Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Reconocimiento de Objetos en Tiempo Real para un Robot Humanoide usando Técnicas de Inteligencia Artificial

Tesis presentada para obtener el grado de:

Ingeniero en Tecnologías de la Información

Presenta:

Jorge Molina Lechuga

Dirección de Tesis:

Dr. David Eduardo Pinto Avendaño

Puebla, Pue. Mayo 2023

Dedicatoria

A mis padres y a mi hermano.

A mis abuelos.

Y a todos aquellos, amigos y familiares que desde muy arriba me guían.

MILITIA EST VITA.

Agradecimientos

A mis padres, por todo su cariño, apoyo, esfuerzo y guía que me han dado a lo largo de los años y que me han vuelto quien hoy soy, gracias por todo y por alentarme a seguir mis sueños y metas, así como la presente tesis y entre otros logros no serían posibles sin ustedes. Espero con el presente trabajo devolverles un poco de la dicha que se me ha conferido al ser su hijo.

A mi hermano por siempre darme fuerza para lograr mis metas, eres una gran inspiración en mi vida.

A mis abuelitos, por su cariño y apoyo a todos mis sueños, por mostrarme que el único camino es siempre arriba y adelante.

A mi asesor de tesis el Dr. David Eduardo Pinto Avendaño, por toda la ayuda, paciencia y conocimientos brindados previo y durante la realización del presente proyecto de tesis.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1. Introducción

1.1 Resumen

El presente proyecto de tesis tiene como objetivo diseñar e implementar técnicas basadas en Inteligencia Artificial (IA) para proporcionar la capacidad de reconocer objetos en tiempo real a un robot humanoide que se encuentra en la Benemérita Universidad Autónoma de Puebla. En principio, se considera el uso de redes neuronales profundas (Deep Learning) para lograr tal propósito, sin embargo, hay una visión abierta al uso de cualquier técnica de IA que permita llevar a cabo la tarea planteada.

Para llevar a cabo el proyecto se definirán un conjunto de objetos básicos que se usarán en el proceso de entrenamiento y reconocimiento. Se realizará el proyecto en dos etapas, en la primera etapa se llevará a cabo el entrenamiento de modelos de aprendizaje, junto con la elaboración de sus respectivos conjuntos de entrenamiento y la segunda etapa consistirá en la realización de pruebas. En la etapa correspondiente al entrenamiento, se mostrarán al modelo imágenes que contienen los objetos previamente definidos con la finalidad de que aprenda los rasgos que definen cada objeto. Posteriormente, en la fase de pruebas, un usuario presentará una serie de objetos al robot, el cual deberá indicar "verbalmente" al usuario, mediante un sintetizador de voz, si reconoce o no dicho objeto, y en caso de reconocerlo, deberá expresar el nombre de dicho objeto.

Las pruebas deberán ser ejecutadas en tiempo real. Por esa razón, se propone utilizar un kit de desarrollo de hardware basado en la tecnología de NVIDIA llamado Jetson Nano, el cual cuenta con las prestaciones físicas necesarias para permitir la ejecución de múltiples redes neuronales en paralelo con la finalidad de ejecutar aplicaciones tales como clasificación de imágenes, detección de objetos y procesamiento de voz.

En el presente trabajo de tesis se generará un modelo para la detección de objetos utilizando YOLOv4. El proceso que se llevará a cabo requiere de la generación de un

conjunto de datos el cual será procesado para generar variantes y así entrenar distintos modelos con diferentes configuraciones del modelo YOLOv4, a su vez se realizarán pruebas a dichos modelos para evaluar y comparar su rendimiento.

1.2 Antecedentes del Proyecto

La visión por computadora ha sido un campo de estudio muy recurrente en el área de la robótica, ya que la capacidad de un robot para reconocer los objetos que componen el entorno que lo rodea juega un papel importante en la toma de decisiones, por lo que a lo largo de los años se han implementado diversas soluciones destinadas a dicho propósito, entre ellas destaca el uso de Deep Learning.

Dentro del área de Deep Learning destaca el uso de redes neuronales convolucionales las cuales tuvieron un hitó en 2012 con el lanzamiento del modelo Alexnet (Krizhevsky et al., 2012), a partir del éxito obtenido con el trabajo de Krizhevsky et al. (2012), la comunidad empezó a centrar su atención dentro de las redes neuronales convolucionales para la clasificación de objetos, dando lugar a modelos más exactos que con el incremento al acceso a GPU con mayor potencia computacional, se permitió integrar soluciones basadas en Deep Learning enfocadas en la detección de objetos a diversas plataformas desde robots, hasta dispositivos móviles.

Este proyecto esta precedido por un desarrollo de hardware enfocado en un rostro humanoide en el cual se han implementado articulaciones (brazos y piernas) y un torso. El proyecto se ha venido desarrollando desde hace un par de años en la Dirección de Innovación y Transferencia de Conocimiento de la Benemérita Universidad Autónoma de Puebla, con el objetivo de contar con un robot humanoide articulado que tenga capacidades de inteligencia artificial.

El robot actualmente tiene implementados un total 23 servomotores: 18 motores en el cuerpo, 2 en el cuello y 3 en el rostro. Este robot actualmente tiene capacidad de mostrar expresiones faciales, mover la cabeza con dos grados de libertad y mover las articulaciones.

Sin embargo, los movimientos y expresiones son hasta el momento actuadores básicos que necesitan un proceso de uso y es por esta razón que se necesita implementar métodos computacionales basados en inteligencia artificial para llevar a cabo actividades de aprendizaje y reconocimiento en el mundo real.

1.3 Objetivo General

Diseñar e implementar en un robot humanoide métodos basados en inteligencia artificial para el reconocimiento de objetos en tiempo real.

1.3.1 Objetivos Específicos

- Identificar un conjunto discreto de objetos a ser identificados en tiempo real por el robot humanoide.
- Ubicar y evaluar herramientas exitosas para el reconocimiento de objetos en tiempo real y que empleen métodos y técnicas basadas en inteligencia artificial.
- Implementar técnicas para el reconocimiento de objetos en tiempo real y que puedan ser usadas por un robot humanoide.
- Evaluar el rendimiento de las técnicas implementadas.

1.4 Contribución de la Tesis

El proyecto es de desarrollo tecnológico y su principal aportación es contribuir a que un robot humanoide tenga la característica de reconocimiento de objetos en tiempo real.

Capítulo 2. Marco Teórico

En el siguiente capítulo se presentan las áreas y conceptos que engloban el presente trabajo de tesis, se empezará con una introducción a las áreas junto a sus implementaciones y estructuras derivadas que permiten la ejecución de detección de objetos, una tarea que será descrita para luego presentar los correspondientes modelos que la implementan, para continuar con las métricas utilizadas para evaluar el rendimiento de los modelos una vez ejecutados, por último se contará con una descripción a las herramientas a utilizar durante el proyecto y que permitirán la culminación de este.

2.1 Machine Learning

Es una subárea perteneciente al campo de la Inteligencia Artificial, el término se le atribuye a Arthur Samuel (Samuel, 1959) quien lo aborda desde una perspectiva en la cual se programan computadoras digitales para que se comporten de manera inteligente, involucrando el proceso de aprendizaje.

Machine Learning (ML), es uno de los campos que actualmente tienen mayor relevancia dentro de la Inteligencia Artificial, principalmente por el gran impacto al ser utilizado dentro de diversas tareas como lo son reconocimiento de imágenes, sistemas de recomendación, sistemas texto y de voz, por mencionar algunos, todo a través de computadoras, dispositivos móviles, la web e incluso dentro de automóviles.

Dentro del campo, existen diversos enfoques para interpretar los datos y ajustarlos al problema a resolver, para ello, existen clasificaciones de los algoritmos, de las cuales destacan los algoritmos de tipo supervisado y no supervisado.

2.1.1 Aprendizaje Supervisado

Los modelos de esta línea son entrenados con un conjunto de datos acompañados con sus respectivas etiquetas (resultados/respuestas esperados) con el propósito de que el modelo

aprenda a identificar las asociaciones entre las características de cada uno de los datos pertenecientes al conjunto de datos y su respectivas etiquetas, para que una vez terminado de entrenar, al presentarle nuevo dato de entrada pueda devolver la etiqueta correspondiente (Ekman, 2021; Rebala et al., 2019).

2.1.2 Aprendizaje No Supervisado

Para este tipo de aprendizaje a los modelos se les presentan un conjunto de datos sin ninguna etiqueta, este tipo de modelos se caracteriza por analizar los datos y encontrar patrones dentro de los mismos para agruparlos (Ekman, 2021; Rebala et al., 2019).

2.2 Deep Learning

Es un campo perteneciente a Machine Learning, a manera de una definición un poco abstracta, Ekman realiza una definición de Deep Learning de la siguiente manera: "Una clase de algoritmos de machine learning que utilizan múltiples capas de unidades computacionales en donde cada capa aprende su propia representación de los datos de entrada. Estas representaciones son combinadas por capas posteriores de manera jerárquica" (2021, p. xxvii).

Las capas que contienen las representaciones de los datos casi siempre son aprendidas por modelos llamados redes neuronales, mismas que al ser compuestas por múltiples capas, agregan profundidad a la red, de ahí el "Deep" (Profundo) en Deep Learning (Chollet, 2018).

2.2.1 Red Neuronal Artificial

La idea detrás del cómo aprenden las maquinas a través de las redes neuronales, toma como inspiración la manera en la que las neuronas en el cerebro se componen. La unidad fundamental para la generación de redes neuronales artificiales son las neuronas artificiales, las cuales tiene sus orígenes a través del perceptrón, el cual Ekman (2021) menciona es un tipo de neurona artificial.

Ekman (2021) describe al perceptrón como una unidad computacional, que se compone de datos de entrada x, que abarcan desde 1 hasta n, siendo n el total de datos, a su vez cada dato tiene un peso de entrada identificado como w, también incluye una entrada conocida como bias (sesgo), finalmente cuenta con una salida a la cual identificaremos como

y, la estructura descrita se puede observar en la Figura 1. Los datos de entrada x, junto con sus respectivos pesos w son procesados por el perceptrón a través de una sumatoria ponderada z que será procesada por una función de activación $f(z)$, que permitirá obtener un resultado y.

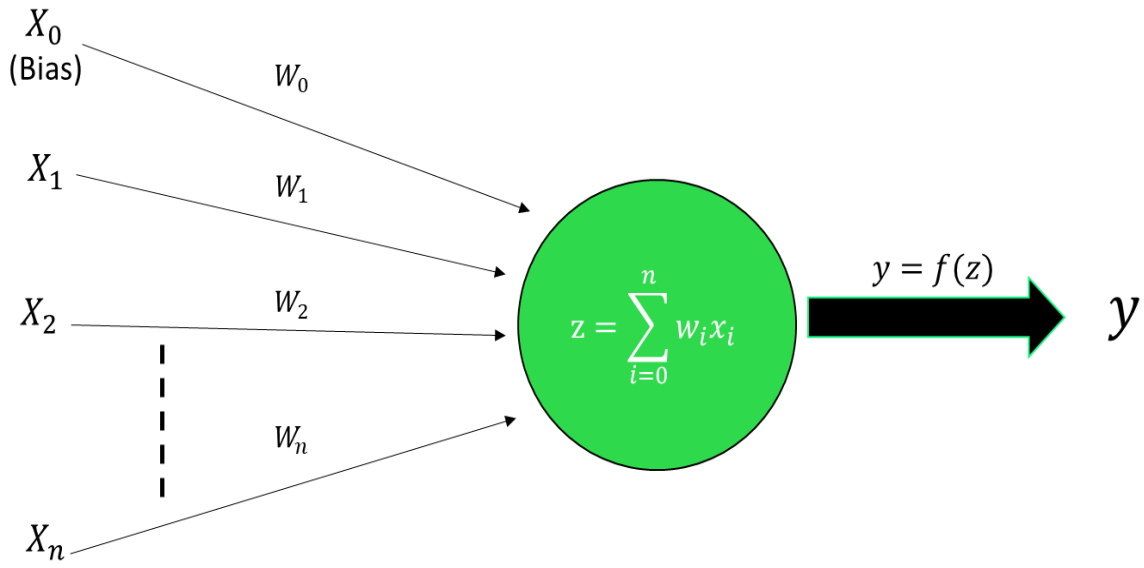

Figura 1: Estructura de un perceptrón.

2.2.2 Redes Neuronales Convolucionales

Maiettini (2020) y Zhang et al. (2021) menciona que fueron inicialmente utilizadas en el trabajo de LeCun et al. (1989). Con los años, las aplicaciones de las redes neuronales convolucionales en su mayoría son relacionadas al campo de la visión por computadora gracias a que se popularizó su uso a través de Deep Learning tras el éxito de AlexNet por Krizhevsky et al. (2012), pero el continuo avance dentro del área de Deep Learning, permitió que fueran usados en otras tareas. Una de las ventajas que trajo la llegada de las redes neuronales convolucionales o *convnets*, como también se les conoce, de acuerdo con Ekman (2021) es la introducción de la propiedad de translación de invariancia, misma que permite detectar objetos sin importar su ubicación en la imagen.

La topología de una red neuronal convolucional es diferente, según Ekman (2021) las capas se componen a través de tensores de 3 dimensiones (largo, alto, canales o mapa de características). En el caso de una imagen a color, al entrar a la red convolucional la dimensión de canales corresponde a 3, que representan los tres canales RGB de una imagen, mismos que serán procesados en la capa de entrada de la red neuronal convolucional (Figura 2).

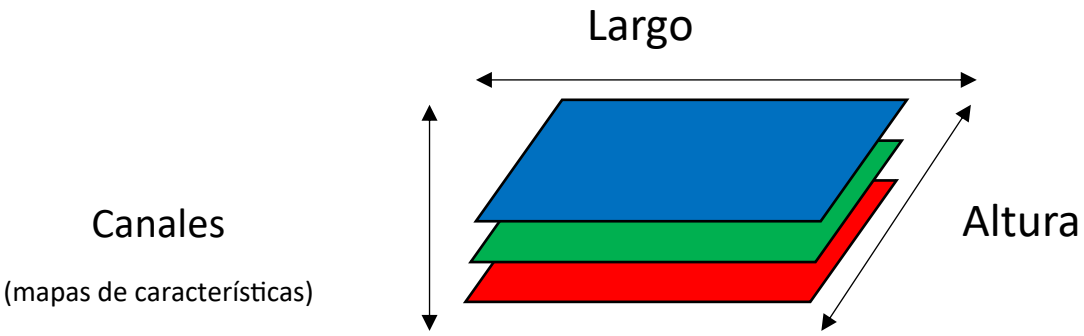

Figura 2: Representación de capa convolucional de entrada.

Las redes convolucionales aprenden rasgos principales de los objetos como resultado de la operación de kernel convolucional (Ekman, 2021), en donde realizan una operación de punto producto entre el kernel (una matriz bidimensional donde se despliegan pesos) y una sección de las mismas dimensiones, el cual es obtenido al recorrer el kernel definido a lo largo de la imagen pixel por pixel. Al terminar la operación, el resultando es un nuevo tensor 3D el cual es mejor conocido como mapa de características en donde la red ha empezado a identificar rasgos de la imagen. El tensor 3D devuelto tiene menor largo y altura, pero mayor número de filtros, cada uno contiene un rasgo que la red va abstrayendo del objeto a tratar.

Dentro de las capas que componen una red neuronal convolucional, se encuentran capas de agrupamiento (*pooling*), mismas que de acuerdo con Maiettini (2020) aparecen entre varias capas convolucionales seguidas y las cuales Chollet (2018) menciona que generalmente se utilizan a través de una operación de agrupamiento máximo (*max pooling*), reduciendo aún más el tamaño de la imagen (largo y alto) (Rebala et al., 2019). Finalmente, luego de procesar las diversas capas convolucionales y *pooling* en la red neuronal convolucional, se utiliza como describe Ekman (2021) una red neuronal completamente

conectada, con la intención de obtener un resultado en base a lo procesado anteriormente por la red convolucional.

2.3 Detección de Objetos

La detección de objetos es una tarea de la disciplina de la visión por computadora (Zhang et al., 2021, p. 612), en la que se combinan dos tareas, la de clasificación de imágenes y la de localización de imágenes (Ekman, 2021). Con lo cual se nos permite obtener qué instancias de clases se encuentran en una imagen, con su respectiva ubicación a través de cuadros delimitadores. La tarea es una oportunidad para diversas aplicaciones, ya que, en el caso de la robótica, permite que los robots interactúen con su entorno dependiendo de las tareas a las que fue asignado.

