USO DE INTELIGENCIA ARTIFICIAL PARA LA IDENTIFICACIÓN DE PIEZAS MECÁNICAS UTILIZANDO UNA CÁMARA

JUNIOR GONZALO HERNÁNDEZ ESPINOSA
BRAYAN ARLEY NIÑO TORRES

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2021

USO DE INTELIGENCIA ARTIFICIAL PARA LA IDENTIFICACIÓN DE PIEZAS MECÁNICAS UTILIZANDO UNA CÁMARA

JUNIOR GONZALO HERNÁNDEZ ESPINOSA
BRAYAN ARLEY NIÑO TORRES

Trabajo de Grado para optar al título de
Ingeniero Electrónico

Director
Jaime Guillermo Barrero Pérez
Mag. Potencia eléctrica

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2021

Dedicado

A mis padres, Gonzalo Hernández & Luz Amparo Espinosa, y a mi hermano Cristian Hernández; quienes me han apoyado en cada decisión que he tomado y han estado para mi en cada momento.

A mi novia, mis amigos, compañeros y profesores, que fueron fundamentales durante mi proceso de formación como profesional; y a todo aquel que por cualquier razón, lea este documento.

Junior Hernández

Para mis abuelas Maria Ninfa Alvarez & Maria Elsa Muñoz, y a mis padres; que son las personas mas importantes de mi vida.

A mi amiga y compañera Andréa Carolina Hernández Joya por toda su colaboración y apoyo en todo momento.

Brayan Niño

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

pág.

RESUMEN

TÍTULO: USO DE INTELIGENCIA ARTIFICIAL PARA LA IDENTIFICACIÓN DE PIEZAS MECÁNICAS UTILIZANDO UNA CÁMARA [*]

AUTORES: JUNIOR GONZALO HERNÁNDEZ ESPINOSA, BRAYAN ARLEY NIÑO TORRES [**]

PALABRAS CLAVE: DETECCIÓN DE OBJETOS, INTELIGENCIA ARTIFICIAL, VISIÓN POR COMPUTADOR, APRENDIZAJE PROFUNDO

DESCRIPCIÓN:
La inteligencia artificial (*AI*) y la industria son dos conceptos que a día de hoy van de la mano. Se dice que esta última ha pasado a una era más "digitalizada" gracias a la tecnología moderna [1]. Por consiguiente, este proyecto busca hacer su aporte a la industria usando las redes neuronales artificiales, inteligencia artificial.

En este trabajo de grado se realizó cabo el entrenamiento de una red neuronal para la clasificación de objetos tales como tuercas, arandelas, tornillos. Para su ejecución, se presentan dos frameworks de trabajo. El primero, la API de tensorflow, la cual se implementó en el sistema operativo Windows, de manera local, usando una tarjeta de gráficos (GPU) Nvidia, y su software de desarrollo: CUDA y CuDNN; y aXeleRate, se ejecutó en la nube usando la herramienta *Google Colaboraty*. Estos objetos se mueven sobre una banda transportadora en medio de otros objetos a clasificar, a lo largo de ella, se captura de una imagen, la cual corresponde a una "sección" de la banda transportadora que será analizada por un microcontrolador, capacitado a nivel de hardware, software y programación para detectar y caracterizar los objetos por su tamaños y clases. Al mismo tiempo, el microcontrolador llevará a cabo un conteo de cuáles y cuantas piezas han sido clasificadas, información que se mostrará en pantalla.

---

[*] Trabajo de grado

[**] Facultad de Ingeniería Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: Jaime Guillermo Barrero Pérez, Mag. Potencia Eléctrica.

[1] Siemens. *La Inteligencia Artificial en la industria | Industria | Siemens Global*.

ABSTRACT

TITLE: USE OF ARTIFICIAL INTELLIGENCE FOR IDENTIFICATION OF MECHANICAL PARTS USING A CAMERA [*]

AUTHORS: JUNIOR GONZALO HERNÁNDEZ ESPINOSA, BRAYAN ARLEY NIÑO TORRES [**]

KEYWORDS: OBJECT DETECTION, ARTIFICIAL INTELLIGENCE, COMPUTER VISION, DEEP LEARNING

DESCRIPTION:

Artificial intelligence (AI) and industry are two concepts that go hand in hand today. The latter is said to have moved into a more “digitized” era thanks to modern technology[1]. Therefore, this project seeks to make its contribution to the industry using artificial neural networks, artificial intelligence.

In this degree work, the training of a neural network for the classification of objects such as nuts, washers, screws is carried out. For its execution, two frameworks are presented. The tensorflow API, which was implemented in the Windows operating system, locally, using an Nvidia graphics card (GPU), and its development software: CUDA and CuDNN; and aXeleRate, it was run in the cloud using the Google Colaboraty tool. These objects move on a conveyor belt in the middle of other objects to be classified, along it, an image is allowed to be captured, which corresponds to a “section” of the conveyor belt that will be analyzed by a microcontroller, trained at the hardware, software and programming level to detect and characterize objects by their sizes and classes. At the same time, the microcontroller will carry out a count of which and how many pieces have been classified, information that will be displayed on the screen

---

[*] Bachelor Thesis

[**] Faculty of Engineering and Physical Sciences. School of Electronic and Electrical Engineering. Director: Jaime Guillermo Barrero Pérez, Master in Electrical Power.

INTRODUCCIÓN

El mundo ha llegado hoy a lo que se denomina la cuarta revolución industrial o industria 4.0, revolución que da inicio en la segunda década del siglo XXI, y su auge se ha dado más específicamente en los últimos años [1]. Se trata una nueva era donde se hace uso total y exhaustivo de la tecnología moderna con el fin de interconectar todas las partes en las empresas y de esta manera lograr una automatización inteligente de sus procesos, mejorando la eficiencia de las mismas. Hoy en día la economía está evolucionando de la mano de esta revolución, que se caracteriza por el uso de *Cyber Physical Systems* (CPS), fábricas inteligentes, transformaciones digitales e innovación de servicios [2].

Entre las tecnologías que involucra la industria 4.0 están presentes: robótica, *big data*, *cloud computing*, sistemas autónomos, *IoT* y *AI*, siendo esta última el enfoque principal de este proyecto. Esta nueva "era digital" será tan beneficiosa para algunos sectores de la industria como tan perjudicial para otros, ya que aquellas empresas que no logren adaptarse a los cambios que esta conlleva, se verán fuertemente afectados y superadas por aquellas que si lo hagan[23].

---

1 Joanna J Bryson. "La última década y el futuro del impacto de la IA en la sociedad | OpenMind". En: *OpenMindBBVA* ().

2 Saqib Shamim y col. "How firms in emerging economies can learn industry 4.0 by extracting knowledge from their foreign partners? A view point from strategic management perspective". En: *International Conference on Advanced Mechatronic Systems, ICAMechS*. Vol. 2019-Augus. IEEE Computer Society, 2019, págs. 390-395. DOI: 10.1109/ICAMechS.2019.8861622.

3 Emanuele Cardillo y Alina Caddemi. "Feasibility Study to Preserve the Health of an Industry 4.0 Worker: A Radar System for Monitoring the Sitting-Time". En: *2019 IEEE International Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2019 - Proceedings*. Institute of Electrical y Electronics Engineers Inc., 2019, págs. 254-258. DOI: 10.1109/METROI4.2019.8792905.

De forma general, los avances tecnológicos han tenido un gran impacto mejorando la industria considerablemente, siendo un aporte positivo para la sociedad en varios aspectos. Sin embargo, el desarrollo industrial implica cambios no deseados. El cambio climático, los niveles de contaminación del aire, el agotamiento de recursos, e incluso la pérdida de biodiversidad son algunas de las consecuencias secundarias [4]. No obstante, se puede usar la tecnología en favor del medio ambiente y contrarrestar parcialmente los efectos negativos mencionados anteriormente. Conceptos como *"smart cities"* [5] se hacen presentes, así como diferentes áreas que deben mejorar: renovación urbana, movilidad, energía, eficiencia y cambio climático, residuos, entre otras [6].

En el campo de residuos y reciclaje, área central de nuestro proyecto es posible realizar una optimización con relación a la selección de objetos que aún tengan vida útil, tales como tornillos, arandelas y tuercas. Durante este proceso, los objetos pasan a través de una banda transportadora para su respectiva clasificación, por ende, se propone un sistema inteligente basado en inteligencia artificial *(AI)* encargado de la detección de las piezas mencionadas.

En este proyecto se desarrolla un método para la detección y clasificación de piezas mecánicas funcionales para su reciclaje, esto con el fin de hacer un aporte importante, y además, entrar en materia con la Industria 4.0. De igual manera, con el debido

---

[4] PwC. *Industria 4.0: ¿cómo puede beneficiar al medio ambiente?* 2018.

[5] Mircea Eremia, Lucian Toma y Mihai Sanduleac. "The Smart City Concept in the 21st Century". En: *Procedia Engineering*. Vol. 181. Elsevier Ltd, 2017, págs. 12-19. DOI: 10.1016/j.proeng.2017.02.357.

[6] Asociación Cluster de Industrias de Medio Ambiente de Euskadi. Aclima. *Tecnología e industria 4.0: la sostenibilidad en la cuarta Era Industrial*. Inf. téc. Madrid: Fundación Conama (Congreso Nacional del Medio Ambiente), 2018.

procedimiento (tal como construir la base de datos, ajustar parámetros, entre otros), se puede llevar a una manera más general en donde se adapta al cualquier clase de objeto que se desee clasificar. Con esto se pretende aportar mediante una solución económica y eficiente, al desarrollo que implica dicha revolución; con esto se espera hacer las empresas mas competitivas, mejorar su economía y generar nuevas fuentes de trabajo en esta área.

1. OBJETIVOS

1.1. OBJETIVO GENERAL

- Identificar diferentes piezas mecánicas en una banda transportadora y clasificarlas según su tipo y tamaño, a partir de un sistema estructurado en redes neuronales.

1.2. OBJETIVOS ESPECÍFICOS

- Entrenar un modelo que a través de técnicas de aprendizaje profundo, identifique piezas mecánicas: tornillos, tuercas y arandelas; en una banda transportadora y clasificarlas según sus tamaños y tipos.
- Obtener una base de datos con imágenes que permitan entrenar el algoritmo (red neuronal) para la identificación de tuercas, tornillos y arandelas.
- Evaluar el rendimiento del modelo implementado.

2. INTELIGENCIA ARTIFICIAL Y APRENDIZAJE PROFUNDO

2.1. ESTADO DEL ARTE

En la actualidad, existen distintas clases de dispositivos con la funcionalidad de clasificación haciendo uso de los algoritmos de inteligencia artificial, algunos de los métodos propuestos para la detección de objetos, usando este tipo de inteligencia, están basados en arquitecturas de redes neuronales tales como Yolo, MobileNet, ResNet, Inception, entre otras. Así mismo, existen marcos de trabajo diseñados para facilitar la creación o adaptación (*transfer learnign*) de modelos existentes que usan estas arquitecturas.

Profundizando en el tema de redes neuronales para la identificación de piezas mecánicas, se pueden encontrar trabajo como el de Lenin Marcel Cadena Castro[7], en el que se hace uso de un sistema inteligente con visión artificial y una red neuronal en el software MATLAB, para el reconocimiento y clasificación por el robot NAO como piezas mecánicas de una linea de producción.

Hoy por hoy, existen grandes compañías tecnológicas que han desarrollado dispositivos o tecnologías para facilitar la implementación de redes neuronales en plataformas con recursos limitados y pequeñas computadoras de placa única, como Raspberry Pi, Arduino y la micro:bit.

Por otra parte, el trabajo de Luis Piñuel Moreno[8] realizó el análisis del procesador

---

[7] Jonathan Alejandro Heredia López Lenin Marcel Cadena Castro. “Sistema inteligente con visión artificial para el reconocimiento de piezas mecánicas en el robot NAO”. B.S. thesis. Universidad Politécnica Salesiana, 2018.

[8] Francisco Igual Peña Luis Piñuel Moreno. “ACELERACIÓN DE AI EN DISPOSITIVOS DE BAJO CONSUMO”. B.S. thesis. Universidad Complutense de Madrid, 2020.

