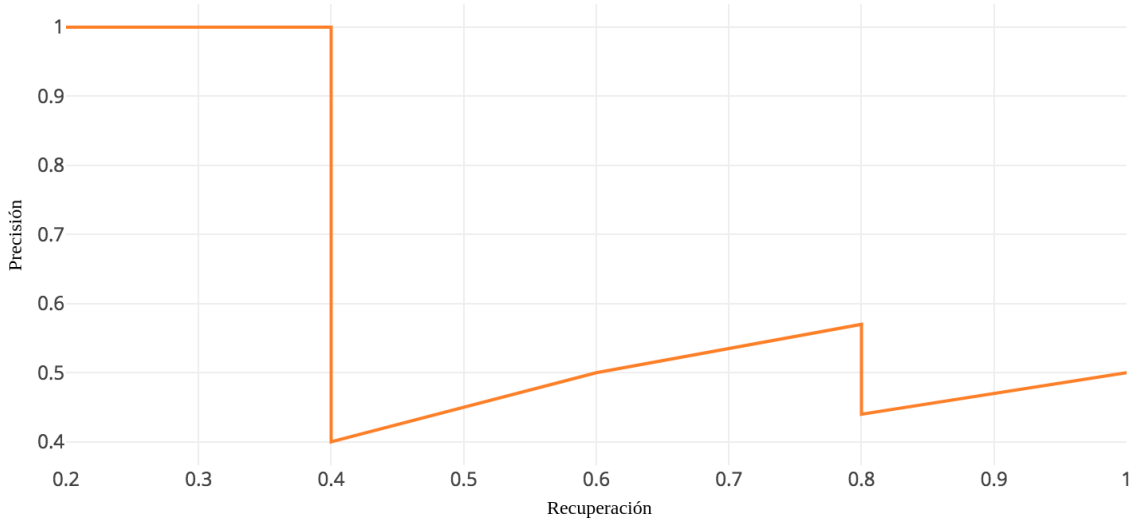


Figura 4.5: Gráfico de la curva PR [14].

Tanto la precisión como la recuperación siempre se encuentran entre 0 y 1. Por lo tanto, el AP también recae en este rango. Antes de calcular el AP para la detección, normalmente primero se suaviza el patrón de zigzag. De manera gráfica, en cada nivel de recuperación se remplaza cada valor de precisión con el valor máximo de precisión (hacia la derecha) de ese nivel de recuperación, tal como se ilustra en la Figura 4.6.

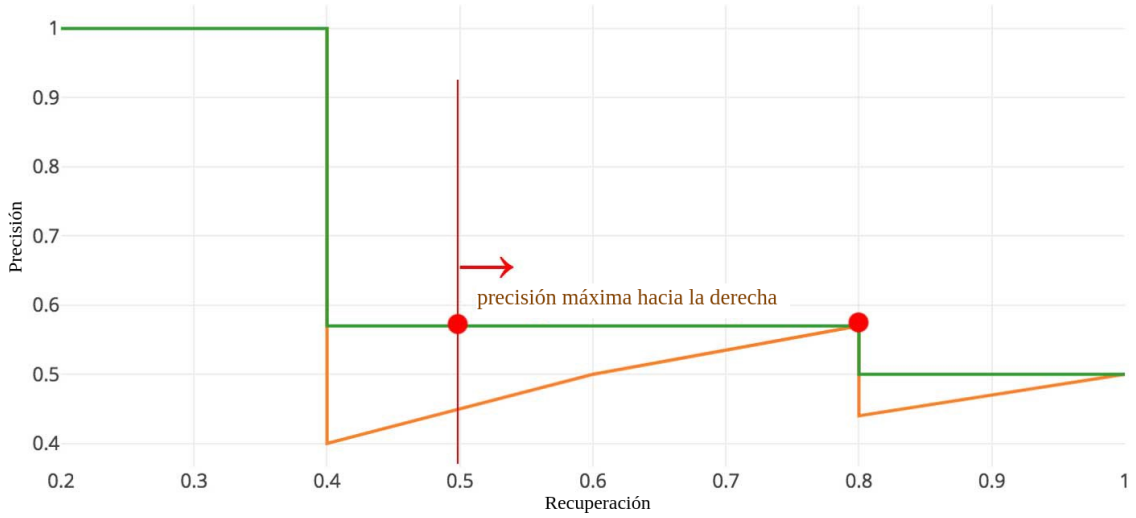


Figura 4.6: Eliminación de patrón zigzag por sustitución de valor máximo de precisión [14].

Así, la curva pasa a decrecer de manera monótona en lugar de hacerlo en un patrón zigzag. El valor de AP calculado sera menos susceptible a pequeñas variaciones de los valores ordenados. Matemáticamente, se remplaza el valor de precisión por el de recuperación $\tilde{r}$ con la precisión máxima para cualquier recuperación $\geq \tilde{r}$:

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \tag{4.6}$$

Interpolación de AP

PascalVOC es un conjunto de datos muy popular para la detección de objetos. Para el reto, que lleva el mismo nombre que el conjunto de datos, una predicción es positiva si IoU $\geq$ 0,5. También, si múltiples detecciones para el mismo objeto se realizaron, solo cuenta la primera como positiva mientras que el resto como negativas [7]. En el conjunto *PascalVOC2008*, se calculó un promedio para los 11 puntos de interpolación para AP. Primero se divide el valor de la recuperación desde 0 hasta 1,0 en 11 puntos

(Figura 4.7).

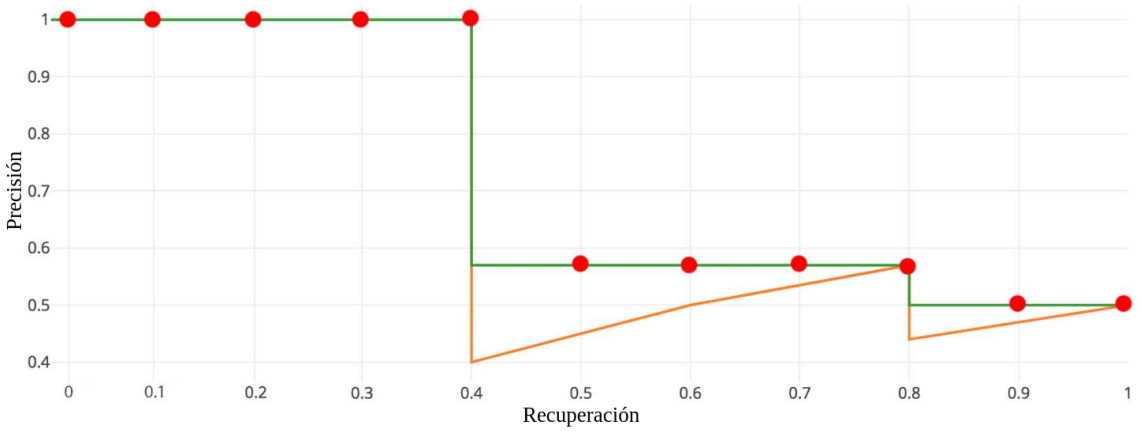


Figura 4.7: División en 11 puntos del valor de recuperación [14].

En seguida, se calcula el promedio del valor de precisión máxima para esos 11 puntos de recuperación. La definición matemática es la siguiente:

$$AP = \frac{1}{11} \times (AP_r(0) + AP_r(0{,}1) + ... + AP_r(1{,}0)) \tag{4.7}$$

$$AP = \frac{1}{11} \sum_{r\epsilon(0,0{,}1,...,1{,}0)} AP_r \tag{4.8}$$

$$AP = \frac{1}{11} \sum_{r\epsilon(0,0{,}1,...,1{,}0)} p_{interp}(r) \tag{4.9}$$

Cuando AP_r se vuelve extremadamente pequeño, se puede asumir que los términos sobrantes son cero, es decir, no necesariamente se hacen perdiciones hasta que la recuperación alcanza un 100 %. Si el nivel de precisión máximo posible cae a un nivel despreciable, se puede detener. De acuerdo con [7], la intención de usar 11 puntos interpolados para calcular AP es para reducir el impacto de las oscilaciones en la curva PR, causados por pequeñas variaciones de los valores ordenados. Sin embargo, este método de interpolación es una aproximación que sufre de dos inconvenientes. Es menos preciso y pierde la capacidad de medir la diferencia para métodos con bajo AP.

AP - Área bajo la curva

Para competiciones posteriores de *PascalVOC* (VOC2010-2012) se hace un muestreo de la curva en todos los valores de recuperación únicos $(r_1, r_2, \dots)$, siempre que el valor de precisión máximo descienda. Con este cambio, lo que se mide es exactamente el área bajo la curva PR después de remover el patrón zigzag (Figura 4.8).

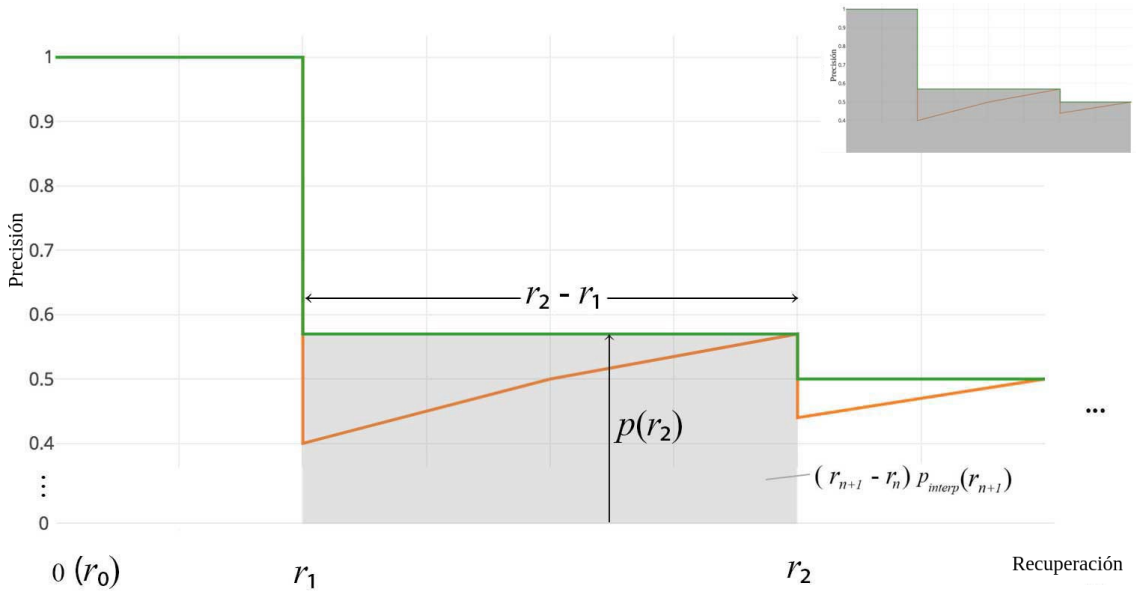


Figura 4.8: Área bajo la curva PR [14].