2.3.1 Modelos Enfocados a la Detección de Objetos

En el campo de Deep Learning han surgido bastantes modelos enfocados en la clasificación y localización de objetos en tiempo real, que previamente eran tratados a través de las ahora técnicas tradicionales, debido a lo computacionalmente costosas que resultaban ser, dichas propuestas dejaron de ser consideradas. El resurgimiento de Deep Learning es comúnmente datado de acuerdo con Ekman (2021) en 2012 cuando el trabajo realizado en el modelo de Alexnet (Krizhevsky et al., 2012), superó a otros modelos durante la competencia de ImageNet Large Scale Visual Recognition Challenge (ILSVRC). A partir de ahí la comunidad científica empezó a desarrollar mejores soluciones con el uso de Deep Learning, gracias al uso de potentes equipos computacionales, así como también describe Ekman (2021) el acceso a inmensas bases de datos y mejores unidades de procesamiento gráfico (GPU).

Posteriormente y como resultado del éxito obtenido con Alexnet (Krizhevsky et al., 2012), las Redes Neurales Convolucionales (CNN) fueron ampliamente integradas para la resolución de diversos problemas al utilizar Deep Learning, en el caso de visión por computadora, surgen dos corrientes: detectores de una y dos etapas, siendo la familia de algoritmos YOLO (Bochkovskiy et al., 2020 ; Redmon et al., 2016; Redmon & Farhadi, 2017; Redmon & Farhadi, 2018) los más representativos de la primera y de la segunda los algoritmos basados en R-CNN (Girshick et al., 2014).

2.3.2 YOLO: You Only Look Once

Es un modelo de una etapa realizado por Redmon et al. (2016) para la detección de objetos en tiempo real (hasta 45 FPS), el cual introduce el modelo como un problema de regresión al predecir directamente los pixeles de la imagen como objectos a la vez que sus correspondientes cuadros delimitadores.

Cuando una imagen entra en la red, **YOLO** la divide en una cuadrícula de $S\ x\ S$, en donde cada celda predice múltiples cuadros delimitadores *(Bounding Boxes)* donde cree que existe un objeto añadiéndole un porcentaje de certidumbre a cada una. Una vez obtenidas todas las predicciones se procede a eliminar los *Bounding Boxes* que tengan un nivel de certidumbre menor. Una vez que se eliminan la mayoría de los *Bounding Boxes*, si se detectan duplicados en la detección de un mismo objeto, se prosigue a utilizar *Non-maximum supression* (NMS), con lo que finalmente se obtiene una imagen en la que se ha realizado clasificación y localización de objetos. Cabe mencionar que el modelo destaca por la rapidez con la que identifica objetos, aunque esto lo lleve a sacrificar un poco la exactitud, especialmente en objetos de menor tamaño. Esto y otras mejoras fueron implementadas en iteraciones posteriores del modelo YOLO (Bochkovskiy et al., 2020; Redmon & Farhadi, 2017; Redmon & Farhadi, 2018).

2.3.3 YOLOv2

Posteriormente Redmon & Farhadi (2017) introduce la siguiente iteración del algoritmo, **YOLOv2** el cual utiliza como base **Darknet-19** el cual contiene 19 capas convolucionales y 5 capas para *max pooling*, dentro de las novedades del modelo destaca el uso de *batch normalization*, lo anterior permitió a **YOLOv2** tener mejores resultados al ser evaluado ante diversas competencias.

2.3.4 YOLOv3

En 2018 el modelo **YOLOv3** (Redmon & Farhadi, 2018) nuevamente mejora la exactitud del modelo y conservando la velocidad de inferencia. Dentro de los cambios destacados se encuentra la incorporación de **Darknet-53** en lugar de Darknet-19 por lo que ahora se cuenta 53 capas convolucionales, además que el modelo extrae características

utilizando un concepto similar a la de una red piramidal de características o por sus siglas en inglés **FPN** (Feature Pyramidal Network).

2.3.5 YOLOv4

Para **YOLOv4** (la versión a usar en el presente trabajo de tesis) Bochkovskiy et al. (2020) introdujeron diversas novedades a la familia YOLO, como lo son el uso de Bag of Specials y Bag of Freebies, así como la implementación de CSPDarknet53 como backbone, SPP y PAN, entre otros que permiten a YOLOv4

2.3.5.1 Bag of Freebies

Son métodos que cambian la estrategia del entrenamiento o el costo del entrenamiento, permitiendo que un modelo adquiera una mejor exactitud sin incrementar el costo de inferencia (Bochkovskiy et al., 2020), dentro de los métodos usados por YOLOv4 destaca *data augmentation* que en el contexto de detección de objetos sería el uso de técnicas de aumento de imágenes mediante copias adicionales de las imágenes presentadas, pero con variaciones que pueden ser una combinación de aumento de brillo con rotación de la imagen, entre otras.

2.3.5.2 Bag of Specials

Se trata de aquellos "métodos que únicamente incrementan el costo de inferencia en una pequeña cantidad, pero pueden mejorar significativamente la exactitud de un modelo" (Bochkovskiy et al., 2020, p. 4). SPP y PAN entran dentro de los métodos usados.

2.3.6 R-CNN

Creado por Girshick et al. (2014), destaca por utilizar regiones de propuestas (*region proposals*), las cuales de acuerdo con Girshick et al. (2014) 2000 son creadas a partir la imagen de entrada, dichas regiones según Ekman (2021) contienen posibles objetos. Posteriormente a través de una red neuronal convolucional se extraen las características de cada región propuesta (Zhang et al., 2021) en donde finalmente se devuelve un vector con 4096 elementos para ser introducido en una máquina de vectores de soporte que devuelve la categoría contenida en cada propuesta (Girshick et al., 2014). Luego de unos cuantos procesos más se logra obtener las detecciones necesarias.

En comparación con YOLO, R-CNN es mejor detectando objetos, pero al contrario que YOLO, es más lenta para realizar las inferencias, lo cual no es apto para ciertos escenarios de detección de objetos en tiempo real en donde el tiempo para toma de decisiones en base a eventos es crucial.

2.4 Métricas de Evaluación de Modelos

2.4.1 Matriz de confusión

Nos permite observar el rendimiento de un modelo al identificar correctamente o no, instancias de clases de objetos en una imagen, las cuales caen en las siguientes categorías:

- **Verdadero Positivo (TP):** sucede cuando el modelo detecta un objeto de manera correcta.
- **Falso Positivo (FP):** sucede cuando el modelo detecta de manera incorrecta un objeto.
- **Falso Negativo (FN):** sucede cuando el modelo no detectó un objeto que en realidad era positivo.

Comúnmente en modelos enfocados en clasificación, tienden a contar con el concepto de **Verdadero Negativo (TN)**, pero dentro de los modelos enfocados a la detección de objetos no es conveniente asumir dicho concepto, debido a las múltiples detecciones que implicaría dentro de una imagen.

A partir de las categorías mencionadas anteriormente se pueden obtener las siguientes métricas:

2.4.2 Sensibilidad (r)

La ecuación (1) muestra corresponde a la frecuencia con la que el modelo detecta todas las predicciones correctas de todo el conjunto de casos que eran verdaderos objetos por identificar.

$$\frac{TP}{TP + FN} \tag{1}$$

2.4.3 Precisión (p)

La precisión, representada en la ecuación (2), corresponde a la frecuencia con la que el modelo detecta todas las predicciones correctas.

$$\frac{TP}{TP + FP} \tag{2}$$

Para que un modelo pueda discernir entre un objeto detectado o no, requiere de un umbral conocido como *Intersection Over Union* (Intersección sobre unión).

2.4.4 F1-Score

Es la media armónica entre la precisión (p) y la sensibilidad (r).

$$2 * \frac{p * r}{p + r} \tag{3}$$

2.4.5 Intersection Over Union (IoU)

La intersección sobre unión (IoU) determina la similitud entre los cuadros delimitadores definidos durante la generación de los datos de entrenamiento (B_{gt}) y los cuadros delimitadores generados durante las predicciones (B_p), si la similitud supera el umbral definido, comúnmente del 50%, el modelo clasifica la detección como positiva. La IoU se obtiene de la siguiente manera:

$$IoU = \frac{\text{área de superposición}}{\text{área de unión}} = \frac{área(B_p \cap B_{gt})}{área(B_p \cup B_{gt})} \tag{4}$$

A manera gráfica la IoU se presenta como en la siguiente figura:

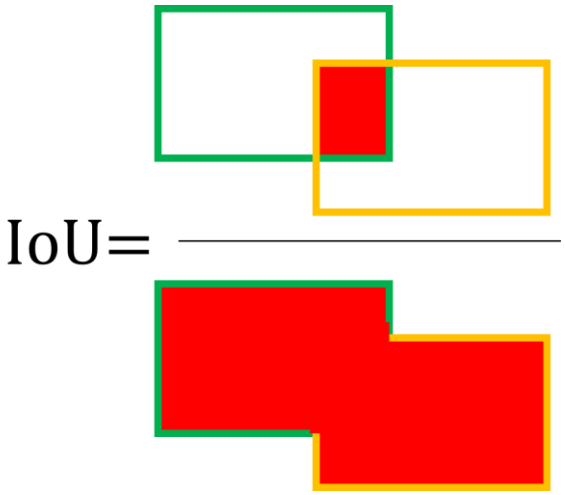

Figura 3: Representación gráfica de IoU.

2.4.6 Average Precision (AP)

Dentro de la literatura existen diversas maneras de calcular el AP, pero comúnmente como describe Maiettini (2020) es el área bajo la curva AUC por sus siglas en inglés de la curva de Precisión-Sensibilidad (*Precision-Recall Curve*).

$$AP = \int_0^1 p(r)dr \tag{5}$$

Dado que la curva de Precisión-Sensibilidad gráficamente no es monótonamente decreciente, varias competiciones dentro de la detección de objetos utilizan diversos métodos para interpolar y obtener un resultado más fiable al obtener el AP a cada clase a detectar por el modelo, entre los métodos, el usado en la competencia de Pascal VOC previo a las ediciones 2010-2012 utiliza una interpolación de 11 puntos y que de acuerdo con Padilla et al. (2020) se puede obtener de la siguiente manera:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 0.9, 1\}} p_{interp}(r) \tag{6}$$

En donde:

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

Posteriormente, a partir de Pascal VOC 2010, Everingham et al. (2015) mencionan que se decidió cambiar la manera en la que se calculaba el AP de 11 puntos para considerar todos los puntos. En el caso del presente trabajo de tesis ambos pueden ser utilizados, pero se optó por utilizar el método que involucra todos los puntos.

Lo anterior es realizado por cada clase, para así calcular el promedio de todas las clases a través de mAP.

2.4.7 Mean Average Precision (mAP)

Es la media de todas las AP de cada clase a detectar por el modelo, para obtener el mAP, se requiere utilizar la siguiente fórmula:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{7}$$

Donde ***N*** representa el total de clases a detectar por el modelo.

2.5 Frameworks enfocados al Aprendizaje Profundo

Durante la realización de modelos enfocados al uso de Machine Learning, existen diversas aproximaciones sobre la manera de implementar modelos, tradicionalmente se solían implementar directamente a través de lenguajes como Python o C++, si bien se tiene un mayor control de los procesos involucrados en el tratamiento de los datos, el anterior método suele ser extenso y propenso a errores durante su desarrollo, por lo que surge la necesidad de frameworks enfocados en Machine Learning permitiéndonos realizar éstas tareas sin incurrir en invertir tiempo en afinar los parámetros y funciones necesarias para el procesamiento de datos, logrando a su vez, que los desarrolladores se enfoquen más en el desarrollo e implementación del modelo.

Para la implementación de modelos por visión por computadora, particularmente el caso de YOLO, se utilizará principalmente el uso de frameworks como son Tensorflow y Darkent.

2.5.1 TensorFlow

TensorFlow (NVIDIA, 2022b) es una librería de software de código abierto para el cálculo numérico usando grafos de flujo de datos. Creado y actualmente bajo mantenimiento por Google, TensorFlow permite a los desarrolladores realizar aplicaciones basadas en Machine Learning. Las aplicaciones de TensorFlow como en NVIDIA (2022a) se menciona son diversas y abarcan proyectos referentes a, por decir algunos, voz, lenguaje, extracción de información, reconocimiento y clasificación de imágenes.

El Framework permite el desarrollo de modelos de Machine Learning en computadoras de escritorio, dispositivos móviles, dispositivos embebidos, la web y la nube, además cuenta con soporte para diversos lenguajes de programación, principalmente destacan el uso de Python, C++ y JavaScript, con lo que TensorFlow provee flexibilidad a los desarrolladores. Aunado a lo anterior, TensorFlow permite el uso de GPU, por lo que convierten a TensorFlow en uno de los Frameworks más populares y robustos.

Para la ejecución de modelos en dispositivos móviles y embebidos, existe TensorFlow Lite (Tensorflow, 2021), el cual es un conjunto de herramientas que permiten aceleración de hardware y optimización de modelos. Entre sus ventajas tenemos baja latencia y consumo de energía.

2.5.2 Darknet

Darknet (Redmon, 2016) es un framework de código abierto creado por Joseph Redmon, para el desarrollo de redes neuronales, escrito en C, es capaz de ejecutar y entrenar modelos utilizando el CPU, aunque se puede aumentar considerablemente el rendimiento de los modelos con ayuda de una GPU en conjunto del uso de la librería CUDA. Darknet permite el uso de librerías como OpenCV para aumentar la compatibilidad con formatos de imágenes. OpenCV permite visualizar imágenes y detecciones sin tener que recurrir a una copia en el sistema.

Si bien Darknet es un framwork capaz de entrenar y ejecutar distintos modelos de Machine Learning, destaca en el uso de modelos de detección de objetos, concretamente **YOLO** por Redmon et al. (2016), el cual fue creado para aprovechar las cualidades del

framework. Por lo que el uso de Darknet es muy común a lo largo de las distintas versiones de YOLO (Bochkovskiy et al., 2020; Redmon & Farhadi, 2017; Redmon & Farhadi, 2018).

2.6 Robot Operating System (ROS)

ROS según Open Robotics (2021) es un SDK (Software Development Kit) de código abierto usado principalmente para la creación de software en el campo de la robótica. Actualmente ROS en su versión 1, se encuentra disponible de manera oficial para sistemas basados en Unix, como lo son Mac OS X y distribuciones Linux. ROS fue creado de acuerdo con Open Robotics (2018b) con la intención de facilitar el reúso de código para la investigación y el desarrollo en la robótica, sin importar el lenguaje de programación, ya que ROS puede ser utilizado en lenguajes como Python, C++ y Lisp, también, aunque de manera experimental con lenguajes como Java y Lua. ROS opera de acuerdo con Open Robotics (2018b) como un marco distribuido de procesos (nodos), en base a una red peer-to-peer (p2p) de procesos ROS.

El grafo computacional de ROS consiste en una red peer-to-peer (p2p) de procesos de ROS que procesa datos de manera conjunta (Open Robotics, 2018a):

- **Nodos**: Son procesos que realizan cálculos computacionales, a manera de módulos, cada nodo puede comunicarse con otros nodos para realizar diversas acciones o mediciones asociadas al funcionamiento de un robot.

- **Maestro**: Se encarga del registro de nodos ante el sistema y de establecer la comunicación entre nodos.

- **Servidor de Parámetros**: Es un diccionario multivariable compartido, el cual los nodos usan para almacenar y recuperar datos, aunque generalmente destinados a ser datos para configuración. Se ejecuta dentro del Maestro.

- **Mensajes**: Son estructuras de datos simple que son de tipos primitivos (entero, punto flotante, booleano, etc.). Los nodos se comunican entre ellos a través de los mensajes que publican para los temas.

- **Temas**: A través de un sistema de transporte con semántica de publicación/suscripción, los mensajes llegan a los nodos con un nombre único, el cual permite a los nodos identificar el contenido del mensaje y así solicitar el envío o recepción de datos, de tal manera que ningún nodo conoce la existencia de otros nodos. Los temas son diseñados para una comunicación unidireccional.