Kendryte K210, incidiendo sobre su rendimiento y consumo energético, usando dos placas, Maix Go y Maix Bit del fabricante Seeed Studio Sipeed y comparando con otras DSAs (Domain-Specific Architectures), como Google Coral e Intel NCS2. De este trabajo se concluyo un mejor consumo energético por parte de la Maix Bit, y en terminos de FPS (fotogramas por segundo) la arquitectura K210 es más rápida que las otras dos arquitecturas de propósito específico. A pesar de los buenos resultados, el procesador K210 presenta limitaciones, ya que no presenta la memoria suficiente que permite implementar modelos mas grandes

Un estudio abordado por Valderas[9] es la comparativa de ejecución de redes neuronales profundas en un sistema embebido como la Raspberry Pi. Este, usa el *Neural Compute Stick* (NCS), una herramienta de desarrollo para inferencia de aprendizaje profundo con un consumo mínimo en computación. Lo anterior crea una red neuronal desde cero, que permite clasificación de imágenes ejecutándolas en la Raspberry Pi con y sin el *Neural Compute Stick*, obteniendo los resultados del rendimiento. Además, para colocar en contexto los resultados con respecto a una maquina de grandes recursos. Concluyendo una mejora de 5 veces en el tiempo de respuesta y estableciendo tiempos cercanos a los conseguidos con plataformas que emplean mas recursos de computación. Asimismo, se evidenciando la utilidad de dispositivos aceleradores de inferencia en sistemas embebidos.

---

[9] Antonio José Toro Valderas. “Implementación de redes neuronales en Raspberry Pi 3 con Movidius Neural Compute Stick”. B.S. thesis. Universidad de Sevilla, 2020.

2.2. INTELIGENCIA ARTIFICIAL

Para empezar debemos hablar del concepto fundamental aplicado en este proyecto: La inteligencia artificial (IA). Ésta ha logrado, en pocas palabras, cambiar la forma en que vivimos. La idea de inteligencia artificial fue introducida por los científicos J. McCarthy, M. Minsky, C. Shannon, A. Newell, y H. Simon en agosto de 1956; sin embargo, es hasta las ultimas décadas en donde se han conseguido los avances tecnológicos necesarios para poder desarrollar esta importante área, y es gracias a esto que ha tomado tanta fuerza en la actualidad, con eventos importantes como la victoria de Deep Blue; la inteligencia artificial de IBM, ante Gari Kasparov; campeón mundial de ajedrez, en 1997; o cuando google ganó con su IA al campeon mundial de *Go* en 2015.

A grandes rasgos se puede definir la inteligencia artificial como el área de la tecnología que estudia los comportamientos, aptitudes y habilidades del ser humano a nivel racional y/o cognitivo (comprensión, percepción, toma de decisiones) para reproducirlos en un sistema embebido o computadora[10]. Un ejemplo particular de inteligencia artificial, aplicado a la industria automotriz es el de una maquina, la cual ha sido entrenada para que sus entradas sean las características de un automotor, y según éstas, genere el diseño del chasis mas apropiado para los requerimientos dados, esto lo hace una IA llamada *"generative-design AI"* ó inteligencia artificial de diseño generativo, un modelo bastante avanzado y que lleva un gran trabajo detrás para su entrenamiento[11]. Así como el anterior, es posible obtener modelos de IA para una gran variedad de aplicaciones, y de esto se han encargado las diferentes

---

[10] Thomas Hardy. *Revista de la Universidad Bolivariana Volumen*. Inf. téc.

[11] Maurice Conti. “The incredible inventions of intuitive AI”. En: *TED Conferences*. 2016.

ramas en las que se divide la inteligencia artificial, estas son algunas de ellas:

- Aprendizaje automático (Machine Learning)
- Aprendizaje profundo (Deep Learning)
- Visión por computador (Computer Vision)
- Procesamiento de lenguaje natural (NLP)
- Generación de lenguaje natural (NLG)
- Chatbot
- Análisis predictivo

Donde cada una de ellas comparten algo en común en su desarrollo, que es su principal combustible y el de la inteligencia artificial en sí: los datos. En aprendizaje automático, un modelo se refiere a un algoritmo que ha sido previamente capacitado o entrenado, para que dada una entrada, generé una predicción asociada a dicha entrada. Un modelo de IA será mas eficiente entre mas cantidad de datos haya sido usado para su entrenamiento, es por esto que junto con la IA, el *Big Data* es también una de las innovaciones mas importantes a día de hoy.

2.2.1. Aprendizaje automático El aprendizaje automático o *machine learning* es una rama de la inteligencia artificial cuyo objetivo es el estudio y desarrollo de técnicas las cuales permitan a las maquinas ejecutar acciones para las que no han sido programadas o entrenadas, sino que han logrado aprender con base a sus experiencias pasadas. Para esto, existen los paradigmas de aprendizaje, que en los humanos, son análogos a los mecanismos que permiten procesar toda la información nueva recibida para transformarla en conocimiento. En el *machine learning*

existen dos principales paradigmas de aprendizaje: el aprendizaje supervisado y el aprendizaje no supervisado.

Aprendizaje supervisado: En este tipo de aprendizaje se busca obtener una relación existente entre variables de entrada y salida, mediante un proceso repetitivo en donde el modelo recibirá múltiples ejemplos de entrada asociados a una salida en particular. Una vez realizado este proceso, llamado comúnmente entrenamiento, el algoritmo habrá sido capaz o no (depende la complejidad, la cantidad de iteraciones, etc) de extraer patrones que relacionen cada ejemplo o característica de entrada con su característica de salida y de esta forma realizar una inferencia con ejemplos incluso, que no fueron utilizados para su entrenamiento. Se llama supervisado por que al decirle como son las características de salida se está participando en su aprendizaje.

Aprendizaje no supervisado: A diferencia del aprendizaje supervisado, el aprendizaje no supervisado solo es alimentado con ejemplos de entrada, sin ninguna información adicional, en este, la red descubre en los datos de entrada y de forma autónoma: características, regularidades, correlaciones y categorías.

2.2.2. Aprendizaje profundo Es una técnica de *machine learning* inspirada en la arquitectura de red neuronal de un cerebro humano, y que busca simular su funcionamiento mediante algoritmos o modelos basados en redes neuronales (2.3). En aprendizaje profundo, un modelo aprende a clasificar directamente archivos de texto, imágenes o sonido, a tal nivel de sobrepasar en algunas ocasiones el rendimiento de un humano en ciertas tareas. Estos modelos son entrenados usando grandes conjuntos de datos etiquetados (características de entrada y de salida), en arquitecturas de redes neuronales profundas, es decir, de varias capas.

Dentro del aprendizaje profundo podemos encontrar varias arquitecturas de aprendizaje con redes neuronales profundas, redes neuronales profundas convolucionales, redes neuronales recurrentes, que han sido aplicadas a campos como visión por computador, reconocimiento de voz, reconocimiento de audio y música.

El concepto de aprendizaje profundo se teorizó en los años 80, este no se había aplicado por dos razones principales: requiere de grandes cantidades de datos y la poca potencia de computo, en esa época, carecía de estos dos componentes fundamentales. El concepto de "profundo" tiene su fundamento en la redes neuronales, las cuales, entre mas capas ocultas tengan mas "profundas" son[12].

2.2.3. Visión por computador Es el proceso de obtención, caracterización e interpretación de la información a partir de imágenes. Este se desarrolla por medio de una técnica de captación óptica muy practica; en donde el análisis de las imágenes obtenidas permite detectar en un objeto diversas clases de características las cuales en muchas ocasiones son imperceptibles al ojo humano. En diversas industrias la visión artificial incrementa la calidad y seguridad en procesos de fabricación y clasificación, esta herramienta posee numerosas aplicaciones y ha tenido grandes avances gracias al desarrollo de un tipo de red neuronal especializada para trabajar con imágenes.

2.2.4. Representación digital de imágenes La visión por computador se lleva a cabo gracias a la digitalización de las imágenes, lo que significa, que se debe emplear un lenguaje que las maquinas interpreten. Una imagen digital es una representación numérica, una función espacial $f(x, y)$ que ha sido discreteada tanto en

---

[12] ML. *What Is Deep Learning? | How It Works, Techniques & Applications - MATLAB & Simulink*. 2019.

coordenadas espaciales como en intensidad. En otras palabras, es una matriz cuya función se basa en identificar un punto de la imagen y el correspondiente valor en ese punto. A estos elementos de distribución digital se les denomina píxeles[13].

Las imágenes digitales pueden ser representadas en diferentes modos de color, siendo lo más comunes el de escala de grises y el RGB (por sus siglas en *inglés red, green, blue*).

La escala de grises es la representación de una imagen en la que cada píxel se dibuja usando un valor numérico individual que representa su intensidad, en una escala que se extiende entre blanco y negro.

Por otra parte, el modelo mas utilizado en representación de imágenes de color digital es capaz de representar el color mediante síntesis aditiva, mezclando por adición de los tres colores de luz primarios (rojo, verde y azul). Una imagen digital con formato de color RGB, se divide en 3 capas, una para cada color, y sus valores de intensidad están comprendidos entre 0 y 255.

2.2.5. Etapas en el proceso de Visión Artificial La visión artificial esta compuesta por procesos como la obtención, caracterización e interpretación de imágenes. Estos procesos se dividen en 5 áreas principales.

- Adquisición.
- Preprocesado.
- Segmentación.
- Representación y descripción.
- Reconocimiento e interpretación.

---

[13] DEA. “Procesamiento digital de imágenes”. En: *Perfiles Educativos* 72 (1996).

La adquisición consiste en obtener la imagen. El preprocesado incluye las técnicas para mejorar la calidad como reducción de ruido y resalte de detalles.La segmentación divide la imagen en objetos de interés. Por medio de la descripción de sus características. El reconocimiento es el proceso que nos lleva a identificar los objetos(tuercas, tornillos, arandelas). Por ultimo la interpretación les da un significado asociado al conjunto de objetos reconocido. Por ello conviene agrupar estos procesos según la complicación y el grado de detalle que tiene su implementación.

- Visión De Bajo Nivel
- Visión de nivel intermedio
- Visión de alto nivel

La visión de bajo nivel son aquellos procesos que no necesitan ningún tipo de inteligencia. Estos son la adquisición y el procesado. La visión de nivel intermedia son procesos que extraen, caracterizan y etiquetan elementos de la imagen obtenido en el visión de bajo nivel. Esto involucra lo que es segmentación, descripción y reconocimiento. La visión de alto nivel es aquella que simula una inteligencia artificial mediante algoritmos.

2.3. REDES NEURONALES ARTIFICIALES

Las redes neuronales artificiales, conocidas por sus siglas en inglés *Artificial Neural Network* (ANN) son un modelo computacional ampliamente utilizado en el aprendizaje automático, y son la base fundamental de los modelos de aprendizaje profundo. Están inspirados en el cerebro humano, e intentan imitar la forma de comunicaciones de una neurona biológica.

Una red neuronal convencional está conformada por capas: capa de entrada, capas ocultas y capa de salida. Cada una de estas capas está a su vez conformada

por nodos, los cuales se caracterizan por tener dos parámetros: la primera llamada ***Weigths*** (W) ó ***pesos*** y otra llamada ***bias*** ó ***sesgos***. Los valores de estas variables son los que se buscan ajustar durante el entrenamiento de una red neuronal. Cuando el valor de salida de un nodo está por encima del valor de umbral especificado, el nodo se activa enviando sus datos a los nodos de la siguiente capa de la red[14].

La información se mueve a través de la red neuronal desde las capas de entrada a las capas de salida, así, a cada neurona o nodo llegan múltiples entradas de nodos anteriores (o las características de entrada, en caso de ser los nodos de la capa de entrada), y las sumas de estas multiplicadas por sus pesos asociados determina el "impulso nervioso" que recibe la neurona. Este valor, es evaluado en el interior de la neurona mediante una función de activación la cual devuelve un último valor que es el valor de salida de la neurona.