Ninguna aproximación o interpolación es necesaria. En lugar de muestrear 11 puntos, se muestrean $p(r_i)$ siempre que disminuya y se calcula AP como la suma de los bloques rectangulares. Esto es:

$$AP = \sum (r_{n+1} - rn) p_{interp}(r_{n+1}) \tag{4.10}$$

$$p_{interp}(r_{n+1}) = \underset{\tilde{r} \geq r_{n+1}}{\text{máx}} \; p(\tilde{r}) \tag{4.11}$$

Esta definición de la ecuación 4.10 es llamada el Área Bajo la Curva o AUC (*Area Under Curve*). Tal como muestra la Figura 4.9, los puntos interpolados no cubren donde disminuye la precisión, por lo tanto, ambos métodos divergirán.

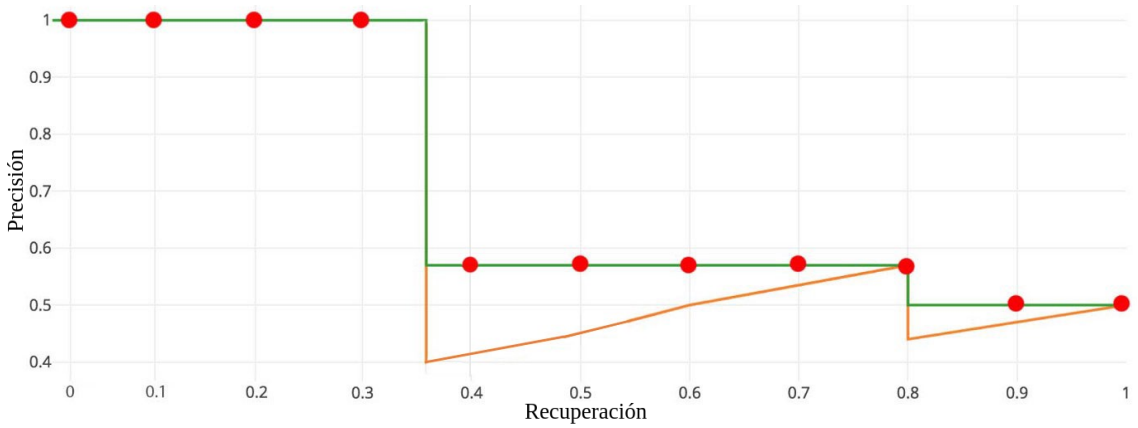


Figura 4.9: Interpolación de puntos de precisión [14].

Finalmente, el término mAP es prácticamente el promedio de AP. En algunos contextos se calcula el valor de AP para cada clase y posteriormente se promedia, en algunos otros, ambos términos pueden significar lo mismo.

4.4. Resultados

Los resultados de los entrenamientos están basados en los resultados de las evaluaciones para los índices de precisión de los archivos de pesos obtenidos de cada entrenamiento realizado. Antes de presentar los resultados es prudente recordar que dichos entrenamientos se han realizado sobre diferentes modelos de arquitectura de red. Por esta razón, este apartado se encuentra dividido entre los resultados obtenidos para el modelo *tiny* de YOLO y los resultados de los modelos que han sido propuestos como alternativa al anterior. Así mismo, se hace una comparación de los tiempos de ejecución registrados para cada uno de los modelos entrenados.

4.4.1. *Tiny* YOLO

Diferentes entrenamientos han sido realizados para la arquitectura del modelo original de la versión pequeña de YOLO. En primera instancia se realiza un ligero cambio para adaptar la estructura al caso de entrenamiento para la detección de una sola clase. Esta es considerada entonces como la arquitectura original, sobre la cual se realizan los primeros dos entrenamientos. Estos entrenamientos difieren entre sí por el hecho

de que uno ha sido llevado a cabo con la implementación de transferencia de aprendizaje y el otro no. La transferencia de aprendizaje tiene diversas finalidades, de las que pueden destacar la mejora de la tarea de aprendizaje y la reducción del tiempo de entrenamiento, en comparación de situaciones en el que no es utilizada. Los siguientes dos entrenamientos registrados se llevan a cabo con la misma estructura anterior, pero aplicando un cambio más relevante para mejorar la detección. Este cambio consiste en recalcular los anclajes del sistema de detección, basado en una nueva resolución de imagen que es más alta que la predeterminada. De la misma manera que para la anterior, esta estructura es entrenada con transferencia de aprendizaje y sin ella. Es importante recordar que la transferencia de aprendizaje que se trata aquí es realizada sobre una arquitectura neuronal que consta de solo 23 capas. Al ser una estructura con tan pocas capas se puede pensar que la cantidad de conocimiento que puede transferirse es poca, sin embargo, como se ilustra en la Tabla 4.1, en este caso, el uso de este método para la mejora del aprendizaje durante los entrenamientos presenta un incremento en el índice de precisión en comparación con los entrenamientos en los que no ha sido aplicado.

Porcentaje de la media del Promedio de Precisión (% mAP)					
Arquitectura de red	*Umbral de Intersección sobre la Unión (IoU)*				
	0.75	0.5	0.25	0.1	0.05
Tiny YOLO	52.7	91	96	96.1	96.1
Tiny YOLO 2	**54.7**	**92.4**	**96.8**	**97**	**97**
Tiny YOLO (TL)	62.7	92.8	96.7	96.8	96.8
Tiny YOLO 2 (TL)	**67.5**	**96**	**97.8**	**97.8**	**97.8**

Tabla 4.1: Resultados de entrenamientos de modelos de *tiny* YOLO con y sin *Transfer Learning* (TL) para detección de robot NAO.

Sobre la Tabla 4.1 se presenta una columna que corresponde a un umbral con valor de 0,5 para el parámetro de IoU. Es importante mencionar que en la mayoría de las competencias en donde es utilizada esta métrica de mAP, consideran este valor de umbral como el predeterminado para medir la precisión de detección del sistema. Sin embargo, como se observa en los resultados, a medida que se incrementa este valor el índice de precisión decrece y, de manera inversa, el porcentaje de precisión aumenta cuando se disminuye el umbral de IoU. Dependiendo de los objetivos específicos para

los cuales son entrenados los modelos, es posible variar este valor para obtener los resultados que sean más convenientes.

Por otro lado, cada implementación puede realizar el cálculo del valor de mAP de diversas maneras o con diferentes parámetros, lo que tiende a generar resultados de evaluación diferentes para cada una de ellas. Por ejemplo, para el caso de la implementación en *Darknet*, la evaluación utiliza un umbral para el valor de confidencia de 0,25, mientras que la implementación en *PyTorch* utiliza un umbral, para el mismo valor, de 0,001. La razón de hacer esto, de acuerdo a [16], es porque al disminuir el umbral de confianza o confidencia del objeto durante la evaluación, puede que se generen predicciones de cuadros delimitadores con muy bajo valor de confidencia pero, que pueden coincidir mejor con la predicción correcta. Cabe aclarar que, a pesar de las diferencias en los cálculos, las evaluaciones han sido llevadas a cabo bajo las mismas condiciones y parámetros. Esto con la intención de obtener resultados homogéneos y congruentes que puedan ser comparados de manera eficiente. Así también, las evaluaciones han sido realizadas considerando un umbral o *threshold* del parámetro confidencia del objeto detectado correspondiente al 50 %. Este valor puede ser ajustado, igualmente, dependiendo de los objetivos de los entrenamientos. Como se menciona antes, a medida que el valor decrece, el resultado de mAP aumenta y de manera inversa, al aumentarlo, la precisión aumenta. La decisión de qué valor asignar a este parámetro puede residir, simplemente, en preguntarse qué porcentaje de confianza para la predicción del sistema de detección se está dispuesto a aceptar para un fin determinado.

Como complemento a las evaluaciones del índice de precisión para calificar el desempeño de la red entrenada, también es posible observar la convergencia de los otros parámetros antes presentados. Lo anterior es posible a través de la información recopilada de dichos parámetros durante el entrenamiento. Las siguientes gráficas presentan la convergencia del valor de los parámetros precisión (Figura 4.10), recuperación (Figura 4.11), puntaje F1 (Figura 4.12), y por su puesto, de la media de las precisiones promedio (Figura 4.13). El comportamiento esperado para los valores de los parámetros aquí presentados, a medida que el entrenamiento es llevado a cabo, es una convergencia a el valor de la unidad.

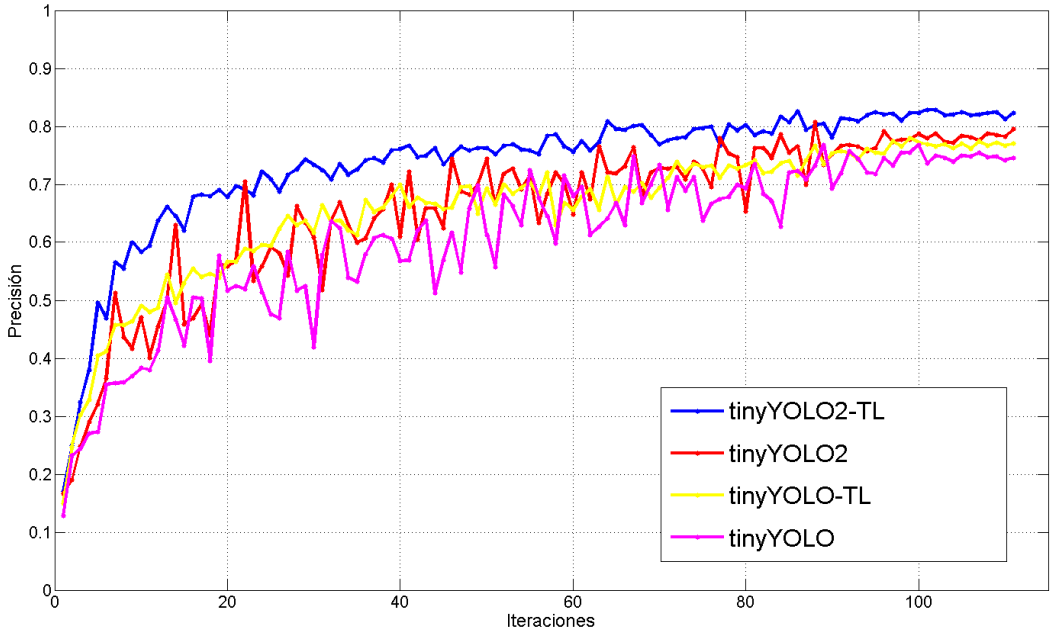


Figura 4.10: Gráficos de resultados para valores de Precisión durante entrenamiento, modelos *tiny* YOLO.