- **Servicios:** Representan el envío de mensajes entre nodos a través de un modelo de solicitud/respuesta. Los nodos que ofrecen datos a través de un servicio bajo un nombre, envían respuestas a los nodos clientes que lo solicitaron.

- **Bolsas:** Son un mecanismo para almacenar y reproducir datos de mensajes ROS.

2.7 NVIDIA Jetson Nano Developer Kit

Es una computadora pequeña y potente capaz de ejecutar múltiples redes neuronales en paralelo. Como se puede observar en la Figura 4 la Jetson Nano cuenta con diversas opciones de entrada/salida (E/S) que permiten a los desarrolladores integrar la Jetson Nano a diversos proyectos, pero dentro de su configuración, lo que destaca de la Jetson Nano es la integración de un procesador ARM Cortex-A57 MPCore de 4 núcleos con una frecuencia máxima de 1.43 GHz, la inclusión de memoria LPDDR4 de 4 GB 64-bit y principalmente una GPU con arquitectura NVIDIA Maxwell de 128 núcleos, lo que para propósitos de la presente tesis permite integrar la jetso Nano al robot humanoide George y así ejecutar a través de ella el modelo de detección de objetos YOLOv4 a través de Darkenet y el uso de otras librerías para aprovechar la potencia de la GPU y no depender del CPU.

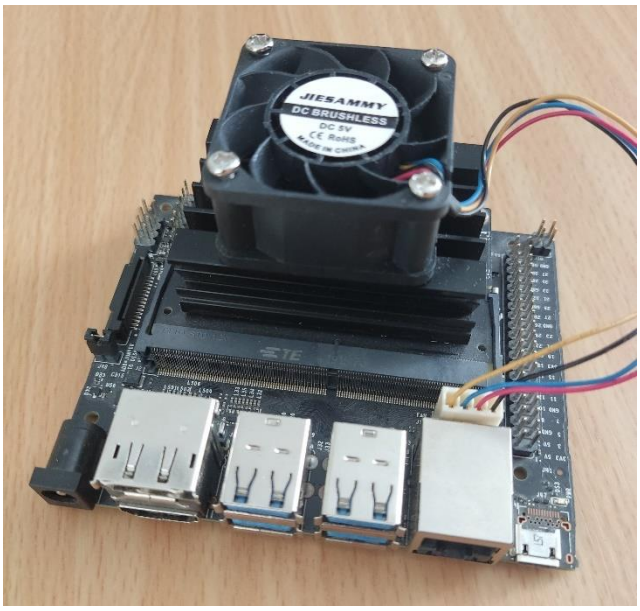

Figura 4: Jetson Nano a utilizar.

2.8 Robot Humanoide George

George es un robot con forma humanoide cuyas extremidades fueron realizadas con el uso de una impresora 3D, además George cuenta con 23 servomotores, de los cuales en el cuerpo (18) se tiene una combinación de servomotores ROBS-251 y servomotores Dynamixel AX-12A dispersos de la siguiente manera:

- 6 en cada pierna.
- 3 en cada brazo.

Finalmente 5 servomotores, 2 en el cuello y 3 en el rostro fueron integrados al robot humanoide. Los servomotores del cuerpo son alimentados con el uso del microcontrolador opencm904 y su expansión opencm 485 EXP permitiendo utilizar ROS en su versión Noetic para controlarlos.

En el presente proyecto se plantea utilizar la tarjeta embebida Jetson Nano para extender las funcionalidades que George presenta, en específico se plantea dotar al robot humanoide George (Figura 5) con la capacidad de detectar objetos en tiempo real y expresar a través de un sintetizador de voz, los resultados obtenidos por el modelo YOLOv4.

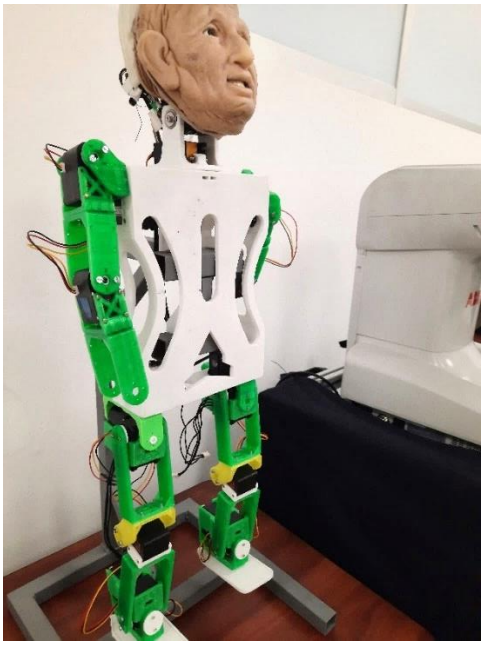

Figura 5: Robot Humanoide George.

Capítulo 3. Estado del Arte

En el campo de Aprendizaje Profundo/*Deep Learning* han surgido bastantes modelos enfocados en la clasificación y localización de objetos en tiempo real, que previamente eran tratados a través de las ahora técnicas tradicionales, debido a lo computacionalmente costosas que resultaban ser, por lo que las pocas propuestas basadas en *Deep Learning* dejaron de ser consideradas. El resurgimiento de *Deep Learning* es comúnmente datado en 2012 cuando el trabajo realizado en el modelo de Alexnet (Krizhevsky et al., 2012), superó a otros modelos y ganó la competencia de ImageNet Large Scale Visual Recognition Challenge (ILSVRC). A partir de ahí y gracias al fácil acceso a mejores unidades de procesamiento gráfico (GPU), así como a potentes equipos computacionales, permitieron a la comunidad científica desarrollar mejores soluciones con el uso de *Deep Learning*.

Posteriormente y como resultado del éxito obtenido con Alexnet (Krizhevsky et al., 2012), las Redes Neurales Convolucionales (CNN) fueron ampliamente integradas para la resolución de diversos problemas al utilizar *Deep Learning*, en el caso de visión por computadora, surgen diversas propuestas de modelos, entre los que destacan modelos como YOLO (You Only Look Once), SSD y Faster R-CNN, los cuales son utilizados dentro de diversas aplicaciones como se muestran en los presentes trabajos que se encuentra a lo largo de la literatura.

3.1 Aplicaciones de modelos de detección de Objetos

A medida que la robótica avanza, es innegable pensar en las interacciones que tendrán los robots humanoides con su entorno, para ello es necesario dotarlos con detección de objetos en tiempo real en la que se les permita tomar decisiones en base a lo reconocido, de los acercamientos que se encuentran a través de la literatura, existen trabajos como el presentado por Chatterjee et al. (2020) en el que se describe una versión modificada de YOLOv3 aplicado en robots humanoides NAO para realizar en tiempo real detección, reconocimiento

y localización de objetos a través de una cámara usando equipos computacionales de bajo rendimiento, así mismo realizaron comparativas entre distintos algoritmos en los que buscaron un balance entre retención de información, tiempo de inferencia y alta precisión para detección y localización de objetos en tiempo real, con lo que finalmente basaron su decisión de utilizar YOLOv3. Utilizando un sistema embebido como lo es la NVIDIA Jetson Nano, extrajeron el video de una cámara con una resolución de 480p a 10 FPS. Como resultado del uso del modelo YOLOv3 con la red modificada, obtuvieron un robot capaz de realizar diversas acciones en base a los objetos presentados, todo con una precisión en mAP_{50} de 48.984 y en promedio un tiempo de inferencia de 17 ms.

Trabajos como el de Maiettini et al. (2017) se enfocan simultáneamente en la localización y reconocimiento de objetos usando un robot humanoide iCub, el cual, a través de un procedimiento interactivo, recolecta y etiqueta datos que serán muestras para el proceso de entrenamiento de modelos de *Deep Learning*. Con los datos adquiridos de manera autónoma se realiza un proceso de afinación a la red neuronal, en este caso, el modelo Faster R-CNN, mismo que se le modificó la red neuronal convolucional encargada de la extracción de características con dos opciones: La red ZF y la red VGG_CNN_M_1024. El recabado de objetos que lleva a cabo el iCub fue realizado en un entorno natural y semi controlado, en el cual un maestro humano utiliza una interfaz de voz para señalar y nombrar al iCub los objetos a clasificar y detectar con procedimientos de estimación y segmentación para diferenciar la profundidad. Una vez obtenidos los datos de manera automatizada, se prosiguió entrenar Faster R-CNN y lograron obtener un mAP de 0.72 usando ZF y 0.71 para VGG_CNN_M_1024 para los objetos detectados. Al mismo tiempo, obtuvieron un mAP de los cuadros delimitadores con 0.71 para ZF y de 0.69 para VGG_CNN_M_1024. Con lo que Maiettini et al. (2017) consideran el entrenamiento de modelos a través del etiquetado automático de objetos como una buena alternativa.

Como parte de los esfuerzos para habilitar a los robots humanoides de interactuar con el entorno, Tian et al. (2019) presentan un sistema para automatizar la manipulación de objetos con robots humanoides basados en microcontroladores, a través de un modelo modificado de YOLOv2 y con el uso de un sensor visual, obtienen dimensiones de un objeto (alto y ancho) para posteriormente procesarlas a través de un robot Inmoov, mismo que en base a lo

detectado usará un brazo robótico para alcanzar el objeto. En el trabajo de Sai et al. (2017) un robot humanoide utiliza una interfaz de voz a través del módulo EasyVR para recibir instrucciones de sus siguientes acciones, a su vez utiliza la librería OpenCV para reconocer rostros y objetos, lo cual expande el uso de algoritmos para la detección de objetos para mayores propósitos. De la misma manera, en el trabajo realizado por Condés & Cañas (2019) la interacción humano-robot se da a través de comandos por voz para el seguimiento de personas. Usando un modelo SSD preentrenado con el framework Tensorflow, en conjunto de FaceNet CNN y un detector de rostros, pudieron habilitar el seguimiento de personas sin riesgo de distraerse con otras personas alrededor, en un robot TurtleBot2. Por otro lado, Jeve et al. (2019) propone un sistema que recibe audio y en base a la instrucción, sigue objetos tomando en cuenta su color.

Otra de las aplicaciones de visión por computadora es en competencias como RoboCup, en donde diversos equipos han realizado propuestas para mejorar el desempeño de robots humanoides para jugar fútbol. En el trabajo de Szemenyei & Estivill-Castro (2019) se presenta ROBO, un modelo basado en el modelo Tiny YOLOv3. ROBO permite obtener mejores resultados durante la detección de objetos relevantes en escenarios de fútbol para robots en tiempo real usando un robot NAO.

Hasta ahora se ha visto que los trabajos realizados para la tarea de detección de objetos son del tipo supervisados, esto quiere decir que previamente los objetos a detectar fueron catalogados manualmente por un humano para que, al momento de entrenar el modelo, se pueda detectar nuevos objetos que guardan las mismas características que los usados durante el entrenamiento. Por lo que existen trabajos que usan una mezcla de técnicas no supervisadas y supervisadas para dotar a robots humanoides con la capacidad de identificar objetos con aproximaciones al aterrizaje simbólico.

En la adquisición del lenguaje existen diversos enfoques y estudios sobre cómo las letras y palabras dejan de ser símbolos y se le añade su significado, esto es un interrogante mejor conocido como el problema del aterrizaje simbólico. El habilitar la manera en la que un humano aprende (no supervisada) aplicado en robots humanoides, es el objetivo de Štepánová et al. (2018). Para ello propusieron la adquisición de lenguaje implementando una arquitectura cognitiva jerárquica a través del procesamiento del lenguaje y de la visión.

El proceso de detección de objetos en el trabajo expuesto por Štepánová et al. (2018) se tuvo que enfocar en 3 capas, la primera corresponde a la capa del lenguaje en la que el robot recibe la descripción hablada de la imagen por un humano, obteniendo así el color, tamaño y forma. En la capa visual se trata de obtener las mismas características de los objetos para ser procesados a través de un modelo de mezcla Gaussiana. Una vez obtenidos los clústeres en ambas capas, los datos se procesan en una última capa que permite asociar los datos de las anteriores capas para que el robot pueda describir los objetos sin necesidad de un humano como asistente.

3.2 Optimización de modelos en sistemas embebidos

Los modelos para la detección de objetos pueden ser ejecutados en equipos embebidos, pero si se requiere implementarlos en tiempo real, el rendimiento puede no ser óptimo. Por ello existen propuestas como la de Wang et al. (2020) en donde se utiliza un modelo basado en YOLOv3 para NVIDIA Jetson TX2, con lo que obtuvieron un mAP de 0.492 y reduciendo el tamaño del modelo en un 96.93%.

A su vez, Wu et al. (2021) buscan la mejora de rendimiento en la detección de objetos en sistemas embebidos al proponer un modelo al cual llamaron Embedded YOLO. Su propuesta se basa en YOLOv5s, modificándolo al introducir un módulo al que se refieren como DSC_CSP a manera de reemplazo para las capas intermedias del modelo. DSC_CSP reduce la complejidad computacional necesaria para usar el modelo, pero se pierde precisión. Para obtener un mejor rendimiento, implementaron una técnica de destilación de conocimiento, así como otras optimizaciones para el tratado de datos, lograron obtener un mAP de 0.59 y un modelo con tamaño de 12MB.

Mezzina & De Venuto (2022) proponen el framework MONOCULAR el cual fue diseñado para ser integrado a robots sociales. A través de un robot pepper, el framework propuesto busca a través de cámaras RGB y un sensor 3D, los objetos a manipular, definir las características de las estanterías en las que se encuentran los objetos y a su vez la implementación de rutinas para acercarse y manipular dichos objetos (Mezzina & De Venuto,

2022). El trabajo realizado por Mezzina & De Venuto (2022) plantea el uso de una versión del modelo YOLOv3 (Mini-YOLOv3) entrenado para detectar 4 empaques.

A su vez Barry et al. (2019) presenta el uso de xYOLO, un modelo basado en YOLOv3 en su versión Tiny, para ser utilizado en una Raspberry Pi 3 B, el modelo propuesto fue entrenado para detectar dos clases de objetos que corresponden a los objetos balón y portería. Los resultados del modelo propuesto por Barry et al. (2019) muestra que alcanza 9.66 FPS, lo que implica una mejora en la inferencia en comparación del uso de YOLOv3 en su versión Tiny (0.14 FPS), pero al enfocarse en la velocidad, Barry et al. (2019) reportan una disminución considerable de la exactitud del modelo.

Utilizando un robot humanoide NAO, Cruz et al. (2021) plantean un sistema que permita detectar las características de elementos clave dentro del campo al procesar imágenes en escala de grises a través del uso de Redes Neuronales Convolucionales CNNs.

Para el campo de la robótica los modelos de detección de objetos han sido muy beneficiosos, pero los avances en *Deep Learning* permitieron la introducción de las redes neuronales a diversos campos, extendiendo así el alcance de los modelos para ser utilizados como herramientas para el trabajo o incluso ser utilizados dentro de la vida cotidiana, como en el caso de áreas como la agricultura o en el uso de sistemas de vialidad.

Dentro del trabajo descrito por Parico & Ahamed (2021), destaca la realización de un estudio para producir un contador de peras en tiempo real utilizando YOLOv4, junto con sus variantes YOLOv4-tiny y YOLOv4-CSP, a su vez se hace el uso de Deep SORT como sistema de respaldo, con lo que los autores (Parico & Ahamed, 2021) buscan que el sistema para el conteo de peras se utilice en aplicaciones para teléfonos.

Otra de las aportaciones del uso de detectores de objetos para el beneficio de la agricultura es el presentado por Perez-Daniel et al. (2020). En el que se describe el uso de RetinaNet bajo diferentes configuraciones de la red, para ejecutar el entrenamiento de un modelo para determinar si las frutas se encuentran frescas o en estado de descomposición.

Otra de las aplicaciones dentro de la detección de objetos, se puede mostrar dentro de la industria automotriz, donde los recientes avances dotan a los automóviles con la capacidad de reconocer diversos elementos del entorno en el que se encuentren. En base a lo descrito

anteriormente, dentro de la literatura existen trabajos como el de Ibrahim et al. (2020), el cual describe un sistema de advertencia de colisiones en tiempo real enfocado a automóviles autónomos al utilizar YOLO para detectar autos en una Jetson TX2, dentro de las funciones del sistema presentado, el sistema cuenta con la capacidad de estimar la distancia de los vehículos, además implementan técnicas de rastreo para evaluar la velocidad límite para mantener seguro el vehículo.