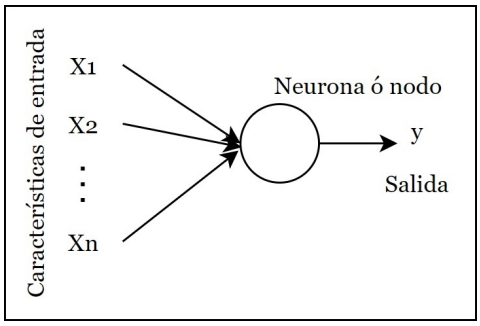
2.3.1. Red Neuronal monocapa La red neuronal monocapa es la red neuronal mas sencilla, compuesta únicamente por la capa de entrada (que son las mismas características de entrada) y una capa de salida. Una red neuronal monocapa se puede observar en la figura 1.

2.3.2. Red neuronal multicapa Esta red neuronal es una generalización de la red neuronal monocapa, tambien cuenta con sus capas de entrada y salida, además dispone de un conjunto de capas intermedias (capas ocultas) entre estas. Una red neuronal multicapa se puede apreciar en la figura 2.

---

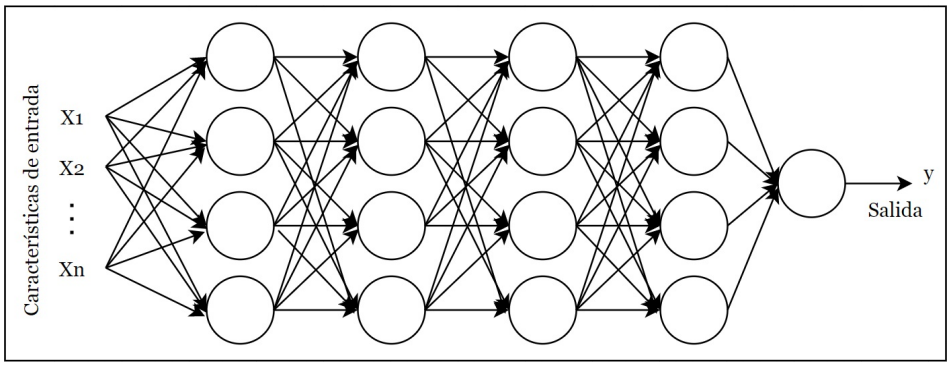
[14] IBM. *What are Neural Networks? | IBM*.

Figura 1. Red neuronal monocapa



Fuente: Elaboración propia

Figura 2. Red neuronal multicapa



Fuente: Elaboración propia

2.3.3. Red Neuronal convolucional Este tipo de red neuronal es las protagonista de los avances que se tienen a la fecha en visión por computador, se caracteriza principalmente por ser capaz de reconocer patrones complejos en grandes conjuntos de imágenes. El motivo por el que esta red neuronal se destaca en este aspecto es porque su arquitectura está dividida en dos partes. En la primera parte se tienen capas de convolución y reducción (*max pooling*), ésta es la que le da el nombre a este tipo de redes; la segunda parte es una red neuronal multicapa, que toma la información recolectada por la primera y es donde se lleva a cabo la clasificación.

Convolución: Es una operación en la cual, dada una imagen de entrada, genera una de salida, después de aplicar una operación con base a un *kernel* o filtro, que no es más que una matriz de números de un tamaño reducido. Esta operación es tan sencilla como la multiplicación y la suma de un píxel central (sobre el que se está aplicando la operación) y sus píxeles vecinos. El resultado variará en función de los parámetros del filtro.

Figura 3. Ejemplo de filtro

1	0	-1
1	0	-1
1	0	-1

Fuente: Elaboración propia

Al aplicar estos filtros a las imágenes que entran en la red neuronal, se obtiene un ***mapa de características***, los cuales contienen información extraída a la imagen según el *kernel* aplicado. Por ejemplo, al aplicar el filtro de la figura 3 en una imagen, se obtiene un mapa de características el cual representa los cambios de contraste o bordes, verticales en la imagen (ver figura 6).

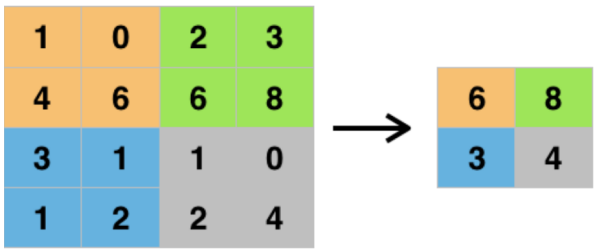
Agrupación máxima: La agrupación máxima o *max pooling* es el proceso de reducir el tamaño de la imagen o mapa de características, suele ir posterior a una capa de convolución y se hace resumiendo regiones. Para llevar a cabo el *max pooling* es necesario definir dos cosas: el ***grid*** y el ***stride***. El gird o *grid* define el tamaño de la sección que se va a agrupar y el paso define que tan separadas se harán estas agrupaciones. Una vez definidos estos dos valores, se ubica la cuadricula en la imagen y se extrae el valor de mayor magnitud en dicha cuadricula, posteriormente se desplaza en el valor del paso y se repite el procedimiento. Un ejemplo de *max*

Figura 4. Filtro de la figura 3 aplicado a una imagen

Fuente: Tomado de Dot CSV: Redes Neuronales Convolucionales - La VISIÓN de la Inteligencia Artificial

pooling se puede observar en la figura 5.

Figura 5. Max pooling



Fuente: Tomado de https://deepai.org/machine-learning-glossary-and-terms/max-pooling

Si bien un filtro de convolución no detecta mas que cambios de contrastes, texturas, superficies planas, la clave de usarlos está en la profundidad de las redes neuronales que usen estos filtros. Cuando se configuran dos o más filtros en cascada, se está haciendo detecciones sobre los resultados de filtros anteriores, y cuando se agrupan filtros de diferentes características, es posible detectar patrones cada vez mas complejos. Además, otra ventaja que traen consigo los filtros es la capacidad de reducir el tamaño de la imagen, según el *kernel* aplicado.

2.3.4. Redes neuronales recurrentes Son un tipo de red neuronal diferente a las que se han presentado, estas tienen la especial característica de tratar datos de secuencias temporales, que permite incluir la dimensión de tiempo. También se conocen como redes espacio-temporales o dinámicas, en donde el cálculo de una entrada en dado paso, depende de un paso anterior y en algunas ocasiones de el paso futuro [15].

2.4. REGRESIÓN LOGÍSTICA Y REDES NEURONALES

2.4.1. Regresión logística Para entender el funcionamiento de una red neuronal, se puede empezar por entender un concepto base, la regresión logística. La regresión logística es un algoritmo de aprendizaje que se utiliza para modelar situaciones donde un evento se puede clasificar como verdadero o falso, ejemplos como 1 ó 0, como perro o gato; es decir, problemas de clasificación binaria. Una regresión logística se describe como una red neuronal muy pequeña. De hecho, la regresión logística, se trata de una red neuronal con exactamente una neurona (2.3.1).
Dado un vector de características x, el cual puede ser, por ejemplo, una imagen que se quiere o no clasificar como imagen que contiene un gato, se quiere que la salida sea $\hat{y}$, y que corresponde con la probabilidad de que la imagen de entrada sea de un gato. No es apropiado definir la salida de una neurona simplemente como $\hat{y} = WX + b$, puesto que $\hat{y}$ al ser una probabilidad, debe corresponder con un valor entre 0 y 1. Para lograr esto, se define una segunda operación dentro de la neurona, la **función de activación**.

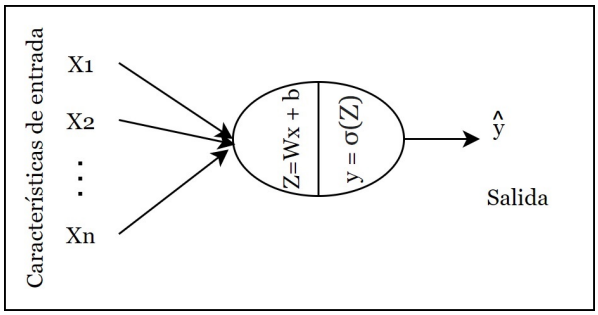
$$Z = WX + b \tag{1}$$

[15] Isis Bonet Cruz y col. “Redes neuronales recurrentes para el analisis de secuencias”. En: (2007).

$$\hat{y} = \sigma(Z) \tag{2}$$

Las ecuaciones (8) y (2) representan las operaciones que se efectúan dentro de una neurona, donde W, X y Z, pueden ser cantidades vectoriales o matriciales.

Figura 6. Representación gráfica de la regresión logística



Fuente: Elaboración Propia

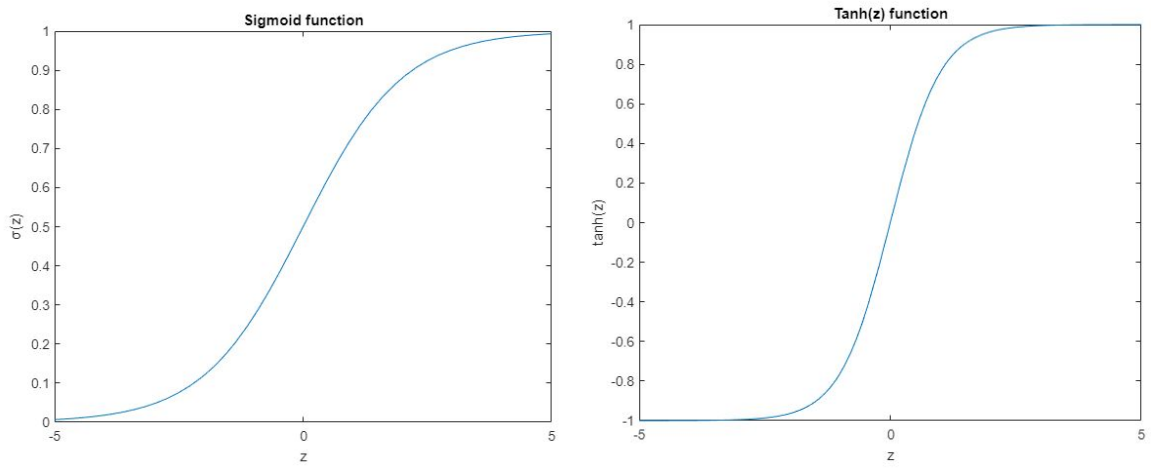
2.4.2. Función de activación Es una función usada para definir la salida de un nodo en una red neuronal. Existen varios tipos de funciones de activación que pueden ser usadas según conveniencia o gusto.

La elección de la función de activación se puede considerar un hiper-parámetro (2.4.6). En la figura 7 se pueden observar las funciones $Sigmoide$ y $tanh(z)$. La función $tanh(z)$ es una versión desplazada de la función $Sigmoide$, normalmente la primera funciona mejor para capas intermedias, mientras que $tanh(z)$ se suele utilizar en capas de salida para ejemplos de clasificación binaria, debido a que su rango va de 0 a 1.

Una función muy popular en *machine learning* es la función $ReLu$ (unidad lineal rectificada), ésta se puede observar en la figura 8.

Una desventaja de las dos primera funciones es que presentadas anteriormente

Figura 7. Funciones de activación



(a) Sigmoid (b) Tanh

Fuente: Elaboración Propia

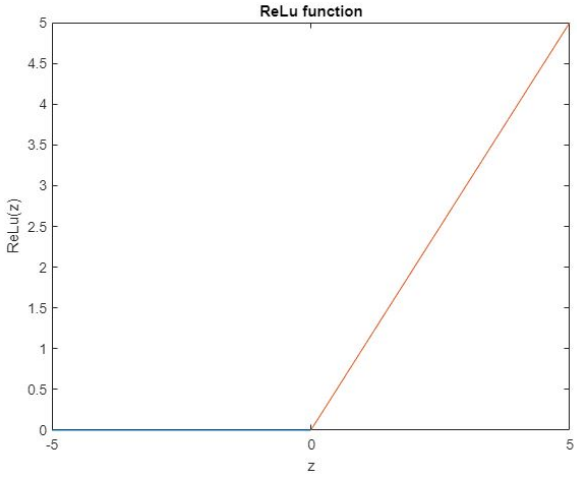
es que para valores muy grandes o muy pequeños, la derivada de la función tiende a 0, lo que dificulta el aprendizaje al detener el proceso de descenso del gradiente, situación que no se presenta en la función *ReLu*.