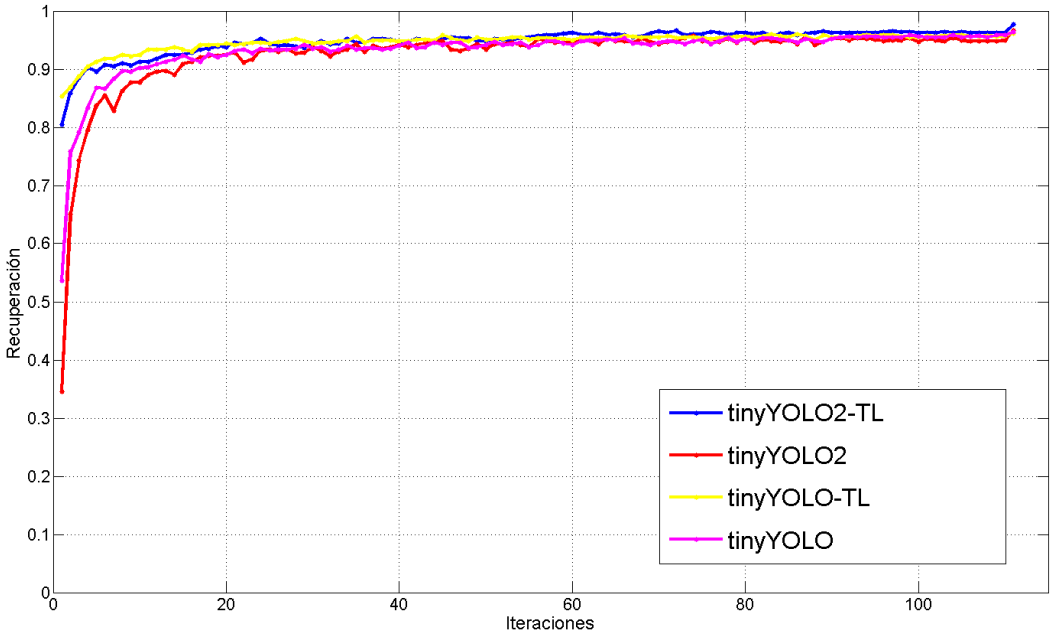


Figura 4.11: Gráficos de resultados para valores de Recuperación durante entrenamiento, modelos *tiny* YOLO.

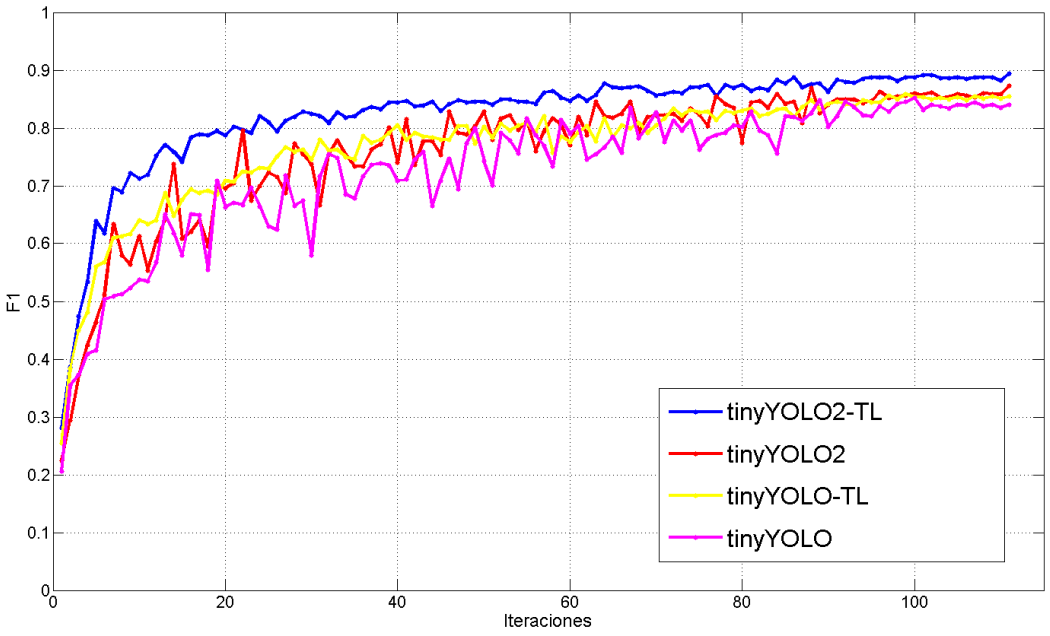


Figura 4.12: Gráficos de resultados para valores de puntaje F1 durante entrenamiento, modelos *tiny* YOLO.

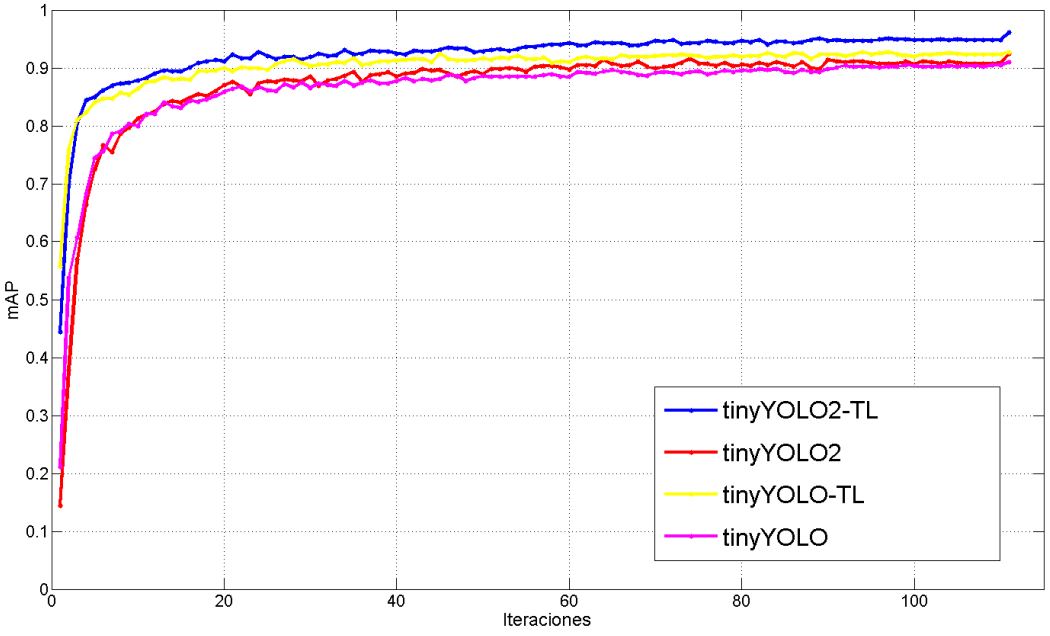


Figura 4.13: Gráficos de resultados para valores de mAP durante entrenamiento, modelos *tiny* YOLO.

Como se menciona anteriormente, la precisión y la recuperación hacen referencia al porcentaje de las predicciones positivas correctas y el porcentaje de positivos verdaderos predichos de manera correcta, respectivamente. En cuanto a la puntuación F1, es el promedio ponderado de los dos anteriores. El mAP es, para fines específicos del proyecto, la métrica de interés para la evaluación del índice de presión de la red. Como puede observarse tanto en la Tabla 4.1, como en la gráfica de la Figura 4.13, los entrenamientos realizados sobre el modelo denominado tinyYOLO2 entregan los mejores resultados en ambos casos de entrenamiento, esto es con la implementación de transferencia de aprendizaje y sin ella.

Otros comportamientos que son importantes de analizar durante los entrenamientos son los valores de pérdida de regresión, conocido en la literatura como *loss*. Recordando un poco, se desea que los valores de pérdida converjan al cero, o en su defecto, a un valor cercano a éste. Para el caso general, esto quiere decir que la diferencia o el error entre las predicciones realizadas durante el entrenamiento y las etiquetas de las bases de datos es muy pequeña, lo que indica que en aprendizaje se está llevando de manera exitosa. De la misma manera, para evitar caer en sobreajuste, es recomendable evaluar estos valores sobre un conjunto de datos diferente al de entrenamiento. Para este caso, se realiza sobre el conjunto de evaluación o prueba.

Existen también diferentes métricas para evaluar otros valores de pérdida, por ejemplo, para evaluar la pérdida de regresión óptima de los cuadros delimitadores se puede utilizar el IoU generalizado o GIoU. Al incorporarlo como métrica para valores de pérdida se produce una mejora consistente en el desempeño del detector de objetos [29].

Como puede observarse en las siguientes gráficas (Figuras 4.14, 4.17, 4.16 y 4.17), el comportamiento de los valores de pérdida, en ambas métricas y para ambos conjuntos de datos, se da de la manera esperada al converger a un valor cercano a cero.