Dentro del campo de los vehículos autónomos, Mobahi & Sadati (2020) por su parte, presentan el entrenamiento y la evaluación de un modelo que, dentro de su configuración, se basa en una combinación de SSD y FPN, para finalmente incorporar dicho modelo dentro de vehículos autónomos para la detección de objetos.

Como se puede observar a lo largo de la literatura, el uso de modelos basados en *Machine Learning* y en particular *Deep Learning* para el campo de visión por computadora, son cada vez más frecuentes dado lo eficientes y veloces que pueden llegar a ser los modelos, lo cual representa una gran oportunidad para el desarrollo de sistemas más complejos en la robótica, en el presente trabajo de tesis el uso del modelo YOLOv4 ayudará a dotar al robot humanoide George de las capacidades de reconocer un conjunto de objetos, por lo que en lo que resta del presente trabajo de tesis, se describirá el desarrollo del modelo.

Capítulo 4. Elaboración e Implementación del conjunto de datos

En el presente capítulo se exponen los pasos realizados previos al entrenamiento del modelo YOLOv4 desde el recopilado de datos hasta su preprocesamiento para finalmente obtener los conjuntos de imágenes usados para el entrenamiento y la validación del modelo. Todo ello a través del uso de scripts realizados en el lenguaje de programación Python y otras herramientas de software que permitieron la realización de dichos conjuntos.

4.1 Figuras para utilizar en el entrenamiento.

Como se mencionó previamente el objetivo del modelo será el de detectar en tiempo real un conjunto finito, el cual será formado por 3 figuras correspondientes a un cuadrado, triangulo y un círculo (Figura 6). Para ello es necesario generar un conjunto amplio de imágenes que permitirán al modelo identificar las figuras.

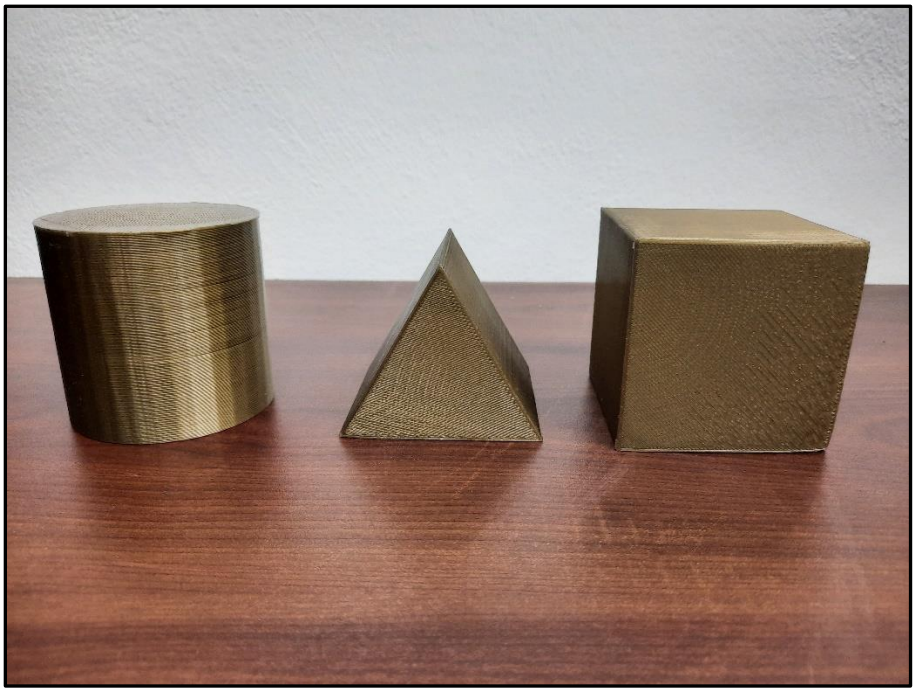

Figura 6: Objetos a utilizar para el entrenamiento.

4.2 Generación/Elaboración de Imágenes

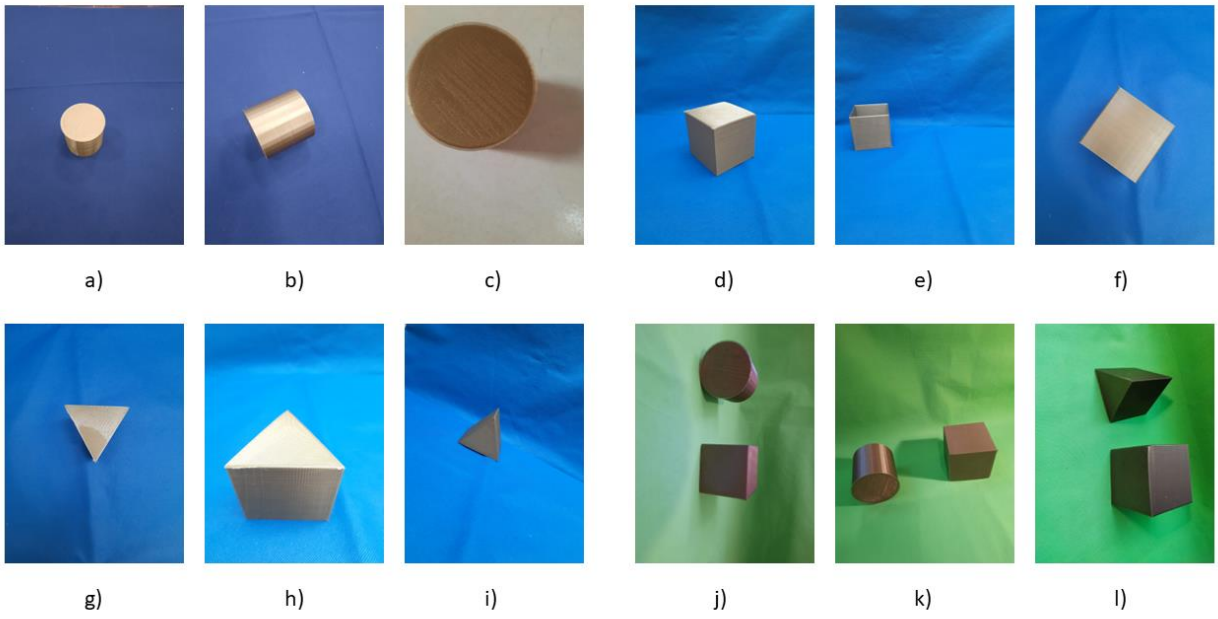

Figura 7: Ejemplos de Imágenes capturadas.

Para que el modelo pueda identificar el conjunto finito de figuras geométricas anteriormente mencionado, es necesario generar el conjunto inicial. Para ello se realizó la captura de 670 imágenes, mismas que contenían una o varias figuras a la vez, en diferentes fondos, ángulos y distancias como se muestra en la Figura 7, con el objetivo de recopilar la mayor cantidad posible de imágenes para que el modelo mejore la detección de objetos ante diversas condiciones.

Posteriormente se desarrolló un script usando Python con el que se disminuyó la resolución de las imágenes a 640x480 y se cambió la orientación de aquellas imágenes que se encontraban orientadas de manera vertical para que todas las imágenes estuvieran orientadas de manera horizontal.

A pesar de que YOLOv4 redimensiona las imágenes al momento de introducirlas a la red neuronal, la reducción del tamaño de imágenes permite agilizar el tratamiento de imágenes en pasos posteriores, así mismo el cambio de orientación de imágenes permitirá a la red que compone YOLO a aprender mejor de las imágenes ajustando un poco las imágenes al tamaño esperado dentro de la red sin distorsionar la imagen, de lo contrario imágenes como un cuadrado podrían deformarse y parecer rectángulos en vez de cuadrados, lo cual no es conveniente para el propósito del proyecto.

4.3 Etiquetado de Imágenes

Para que un modelo pueda aprender a identificar la ubicación de un objeto no basta con introducir una imagen con los objetos a detectar, también es necesario indicar los cuadros delimitadores. A través de la herramienta labelImg se colocan los cuadros delimitadores a cada una de las imágenes. En la Figura 8 se puede apreciar el resultado de dibujar el cuadro delimitador indicando la posición de este a través de labelImg, generando así un archivo bajo el mismo nombre de la imagen y con la extensión .txt que contiene la información de la etiqueta generada en el formato de anotaciones que soporta YOLO.

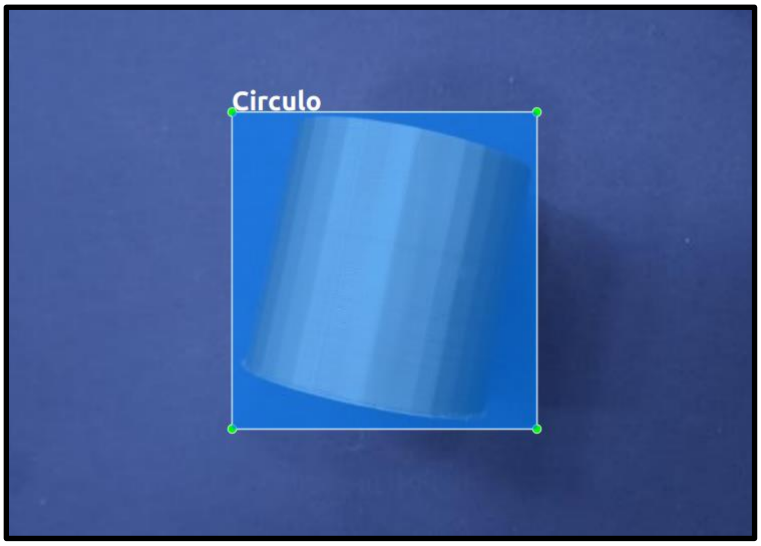

Figura 8: Objeto con cuadro delimitador aplicado.

El formato que YOLO utiliza para representar los cuadros delimitadores se compone como se puede observar en la Figura 9.

$$(ID\_clase, X\_centro, Y\_centro, Anchura, Altura)$$

Figura 9: Formato utilizado en archivos para indicar cuadros delimitadores.

Donde **ID_clase** es un identificador cuyo valor numérico es correspondiente a las clases definidas en el archivo de configuración obj.names. Tanto **X_centro** y **Y_centro** corresponde la coordenada del centro del cuadro delimitador. Finalmente se colocan la **altura** y **anchura** en pixeles de la imagen, en la Figura 10, se puede observar el archivo correspondiente a la imagen presentada en la Figura 8 tras salvar la anotación del cuadro delimitador.

```
2 0.514500 0.521833 0.275000 0.391667
```

Figura 10: Extracto de archivo con formato para cuadros delimitadores.

4.4 Técnicas de Generación de Imágenes

Previamente se generó el conjunto de datos inicial para el entrenamiento del algoritmo, pero la cantidad de dicho conjunto era aún insuficiente para poder entrenar de manera ideal el

algoritmo, por lo que fue necesario aumentar la cantidad de muestras que se usarán para el entrenamiento, una opción sería volver a recopilar nuevas imágenes con lo cual se podrían obtener las necesarias para tener un entrenamiento óptimo del modelo, pero existe la posibilidad de exceder la cantidad requerida o por el contrario el conjunto de datos se encuentre ante una aún falta de imágenes, independientemente del resultado a cada una de las nuevas imágenes se le tendría que etiquetar a mano, todo lo mencionado es un problema latente en el campo de visión por computadora por lo complejo y extenso que el procedimiento se vuelve. Afortunadamente existen técnicas de generación de imágenes como lo son cambio de fondo y transformaciones geométricas, mismas que fueron utilizadas durante la generación de nuevas imágenes para el conjunto de datos hasta ese momento del presente trabajo de tesis.

Antes de generar nuevas imágenes, primero se dividieron todas las imágenes en 4 carpetas, las 3 primeras carpetas agrupan imágenes correspondientes a la misma figura y tituladas de la misma manera (**triángulo**, **cuadrado** y **círculo**), dejándonos con una cuarta utilizada para almacenar aquellas imágenes que contenían más de una figura la cual sería identificada como **mix**, todo lo anterior fue realizado con la intención de tener una mejor perspectiva de la cantidad de imágenes representantes a cada figura.

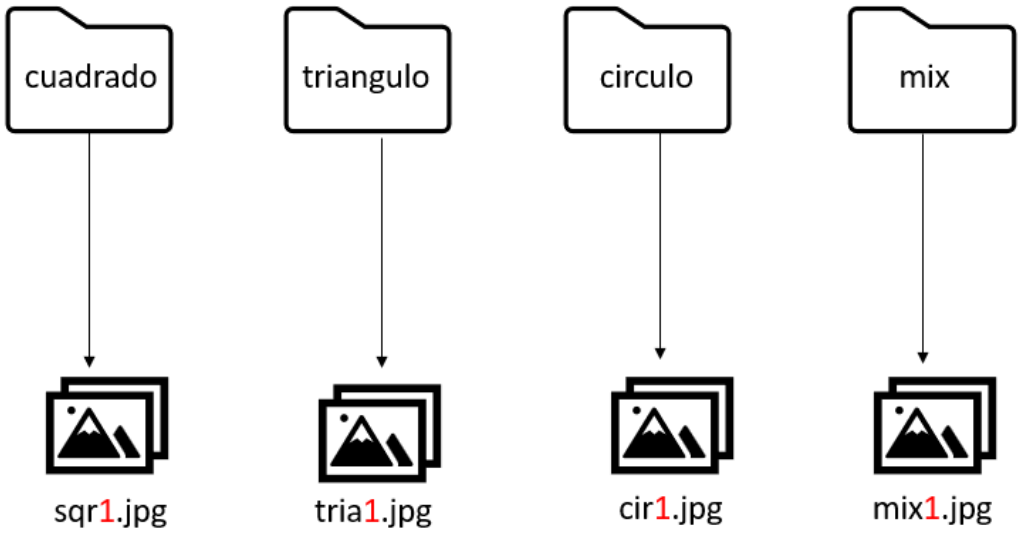

Figura 11: Estructura del almacenamiento y organización de las imágenes.

4.4.1 Renombrar Datos

Acto seguido, se prosiguió a renombrar todas las imágenes para identificar la figura contenida para facilitar tanto la manipulación de imágenes con el uso de código, como a la vez la posterior reagrupación que tendrán los conjuntos de imágenes, misma que se describirá más adelante cuando se generen los conjuntos finales para el entrenamiento.

El script desarrollado en Python, es modificado para renombrar las imágenes dependiendo de la figura a la que pertenecen, sqr para cuadrado, tria para triángulo, cir para círculo y mix para, como se mencionó previamente, identificar al conjunto de imágenes que albergan más de una figura. Para poder identificar dentro del mismo conjunto a cada imagen, al nombre le precede un identificador que abraca desde el 1 hasta n donde n representa el número total de imágenes en ese conjunto, con lo que finalmente terminamos obteniendo una estructura de archivos por el momento parecida a la que se muestra en la Figura 11.

4.4.2 Cambio de Fondo

Una vez obtenidas las etiquetas en cada foto y de tener una mejor estructura en la cual tener almacenados los datos, se aplicará la primera técnica para aumentar la cantidad de datos que se contaba, dicha técnica consistiría en utilizar todas aquellas imágenes que tuvieran un fondo azul y sustituir la región azul de las imágenes por alguna otra imagen dándonos el resultado que se muestra en la Figura 12 y logrando así duplicar la cantidad de imágenes que hasta ese entonces pertenecían a imágenes con fondo azul.

El contar con una copia en términos de posición de las figuras geométricas contenidas en las imágenes con fondo azul, podría suponer un riesgo de “overfitting”, pero el cambio de fondo ayudará a mitigar el riesgo si se introducen fondos que puedan agregar al modelo diferenciadores con lo que a su vez se evitará que el modelo realice asociaciones entre el fondo y las figuras geométricas presentadas en una imagen durante el entrenamiento.