Softmax por otra parte es una función que convierte un vector de números en un vector de probabilidades, se utiliza para normalizar las salidas, convirtiendo los valores de suma ponderada en probabilidades que suman uno. Cada valor de la salida de la función se interpreta como la probabilidad de pertenencia a cada clase.

$$\text{f(z)j} = \frac{e^{zj}}{\sum_{k=1}^{k} e^{zk}} \quad \text{para j = 1, . . . , K.} \tag{3}$$

2.4.3. Función de pérdida y funcion de costo La **función de pérdida** (L) es utilizada en modelos de aprendizaje supervisado. Sirve para estimar que tan bien funcionan estos, en términos generales, esta función mide la capacidad que un modelo tiene para hacer una estimación correcta de la relación entre la entrada y la

Figura 8. Función de activación



Fuente: Elaboración Propia

inferencia de salida. Un modelo tiene mejores resultados cuando su función de pérdida tiende a cero. Una función de pérdida comúnmente utilizada es la del error cuadrático medio, sin embargo, esta función puede presentar varios mínimos locales, lo que dificultaría la optimización de la misma cuando se lleve a cabo el descenso del gradiente (2.4.4), otra opción que es bastante elegida en redes neuronales y regresión logística es la siguiente:

$$L(\hat{y}, y) = -(ylog(\hat{y}) + (1-y)log(1-\hat{y})) \tag{4}$$

Donde $\hat{y}$ es la salida estimada e y el valor correcto. Esta función se define respecto a un único ejemplo de entrenamiento. Para medir todo el conjunto de datos, se usa la **función de costo** (J), la cual se define como:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^i, y^i) = \frac{1}{m}\sum_{i=1}^{m} -(ylog(\hat{y}) + (1-y)log(1-\hat{y})) \tag{5}$$

En donde i representa el i-ésimo ejemplo de entrenamiento.

2.4.4. Descenso del gradiente El Descenso del gradiente es uno de los algoritmos mas sencillos y conocidos debido a que solo hace uso del vector gradiente para su ejecución, por esto se clasifica como un método de primer orden. Se utiliza en la regresión logística como en redes neuronales profundas. Este método se usa para hallar el valor mínimo de la función de error con respecto a la variación de los pesos en un nodo de la red neuronal:

$$w_{i+1} = w_i - \alpha * \frac{\partial L(W,b)}{\partial W} \tag{6}$$

El hiper-parámetro α denomina la tasa de aprendizaje, i representa el i-ésimo peso en la capa de la red neuronal. La dirección en la que mayor decrece la función de error se obtiene al derivarla respecto a sus pesos asociados:

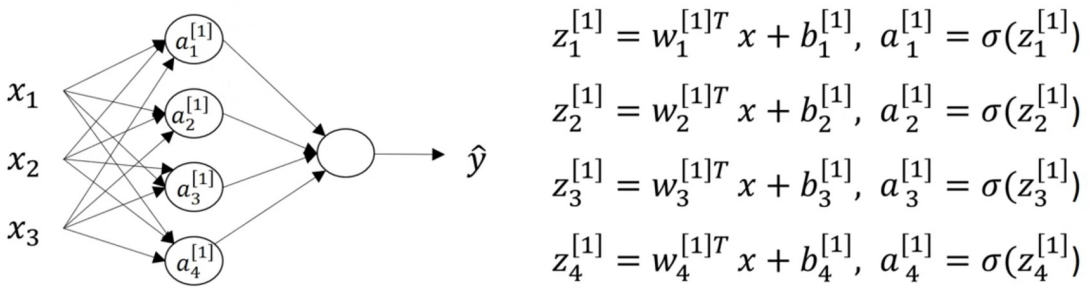
$$\bigtriangledown L(\vec{w}) = [\partial L/\partial w_0, \partial L/\partial w_1, ..., \partial L/\partial w_n]$$

Una consecuencia de implementar este algoritmo es que el aprendizaje puede converger en un punto bajo, pero no garantiza que sean los mínimos globales, los cuales indican el punto mas pequeño de la superficie de error. Aun así, esto sucede en pocas ocasiones, logrando que el descenso del gradiente alcance buenos resultados en las redes neuronales.

2.4.5. Salida de una red neuronal En la sección 2.4 se definió la regresión logística, que al final es una red neuronal con una única capa oculta. La forma operar de una red neuronal es igual, pero repetida muchas veces. Es decir las operaciones que efectúa un nodo, representada en las ecuaciones 8 y 2, se repiten tantas veces como nodos existan dentro de una red neuronal.

Como se puede observar en la figura 9, la salida de la capa intermedia (capa de 4 nodos), no es directamente la salida de la red neuronal como en regresión logística, en este caso, la salida de cada nodo, representadas como a_1, a_2, a_3, a_4 son a su vez

Figura 9. Activaciones en una red neuronal de dos capas



Fuente: Coursera; *Neural networks and deep learning; Computing a Neural Network's Output*

entradas para los nodos de la siguiente capa, en este caso la capa de salida, donde se computa $\hat{y}$.

2.4.6. Hiper-parámetros Anteriormente se nombraron los parámetros de una red neuronal, los cuales se encuentran presentes en cada nodo de la red, estos son los pesos (W) y los sesgos (b). El objetivo es ajustar (entrenar) estos parámetros de tal forma que dada una entrada a la red, se obtenga la salida correspondiente. Sin embargo durante el proceso de entrenamiento entrar a jugar otros parámetros adicionales, que dentro del aprendizaje automático, se denominan hiperparámetros:

Tasa de aprendizaje: También llamada *learning rate*, se denota como α y está presente en el descenso del gradiente (2.4.4). Determina que tan rápido "aprende" la red neuronal. Si este valor es muy grande, cada paso en el descenso del gradiente sería tan largo, que no sería posible llegar a ajustar los parámetros (ver ecuación 6), es decir, llegar al mínimo de error en la función de error; si este es muy pequeño, el entrenamiento tardaría un tiempo adicional innecesario. La manera de ajustar este parámetro es mediante ensayo y error. Los valores suelen estar comprendidos entre $0{,}01$ y $0{,}0001$.

Número de iteraciones: También llamado *epocs* ó épocas, representa la cantidad de veces que va a repetirse el conjunto de datos en la red neuronal. Este valor es directamente proporcional al tiempo de entrenamiento, así mismo, entre mas épocas se entrena un algoritmo este puede presentar mejores resultados. Sin embargo a partir de cierto valor ya no se verán cambios, por el contrario, si se tienen muchas épocas y un bajo número de ejemplos de entrenamiento, es muy probable que se produzca *overfitting* o sobre-ajuste, esto significa que la red neuronal arrojará buenos resultados haciendo pruebas con el conjunto de datos de entrenamiento, mas no con el conjunto de datos de prueba, o con nuevos ejemplos, es decir, el modelo "memoriza" los datos contenidos en el conjunto de entrenamiento.

Número de capas ocultas: Define el número de capas presentes en la red neuronal, es decir, su profundidad, en general, una red neuronal con más capas es capaz de aprender patrones mas complejos.

Elección de función de activación: La elección de la función de activación depende de la capa en la red neuronal, y del propósito para el cual está siendo entrenada.

Tamaño de lote: Más conocido como *batch size*; es la cantidad de ejemplos de entrenamiento (imágenes) que pasan a través de la red neuronal simultáneamente. Consiste en dividir el total estos en conjuntos mas pequeños llamados *mini-batches* los cuales están divididos usualmente en tamaños cuya cantidad es una potencia de 2, es decir: 2, 4, 8, ... , 256, 512, etc. Esto hace que el entrenamiento sea más rápido a costas de tomar más recursos para hacerlo efectivo, en particular, consume gran cantidad de memoria RAM. Usualmente para conjuntos de datos pequeños, el *batch size* puede ser grande, es decir, del mismo tamaño del conjunto de entrenamiento. Si el conjunto de datos es grande, el valor puede varia desde 4 hasta 512, según la

cantidad de memoria disponible.

2.4.7. Regularización La regularización es una técnica utilizada en modelos más complejos que consiste en añadir una penalización a la función de coste, con el fin de reducir el *overfitting*. Esta penalización produce modelos más simples que generalizan mejor. En la práctica, la regularización es un término que se añade a la función de costo[16].

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^i, y^i) + \frac{\lambda}{2m}||w||_2^2 \tag{7}$$

Donde λ es el parámetro de regularización. La ecuación 7 corresponde con la regularización L2.

2.5. TENSORFLOW

Es una librería de código abierto adecuado para computación numérica utilizando grafos con flujo de datos entre sus nodos y aristas. Los nodos son representaciones de operaciones matemáticas y las aristas representan los tensores, que son *arrays* de datos multidimensionales. Es desarrollada por Google y esta orientada a problemas de *Deep Learning* enfocándose en la creación de arquitectura de redes neuronales. Su arquitectura flexible permite implementar miles de algoritmos en una o varias CPUs, GPUs, servidores, dispositivos móviles y sistemas embebidos usando una sola API.

La principal estructura de datos que usa TensorFlow son los “tensores”, un tensor es un conjunto de valores primitivos organizados en un *array* de 1 o N dimensiones donde el rango es el numero de dimensiones, estos se representan tensores repre-

---

[16] Jose Martinez. *Regularización Lasso L1, Ridge L2 y ElasticNet - IArtificial.net*.

sentan la información del problema y en esta estructura se aplican los algoritmos para hacer las operaciones y transformaciones sucesivamente, como es el caso de las capas de una red neuronal.
Tensorflow esta implementado en C++ y Python, una de las maneras mas cómodas de usarlo es a través de API que ofrece Python.

2.6. KERAS

Es una librería de código abierto escrita en python. Funciona como un modelo, proporcionando bloques modulares donde se pueden desarrollar modelos complejos de aprendizaje profundo. Keras no funciona como un *framework*, sino como una interfaz de uso intuitivo (API), accediendo a diferentes *frameworks* de aprendizaje automático vinculados. Su estructura modular que relaciona las capas de la red neuronal, implica que el usuario no debe tener el conocimiento o controlar directamente el propio *backend* del *framework* elegido. Todo esto hace que la curva de aprendizaje de esta API sea muy suave, en comparación de otras herramientas que se usan para la implementación de redes neuronales.