Finalmente, a pesar de que el incremento que se presenta en los entrenamientos realizados con *Transfer Learning* podría parecer no es muy significativo, con respecto a los entrenamientos sin esta técnica. Una diferencia más considerable, al menos visualmente hablando, puede encontrarse al momento de utilizar el archivo de pesos obtenido del entrenamiento con el sistema de detección. Con ello se puede observar como el uso

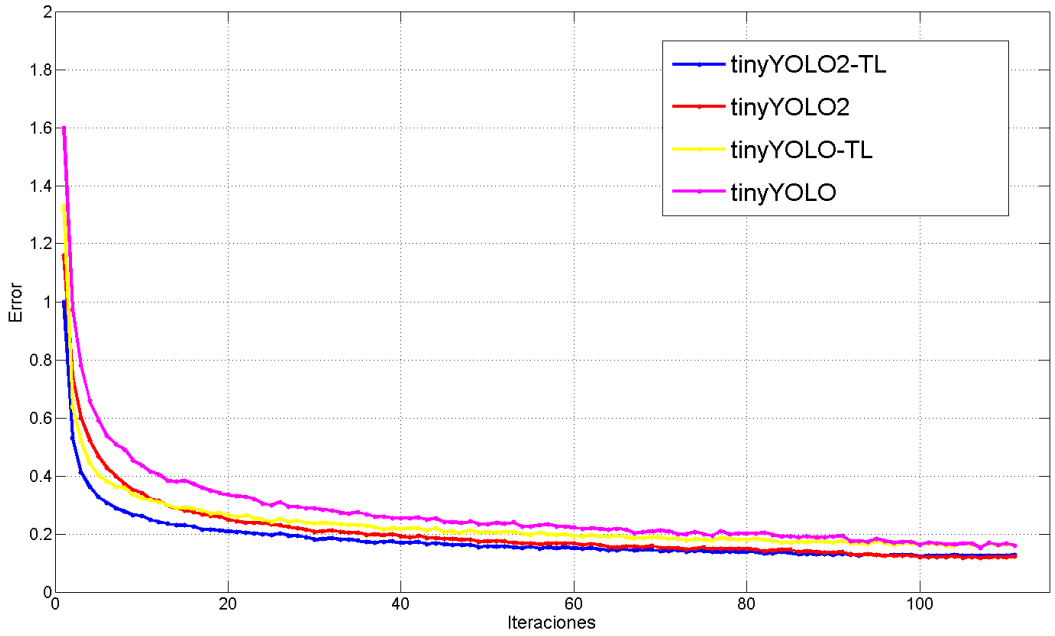


Figura 4.14: Gráficos de resultados para valores de pérdida (error) durante entrenamiento, modelos *tiny* YOLO.

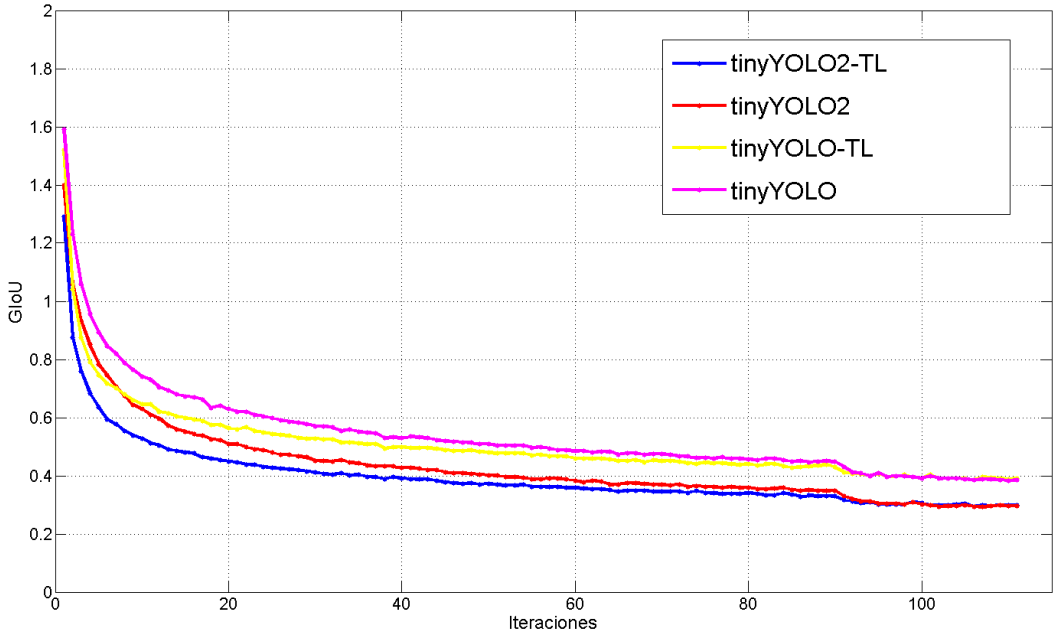


Figura 4.15: Gráficos de resultados para valores de GIoU durante entrenamiento, modelos *tiny* YOLO.

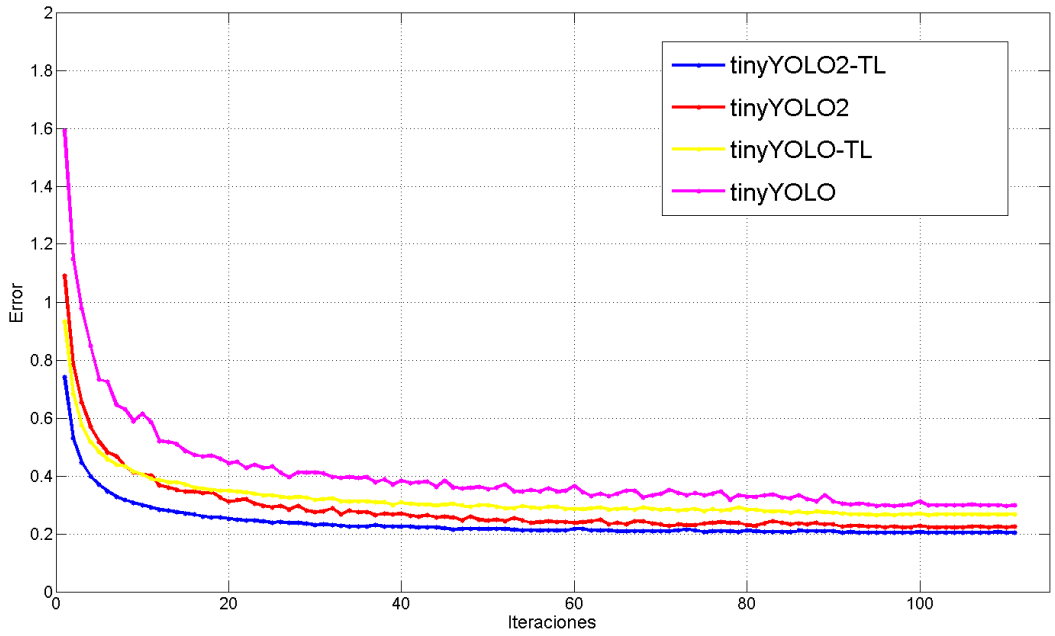


Figura 4.16: Gráficos de resultados para valores de pérdida (error) durante entrenamiento sobre el conjunto de datos de validación, modelos *tiny* YOLO.

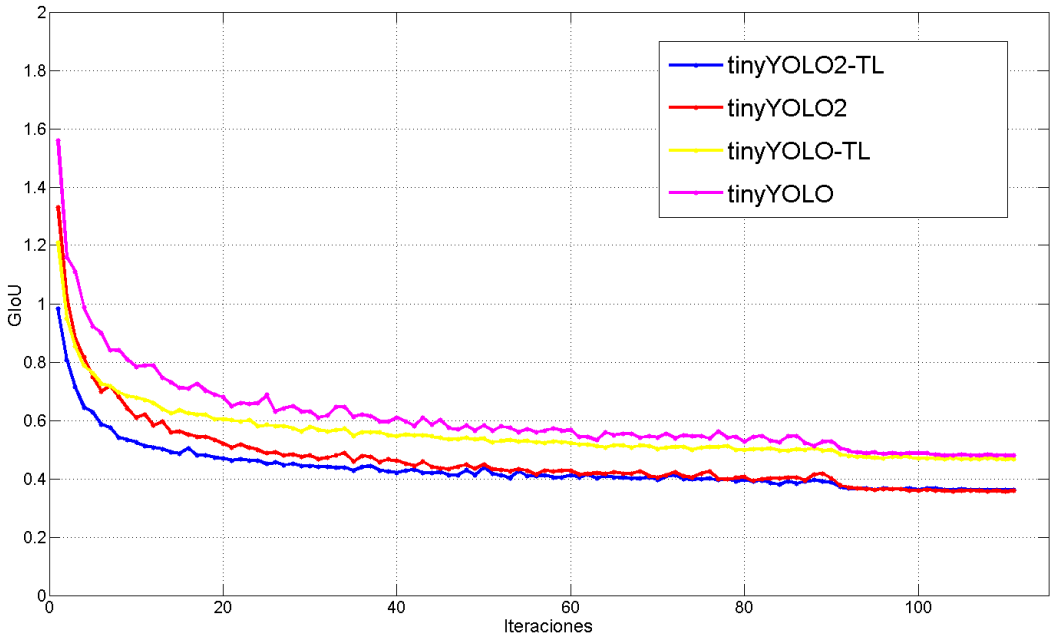


Figura 4.17: Gráficos de resultados para valores de puntaje GIoU durante entrenamiento sobre el conjunto de datos de validación, modelos *tiny* YOLO.

de la transferencia de aprendizaje, además de acelerar el tiempo de entrenamiento al solo entrenar las últimas capas de la red, también mejora el desempeño de la tarea de detección al momento de ser ejecutada sobre datos nuevos para el sistema. La figura 4.18 presenta algunos ejemplos de la detección con el modelo tinyYOLO2-TL.

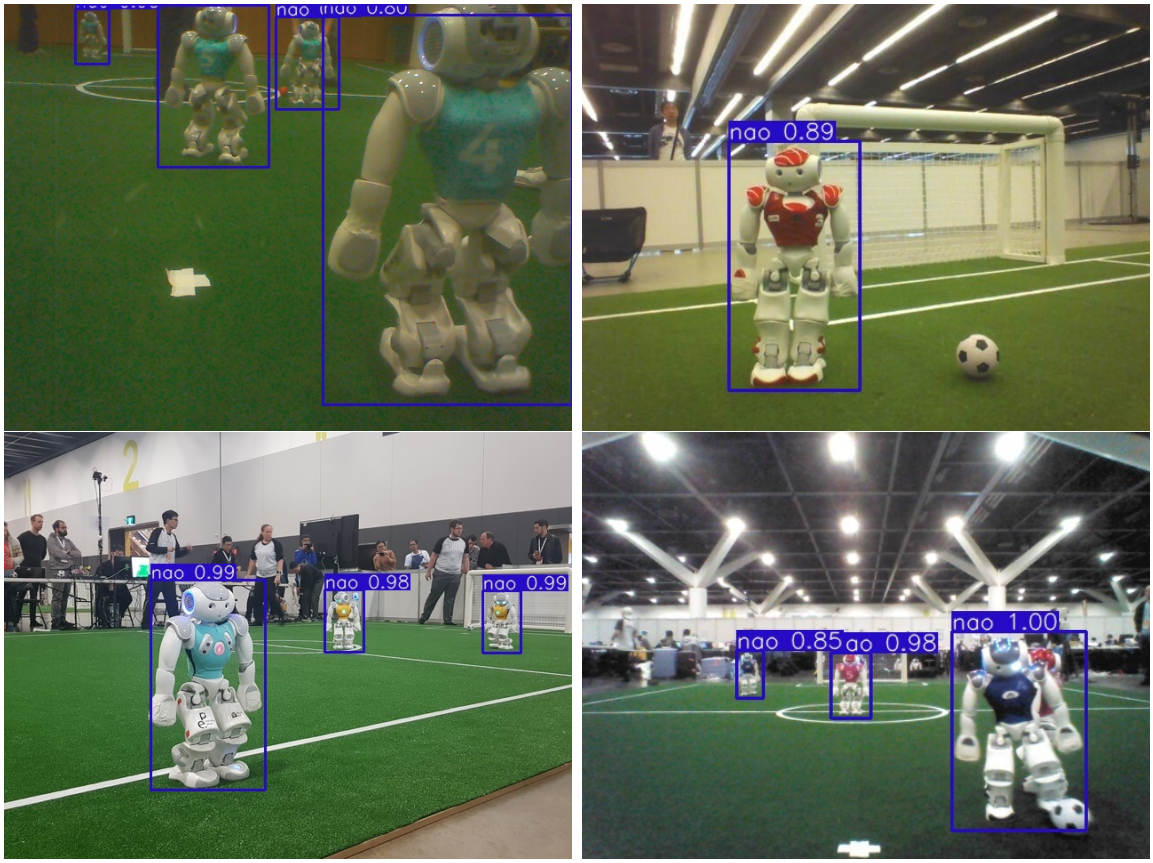


Figura 4.18: Ejemplos de detección de robot NAO con modelo tinyYOLO2-TL.