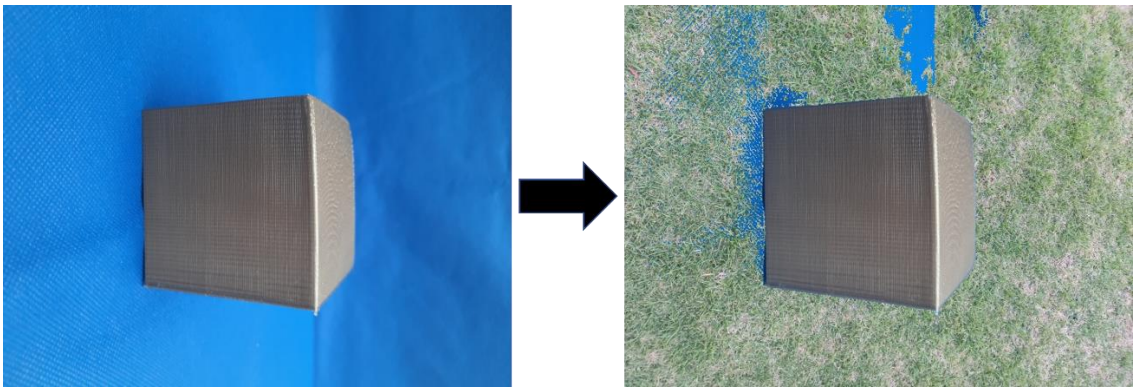

Figura 12: Resultado de aplicar un cambio de fondo a una imagen con fondo azul.

El proceso dentro del script que permite el cambio de fondo inicia con la captura de los limites superiores e inferiores de los valores RGB que más se aproximaran al color azul que poseen las imágenes por fondo y serían almacenadas con las variables su_blue y sl_blue (Figura 13), con la finalidad de que en un paso posterior eliminar dicho color de ser detectado por el programa. Por lo que para asegurar la extracción del fondo azul lo más posible se decidió aplicar la captura de un segundo rango de colores representantes a otras tonalidades del azul (tu_blue y tl_blue).

```
su_blue = np.array([62,135,255])
sl_blue = np.array([13, 85, 137])

tu_blue = np.array([110, 188,255])
tl_blue = np.array([10, 100, 169])
```

Figura 13: Variables dentro del script para almacenar las tonalidades de azul contenidas en las imágenes con fondo azul.

Una vez definidos los límites se procedería con la carga de cada una de las imágenes con fondo azul para sustituirlo con 1 de 3 las imágenes que de manera aleatoria se aplicaron a cada imagen para sustituir el fondo.

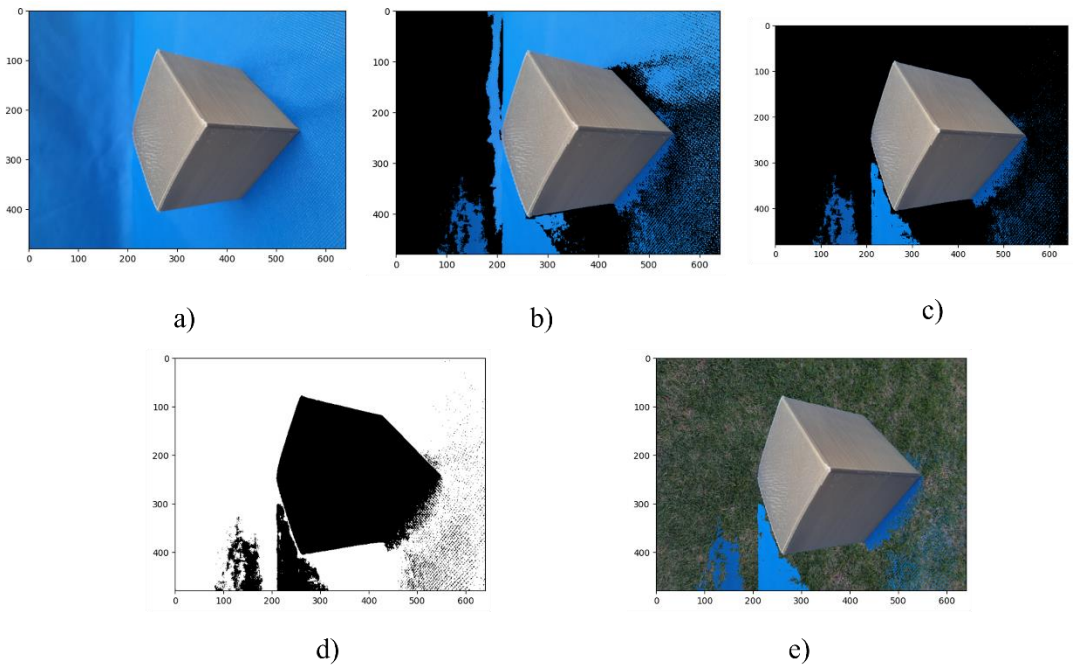

Figura 14: Proceso para generar nuevas imágenes.

Para entender el siguiente procedimiento aplicado a las imágenes se tiene que entender que el valor [0,0,0] en una imagen en el rango RGB, representa el color negro, mientras que el valor [255,255,255] representaría el color blanco. Posteriormente se aplicará el primer filtro, que creará una máscara usando la función “inRange” que devuelve un arreglo bidimensional con la misma dimensión de la imagen a) y que contiene 0 para aquellos pixeles que no eran pertenecientes al rango y 255 para aquellos que lo fueran. Por lo que el procedimiento utilizado anteriormente se aproximaría a la apariencia de la imagen d) de la Figura 14, ya que la máscara representada en d) es el resultado de aplicar el segundo filtro.

```
mask = cv2.inRange(video, sl_blue, su_blue)
plt.imshow(mask , cmap='gray')
```

Figura 15: Extracto de un código de programación para aplicar máscara.

Como se puede observar en b) aún quedan de color azul ciertas secciones tras aplicar el primer filtro, por lo que el segundo rango de colores se utilizaría con la función “inRange”, pero aplicado sobre la imagen b) resultando en c) y tras combinar las máscaras obtenidas en

los anteriores filtros, se obtiene d) y al aplicar combinaciones entre d), c) y la imagen de fondo deseada, se produce el resultado obtenido en e).

Ante la generación de nuevas imágenes previamente se comentó que el etiquetado de imágenes sería automatizado y a través de otro script en Python, se realiza la copia el archivo con extensión txt que contiene el etiquetado en el formato admitido por YOLO, a su vez se renombra el archivo copiado para igualar el nombre de la imagen cuyo fondo fue cambiado.

4.4.3 Transformaciones Geométricas

Hasta ahora se logró tener un conjunto de datos lo suficiente extenso para realizar un entrenamiento apropiado del modelo, aunque el conjunto inicial creado, contiene una ligeramente mayor aparición de figuras sobre otras dentro de las imágenes, por lo que a manera de experimentación se crearía un nuevo conjunto de datos bajo los mismos criterios del conjunto inicial que se tiene hasta el momento, pero con la diferencia de que se emplearon transformaciones geométricas a las imágenes. Dentro de las técnicas usadas, el proceso involucraría principalmente el uso de 2 transformaciones que consisten en voltear las imágenes; la primera voltearía las imágenes de manera horizontal; y la segunda las voltea de manera vertical.

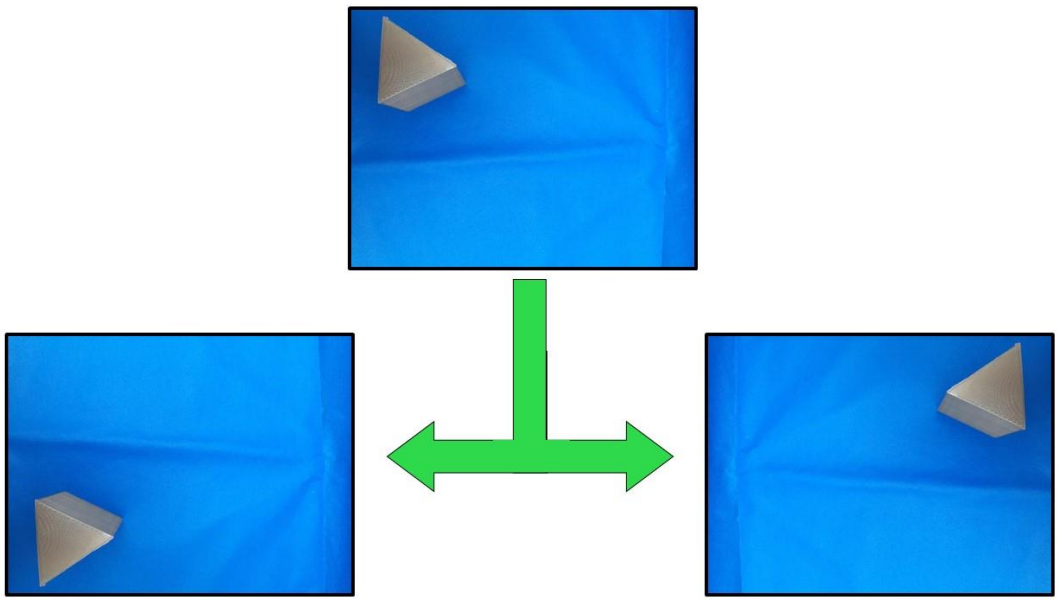

Figura 16: Ejemplo del uso de transformaciones geométricas en una imagen.

Durante el proceso se tuvo cuidado de ajustar los archivos que definen los cuadros delimitadores a través del uso de Python para evitar perder la ubicación de los objetos en las nuevas imágenes que derivaran de las originales al aplicarles transformaciones geométricas.

Con lo anterior se logró hasta en ciertos casos duplicar la cantidad de imágenes que representan cada figura geométrica, por lo que ahora que se contaba con una mayor cantidad de datos, se procedió a realizar una poda de datos para tener una cantidad equitativa entre cada figura y asegurar que el modelo aprenda a discernir entre figuras dentro del nuevo conjunto, cuyos datos serán agrupados y tratados de la misma manera que el conjunto inicial.

4.5 Creación de los sets de entrenamiento y de validación

Como parte del entrenamiento del modelo se requieren dos conjuntos de imágenes denominados de entrenamiento y de validación, el primero tiene la función de ajustar los pesos dentro de la red neuronal convolucional, logrando así que el modelo pueda generalizar los rasgos que hacen de un objeto parte de una clase a detectar, mientras que el conjunto de validación nos ayuda a evaluar el rendimiento del modelo así como a su vez, conocer cuando durante el entrenamiento se necesiten realizar ajustes tanto al modelo, como a los datos que forman al conjunto de entrenamiento.

Para el presente trabajo la creación de los conjuntos de entrenamiento y validación se realizó en 3 fases, la primera consistió en la creación temporal de los conjuntos de validación y entrenamiento aplicados de manera individual para cada carpeta (cuadrado, círculo, triangulo y mix).

Para ello, un script permitió dividir las imágenes de cada carpeta de manera aleatoria en los subconjuntos antes mencionados, en el caso de los conjuntos para el entrenamiento, el 80% fue utilizado para el entrenamiento y el 20% restante se usó para la validación. La razón por la cual se divide cada carpeta en subconjuntos y no realizar directamente el conjunto de validación y entrenamiento, es para obtener un 80% de imágenes de entrenamiento que verdaderamente represente a cada 80% de cada carpeta, logrando así tener conjuntos

mayormente balanceados en términos de apariciones de figuras para cada conjunto de entrenamiento y validación.

En la segunda fase se unificarían todos los subconjuntos creados en la fase anterior para formar los conjuntos deseados para el entrenamiento y la validación, lo anterior se representa mejor en la figura 17.

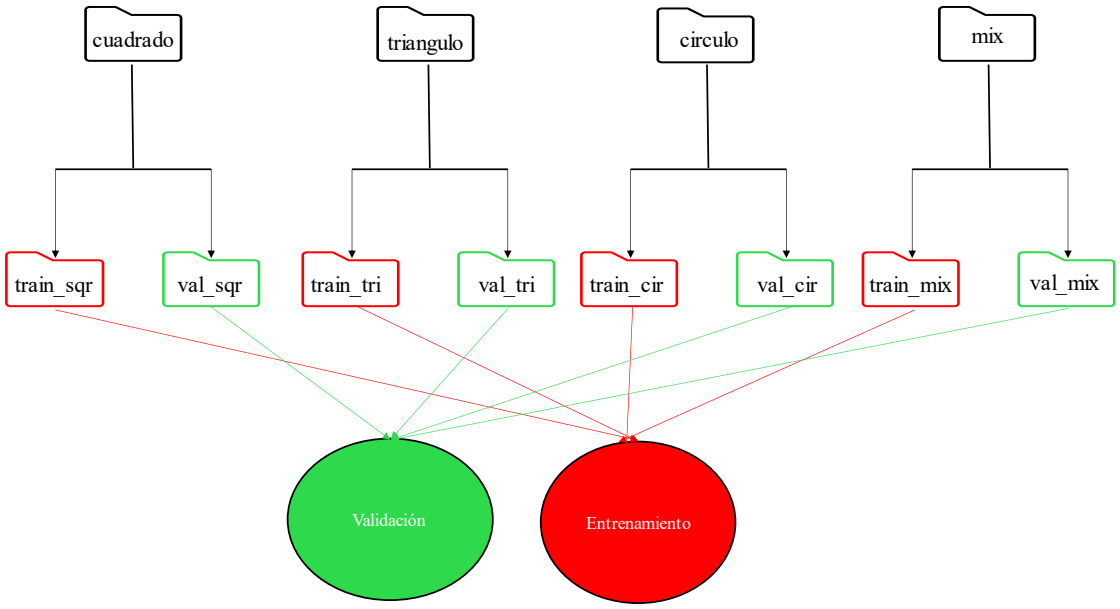

Figura 17: Estructura de conjuntos de validación y entrenamiento.

Se tiene que recordar que el contenido de la carpeta “mix” es de unas cuantas imágenes en las cuales la aparición en total de ciertas figuras dentro de la carpeta son de menor cantidad que otras, debido a ello es que la cantidad de imágenes en la carpeta “mix” es muy pequeña y no supone un gran riesgo entre las clases involucradas y se permite que el contenido de “mix” enriquezca el conjunto de datos deseado, de lo contrario, un aumento notable de la cantidad de iteraciones de determinadas figuras durante el entrenamiento del modelo, podría causar ciertas complicaciones, principalmente que el modelo aprenda mejor una clase que otras, por tal motivo, tanto en el conjunto inicial y en el conjunto que contiene imágenes con transformaciones geométricas, las imágenes pertenecientes a la carpeta “mix” no fueron aumentadas significativamente.

Por último, se generaron 2 archivos en formato txt (train.txt y valid.txt) uno de ellos representado en la figura 18, dentro de cada uno se encuentran listados la ruta y nombre de las imágenes que forman los respectivos conjuntos de validación y entrenamiento.

```
data/objNew/mix12.jpg
data/objNew/tri108.jpg
data/objNew/cir33.jpg
data/objNew/sqr17.jpg
data/objNew/mix44.jpg
data/objNew/tri142.jpg
data/objNew/cir121.jpg
data/objNew/sqr167.jpg
data/objNew/mix11.jpg
data/objNew/tri251.jpg
data/objNew/cir168.jpg
data/objNew/sqr328.jpg
data/objNew/mix48.jpg
data/objNew/tri185.jpg
data/objNew/cir100.jpg
data/objNew/sqr90.jpg
data/objNew/mix6.jpg
data/objNew/tri256.jpg
data/objNew/cir43.jpg
data/objNew/sqr117.jpg
data/objNew/mix35.jpg
data/objNew/tri164.jpg
data/objNew/cir104.jpg
data/objNew/sqr282.jpg
```

Figura 18: Contenido del archivo train.txt con rutas de imágenes para el entrenamiento.

Capítulo 5. Entrenamiento

En el siguiente capítulo se proseguirá a describir todos aquellos pasos seguidos para el entrenamiento del modelo, mismo que será utilizado por el robot humanoide George a través de la tarjeta embebida Jetson Nano. El entrenamiento se realizaría a través del repositorio oficial para YOLOv4 utilizando Darknet del repositorio en Github de AlexeyAB[1]. La razón por la cual se utiliza Darknet es que además de ser el framework bajo el cual las versiones anteriores de YOLO fueron creadas, existe una mayor documentación sobre las implementaciones de YOLOv4 bajo el uso del framework Darknet a comparación de las encontradas en otros frameworks como Tensorflow o Pytorch.

Para el entrenamiento se usará un equipo con sistema operativo Linux, 64 GB de RAM, una tarjeta gráfica NVIDIA GTX 1070 Ti con 8GB de VRAM y un procesador Intel i7-6700, requerimientos suficientes para la versión de Darknet previamente mencionada.