Keras es un lenguaje de alto nivel basado en las herramientas **tensorflow**, **theano** y **Microsoft cognitive toolkit** las cuales tienen interfaces que son de rápido acceso y muy intuitiva al ***backend*** correspondiente, por ello no es necesario elegir un único ***framework*** porque se puede cambiar el ***backend*** de forma sencilla. La mayoría de aprendizaje profundo consiste de unir capas sencillas, además del soporte para las redes neuronales estándar, Keras ofrece soporte para las Redes Neuronales convolucionales y para las Redes Neuronales Recurrentes.
Para construir un modelo en Keras, primero se deben configurar sus capas, por ejemplo, un modelo sencillo con una capa de entrada, una intermedia y una de salida, como se puede apreciar a continuación:

```
    model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(X, Y, Z)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

$tf.keras.layers.Flatten$: Esta función crea una capa de entrada, la cual se encarga de transformar la matriz (imagen) de entrada; tridimensional para imágenes RGB, bidimensional para imágenes en escala de grises, en un único vector plano, en esta capa no se entrena ningún parámetro, solo se modifican los valores de entrada.

$tf.keras.layers.Dense$: Esta función crea una capa intermedia convencional, se especifica la cantidad de unidades ocultas o nodos que va a contener y la función de activación.

$tf.keras.layers.Conv3D$: Crea una capa convolucional, en donde la entrada es convolucionada con un filtro y su salida es un mapa de características. Igual que en la capa dense, se especifica una función de activación

2.7. TRANSFERENCIA DE CONOCIMIENTO (TRANSFER LEARNING)

Uno de los recursos más importantes dentro del *Deep Learning* es el *Transfer Learning* o transferencia de conocimiento, el cuál se basa en el uso de patrones o modelos que han sido entrenados para una tarea; y reutilizarlos en otras similares, estos modelos se conocen como modelos pre-entrenados. Esta técnica ahorra mucho tiempo de trabajo y una excesiva magnitud de cálculo, al usar los pesos y los sesgos de un modelo ya entrenado y ajustando los pesos de las últimas capas. Entre los modelos más conocidos y precisos para hacer *transfer learning* se destacan: ResNet, Inception, MobileNet, VGG16. Después de escoger el modelo pre-entrenado

mas adecuado para el problema, se ajustan sus parámetros según las muestras que hay para entrenar el modelo y la potencia computacional. Tres estrategias a las que se pueden recurrir son:

- Entrenar todo el modelo: Utilizando la arquitectura del modelo pre-entrenado comenzando el entrenamiento de las redes neuronales desde cero.
- Entrenar algunas capas y congelar otras: Dependiendo de las características del problema se dejan congeladas las capas superiores que son las tienen la capacidad de abstraer características específicas o las inferiores que abstraen características generales.
- Congelar la base convolucional: Si el modelo pre-entrenado posee un mecanismo de extracción de características análogas a los que aborda el problema, se mantiene la base convolucional original y luego se usa las salidas para alimentar el clasificador propio.

2.7.1. Extractores de características

ResNet: Es una arquitectura CNN que puede agregar una gran cantidad de capas con un rendimiento sólido. Estas redes neuronales se entrenan por medio del proceso de retro-propagación, la cuál consiste en cambiar los pesos de las neuronas para reducir al mínimo la función de error. ResNet analiza las capas que inicialmente no hacen nada, las omite y reactiva capas anteriores; es decir, al iniciar el entrenamiento comprime la red para hacerlo mas rápido y a medida que avanza el entrenamiento va expandiendo las capas con el fin de mejorar la exploración y así las características de la imagen. El salto entre capas elimina las complicaciones, usando pocas capas y haciendo la red mas sencilla, esto a su vez acelera el aprendizaje y reduce los efectos de la desaparición de los gradientes.

Uno de los principales inconvenientes que hace difícil ejecutar ResNet, es que puede tener entre docenas y miles de capas convolucionales, lo que implica mucho tiempo para entrenar y ejecutar, razón por la cual, esta arquitectura no se puede implementar en dispositivos móviles. Keras se ha encargado de capacitar la arquitectura de Resnet con el conjunto de datos de ImageNet. Por lo que se puede aprovechar el aprendizaje por transferencia. Las aplicaciones proporcionadas por Keras son: ResNet V1 y ResNet V2 con 50, 101 o 152 capas y ResNeXt con 50 o 101 capas.

Inception: La red *Inception* o popularmente conocida como *GoogleNet*, es una de las arquitecturas CNN que consiste en la utilización de métodos como: Convolución 1x1, Agrupación promedio global, Módulo de inicio, Clasificador auxiliar para entrenamiento, esta arquitectura general tiene 22 capas de profundidad y fue diseñada teniendo en cuenta la eficiencia computacional.

VGG16: La particularidad de esta arquitectura CNN es que, en vez de tener una gran cantidad de hiperparámetros, se centra en que a medida que aumenta la profundidad, es decir, a medida que nos movemos en la red, se va añadiendo más y más capas convolucionales en cascada en los cinco conjuntos de capas convolucionales. En esta arquitectura se empieza con un tamaño de canal de 64, que luego, va aumentando en un factor de 2 después de cada capa, hasta llegar a una agrupación máxima de 512.

MobileNet: MobileNet se caracteriza por usar convoluciones separables en profundidad, excepto por la primera capa, para reducir el tamaño del modelo y el cálculo, en otras palabras, usa redes neuronales profundas y livianas que pueden ser implementadas en dispositivos móviles o sistemas embebidos. La convolución separable en profundidad se compone de dos operaciones: la convolución en profun-

didad, que se encarga de aplicar convoluciones por separado a cada tensor de la entrada y la convolución por punto, que es una convolución tradicional 1x1 aplicada al tensor resultante. Otra característica de esta arquitectura es que todas las capas son seguidas de *Batch Normalization* y tiene una función de activación *ReLU*, exceptuando la última capa *fully-connected*. MobileNets introduce dos nuevos hiperparámetros globales (multiplicador de ancho y multiplicador de resolución) que permiten a los desarrolladores de modelos variar entre latencia o precisión por velocidad, dependiendo de en sus requisitos. MobileNet cuenta con 28 capas, El tamaño de la imagen de entrada es 224×224×3.

2.8. YOLOV2

Yolo es un algoritmo para el reconocimiento de objetos que se destaca principalmente en términos de velocidad, siendo capaz de reconocer objetos a altas cantidades de imágenes por segundo en redes pequeñas. Aunque en términos de precisión *mAP (mean average precision*) Yolo posee una precisión media de alrededor del 70 %.

En Yolo, la imagen de entrada es dividida en una cuadrícula S x S, en donde cada celda de ésta se encarga de predecir un número determinado B de cuadros delimitadores con unas componentes asociadas: $[x, y, w, h, confianza]$, donde x y y denotan la posición central de cada cuadro, w y h sus dimensiones. Por último, el índice de confianza, cada índice de confianza expresa la probabilidad de que dicho cuadro delimitador contenga un objeto de interés[17].