4.4.2. Modelos propuestos

Es importante señalar que, a diferencia de los entrenamientos presentados previamente, estas propuestas de modelos no han sido entrenadas con técnicas de transferencia de aprendizaje. Esto se debe, principalmente, al hecho de que se trata de arquitecturas "nuevas", por lo que no existe un precedente de archivo de pesos que pueda ser usado como referencia para hacer el traspaso de conocimiento. A pesar de ello, los resultados para el índice de presión obtenidos, y presentados en la Tabla 4.2 son satisfactorios.

Porcentaje de la media del Promedio de Precisión (% mAP)					
Arquitectura de red	*Umbral de Intersección sobre la Unión (IoU)*				
	0.75	0.5	0.25	0.1	0.05
Tiny YOLO R1	44.7	**91.4**	95.5	95.7	95.8
Tiny YOLO R2	35.3	**77.9**	87.4	88.7	89.2
TYR-NAO	56	**81**	88	88.3	88.4

Tabla 4.2: Resultados de entrenamientos de modelos propuestos para detección de robot NAO.

Tal como se puede observa en dicha tabla, para la primera de las estructuras, denominada R1, se presentan los resultados más altos en cuanto a mAP se refiere. Sin embargo, es importante notar que la reducción de la arquitectura y las modificaciones sobre este modelo no son muy significativas con respecto al original, por lo que es lógico que se conserve un buen nivel de precisión. Por su parte, la segunda de las propuestas (R2) está constituida por solo 15 capas, y presenta un cambio mucho más significativo, lo que en un inicio lleva a suponer un procesamiento de ésta más rápido que la anterior. A pesar de esto, se observa que debido a la misma reducción y modificaciones realizadas, esta arquitectura presenta un sacrificio en lo que a los valores de precisión respecta.

Por último, se tiene la propuesta denominada TYR-NAO. Recordando que ésta presenta una arquitectura de modelo diferente, la cual no solo contiene menos capas (16) que el modelo *tiny*, sino que también el número de filtros en cada capa es menor y que trabaja sobre volúmenes más reducidos. Esto la hace prospecto a ser procesada mucho más rápido que los todos los modelos anteriormente analizados. Además, de acuerdo a la información de la tabla 4.2, los resultados del índice de precisión alcanzan un valor por arriba del 80 %, lo que la coloca en el segundo lugar entre las dos propuestas de reducción, esperando solo resultados de tiempo de ejecución. Los resultados para los parámetros de evaluación de estos entrenamientos se comparan en las siguientes gráficas.

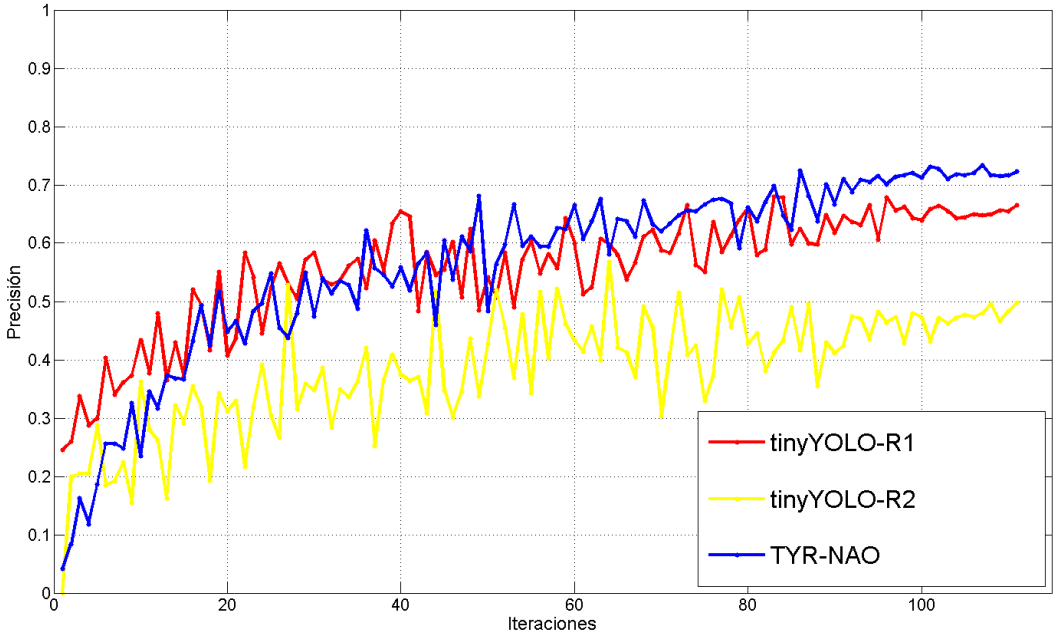


Figura 4.19: Gráficos de resultados para valores de Precisión durante entrenamiento, modelos propuestos.

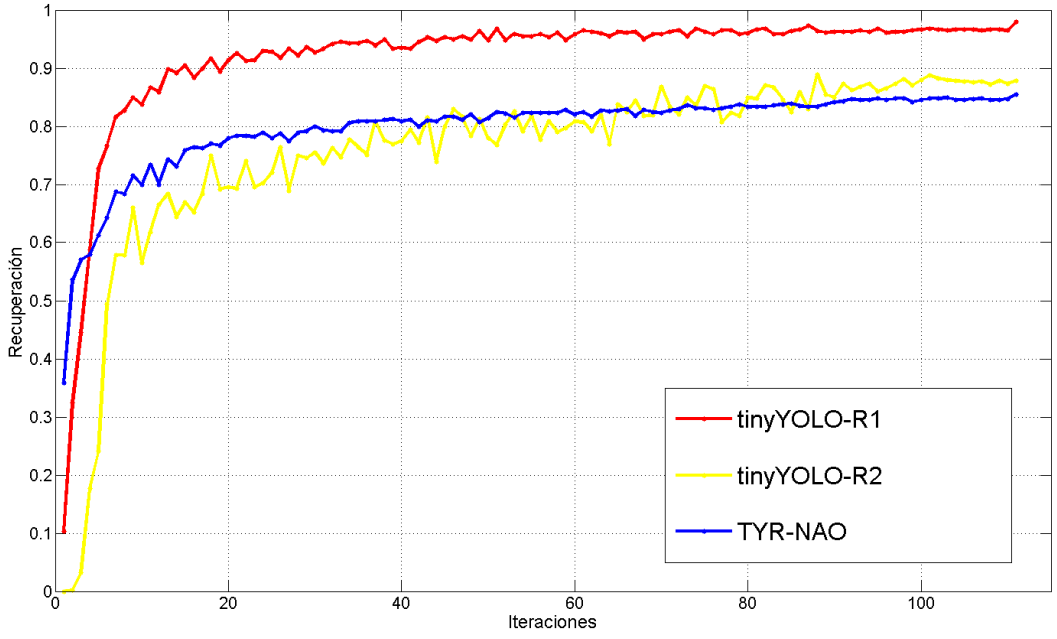


Figura 4.20: Gráficos de resultados para valores de Recuperación durante entrenamiento, modelos propuestos.

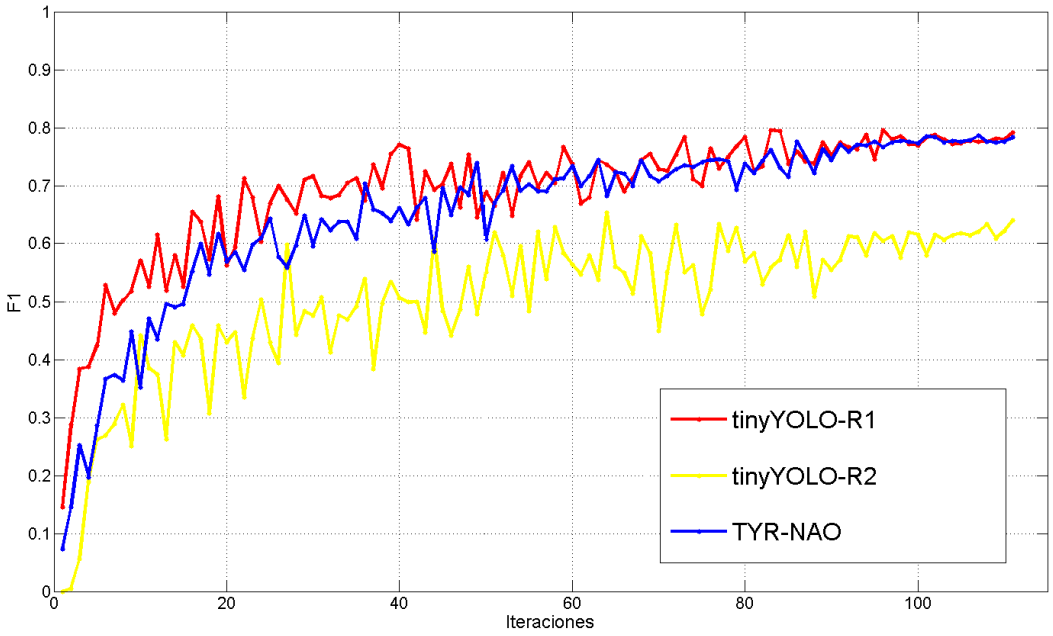


Figura 4.21: Gráficos de resultados para valores de puntaje F1 durante entrenamiento, modelos propuestos.