5.1 Compilando Darknet

Antes de utilizar Darknet, se debe de modificar diversos valores en el archivo Makefile que contiene Darknet para así compilarlo para su posterior uso, dichos cambios serían los siguientes:

- GPU = 1
- CUDNN= 1
- OPENCV=1
- ARCH = compute_53, code =sm_53 ()

[1] https://github.com/AlexeyAB/darknet.

Los anteriores cambios permiten a Darkent hacer uso de la GPU y de la librería OPENCV para tener un mejor rendimiento al momento de entrenar y probar el modelo, dichos ajustes serían usados de igual manera dentro de la Jetson Nano, por lo que se procedería a la compilación del framework en ambos equipos, con la única diferencia sería en el valor en ARCH, ya que la GPU integrada en la Jetson Nano utiliza otra arquitectura y se utilizaría para potenciar el rendimiento de la inferencia.

5.2 Configuración del modelo

Dado que el modelo requiere aproximadamente de H horas para finalizar el entrenamiento, se decidió entrenar YOLOv4 con el archivo ***yolov4.conv.137*** que se puede obtener del repositorio que incluye Darkent el cual consiste en los pesos pre-entrenados usados con el conjunto de datos MS COCO. Aunque las clases que forman el conjunto de COCO contienen rasgos de mayor complejidad, esa misma particularidad debería ayudar a que el modelo generalice los objetos a detectar y a su vez reducir el tiempo de entrenamiento requerido.

La manera en la que las capas del modelo YOLO interactúan entre ellas, entre otras características, es definida a través de un archivo .cfg, por lo que, para entrenar el modelo para la detección de figuras se utilizó el archivo incluido dentro del framework, el cual se encuentra como yolov4-custom.cfg, al cual se le realizó una copia (yolov4-objNF608.cfg) y a dicha copia se le modificaron ciertos parámetros. Al reutilizar el modelo se permite aprovechar de mejor manera durante el entrenamiento el archivo con los pesos pre-entenados que anteriormente se obtuvieron, ya que se ajustan a la red en la que fue entrenada originalmente.

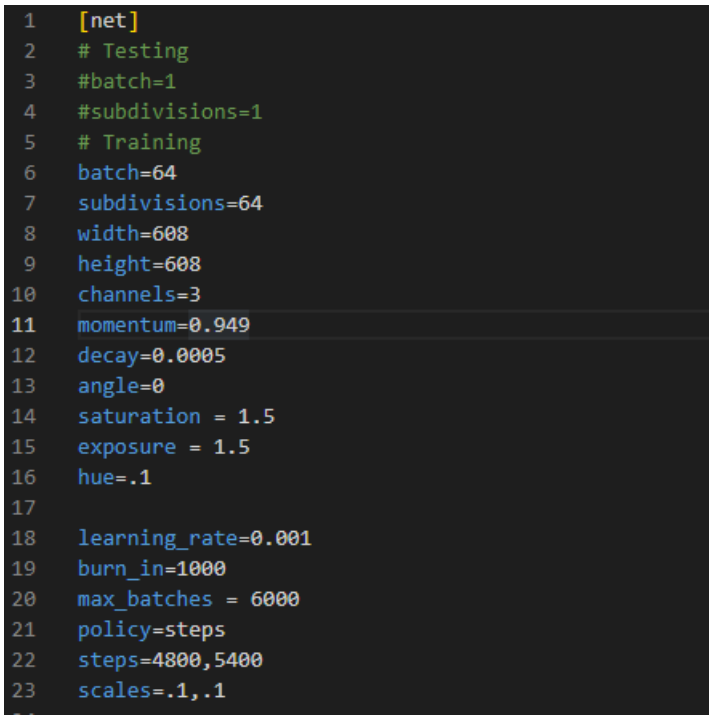

Figura 19: Extracto del archivo con extensión .cfg con la configuración de la red YOLOv4.

Dentro del archivo de configuración de la red, en el caso del parámetro batch, se entrenó con el valor de 64 y en el campo subdivisions dado a los errores que se tenían durante el entrenamiento, al final se le asignó un valor de 64, lo último a sugerencia de la documentación del framework encontrada en repositorio de Darkent. A su vez se configuró que el entrenamiento fuera con un tamaño de la red de 608 de ancho y 608 de alto y que se realizara durante 6000 iteraciones, valor que corresponde al campo max_batches de la Figura 19. Para calcular las iteraciones, en el repositorio de darknet se sugiere realizar el siguiente cálculo:

$$max\ \_batches = clases * 2000$$

Como se tiene pensada la detección de 3 clases, se tendría un valor de 600, lo cual cumple con lo mencionado en el repositorio de Darknet, donde max_batches no tiene que ser un número menor a las imágenes de entrenamiento y no menor que 6000. Además de los anteriores ajustes, otros fueron realizados al archivo .cfg (Figura 19), con los que se permitirían al framework entrenar de manera más adecuada YOLO, para obtener el modelo deseado.

Los conjuntos de entrenamiento y validación creados anteriormente serían reubicados dentro de los archivos de configuración del framework, en específico la carpeta **"data"** donde se almacenarán junto con los siguientes archivos a realizar.

Como parte de la configuración del modelo, es necesario especificar los elementos que se involucran en el entrenamiento, entre ellos se creó un archivo para especificar los nombres de las clases a detectar, uno por cada línea como se muestra en la Tabla 1.

Tabla 1: Clases de objetos a detectar.

clases
cuadrado
circulo
triangulo

Para que el framework conozca dónde se encuentran los archivos y datos necesarios, se creó un archivo para especificar la ruta del archivo previamente creado para los nombres de las clases, a su vez se listan las rutas de los archivos correspondientes a los conjuntos de entrenamiento y de validación que anteriormente se definieron, así como el número de clases que se espera para detectar (3 para el modelo) y la ruta de una carpeta backup, finalmente obteniendo un archivo como la Figura 20.

```
1   classes = 3
2   train   = data/train.txt
3   valid   = data/test.txt
4   names = data/obj.names
5   backup = backup/
```

Figura 20: Extracto de archivo de configuración con rutas.

El campo "**backup**" hace referencia a la carpeta en la que se almacenan copias de los pesos que durante el entrenamiento se van generando, todo ello con dos motivos, en el caso de que sucediera un error o un evento inesperado al momento de entrenar no se tuviera que iniciar desde el principio y simplemente se continuaría desde la copia deseada. La segunda

es para probar si hasta cierto punto del entrenamiento del modelo, se obtienen mejores resultados y optar por utilizar en el modelo final dichos pesos.

Una vez definidos y configurados todos los elementos que se usarán por el modelo, ahora se procederá con el entrenamiento.

5.3 Entrenamiento del Modelo YOLOv4

El entrenamiento, como anteriormente se mencionó, dotará al robot humanoide George con la capacidad de detectar las figuras geométricas, mismas que forman parte de los conjuntos de datos de entrenamiento y de validación. Para entrenar el modelo, se utilizará el comando dentro de una terminal Linux, el cual se encuentra representado en la siguiente Figura 21.

```
./darknet detector train <Archivo con rutas> <Archivo .cfg de la estructura de YOLOv4> yolov4.conv.137 -map
```

Figura 21: Comando para el entrenamiento del modelo.

El inicio del entrenamiento del modelo resultaría en la Figura 22, y seguido de aproximadamente 18 horas finalizó el entrenamiento, en las que gracias a la opción opción "-map" dentro del comando para entrenar, el framework generó un gráfico en el que se muestra el *mAP@50* y el *Loss* promedio, a su vez, se obtuvieron otros resultados, los cuales se discutirán en breve.

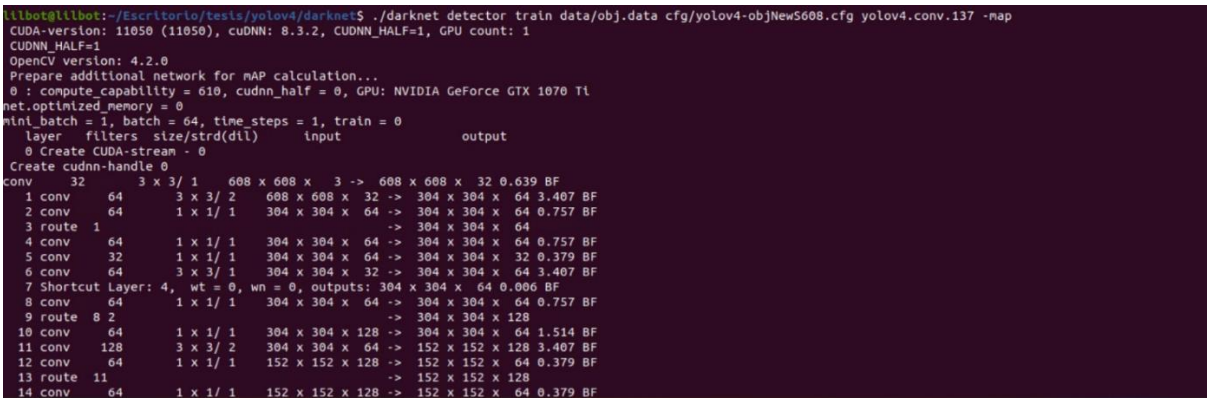

Figura 22: Captura inicio del entrenamiento del modelo.

Capítulo 6. Resultados

Una vez completado el entrenamiento del modelo, tras 6000 iteraciones finalmente se obtuvo el modelo que permitirá al robot humanoide "George" detectar en tiempo real los objetos propuestos anteriormente. Al término del entrenamiento el framework generó una gráfica con el *mAP@50* aplicado al test de validación y el *Loss* generado a lo largo de las iteraciones del entrenamiento, lo cual nos permite conocer en aspectos generales el rendimiento del modelo, pero es necesario la obtención de otras métricas, por lo que en lo que resta del presente capítulo se procederá a evaluar el modelo obtenido a través del framework utilizado durante el entrenamiento, logrando así generar las diversas métricas necesarias. A su vez se generarán diversas pruebas con el conjunto de validación para comparar el rendimiento del modelo tanto en el equipo computacional utilizado en el entrenamiento, como en la Jetson Nano y su integración con el robot humanoide "George".

Tabla 2: Resultados del modelo ante el conjunto de validación.

	YOLOv4
True Positive (**TP**)	178
False Positive (**FP**)	9
False Negative (**FN**)	3
Precisión	95.19%
Sensibilidad	98.34%
F1-Score	96.74%

Al utilizar el framework, se reportó diversas métricas las cuales se encuentran representadas en la Tabla 2 y Tabla 3 las cuales fueron obtenidas utilizando el conjunto de validación. Dentro de la Tabla 2 se puede apreciar el modelo cuenta con una precisión(p) de 95.19% y una sensibilidad(r) de 98.34%, los cuales son buenos resultados ya que idealmente

se busca tener ambos valores lo más alto posible. La sensibilidad obtenida significa que el modelo es ligeramente propenso a detectar de manera exitosa todos los objetos del mundo real como alguna de las figuras por lo que, de los objetos presentados, 3 fueron ignorados (FN) por el modelo, mientras que por el lado de la precisión el modelo errará ligeramente (FP) sobre la categoría en la que dichos objetos detectados y aquellos que creyó detectar pertenecen. Además, se obtuvo un F1-Score de 96.74% como se muestra en la Tabla 2, lo cual es un buen indicio de que el modelo se entrenó de manera correcta.

Como se puede observar en la Figura 23 el ***Loss*** obtenido a lo largo de las iteraciones que tomaron para entrenar el algoritmo y cuyo promedio fue de 0.3288, se basa en el conjunto de entrenamiento, además se puede observar que el modelo aprende a identificar de manera correcta los objetos entre las iteraciones 1200 y 1800, a partir de ahí la curva muestra que el modelo empieza a estabilizarse, lo cual se puede comprobar con el mAP@50 graficado con la línea roja, la cual utiliza el conjunto de validación y permite evaluar la precisión general del modelo para detectar las figuras.

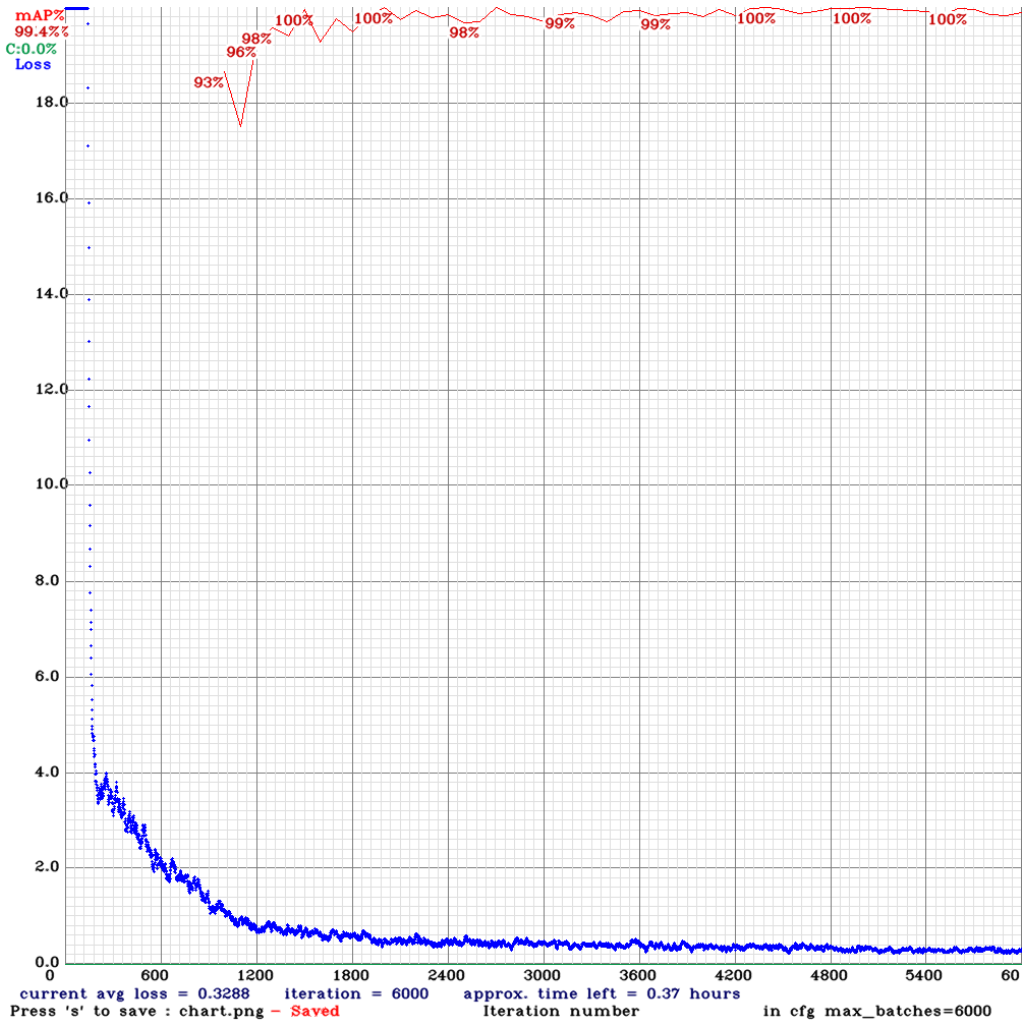

Figura 23: mAP@50 y Loss obtenidos durante el entrenamiento del modelo a través de 6000 iteraciones.

En la Tabla 3 se puede observar el mAP@50 obtenido en la iteración final de la Figura 23, la cual corresponde al valor de 99.40%, además se muestran en la Tabla 3 los valores de *Average Precision* (***AP***) de cada una de las tres clases con las que se entrenó el modelo. En el caso de la clase "círculo" el framework reportó un AP del 100%, para la clase "cuadrado" un 99.52% y para la clase "triángulo", un 98.68%. Lo cual es un buen indicio de que el modelo ha aprendido a reconocer los rasgos que componen un círculo, así como lo ha hecho con el cuadrado y el triángulo.

Tabla 3: mAP@50 y AP del modelo YOLOv4

Clase Detectada	**AP YOLOv4**
Triángulo	98.68%
Cuadrado	99.52%
Círculo	100%
mAP@50	99.40%

Para conocer si la cantidad y balance de las imágenes mejoraría el rendimiento del modelo, se entrenaron otros 2 modelos los cuales a deferencia del modelo entrenado hasta ahora, fueron entrenados usando el conjunto de datos al cual se le aplicó transformaciones geométricas (TG) como paso adicional a las imágenes generadas con técnicas de aumento aplicadas al conjunto original y como se explicó anteriormente tiene el propósito de tener un mayor control sobre el número de apariciones de cada imagen en el conjunto de datos, lo cual se puede observar en la siguiente gráfica de la Figura 24 donde se muestra la cantidad de imágenes aplicada a cada conjunto de datos.