---

[17] Joseph Redmon y col. "You only look once: Unified, real-time object detection". En: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2016-Decem. 2016, págs. 779-788. DOI: 10.1109/CVPR.2016.91. arXiv: 1506.02640.

YoloV2 es una versión mas precisa y rápida. Esto se debe a que YoloV2 usa técnicas que no se usaron anteriormente como Batch-Normalization y Anchor-Boxes.

Batch Normalization o normalización del lote, es un método que normaliza las salidas en cada una de las capas ocultas. Esto acelera el proceso de aprendizaje. Los *Anchor-Boxes*, son cuadros delimitadores predefinidos con cierta altura y ancho. Estos cuadros tienen una escala en relación al aspecto del objeto que se desea detectar, por ejemplo, si se quieren detectar carros y personas, los *anchor-boxes* deseables serían verticales para personas, y horizontales para los autos. Esto hace que la predicción sea mucho más rápida.

Para la detección de objetos también se necesita predecir su ubicación y su forma. por lo que la salida de la red es una matriz tridimensional (o Tensor en *TensorFlow*). La salida de YoloV2 es de $13x13xD$, en donde D depende de cuantas clases de objetos se quiere detectar. Cada elemento esta matriz bidimensional (13x13) se llama celda de cuadrícula.

Cada celda en la cuadrícula tiene una profundidad D. El valor de D es directamente proporcional a las clases C, y al numero de cuadros delimitadores B, el 5 simboliza la predicción de la posición del cuadro delimitador y la probabilidad de que exista un objeto en este cuadro:

$$D = B(5 + C) \tag{8}$$

Esto es suficiente para detectar objetos grandes. Sin embargo, si se busca detectar objetos más finos, la arquitectura se puede modificar de tal forma que la salida de la capa anterior 26x26x512 a 13x13x2048 se concatene con la capa de salida original

13x13x1024.

YoloV2 basa su arquitectura convolucional en *Darknet-19*, Este modelo consta de 19 capas de convolución que principalmente usan filtros 3x3 con 5 capas de agrupación máxima, también utiliza filtros 1x1 para reducir los canales de profundidad entre las convoluciones 3x3.

2.9. OPENCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de software de visión artificial y aprendizaje automático de código abierto. Incluye un gran numero de algoritmos de visión con computadora. Esta centrado principalmente en el procesamiento de imágenes, la captura y el análisis de vídeo. La biblioteca posee mas de 2500 algoritmos optimizados que pueden usarse para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en videos, rastrear movimientos de cámara. Tiene interfaces C ++, Python, Java y MATLAB y es compatible con Windows, Linux, Android y Mac OS. OpenCV se inclina principalmente hacia aplicaciones de visión en tiempo real [18].

[18] OpenCV. *About OpenCV*. 2019.

3. SISTEMAS DE DESARROLLO

Para la implementación del modelo entrenado se proponen dos opciones: La raspberry pi, debido a su fuerza de computo en un reducido tamaño, su versatilidad y gran documentación existente; y la tarjeta Sipeed Maixduino, una propuesta China lanzada apenas en junio de 2019, que cuenta con un acelerador de hardware para redes neuronales. Como veremos a continuación, ambas opciones son apropiadas para este trabajo, con algunos puntos a favor y en contra para cada una de ellas.

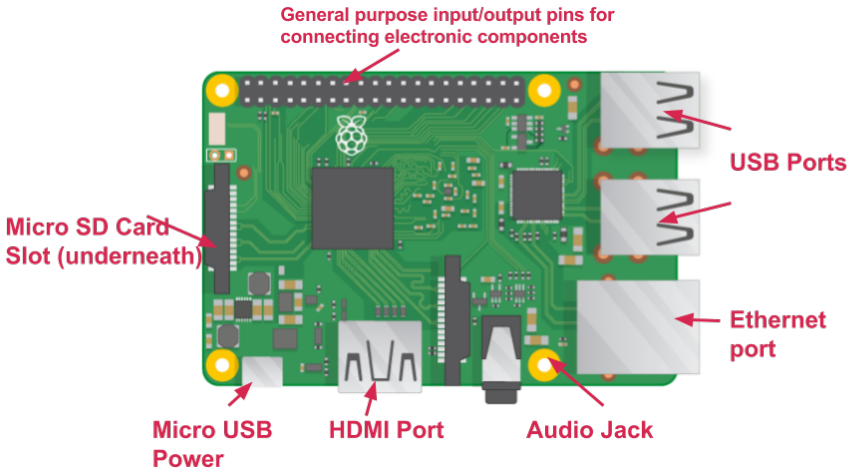
3.1. RASPBERRY PI

La Raspberry PI es una placa de computadora *Single Board Computer (SBC)*, con software de código abierto, que es capaz de comportarse como un ordenador común y permite programar y compilar programas que se ejecuten en él. Su tamaño reducido le dan una gran versatilidad y posibilidad de conectar varios accesorios como teclado, monitor, mouse y componentes de un computador común, esto permite su uso en distintas áreas [19]

---

[19] Gareth Halfacree. *THE OFFICIAL Raspberry Pi Beginner's Guide How to use your new computer*. 2018.

Figura 10. Rapsberry Pi



Fuente: https://www.bigtronica.com/centro/content/9-introduccion-gpio

Para que la Raspberry Pi funcione, necesita un sistema operativo. El mas recomendado actualmente es el Raspbian el cual es un sistema operativo libre y optimizado para el hardware de la Raspberry Pi, entre sus ventajas encontramos facilidad en el modo de uso y buen soporte al usuario. La Raspberry Pi 3 Modelo B+, que es el modelo usado en este proyecto cuenta con las siguientes especificaciones: [20]

---

[20] JA Serrano Vázquez. *Raspberry Pi 3 Model B+*. 2019.

Tabla 1. Especificaciones Raspberry Pi 3B+

Raspberry Pi Modelo B+	
CPU	BCM2837B0 quad-core A53 64-bit @ 1.4GHz
GPU	Broadcom Videocore-IV.
Arquitectura	ARMv8
RAM	1GB LPDDR2 SDRAM 7.6.5 Windows
Almacenamiento	Tarjeta micro-SD
Wi-Fi	2.4GHz and 5GHz 802.11b/g/n/ac
Bluetooth	Bluetooth 4.2 & Bluetooth low energy (BLE),
GPIO	40-pin GPIO header
Puertos	HDMI, jack 3.5mm, 4 USB 2.0, Ethernet, CSI, DSI
Dimensiones	82mm x 56mm x 19.5mm, 50g
Precio	$169.227,875 COP

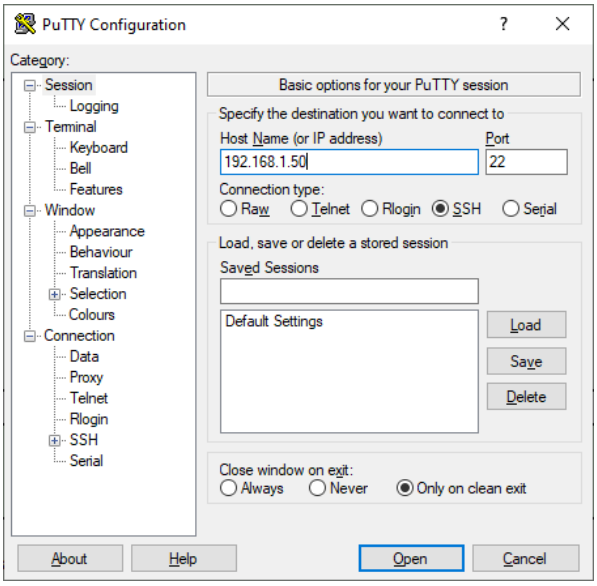
Fuente:
https://magpi.raspberrypi.org/articles/raspberry-pi-3bplus-specs-benchmarks

3.1.1. Protocolo SSH Es un protocolo de administración remota que permite la comunicación segura entre dos sistemas. Esto se logra por su arquitectura cliente/-servidor.

Primero, se debe habilitar el servicio SSH en la Raspberry, ya que este viene desactivado por defecto por cuestiones de seguridad. Windows necesita un programa específico ya que este no posee un cliente SSH, para esto, Putty es una herramienta muy útil para acceder a otros sistemas que sean o no compatibles con el formato que se está usando. Conociendo la dirección IP de la raspberry, es posible conectar con la terminal. Para acceder al escritorio, se puede hacer con el programa de

software libre VNCViewer[21]. De esta manera, se evita conectar un monitor, teclado y ratón para hacer uso de la raspberry.

Figura 11. PuTTY



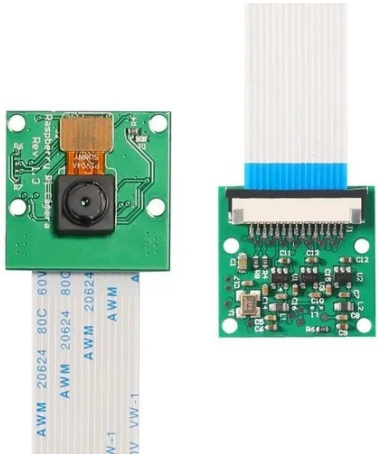
Fuente: https://turinconinformatico.com/raspberrypi/conectarse-por-ssh/

El sensor usado con la raspberry es la cámara Raspberry Pi de 5 megapixeles (ver figura 12).

3.1.2. Implementación en Raspberry Pi Para implemetnar el modelo en la raspberry, se instaló el sistema operativo Raspberry Pi OS, una distribución del sistema operativo GNU/Linux basado en Debian; se activo el servicio SSH y el acceso a la cámara desde la configuración de la raspberry. Raspberry Pi Os permite instalar y ejecutar el interprete de TensorFlow Lite. Este paquete es una fracción del paquete completo de TensorFlow e incluye el código mínimo necesario para ejecutar inferen-

[21] Diana C. *¿Qué Es El Protocolo SSH Y Cómo Funciona?*

Figura 12. Cámara Raspberry Pi

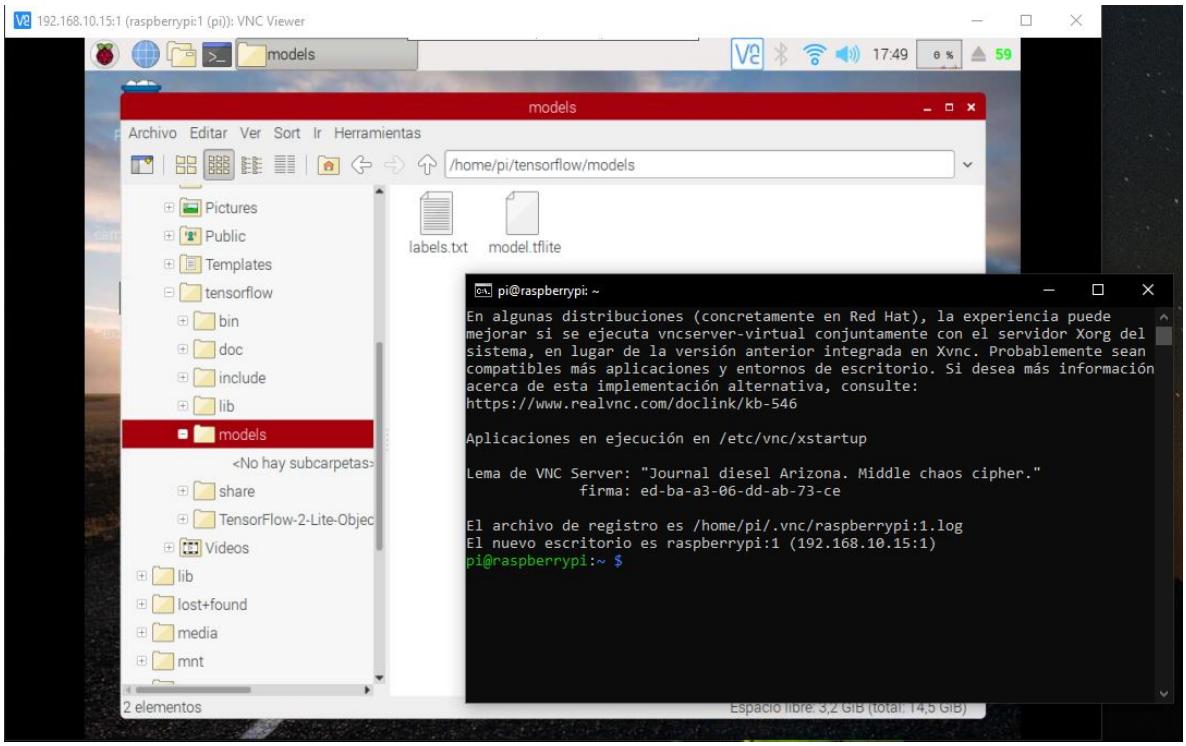


cias con TensorFlow Lite; incluye solo la clase tf.lite.Interpreter de Python. Tambien se instaló OpenCV para realizar la clasificación y visualizar las inferencias realizadas por el modelo. A continuación se muestran los comandos necesarios para la instalación de OpenCV y el intérprete de tensorflow, desde la consola de comandos del sistema operativo Raspberry Pi OS:

```
#Instalación de OpenCV
sudo apt-get install libhdf5-dev libhdf5-serial-dev libhdf5-103
sudo apt-get install libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5
sudo apt-get install libatlas-base-dev
sudo apt-get install libjasper-dev
pip install opencv-python==4.1.0.25

#Interprete de tensorflow
pip install https://github.com/google-coral/pycoral/releases/download/
release-frogfish/tflite_runtime-2.5.0-cp35-cp35m-linux_armv7l.whl
```

Con esto, la tarjeta cuenta con lo necesario para realizar inferencias sobre el modelo que se entrenará para la clasificación.

Figura 13. Acceso remoto a la Raspberry pi mediante VNC Viwer



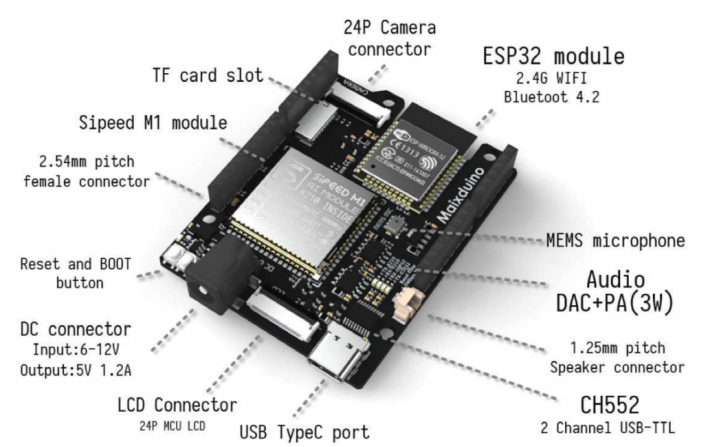
3.2. SIPEED MAIXDUINO

Maixduino es una placa de desarrollo RISC-V 64 para aplicaciones *AI + IoT*. A diferencia de otros desarrollos de Sipeed Maix. Maixduino fue diseñada basándose en Arduino. Además incluye en la misma placa, un módulo ESP-32 y un acelerador de *hardware* para redes neuronales: el módulo Kendryte K210.

El Kendryte K210 es un *System on a chip* (SoC) que integra visión artificial y reconocimiento de audio. Usa el proceso de 28 [nm] de consumo ultrabajo de la empresa TSMC con procesadores doble núcleo de arquitectura de 64 bits que mejoran la eficiencia energética, estabilidad y confiabilidad.

En términos de informática de IA, la potencia de cálculo del K210 es bastante impresionante. Según la descripción oficial de Jia Nan, el KPU del K210 tiene una potencia de 0.8 teraflops (operaciones de coma flotante por segundo).

Figura 14. Tarjeta de desarrollo Sipeed Maixduino



Fuente: https://www.seeedstudio.com/Sipeed-Maixduino-Kit-for-RISC-V-AI-IoT-p-4047.html

A comparación de otros *System on a chip (Soc)* de gama alta, la tarjeta Maixduino tiene una potencia de cómputo considerable, gracias a que cuenta con el módulo Kendryte K210, que alcanza una potencia aproximada de 0.8 Tera-FLOPS (operaciones de coma flotante por segundo).

A pesar de que Sipeed ofrece otras tarjetas que incorporan el módulo Kendryte K210 (como Maix-bit, Maix-Go o Maix-dock), se seleccionó esta por gusto personal, ya que es una tarjeta que puede ofrecer muchas más cosas gracias a que tiene integrada un ESP-32. Otras tarjetas usadas en inteligencia y visión artificial se puede ver en la tabla 3, siendo el módulo K210 la mejor opción por su rango de precio (29

Tabla 2. Especificaciones Sipeed Maixduino

Sipeed Maixduino	
CPU	RISC-V Dual Core 64bit, con FPU @ 400MHz
Puertos	USB tipo C, 2x 24P FPC connector, Tf card, 1.25mm pitch speaker connector, 2.54mm pitch female connector,DC connector
Entorno de desarrollo	Arduino IDE
WIFI	2.4G WIFI
Bluetooth	Bluetooth 4.2
Micrófono	Micrófono MEMS omnidireccional
Almacenamiento	Tf card
Cámara	OV2640 2MP
Dimensiones	68mm x 54mm
Software	FreeRtos & Standard SDK, MicroPython, Machine vision, Machine hearing
Entrada de alimentación	6-12 V
Potencia nominal de alimentación	>3 W
Precio	$83.586,47 COP

Fuente: https://store.comet.bg/download-file.php?id=19012

USD para la Sipeed Maixduino) [22].

[22] Sipeed. *Sipeed Maixduino Kit for RISC-V AI + IoT*.

Tabla 3. Comparativa entre el modulo K210 y otro módulos para IA

Especificación	K210	ESP32	Open MV M7	Pixy 2
CPU cores	2	2	1	2
Frecuencia	400 MHz	160 MHz	216 MHz	204 MHz
Hardware para RN	Si	No	No	No
Wi-Fi	No	Si	No	No
Bluetooth	No	Si	No	No
RAM	8192 kB	512 kB	512 kB	264 kB
FLASH	16 MB	16 MB	2 MB	2 MB
Pines GPIO	48	36	10	6
Pines ADC	0	18	1	0
Buses	SPI,I2C,I2S	SPI,I2C,I2S, UART,CAN	SPI,I2C,CAN	SPI,I2C,USB

El sensor utilizado con esta tarjeta es el OV2640 de 2 megapixeles (ver figura 15).

Figura 15. Sensor OV2640

3.2.1. Implementación en Maixduino La implementación en la tarjeta Sipeed requiere de un proceso de configuración previo a trabajar con ella. el primero es la instalación del controlador, la tarjeta Sipeed Maixduino tiene el microcontrolador CH552. Para lograr la comunicación mediante un puerto serial con la tarjeta, es necesario la instalación del controlador FT2232. La memoria flash tarjeta no tiene

ningún *firmware* instalado, por lo que este debe ser descargado e instalado de la página web de Sipeed, se seleccionó la versión 0.5.0_29.

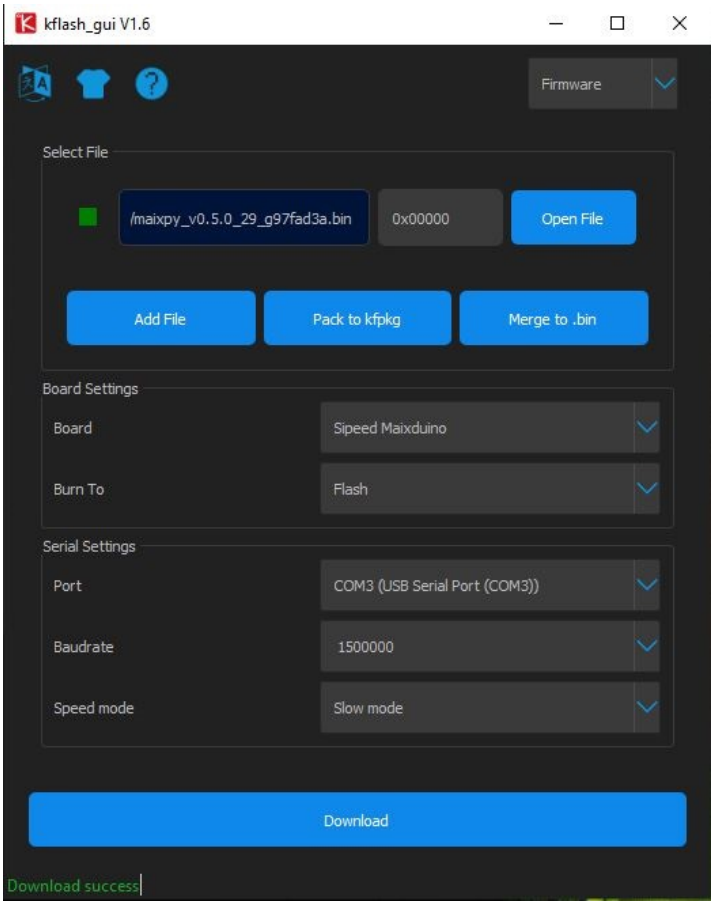
Figura 16. *Firmware* seleccionado

Tomado de: https://dl.sipeed.com/MAIX/MaixPy/release/master/

La herramienta kflash permite grabar el *firmware* en la ubicación de la memoria flash correspondiente (0x00000), así como también para grabar modelos de inteligencia artificial, esta esta disponible en el repositorio de Sipeed. Para las tarjetas Sipeed que tienen el módulo kendryte K210, se debe llevar el modelo creado anteriormente en formato .tflite al formato kmdel, que es el formato compatible con la tarjeta que aprovecha el acelerador de hardware para redes neuronales.

Finalmente, se descargó la versión 0.2.5 del IDE oficial de MaixPy, un IDE muy intuitivo que permite la compilación de *scripts* en el lenguaje micropython.

Figura 17. Herramienta kflash



4. METODOLOGÍA Y DESARROLLO

El objetivo general del proyecto es identificar y clasificar piezas mecánicas según su tipo y tamaño, con el fin de ser usado en una banda transportadora. La clasificación de objetos se llevará a cabo usando inteligencia artificial, para la clasificación se usará tratamiento de imágenes.

4.1. CONJUNTO DE DATOS *(DATASET)*

Para la obtención del conjunto de datos es posible usar los existentes y disponibles en internet. Sin embargo, en este caso, no se encontró un conjunto de datos apropiado por lo que se optó por elaborar uno propio. Para la creación este, se tomaron un cantidad considerable de fotos de piezas mecánicas, el conjunto de imágenes está enfocado exclusivamente para la clasificación de pernos, arandelas y tuercas.

Existen una gran variedad de este tipo de piezas que poseen diferentes tamaños y formas. Se observa a continuación algunas imágenes usadas para la creación de la base de datos.

Figura 18. Pernos

Figura 19. Tuercas

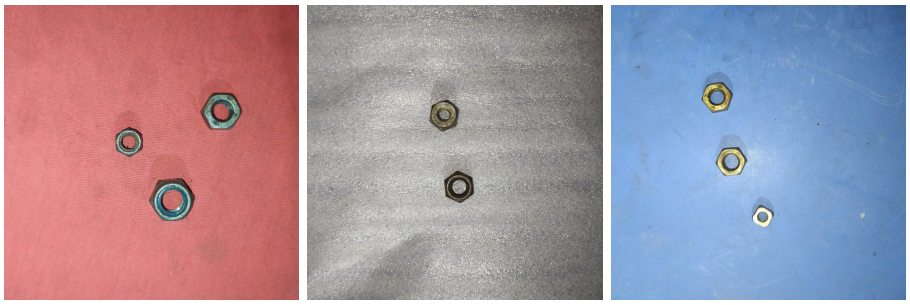


Figura 20. Arandelas

Para la captura de las imágenes se usaron cámaras de teléfonos celulares. Sin embargo, la calidad de estas fotografías es alta en comparación a la de una tomada por cualquiera de los sensores disponibles para las tarjetas. Así mismo, el entrenamiento esta asociado al tamaño de las imágenes de entrada, por lo que conlleva mas tiempo y el uso de mas recursos. Dicho esto, se hace necesario reducir la calidad del conjunto de imágenes.

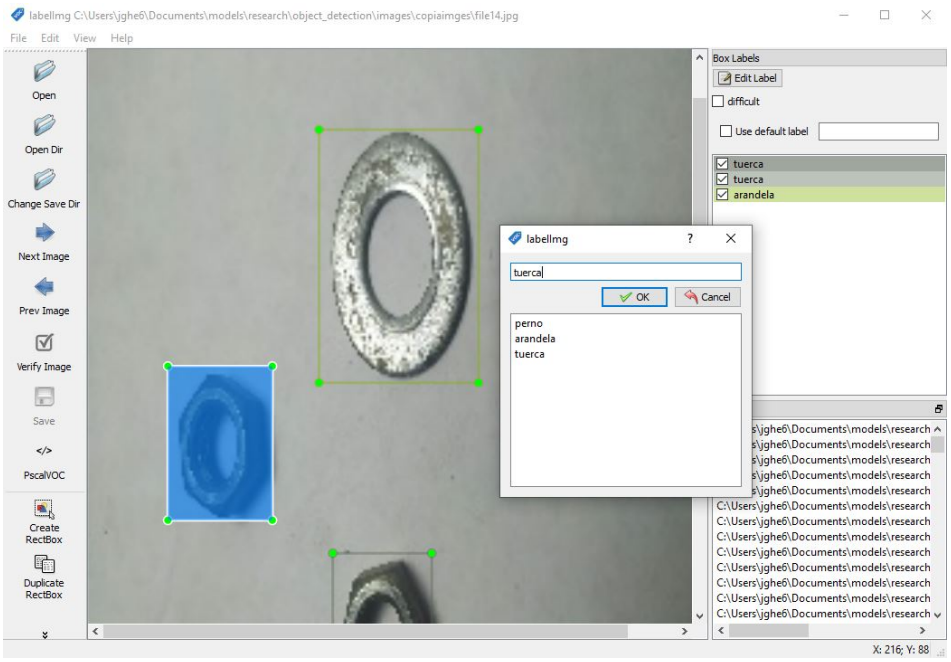
4.1.1. Compresión de imágenes El tamaño elegido para las imágenes fue de 240x320 píxeles, se usó esta resolución con el fin de entrenar un modelo robusto, en donde una baja resolución en las imágenes de entrada no sea un problema; con un peso aproximado de 20 kB por imagen, y donde la calidad no se baja lo suficiente como para perder información importante. Para comprimir las imágenes se uso Python, con ayuda de las librerías numpy, opencv, os.

Figura 21. Código en python para comprimir las imágenes

```python
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2

directory = "Drivefotos/Tuercas"

img_y= 320
img_x= int(img_y*0.75)
indice = 989
path = os.path.join(directory)  # Crear path a imagenes
for img in os.listdir(path):  # Iterar sobre cada imagen
    img_array = cv2.imread(os.path.join(path,img) )

    new_img = cv2.resize(img_array, (img_x, img_y))
    cv2.imwrite(r"C:\Users\jghe6\Documents\Drivefotos\TuercasC\file{}.jpg".format(indice), new_img)
    indice = indice + 1;


```

4.1.2. Etiquetado de imágenes La detección de objetos hace parte de el aprendizaje supervisado, por lo que para entrenar un algoritmo es necesario que las imágenes tengan sus respectivas etiquetas. En este caso se debe etiquetar cada imagen, especificando la sección de la misma donde se encuentran los objetos de interés. Para llevar a cabo el etiquetado, se usa la herramienta *LabelImg*, el uso de esta herramienta se puede observar en la figura 22.

Figura 22. Herramienta de etiquetado LabelImg



Una vez etiquetadas todas las imágenes, está completo el conjunto de datos, cada imagen tendrá un archivo en formato .xml, que contiene la información de importancia dentro de cada una de ellas.

En total, la base de datos obtenida es de 1856 etiquetas, distribuidas en 1122 imágenes. Donde alrededor del 90 % de las imágenes se usan para entrenar el modelo y el 10 % restante se usan para validar el sistema.

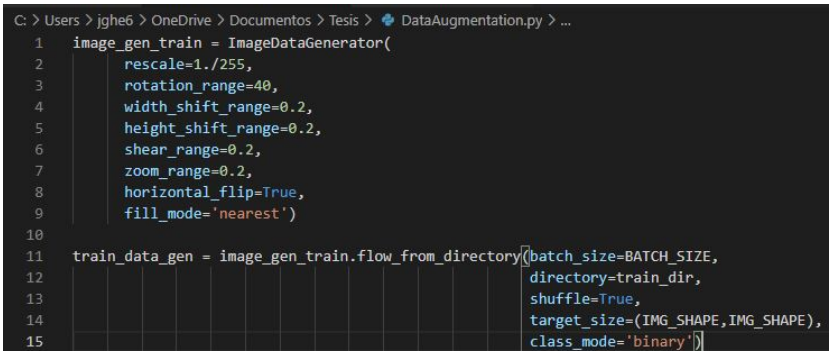
Tabla 4. Objetos etiquetados

Pieza	Entrenamiento	Prueba	Total
Pernos	670	35	705
Arandelas	514	43	557
Tuercas	548	46	594
		Total	1856

4.1.3. Aumento de datos El aumento de datos consiste en generar mas imágenes a partir de la muestras de entrenamiento existentes mediante transformaciones

aleatorias como invertir, rotar y aplicar zoom a la imgen. Keras permite realizar este proceso usando su clase *ImageDataGenerator*, como se muestra en la figura 23.

Figura 23. Aumento de imágenes con keras



4.2. ENTRENAMIENTO DEL MODELO

4.2.1. Tensorflow API Es una interfaz de programación de aplicaciones (API de sus siglas en inglés) la cual ofrece modelos que han sido previamente entrenados de diferentes arquitecturas. También permite re-entrenar dichos modelos para detectar objetos que desee el usuario. Existen repositorios de estas API que manejan tanto tensorflow 1 como tensorflow 2. Algunos de estos modelos se puede observar en la tabla 5.

Tabla 5. Velocidad vs. mAp de Modelos TFODAPI

Modelo	Velocidad (ms)	mAp
CenterNet HourGlass104 512x512	70	41.9
EfficientDet D0 512x512	39	33.6
SSD MobileNet V2 FPNLite 320x320	22	22.2
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	46	34.3
Faster R-CNN ResNet50 V1 1024x1024	65	31
Faster R-CNN Inception ResNet V2 1024x1024	236	38.7

Fuente: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

Los valores de tiempo son una referencia a la velocidad de cada modelo, y fueron tomados usando una tarjeta gráfica TITAN X. A continuación se presenta como se realizó el entrenamiento tanto en la API de tensorflow, y posteriormente, con el marco de trabajo aXeleRate.

4.2.2. Entrenamiento local con la API de tensorflow 2

Adecuación del entorno: En primer lugar, para llevar a cabo el entrenamiento mediante el API de tensorflow a nivel local, es necesario tener un ordenador que cuente con una tarjeta de gráficos (GPU) Nvidia, así como las versiones adecuadas de CUDA y CuDNN, y los controladores de la GPU actualizados. Todo esto se puede descargar desde la pagina web oficial de Nvidia, y en la pagina oficial de desarrolladores de Nvidia. En la tabla 6 describe el hardware y software usados para este trabajo.

Tabla 6. GPU & Herramientas CUDA

Requerimiento	Versión
GPU Nvidia	GTX1650S 4GB
CUDA Toolkit	10.1 Windows
CuDNN	7.6.5 Windows

Posteriormente se instalaron múltiples paquetes y librerías, para lo cual se usó de la herramienta **Anaconda**. Anaconda es una distribución libre de los lenguajes Python y R, esta permite crear administrar paquetes mediante la configuración de nuevos entornos virtuales, previniendo posibles inconvenientes de incompatibilidad de versiones al tener ambientes independientes para cada proyecto.

Una vez instalado anaconda, accedimos a su ventana de comandos, creamos y activamos un nuevo entorno virtual con Python 3.8.5; el cual llamaremos “Tensorflow", luego se instaló tensorflow en su versión 2.3.1.