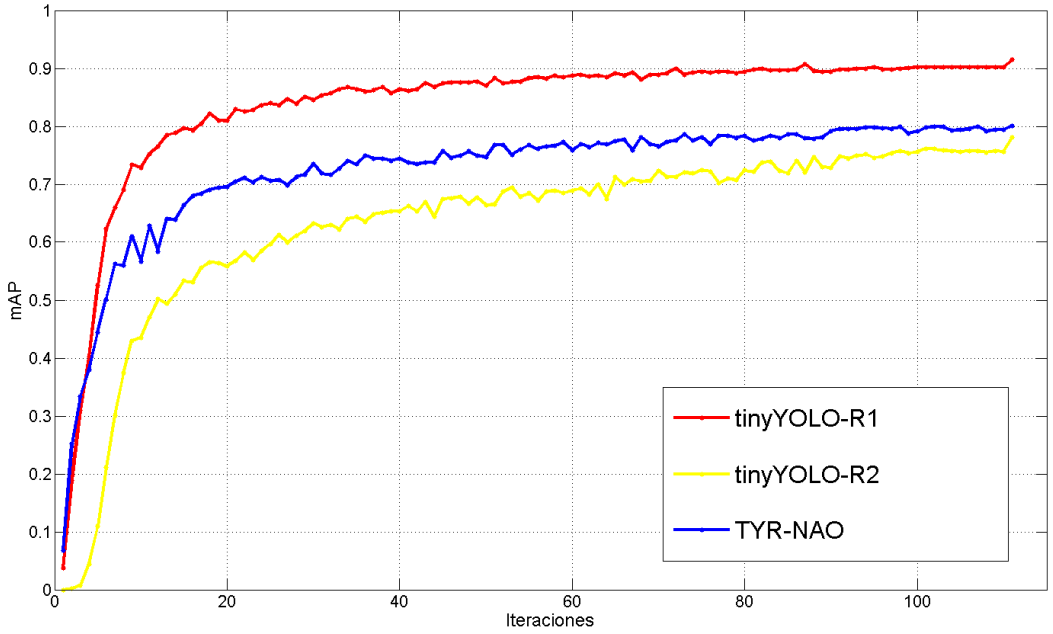


Figura 4.22: Gráficos de resultados para valores de mAP durante entrenamiento, modelos propuestos.

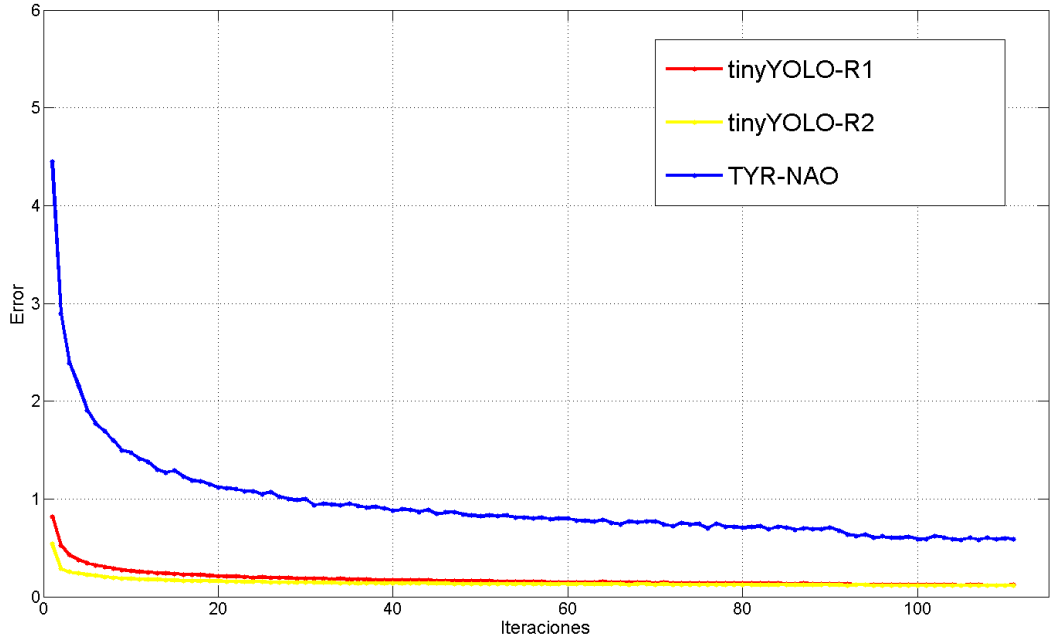


Figura 4.23: Gráficos de resultados para valores de pérdida (error) durante entrenamiento, modelos propuestos.

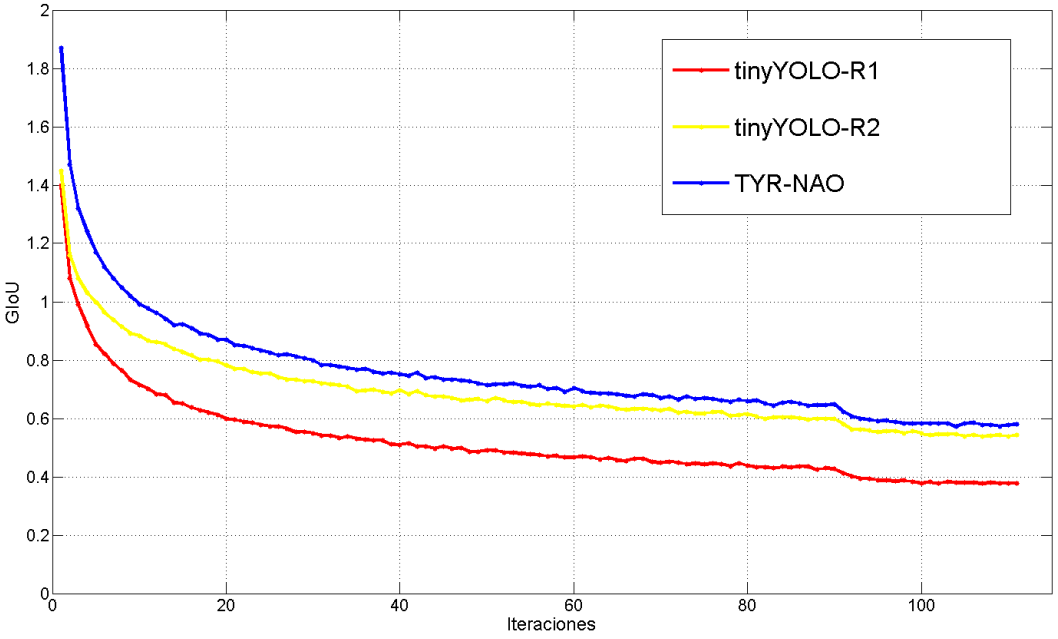


Figura 4.24: Gráficos de resultados para valores de GIoU durante entrenamiento, modelos propuestos.

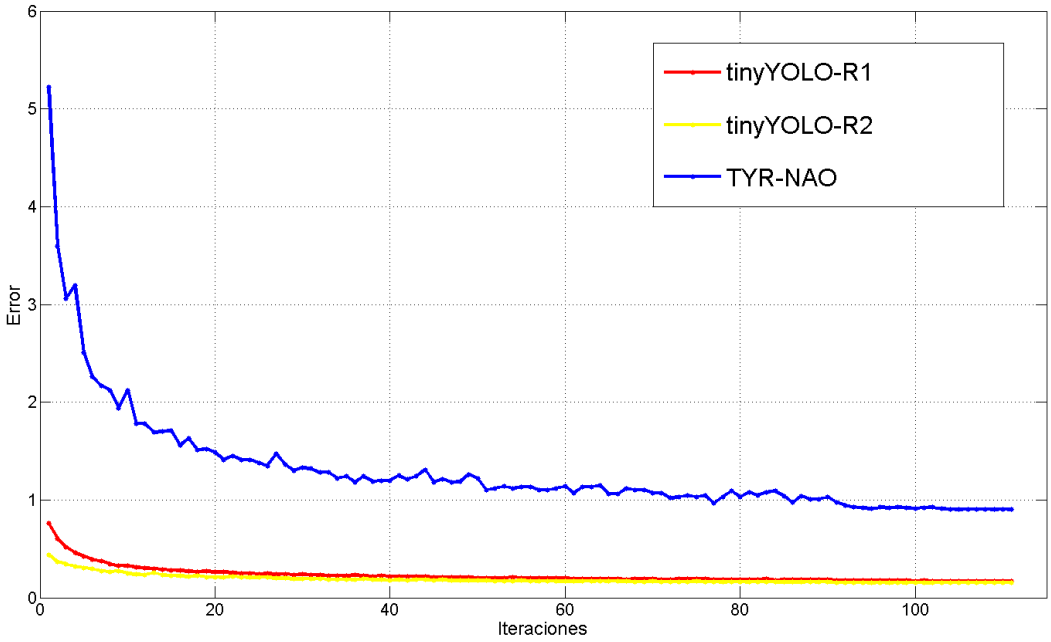


Figura 4.25: Gráficos de resultados para valores de pérdida (error) durante entrenamiento sobre el conjunto de datos de validación, modelos propuestos.

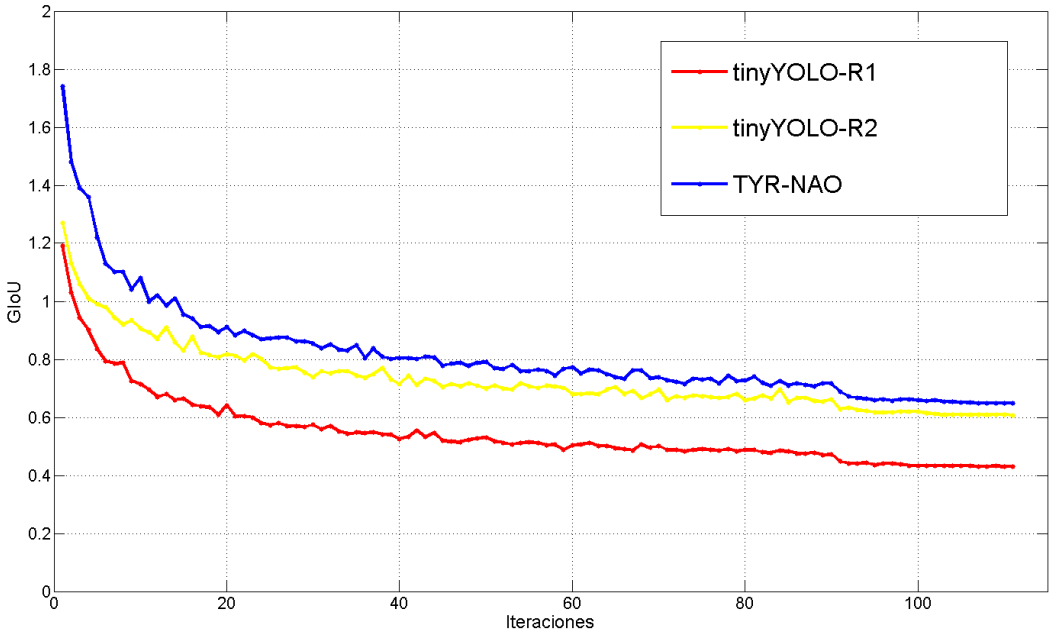


Figura 4.26: Gráficos de resultados para valores de puntaje GIoU durante entrenamiento sobre el conjunto de datos de validación, modelos propuestos.