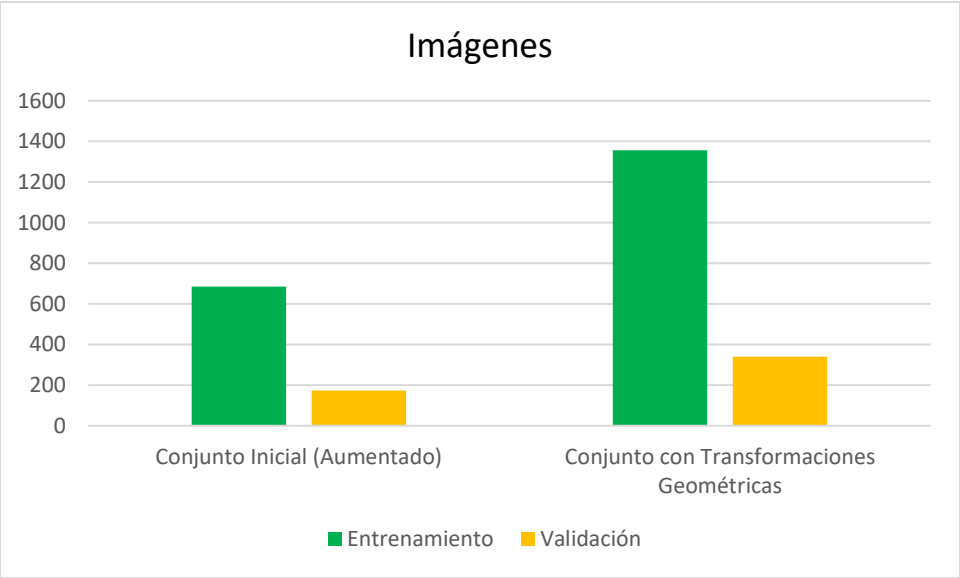

Figura 24: Comparativa de la cantidad de imágenes por conjunto.

Se puede observar que en la fFigura 24, la mayor cantidad que los nuevos modelos con aumento con transformación geométrica tienen tanto en los conjuntos de entrenamiento, como los de validación a comparación del modelo entrenado con aumento de imágenes sin transformaciones geométricas (TG).

Al mismo tiempo en la gráfica encontrada en la Figura 25 se muestra el balance otorgado a cada clase, exceptuando a las imágenes que contienen más de una figura como son las imágenes catalogadas como “mix”, donde no se aplicó ningún aumento con transformaciones geométricas (TG) para evitar perder el ligero balance entre clases.

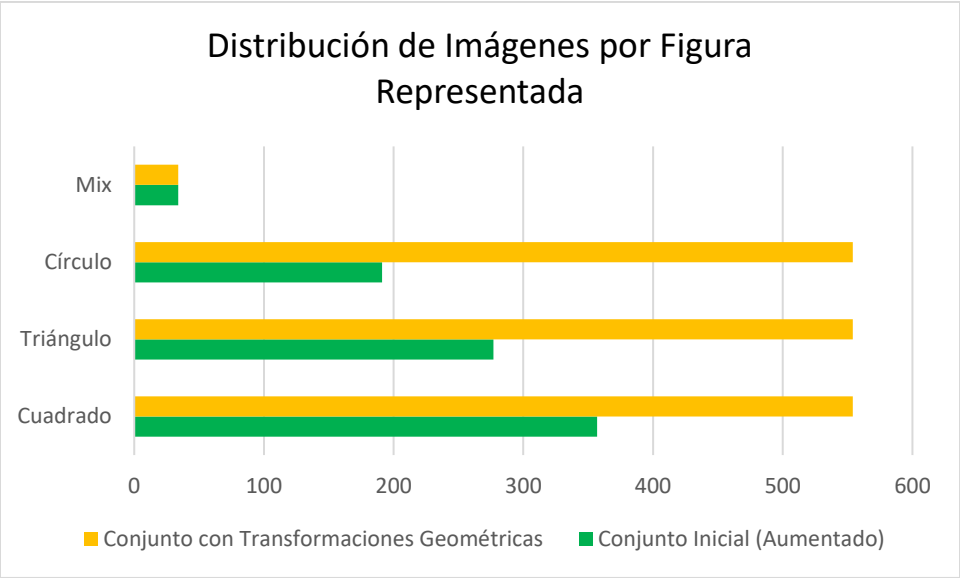

Figura 25: Distribución de imágenes por figura representada en los conjuntos con TG y sin TG.

Dentro de la estructura de la red en los nuevos modelos, el primero conserva la misma estructura que el modelo entrenado sin TG, mientras que el segundo cambia el tamaño de entrada de los datos de una resolución de 608x608 a una resolución de 416x416 para incrementar la los FPS durante la detección de objetos en la Jetson Nano. Tras aplicar las pruebas a los nuevos modelos junto con sus respectivos conjuntos de validación, se obtuvieron los resultados mostrados en la tabla Tabla 4.

Tabla 4: Resultados YOLOv4 con TG con distintas resoluciones de entrada.

	YOLOV4 con TG (608)	YOLOv4 con TG (416)
True Positive (**TP**)	345	346
False Positive (**FP**)	7	7
False Negative (**FN**)	1	0
Precisión	98.01%	98.02%
Sensibilidad	99.71%	100.00%
F1-Score	98.85%	99.00%
mAP@50	99.95%	99.98%

Comparando los resultados entre la tabla Tabla 4 y la Tabla 2 (yolov4 sin TG), el entrenamiento con una cantidad mayor de imágnenes parece haber mejorado la detección de

los objetos en todos los rubros en ambos modelos con TG y la disminicuión de resolución del modelo con TG de igual manera, aunque ligeramente aumentó el rendimiento del modelo a comparación del modelo con TG que mantuvo la resolución de 608x608.

6.1 Pruebas en Jetson Nano

Hasta ahora todas las pruebas presentadas fueron realizadas dentro del equipo con el que se entrenó el modelo YOLOv4, por lo que ahora se proseguirá a replicar las pruebas anteriores dentro de la Jetson Nano para obtener un panorama de lo que será la detección de objetos en imágenes que reciba el robot Humanoide “George”.

6.1.1 Pruebas en conjunto de validación

Tabla 5: Evaluación del modelo YOLOv4 en distintas plataformas.

	YOLOv4 en Jetson Nano	**YOLOv4 en PC Escritorio**
True Positive (**TP**)	178	178
False Positive (**FP**)	9	9
False Negative (**FN**)	3	3
Precisión	95.19%	95.19%
Sensibilidad	98.34%	98.34%
F1-Score	96.74%	96.74%
mAP@0.50	99.40%	99.40%

Se realizaron pruebas al modelo bajo el mismo conjunto de validación usado anteriormente para determinar si al detectar objetos existiera una diferencia cuando se ejecutara las pruebas del algoritmo en un equipo como la Jetson Nano. En la Tabla 5 se puede observar que no hubo diferencia en los resultados obtenidos anteriormente en el equipo usado para el entrenamiento del modelo a excepción del tiempo transcurrido para generar los resultados, en el caso de la Jetson fueron aproximadamente de unos 80 segundos, mientras que en la PC de escritorio fueron unos 7 segundos. A su vez, se realizaron las mismas pruebas a la Jetson Nano con los modelos entrenados con el conjunto de datos de TG, resultando en los mismos resultados que las pruebas realizadas en una PC de escritorio, por lo que no se listaron en la Tabla 5. Todo lo anterior representa un rendimiento favorable al momento de evaluar

imágenes por separado al utilizar la Jetson Nano, por lo que ahora se procedería a evaluar el rendimiento en tiempo real de los modelos.

6.1.2 Pruebas en tiempo real

Para la realización de las pruebas del modelo en tiempo real se planteó realizar pruebas bajo diversos escenarios en donde cada uno tuviera una figura o una mezcla de ellas, además se analizó si los modelos pueden detectar objetos en diversas distancias, por lo que se propuso capturar objetos en los escenarios a utilizar en 3 distancias diferentes a las que se categorizarán como distancia cercana, media y lejana como se presenta en la figura.

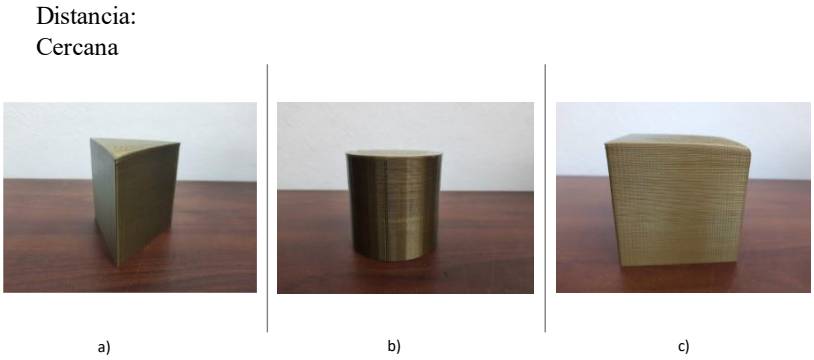

Figura 26: Representación de pruebas con imágenes con distancia cercana.

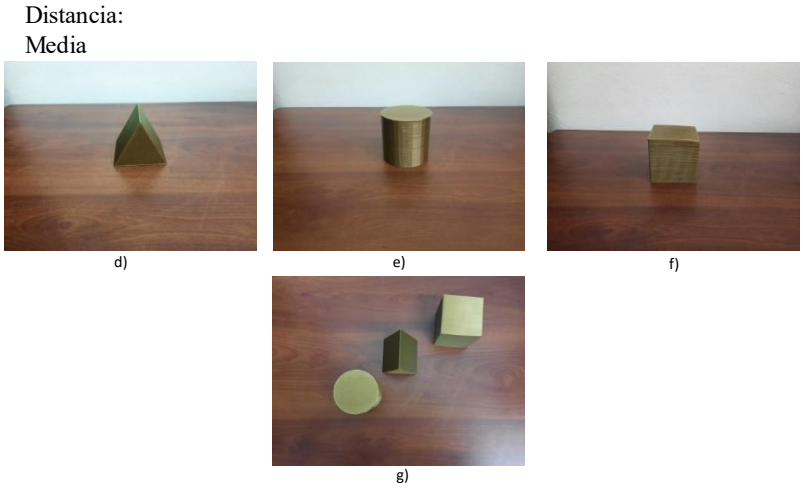

Figura 27: Representación de pruebas con imágenes con distancia media.

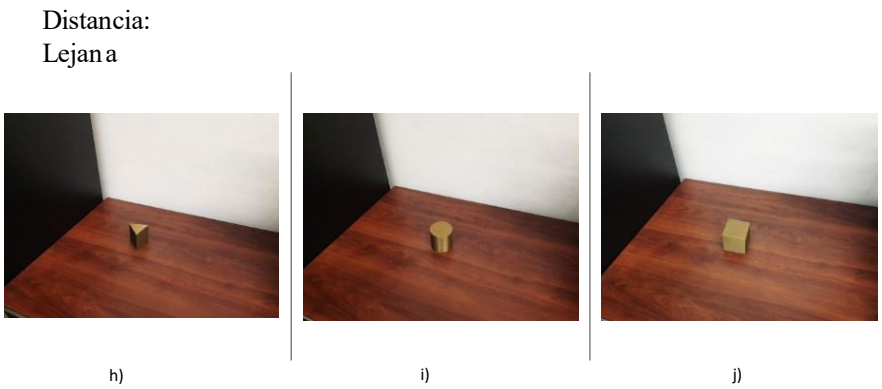

Figura 28: Representación de pruebas con imágenes con distancia lejana.

Tras las pruebas realizadas en tiempo real los cuadros por segundo (FPS) de los modelos se muestran en la Tabla 6, en donde se puede apreciar que el modelo entrenado inicialmente y catalogado como YOLOv4, tiene el mismo rendimiento en términos de velocidad que el modelo entrenado con TG y con la misma resolución, mientras que el modelo entrenado con YOLOv4 con TG y una resolución inferior, obtiene un resultado mejor que los dos anteriores, lo anterior sucede a que al disminuir la resolución, pierde precisión, pero gana velocidad.

Tabla 6: Rendimiento en FPS del modelo en las pruebas en tiempo real.

	YOLOV4	YOLOV4 con TG (608)	YOLOv4 con TG (416)
FPS	1	1	3.7

A través de las siguientes figuras, se presentan los resultados de utilizar el modelo YOLOv4 sin TG para detectar las figuras tomando en cuenta las distancias anteriormente mencionadas para realizar las pruebas. En general todos los modelos de YOLOv4 con y sin TG, independientemente de la velocidad en materia de FPS, muestran buenos resultados al ser puestos a prueba en tiempo real, a excepción del modelo de YOLOv4 con TG (416), el cual, si bien presenta una mejora en FPS a lado de los otros modelos entrenados, es propenso a devolver de manera incorrecta ciertas detecciones en distancias lejanas. En cuanto a los

modelos YOLOv4 con TG (608) y sin TG las detecciones son mayormente favorables a lo largo de las pruebas en las distintas distancias, aunque los resultados del modelo YOLOv4 entrenado sin TG presenta ser el modelo con mejores resultados en la mayoría de las pruebas.

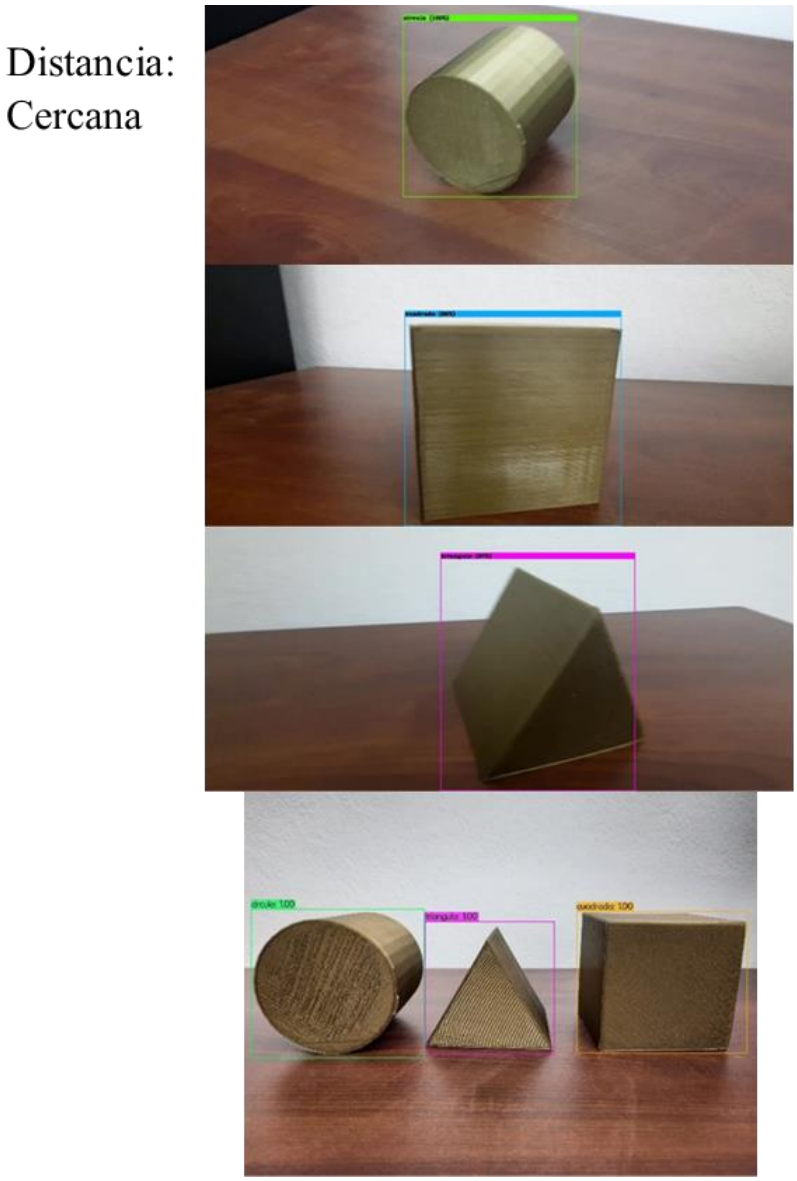

Figura 29: Pruebas en tiempo real con distancia cercana en YOLOv4 sin TG.

Distancia: Media

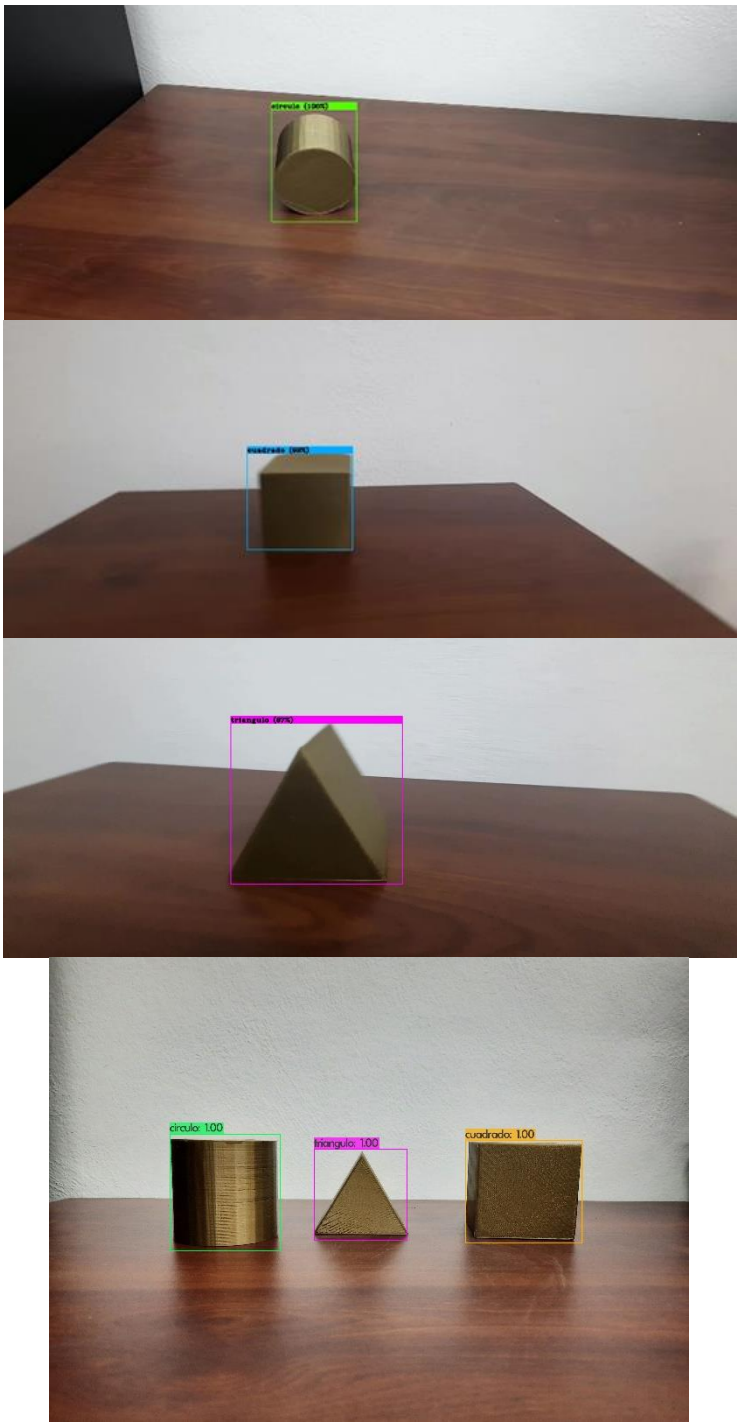

Figura 30: Pruebas en tiempo real con distancia media en YOLOv4 sin TG.

Figura 31: Pruebas en tiempo real con distancia lejana en YOLOv4 sin TG.

Como se puede observar en la imagen del circulo del lado derecho de la figura 31, aún con el modelo YOLOv4 sin TG presenta de vez en cuando errores, lo cual entra dentro de lo esperado, ya que la cualidad principal del modelo es el balance entre velocidad y precisión, a comparación de otras familias de modelos para la detección de objetos donde el aumento de un atributo, disminuye otro.

Capítulo 7. Conclusiones y trabajo a futuro

A lo largo del presente trabajo de tesis, se presentó un modelo de detección de objetos en tiempo real basado en YOLOv4 para ser utilizado dentro del robot humanoide George a través de la tarjeta embebida Jetson Nano. Para lograrlo se inició con la recopilación de las imágenes que se usarán para el entrenamiento del modelo, posteriormente se realizó un tratamiento en términos de formato de las imágenes para ajustarlas de mejor manera a la red de entrada de lo modelo, una vez realizado lo anterior, se etiquetaron las imágenes en el formato que requerido por YOLOv4 para identificar los objetos y su posición dentro de las imágenes y así aprender de ellas durante la fase de entrenamiento, por lo que al finalizar con el etiquetado, se decidió aumentar el tamaño del conjunto de datos que hasta el momento se tenía a través del cambio de fondo de aquellas imágenes con un fondo azul y obteniendo así el primero de dos conjuntos principales de datos que se usaron para entrenar el modelo YOLOv4 sin TG, el segundo se creó a manera de experimentación al utilizar Transformaciones Geométricas para generar un nuevo conjuntos de datos que serviría para entrenar modelos de YOLOv4 con TG y conocer si un mayor número de imágenes podría ayudar al modelo final a aprender de mejor manera los rasgos que componen a las figuras presentadas durante el entrenamiento. Con lo que, una vez generado el conjunto de datos principal, se dispuso a dividirlo en un 80% para el entrenamiento y el 20% para la validación, dicho proceso también se realizó con el conjunto al cual se le aplicó TG.

Posteriormente se prosiguió con la configuración del framework y a su vez se definió la configuración respectiva a cada uno de los 3 modelos que se entrenaron con YOLOv4. Al terminar con las configuraciones previas al entrenamiento, se empezó el entrenamiento de cada modelo YOLOv4.

Tras varias horas de entrenamiento entre modelos, finalmente se obtuvieron los modelos YOLOv4 sin TG, YOLOv4 con TG (608) y YOLOv4 con TG (416) siendo los 2 últimos aquellos modelos entrenados con el conjunto de datos con TG y con resoluciones de

entrada de 608 y 416 respectivamente. Los resultados de los modelos ante los conjuntos de validación de cada modelo son muy favorables a través de los 3 modelos, en el caso del modelo YOLOv4 sin TG, el cual es el modelo principal dentro del presente trabajo de tesis, muestra un **mAP@50** de 99.4% y un **F1-Score** de 97.74%, mientras que el modelo YOLOv4 con TG (608) presenta un **mAP@50** de 99.95% y un **F1-Score** de 98.85, al mismo tiempo YOLOv4 con TG(416) terminó con un **mAP@50** de 99.98% y un **F1-Score** de 99.00%, lo cual muestra que los modelos entrenados con el conjunto de datos al cual se le aplicó TG, son ligeramente mejores al detectar objetos.

Finalmente, tras obtener los resultados de los modelos ante el conjunto de validación, se prosiguió a utilizar los modelos creados anteriormente con la Jetson Nano y se realizaron pruebas en tiempo real de los objetos a través de 3 distancias y en escenarios diferentes. Con lo anterior se pudo observar que el modelo que tenía mejor velocidad en términos de FPS era el modelo YOLOv4 con TG (416), el cual tuvo un rendimiento de 3.7 FPS, mientras que los otros 2 modelos solo mostraron 1 FPS. A pesar de que el modelo YOLOv4 con TG (416) mejores resultados, es el modelo que se equivocaba más de los 3 durante las pruebas en tiempo real en especial en distancias lejanas, mientras que en el caso de YOLOv4 con TG (608) durante las pruebas en tiempo real mostró mejores resultados que la versión con TG (416), aunque seguía presentando algunas detecciones incorrectas a comparación del modelo YOLOv4 sin TG, el cual resultó ser un poco más estable que los demás.

Las pruebas anteriormente descritas también nos muestran que los modelos lograron generalizar los rasgos que identifican a cada figura y que dependiendo de la exactitud que se requiera al detectar objetos, se pueden utilizar modelos de YOLOv4 que mejoren su velocidad, pero disminuyan la exactitud.

Todo lo realizado hasta ahora, se integró al robot humanoide George a través de la tarjeta Jetson Nano, aunque por razones de rendimiento no se pudo ejecutar de manera conjunta la detección de objetos a través del sintetizador por voz.

Como trabajo a futuro, se planea probar el modelo en otros robots humanoides, así como mejorar el modelo entrenado a través de distintos ajustes dentro del modelo, como a su vez con una mejora en el conjunto de datos a través de imágenes con mayor diversidad en su fondo. Con lo anterior, también se buscará permitir comunicar a través de un sintetizador de

voz las detecciones de las figuras presentadas ante él cada 5 segundos, ya que al intentar utilizar el módulo correspondiente a la identificación de objetos por voz dentro de la Jetson Nano, la ejecución del modelo se detenía a causa de la carga computacional que representa el modelo para la Jetson Nano. Al mismo tiempo se plantea utilizar dentro de la Jetson Nano diversas versiones de YOLO en otros frameworks.

Referencias Bibliográficas

Barry, D., Shah, M., Keijsers, M., Khan, H., & Hopman, B. (2019). xYOLO: A Model For Real-Time Object Detection In Humanoid Soccer On Low-End Hardware. *2019 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, 1–6. https://doi.org/10.1109/IVCNZ48456.2019.8960963.

Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

Chatterjee, S., Zunjani, F. H., & Nandi, G. C. (2020). Real-Time Object Detection and Recognition on Low-Compute Humanoid Robots using Deep Learning. *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*, 202–208. https://doi.org/10.1109/ICCAR49639.2020.9108054.

Chollet, F. (2018). *Deep Learning with Python.* Manning Publications.

Condés, I., & Cañas, J. M. (2019). Person Following Robot Behavior Using Deep Learning. *Advances in Physical Agents*, *Vol.* 855, 147–161. https://doi.org/10.1007/978-3-319-99885-5_11.

Cruz, N., Leiva, F., & Ruiz-del-Solar, J. (2021). Deep learning applied to humanoid soccer robotics: Playing without using any color information. *Autonomous Robots*, *45*(3), 335–350. https://doi.org/10.1007/s10514-021-09966-9.

Ekman, M. (2021). *Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, NLP, and Transformers using TensorFlow.* Addison-Wesley.

Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2015). The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, *111*(1), 98–136. https://doi.org/10.1007/s11263-014-0733-5.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 580–587. https://doi.org/10.1109/CVPR.2014.81.

Ibrahim, A. M., Hassan, R. M., Tawfiles, A. E., Ismail, T., & Darweesh, M. S. (2020). Real-Time Collision Warning System Based on Computer Vision Using Mono Camera. *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, 60–64. https://doi.org/10.1109/NILES50944.2020.9257941.

Jeve, K. S., Gaikwad, A. T., Yannawar, P. L., & Kumbhar, A. B. (2019). A Robust Voice-Based Color Object Detection System for Robot. *Innovations in Computer Science and Engineering, Vol.* 32, 105–112. https://doi.org/10.1007/978-981-10-8201-6_12.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, 1097–1105.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541–551. https://doi.org/10.1162/neco.1989.1.4.541.

Maiettini, E., (2020) *From Constraints to Opportunities: Efficient Object Detection Learning for Humanoid Robots* [Tesis doctoral, University of Genova]. CORE. https://core.ac.uk/display/322783660.

Maiettini, E., Pasquale, G., Rosasco, L., & Natale, L. (2017). Interactive data collection for deep learning object detectors on humanoid robots. *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 862–868. https://doi.org/10.1109/HUMANOIDS.2017.8246973.

Mezzina, G., & De Venuto, D. (2022). An Embedded Framework for Fully Autonomous Object Manipulation in Robotic-Empowered Assisted Living. *Sensors*, *23*(1), 103. https://doi.org/10.3390/s23010103.

Mobahi, M., & Sadati, S. H. (2020). An Improved Deep Learning Solution for Object Detection in Self-Driving Cars. *2020 28th Iranian Conference on Electrical Engineering (ICEE)*, 1–5. https://doi.org/10.1109/ICEE50131.2020.9260870.

NVIDIA. (2022a). *NVIDIA Deep Learning Frameworks Documentation*. Recuperado Julio 1, 2022. Sitio web: https://docs.nvidia.com/deeplearning/frameworks/index.html.

NVIDIA. (2022b). *TensorFlow User Guide*. Recuperado Julio 1, 2022. Sitio web: https://docs.nvidia.com/deeplearning/frameworks/tensorflow-user-guide/index.html.

Open Robotics (2018a). *Concepts*. Mayo23, 2022. Sitio web: http://wiki.ros.org/ROS/Concepts.

Open Robotics. (2018b). *Introduction*. mayo 23, 2022. Sitio web: https://wiki.ros.org/ROS/Introduction.

Open Robotics. (2021). *The ROS Ecosystem*. Mayo 23, 2022. Sitio web: https://www.ros.org/blog/ecosystem/.

Padilla, R., Netto, S. L., & da Silva, E. A. B. (2020). A Survey on Performance Metrics for Object-Detection Algorithms. *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 237–242. https://doi.org/10.1109/IWSSIP48289.2020.9145130.

Parico, A. I. B., & Ahamed, T. (2021). Real Time Pear Fruit Detection and Counting Using YOLOv4 Models and Deep SORT. *Sensors*, *21*(14), 4803. https://doi.org/10.3390/s21144803.

Perez-Daniel, K., Fierro-Radilla, A., & Peñaloza-Cobos, J. P. (2020). Rotten Fruit Detection Using a One Stage Object Detector. *Advances in Computational Intelligence*, Vol. 12469, 325–336. https://doi.org/10.1007/978-3-030-60887-3_29.

Rebala, G., Ravi, A., & Churiwala, S. (2019). *An Introduction to Machine Learning*. Springer International Publishing. https://doi.org/10.1007/978-3-030-15729-6.

Redmon, J. (2016) *Darknet: Open Source Neural Networks in C.* pjreddie https://pjreddie.com/darknet/.

Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6517–6525. https://doi.org/10.1109/CVPR.2017.690.

Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779–788. https://doi.org/10.1109/CVPR.2016.91.

Sai, U. B., Sivanagamani, K., Satish, B., & Rao, M. R. (2017). Voice controlled Humanoid Robot with artificial vision. *2017 International Conference on Trends in Electronics and Informatics (ICEI)*, 505–508. https://doi.org/10.1109/ICOEI.2017.8300979.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 3*(3), 210–229. https://doi.org/10.1147/rd.33.0210.

Štepánová, K., Klein, F. B., Cangelosi, A., & Vavrečka, M. (2018). Mapping Language to Vision in a Real-World Robotic Scenario. *IEEE Transactions on Cognitive and Developmental Systems*, *10*(3), 784–794. https://doi.org/10.1109/TCDS.2018.2819359.

Szemenyei, M., & Estivill-Castro, V. (2019). ROBO: Robust, Fully Neural Object Detection for Robot Soccer. *RoboCup 2019: Robot World Cup XXIII, Vol.* 11531, 309–322. https://doi.org/10.1007/978-3-030-35699-6_24.

TensorFlow. (2021). *TensorFlow Lite*. Recuperado en Julio 1, 2022. Sitio web: https://www.tensorflow.org/lite/guide.

Tian, L., Thalmann, N. M., Thalmann, D., Fang, Z., & Zheng, J. (2019). Object Grasping of Humanoid Robot Based on YOLO. *Advances in Computer Graphics: 36th Computer Graphics International Conference CGI 2019, Vol.* 11542, 476–482. https://doi.org/10.1007/978-3-030-22514-8_47.

Wang, Z., Zhang, J., Zhao, Z., & Su, F. (2020). Efficient Yolo: A Lightweight Model For Embedded Deep Learning Object Detection. *2020 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, 1–6. https://doi.org/10.1109/ICMEW46912.2020.9105997.

Wu, W.-K., Chen, C.-Y., & Lee, J.-S. (2021). Embedded YOLO: Faster and Lighter Object Detection. *Proceedings of the 2021 International Conference on Multimedia Retrieval*, 560–565. https://doi.org/10.1145/3460426.3463660.

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into Deep Learning. *arXiv prepreint arXiv:2106.11342v3*.