4.4.3. Tiempos de ejecución

A pesar del hecho de que se han obtenido resultados de precisión por encima del 90 % para algunos entrenamientos, hay que recordar que la idea principal es ejecutar estos modelos sobre la plataforma del robot NAO, que es un sistema embebido de bajo poder computacional. Por esta razón, analizar los tiempos de procesamiento para los modelos entrenados es de gran importancia. Debido a la dinámica del juego de fútbol, se busca obtener los tiempos más bajos posibles. A continuación, se presentan los tiempos de ejecución promedio para cada uno de los modelos, este tiempo representa cuánto se tarda el sistema en procesar la imagen y realizar la detección sobre la misma.

Tiempos de ejecución (ms)			
Arquitectura de red	*Resolución de imagen*	*Tiempo*	*f/s*
Tiny YOLO	416x416	**295**	3.4
Tiny YOLO 2	608x608	**545**	1.8
Tiny YOLO R1	416x416	**484**	2.1
Tiny YOLO R2	224x224	**145**	6.9
TYR-NAO	512x384	**86**	11.6

Tabla 4.3: Resultados de tiempos de ejecución de modelos entrenados para detección de robot NAO.

La Tabla 4.3 enlista los modelos presentados anteriormente. Al mismo tiempo contiene, no solo el tiempo que tarda el sistema en realizar la detección con cada una de las estructuras, sino también la resolución de imagen con la que se inicia este proceso y, adicionalmente, el número de cuadros por segundo (f/s) que se procesan a partir del tiempo de detección especificado. Como se puede observar, el modelo TYR-NAO que ha sido propuesto resulta ser el que es procesado con mayor velocidad con respecto a los demás. Seguido a este se encuentra la segunda reducción propuesta, la cual contiene una capa menos que el anterior y una resolución menor. Posteriormente, se tiene el modelo original *tiny* YOLO con su resolución predeterminada. Curiosamente, por debajo de éste se posiciona la primera de las reducciones propuestas. Ambas trabajan sobre la misma resolución de imagen pero, aun cuando el modelo R1 presenta una arquitectura de menor tamaño, termina manejando volúmenes más grandes, lo que produce que su procesamiento sea más lento. Finalmente, se tiene de nuevo el modelo original modifi-

cado para utilizar una resolución más grande que la predeterminada y, a pesar de ser el modelo con el índice de precisión más alto de todos, es el modelo que más lento se procesa para realizar la detección.

Cabe señalar que el tamaño o resolución de imagen también juega entonces un papel importante al momento de ejecutar el sistema de detección. Por esta razón, se ha probado el modelo TYR-NAO con las diferentes resoluciones utilizadas por los modelos restantes para poder hacer una comparación directa con cada uno de ellos. Así, la Tabla 4.4 presenta los resultados del tiempo de ejecución de este modelo para las diferentes resoluciones y los cuadros por segundo que procesa el mismo.

Tiempos de ejecución (ms)		
Resolución de imagen	*Tiempo*	*f/s*
608x608	**134**	7.5
512x384	**86**	11.6
416x416	**70**	14.3
256x192	**33**	30.3
224x224	**30**	33.3

Tabla 4.4: Resultados de tiempos de ejecución de modelo TYR-NAO para diferentes resoluciones de imagen.

Tal y como presenta la tabla anterior, la propuesta TYR-NAO resulta ser el más veloz de todos los modelos entrenados, sin importar la resolución que se utilice. Así, es 6 veces más rápida, con su configuración de resolución predeterminada, que la arquitectura *tiny* YOLO que presenta el mayor índice de precisión, y está solo 15 puntos por debajo de ésta. A su vez, cuando se comparan ambas estructuras bajo la misma resolución, el modelo propuesto sigue siendo 4 veces más veloz. Por lo que con su porcentaje de precisión, que es alrededor del 80 %, es la opción más viable para la ejecución sobre la plataforma del robot NAO. Las siguientes figuras (Figura 4.27 y 4.28) presenta algunos ejemplos de la detección realizada con este modelo.

Figura 4.27: Ejemplos de detección de robot NAO con modelo TYR-NAO.

Figura 4.28: Ejemplos de detección de robot NAO con modelo TYR-NAO.

Capítulo 5

Conclusiones y Trabajos Futuros

Este apartado presenta las conclusiones generales y específicas de los puntos más importantes, con respecto a los resultados obtenidos durante el desarrollo de lo aquí planteado. Así también, hace algunas propuestas de actividades para darle continuidad y expandir el alcance de este trabajo en un futuro.

5.1. Conclusiones

En un principio se establecieron una pequeña serie objetivos que fueron pensados para alcanzar el objetivo principal de este proyecto. Sin embargo, conforme la investigación sobre las áreas necesarias se realizaba, algunos de esos objetivos se volvieron innecesarios y se decidió enfocarse en otras actividades para que el trabajo pudiera llegar a la meta esperada.

En primer lugar, se tenía la idea de que aplicar un procesamiento a las imágenes que entraban a la red neuronal podía mejorar el entrenamiento. Esto se debía, principalmente, a que sistemas de detección anteriores, basados en el enfoque clásico del procesamiento digital de imágenes (PDI), realizaban este procedimiento para *mejorar* las imágenes para que se adaptaran, de manera más adecuada, al propósito de dicho sistema. A pesar de esto, conforme se fue profundizando en los sistemas de detección basados en CNNs, específicamente en el sistema de detección de YOLO, se ha encontrado que, debido a las propiedades de generalización que las redes convolucionales

aportan al sistema, hacer un procesamiento de las imágenes antes de realizar el entrenamiento no es necesario. La razón principal para justificar esto se da a partir de la implementación de las técnicas de aumento de datos que normalmente se utilizan en los entrenamientos para este tipo de aplicaciones. El aumento de datos consiste, de manera general, en la creación de nuevos datos (imágenes para el caso en cuestión), a partir de la manipulación de los datos originales. Así, por ejemplo, en los entrenamientos realizados para la detección del robot, se realizan algunas modificaciones a las imágenes de la base de datos. Estos cambios, como rotaciones y cambios de escala, ayudan a generalizar la detección del objeto para diferentes tamaños y posiciones. De la misma manera, y enfocado en el procesamiento de las imágenes, también es posible realizar cambios o distorsiones en los canales o espacios de color de las imágenes. Tomando en cuenta que el procesamiento pensado inicialmente para las imágenes estaría enfocado en homogeneizar, en cuanto a espacios de color se refiere, todas las imágenes de la base de datos, se ha decido entonces, que la mejor opción para los fines de este proyecto es la de dirigir el enfoque, para este objetivo en particular, en definir de la manera más óptima posible los parámetros necesarios para el aumento de datos. Lo anterior con la finalidad de obtener los mejores resultados de los entrenamientos.

Como segundo objetivo específico se planteo el desarrollar un extractor de características para las aquellas que se deseaban obtener de las imágenes de la base de datos. Una vez más, es importante aclarar que estos objetivos se plantearon sin un conocimiento completo del funcionamiento de los modelos computacionales que son las redes neuronales convolucionales. Por lo que la intención de este objetivo también esta inspirada en sistemas que realizaban la detección siguiendo una metodología diferente. Sin embargo, tal y como se menciona previamente (sección 2.3.2.1) las CNNs se encargan de extraer las características, tanto de bajo, como de alto nivel, de las imágenes que pasan entre cada una de las capas que componen la red. Por esta razón se ha tomado la decisión de centrarse en una recopilación de imágenes exhaustiva para la creación de la base de datos que sería utilizada para los entrenamientos del sistema de detección. En un principio, debido a que era desconocida la magnitud que tiene, no solo la importancia sino también, el trabajo requerido para la elaboración de una base de datos

que cumpla con los requisitos para poder satisfacer los propósitos del trabajo, no fue contemplado definir este proceso como uno de los objetivos específicos. No obstante, y ya que el desarrollo de un algoritmo para la extracción de características resulta ser innecesario para el caso en cuestión, la concentración que se pensaba dedicar para esta tarea, ha sido dirigida a la construcción de la base de datos. Dicha base de datos, conformada por 3500 imágenes y 3500 archivos de etiqueta para los objetos de clase nao, se presenta como una de las aportaciones que se ofrecen en este proyecto. La importancia de esta tarea recae, principalmente, en que todo el aprendizaje que realiza el sistema a lo largo del entrenamiento depende completamente de los datos y sus etiquetas. Siendo un aprendizaje supervisado, dedicar el tiempo y esfuerzo necesario en la preparación adecuada de las bases de datos es vital para la obtención de los resultados deseados.

Finalmente, el último de los objetivos específicos agrupaba la implementación de algunas redes, así como el diseño de algunas otras para implementarse también en el entrenamiento del sistema para la detección del robot NAO. Al mismo tiempo, pretendía la aplicación técnicas de transferencia de aprendizaje para llevar a cabo dicha detección de manera exitosa. En cuanto a este objetivo se refiere, las actividades que lo componen han sido realizadas y presentadas en este documento de manera satisfactoria. En un inicio, las primeras implementaciones han sido realizadas sobre el modelo *tiny* de YOLO. Una vez que la confección de la base de datos fue terminada, han sido efectuados los primeros entrenamientos con esta arquitectura. Posteriormente, cuando se obtuvieron los primeros resultados, se ha procedido a realizar la implementación de la misma red, pero con la aplicación de la transferencia de aprendizaje. Aquí, tal y como se presenta en la sección 4.4.1 existe un aumento en los porcentajes de precisión al momento de transferir el aprendizaje de los pesos del modelo preentrenado. Esto demuestra que la aplicación de estas técnicas puede, no solo ahorrar tiempo de entrenamiento pues se conserva parte de los pesos que se transfieren, sino también, aumentar el índice de precisión. Además, esta mejora en el incremento de precisión también puede ser apreciada de manera visual, ya que con las pruebas realizadas se ha encontrado que el desempeño de la detección mejora al ejecutarse con el archivo de pesos obtenido del entrenamiento al cual le fue transferido un aprendizaje previo.

A pesar de los buenos resultados en desempeño que presentan los modelos de YOLO, los tiempos de ejecución para estos sistemas no son los más adecuados para la aplicación deseada. Por esto, fueron realizadas algunas modificaciones de la arquitectura original para buscar mantener un porcentaje alto de precisión, pero enfocado en reducir el tiempo de ejecución. En este punto, tanto para la modificación, como para el diseño de una propuesta, es importante considerar que son necesarias un número indefinible de pruebas para encontrar los parámetros que mejor se ajusten y que entreguen los mejores resultados. Esto porque los parámetros de entrenamiento y configuración de un mismo modelo pueden variar entre una tarea en particular y otra. Para este caso en particular, la mayoría de los parámetros han sido conservados entre una estructura y otra.

Los cambios más importantes se han presentado en la arquitectura y tamaño de la red, buscando la reducción en el procesamiento sobre la plataforma. Así, se han obtenido los resultados de los entrenamientos para los modelos presentados en la sección 4.4.2. Una diferencia considerable, con respecto a los modelos de YOLO, es que los modelos propuestos no han podido ser entrenados con transferencia de aprendizaje, al menos no para este caso en particular. La principal razón se debe a que son arquitecturas sin precedentes, por lo que no hay archivos con los que se pueda transferir un aprendizaje previo, sin embargo, no se descarta la posibilidad de hacerlo a partir de los pesos obtenidos durante la elaboración del proyecto y, para entrenamientos con bases de datos similares.

Partiendo de los valores de precisión obtenidos para cada modelo, incluso cuando las arquitecturas expuestas como la propuesta principal de este documento presentan algunas unidades por debajo de los porcentajes de precisión logrados con los entrenamientos del modelo *tiny*, cabe resaltar que la principal razón de esta proposición ha sido la de mejorar los tiempos de procesamiento para cada estructura sobre el robot. Así, y de acuerdo a los resultados presentados en la sección 4.4.3, la propuesta de reducción denominadas R1 no presenta disminución alguna en tiempo de ejecución, al contrario, su procesamiento demora más que la versión predeterminada del modelo *tiny*. Esto, seguramente, provocado debido a las dimensiones de volúmenes que maneja dicha arquitectura. Aunado a eso, existe un sacrificio presentado en los índices de precisión, que

hacen que las modificaciones aplicadas a esta arquitectura no valgan la pena. Por su parte, el modelo R2 ofrece una reducción en cuanto a estructura que más significativa, disminuyendo en gran medida el tiempo de procesado. No obstante, las modificaciones que presenta esta configuración en su arquitectura han provocado que el valor de precisión exhiba una caída importante en el rendimiento del sistema de detección al implementar este modelo.

Por otro lado, se hace una mención especial al modelo propuesto TYR-NAO, ya que debido a los requerimientos para llevar acabo la detección sobre el robot NAO, éste presenta los resultados más adecuados en cuanto a tiempo y precisión se refiere. Comparado con los dos modelos analizados anteriormente, la disminución que presenta en sus valores de precisión, con respecto a la reducción de la duración de detección sobre el robot, hace que se considere como la opción más adecuada. De este modelo se puede rescatar, en primer lugar, la eliminación de las capas de reducción *maxpool* y el aumento, en número, de capas convolucionales. Esta acción conlleva a la extracción de más características y la conservación de más información de los datos conforme se pasa de una capa a otra sobre la red. Por lo que a pesar de tener aproximadamente el mismo número de capas que el modelo R2, su porcentaje de precisión es mayor. Además, su estructura reducida en tamaño, número de capas y número de filtros en cada capa la hace, evidentemente, más veloz que el modelo *tiny*. Ciertamente, presenta una reducción de más de 10 unidades en precisión, respecto al mejor resultado obtenido con YOLO, pero lo compensa al ser, al menos, 6 veces más veloz en su ejecución sobre la plataforma. De esta manera, la propuesta de este modelo para la detección específica del robot NAO, junto con la base de datos para su entrenamiento, es la principal aportación que se presenta en este trabajo, el cual está enfocado, de manera adicional, a su implementación en la categoría SPL de la RoboCup.

5.2. Trabajos futuros

A partir de los resultados obtenidos y las conclusiones expuestas en este documento, se proponen algunas actividades para darle seguimiento a la línea de investigación que aquí se plantea. Como primer punto, se presenta la opción de implementar el modelo propuesto y su entrenamiento para la detección de cualquier otra clase de objeto. Principalmente para evaluar si la red mantiene su índice de desempeño en diversos ambientes y/o escenarios, y sobre todo, que sean diferentes al utilizado en cuestión. Esto con la intención de identificar qué otras áreas o grupos de investigación pueden apoyarse y verse beneficiadas con los resultados que presenta este modelo y, si es posible la existencia de alguna mejora, tanto en tiempo, como en precisión, para diferentes aplicaciones.

Continuando con la presente rama a la que se encuentra enfocado este proyecto, se propone construir nuevas bases de datos y entrenar el modelo TYR-NAO para la detección de cada una del resto de clases de objetos que son relevantes para el juego de fútbol de robots. Realizar una evaluación de los resultados de estos entrenamientos y, en caso de ser necesario, las modificaciones necesarias en cada clase, para mejorar dichos resultados. Como complemento, agrupar las diferentes bases de datos en una sola que consista en la inclusión de todas las diferentes clases de objetos disponibles. Posteriormente, realizar el entrenamiento del modelo propuesto para dichas clases, así como evaluar el desempeño y el comportamiento del mismo para la detección de más de una sola clase y su ejecución sobre la plataforma NAO.

Adicionalmente, buscar hacer una comparación entre los resultados de la propuesta anterior y resolver la interrogante acerca de si la opción más adecuada para la detección de múltiples objetivos de diferentes clases en el juego de fútbol, está conformada por la implementación de un solo sistema que se encargue de la detección de todos los objetos en cuestión, o, si el incorporar diferentes sistemas, cada uno especialmente entrenado para la detección de los objetos de manera individual, genera los mejores resultados. Lo anterior, basado en los tiempos de ejecución y en los porcentajes de precisión que se presenten en cada caso.

Bibliografía

[1] M. Alam, L. Vidyaratne, T. Wash, and K. Iftekharuddin. Deep srn for robust object recognition: A case study with nao humanoid robot. In *SoutheastCon, 2016*, pages 1–7. IEEE, 2016.

[2] 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*, pages 392–403. Springer, 2016.

[3] AlexeyAB. Darknet: Windows and linux version of darknet yolo v3 and v2 neural networks for object detection. https://github.com/AlexeyAB/darknet-how-to-train-to-detect-your-custom-objects, 2016.

[4] AlexeyAB. Yolomark: Windows and linux gui for marking bounded boxes of objects in images for training yolo. https://github.com/AlexeyAB/Yolomark, 2016.

[5] 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*, pages 19–30. Springer, 2017.

[6] A. Deshpande. A beginner's guide to understanding convolutional neural networks. *A Beginner's Guide to Understanding Convolutional Neural Networks–Adit Deshpande–CS Undergrad at UCLA ('19). Np*, 20, 2016.

[7] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[8] H. Farazi and S. Behnke. Online visual robot tracking and identification using deep lstm networks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6118–6125. IEEE, 2017.

[9] R. Federation. Robocup: Objective. http://www.robocup.org/objective, 2018.

[10] R. Federation. Robocup: Official site. http://www.robocup.org, 2018.

[11] R. C. Gonzalez and P. Wintz. Digital image processing. Technical report, Addison-Wesley publishing company, 1987.

[12] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[13] J. R. Hilera González, V. J. Martínez Hernando, et al. *Redes neuronales artificiales: fundamentos, modelos y aplicaciones*. Alfaomega, 2000.

[14] J. Hui. map (mean average precision) for object detection. *Jonathan Hui*, 2018.

[15] M. Javadi, S. M. Azar, S. Azami, S. S. Ghidary, S. Sadeghnejad, and J. Baltes. Humanoid robot detection using deep learning: a speed-accuracy tradeoff. In *Robot World Cup*, pages 338–349. Springer, 2017.

[16] G. Jocher, guigarfr, perry0418, Ttayu, J. Veitch-Michaelis, G. Bianconi, F. Baltacı, D. Suess, WannaSeaU, and IlyaOvodov. ultralytics/yolov3: Rectangular Inference, Conv2d + Batchnorm2d Layer Fusion, Apr. 2019.

[17] R. Joshi. Accuracy precision recall f1 score: Interpretation of performance measures. *URL:https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/*, 1, 2016.

[18] A. Kathuria. What's new in yolo v3? https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b, 2018.

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

[20] R. Lopez. Introducción al deep learning. https://iaarhub.github.io/introduccion-al-deep-learning/, 2017.

[21] S. Nayak. Deep learning based object detection using yolov3 with opencv (python / c++). https://www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/, 2018.

[22] S. Nayak. Training yolov3 : Deep learning based custom object detector. https://www.learnopencv.com/training-yolov3-deep-learning-based-custom-object-detector/, 2019.

[23] M. A. Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

[24] B. Poppinga and T. Laue. Jet-net: Real-time object detection for mobile robots. In *RoboCup 2019: Robot World Cup XXIII*, Lecture Notes in Artificial Intelligence. Springer, 2019.

[25] J. Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *arXiv*, 2016.

[27] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[28] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[29] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2019.

[30] S. Robotics. Official site: Nao. *https://www.softbankrobotics.com/emea/en/nao*, 2018.

[31] M. Szemenyei and V. Estivill-Castro. Real-time scene understanding using deep neural networks for robocup spl. In *Robot World Cup*, pages 96–108. Springer, 2018.

[32] M. Szemenyei and V. Estivill-Castro. Robo: Robust, fully neural object detection for robot soccer. In *RoboCup 2019: Robot World Cup XXIII*. Springer, 2019.

[33] L. Torrey and J. Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global, 2010.

[34] J. I. Zannatha, L. F. Medina, R. C. Limón, and P. M. Álvarez. Behavior control for a humanoid soccer player using webots. In *CONIELECOMP 2011, 21st International Conference on Electrical Communications and Computers*, pages 164–170. IEEE, 2011.