```
conda create -n tensorflow pip python=3.8
conda activate tensorflow
pip install tensorflow==2.3.1
```

Con tensorflow instalado, se clonó el repositorio de GitHub con la API de tensorflow para detección de objetos y con la herramienta “protocolbuffers" se compilaron algunos *protbuffs* que son archivos necesarios para el entrenamiento, usando el siguiente comando:

```
protoc object_detection/protos/*.proto --python_out=.
```

Posteriormente se procedió a instalar la API en sí, esto es, todos las librerías necesarias para su correcto funcionamiento. Para esto, se debe clonar : el repositorio en

GitHub de la API de tensorflow, `https://github.com/tensorflow`; y luego configurarse mediante el archivo de configuración llamado “setup.py” que se encuentra dentro del repositorio en la ruta “tensorflow/models/research/object_detection/packages /tf2/setup.py”. Para esto, vamos a la carpeta $research$ terminal CMD con el entorno virtual activado y ejecutamos los siguientes comandos:

```
cp object_detection/packages/tf2/setup.py .
python -m pip install
```

Las librerías necesarías que fueron instaladas se pueden observar en la figura 24.

Figura 24. setup.py con librerías necesarias

```
# Note: adding apache-beam to required packages causes conflict with
# tf-models-offical requirements. These packages request for incompatible
# oauth2client package.
REQUIRED_PACKAGES = [
    # Required for apache-beam with PY3
    'avro-python3',
    'apache-beam',
    'pillow',
    'lxml',
    'matplotlib',
    'Cython',
    'contextlib2',
    'tf-slim',
    'six',
    'pycocotools',
    'lvis',
    'scipy',
    'pandas',
    'tf-models-official'
]
```

Preparación del conjunto de datos: Teniendo el conjunto de datos y sus respectivas anotaciones (4.1), se dividirán en dos partes, el conjunto de datos de entrenamiento y el conjunto de datos de prueba.

Conjunto de datos de entrenamiento: Corresponde a la mayoría de los datos, estos son los que se usarán durante el proceso de aprendizaje para ajustar los parámetros

del modelo, en *datasets* es norlmamente del 80 % al 90 % de los datos totales, mientras que en *datasets* mas grandes (1 millón de ejemplos), es del 99 % o incluso más.

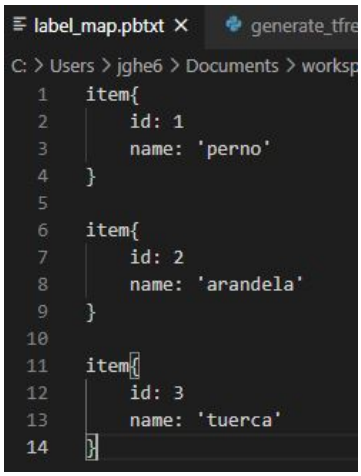
Conjunto de datos de prueba: Es un conjunto que se utiliza únicamente para evaluar el rendimiento del modelo, para esto, el modelo final toma datos de este conjunto y las predicciones realizadas se comparan con las clasificaciones reales, de esta forma se optiene la precisión del modelo. Su proporción suele ser de 10 % al 20 % en conjuntos de datos pequeños, y del 1 % o menos en conjuntos de datos mas grandes.

Una vez dividido el *dataset*, toda su información se debe comprimir en un archivo de extensión que tensorflow pueda comprender, este es un archivo de formato TFRecord, un formato simple para almacenar una secuencia de registros binarios. Este se crea a partir de imágenes y anotaciones, usando python.

Creación del mapa de etiquetas: Tensorflow requiere de un archivo donde se especifican los nombres o etiquetas de los objetos que se quieren detectar, el archivo se nombra label_map.pbtxt y su contenido viene dado por las etiquetas de nuestros 3 objetos (Ver figura 25).

Selección del modelo: Como ya se mencionó antes, la API de tensorflow ofrece variedad de modelos con diferentes arquitecturas. La arquitectura elegida fue “SSD MobileNet V2 FPNLite 320x320". Donde SSD hace referencia a las siglas en inglés *Single Shot Detector*; es una arquitectura diseñada para funcionar de manera óptima en dispositivos móviles y sistemas embebidos (como Raspberry Pi), ya que soporta el formato TFLite. Su arquitectura contiene una capa inicial completa de convolución con 32 filtros, seguida de 19 capas de cuello de botella con función de

Figura 25. Archivo de etiquetas: label_map.pbtxt



activación *ReLu*[23]. Esta arquitectura de red ha sido entrenada con la base de datos ImageNet. Como resultado, la red ha aprendido representaciones de características para una amplia gama de objetos, lo que la hace ideal para uso en *trasnfer learning*. El tamaño de la capa de entrada originalmente es de 224x224.

Configuración y entrenamiento: El archivo descargado desde el repositorio de github de tensorflow, contiene un archivo *checkpoint* que contiene el ultimo punto de entrenamiento que se hizo sobre la red pre-entrenada, un modelo congelado del mismo “saved_model.pb¨ y un archivo de configuración “pipeline.config"; este ultimo es un archivo extenso en el cual se especifican hiperparámetros y rutas. Las modificaciones hechas sobre este archivo se presenta a continuación:

- `num_classes: 3 # Numero de clases totales`
- `batch_size: 4 #Es bajo puesto que no se dispone de mucha memoria gráfica`

[23] Mark Sandler y col. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. En: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2018, págs. 4510-4520. DOI: 10.1109/CVPR.2018.00474. arXiv: 1801.04381.

```
- fine_tune_checkpoint: "models/my_SSD_mobilenet_v2_FPNLite_320x320
  /ckpt-0" #Ruta al checkpoint
- num_steps: 50000 #Número de pasos a entrenar
- fine_tune_checkpoint_type: "detection" #Detección de objetos
- label_map_path: "datas/label_map.pbtxt" #Ruta a label_map
- input_path: "datas/train.record" #Ruta a train.record
- label_map_path: "datas/label_map.pbtxt" #Ruta a label_map
- input_path: "datas/test.record" #Ruta a test.record
```

Este archivo de configuración completo y los códigos utilizados para llevar a cabo el entrenamiento se pueden encontrar en `https://github.com/JrHdez/PATObjectDetection/blob/main/models/SSD_mobilenet_v2_FPNLite_320x320/pipeline.config`

Los requisitos para poder empezar el entrenamiento son:

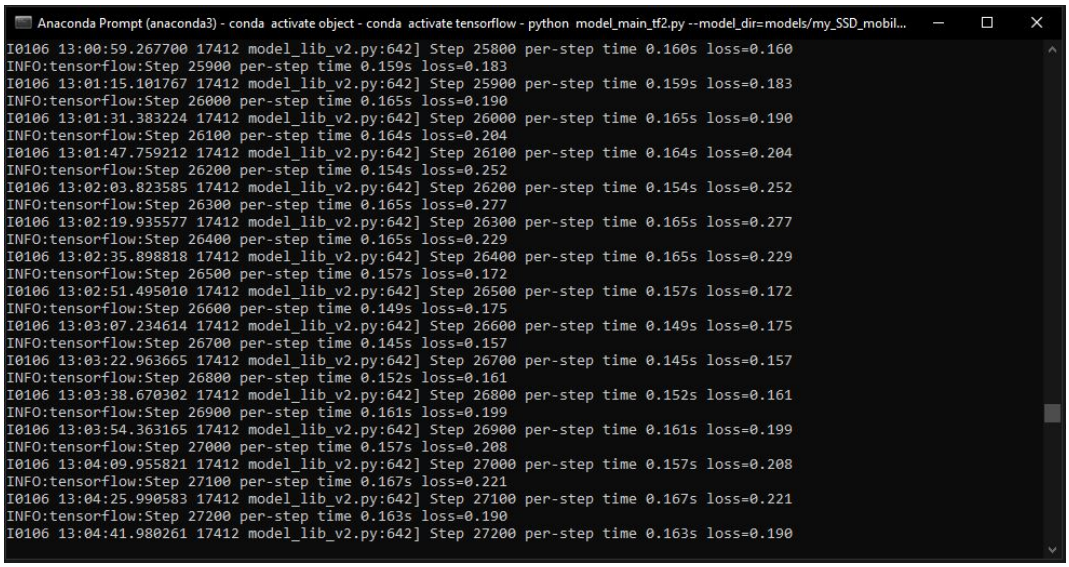
- Modelo: *checkpoint* y archivo de configuración.
- Archivos de datos: train.record, test.record y label_map.pbtxt

Para re-entrenar el modelo, se ejecuta desde la terminal con el ambiente virtual activado el archivo “model_main_tf2.py", al cual se le debe especificar la ruta del modelo de salida y la ruta del archivo de configuración.

```
python model_main_tf2.py --model_dir=models/
my_SSD_mobilenet_v2_FPNLite_320x320--pipeline_config_path=models/
SSD_mobilenet_v2_FPNLite_320x320/pipeline.config
```

El entrenamiento tomó varías horas con el fin de obtener los mejores resultados posibles, y analizar para cuantas épocas se estabilizaba la función de error, los resultados se presentan en la figura 27.

Figura 26. Entrenamiento

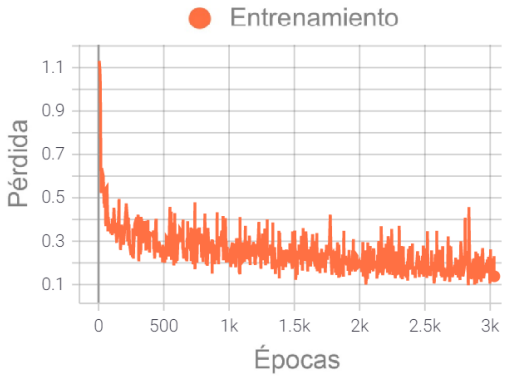


En donde se puede apreciar que la función de pérdida inicia alrededor en **1.1**, logrando descender hasta **0.13** después de 3000 épocas, sin embargo, se puede apreciar que desde la época 500 e incluso menos, el modelo ya había llega casi a estabilizarse.

Congelamiento del modelo: Terminado el entrenamiento, en la ruta de salida asignada tendremos una carpeta con los *checkpoints* que se guardaron durante el mismo. Estos archivos aún no permite hacer inferencias, es necesario exportar un “gráfico de inferencia", mediante uno de los *scripts* para congelar el modelo que podemos encontrar en el repositorio, en este caso usaremos, “export_tflite_graph_tf2.py", que es el que nos permite luego convertir el modelo para ser ejecutado con tensorflow lite.

```
python export_tflite_graph_tf2.py --pipeline_config_path
models/SSD_mobilenet_v2_FPNLite_320x320/pipeline.config
--trained_checkpoint_dir models/my_SSD_mobilenet_v2_FPNLite_320x320/ckpt-51
--output_directory exported_models_ssdmnv2lite320
```

Figura 27. Funcion de pérdida vs. épocas MobileNet V2 FPNLite 320x320



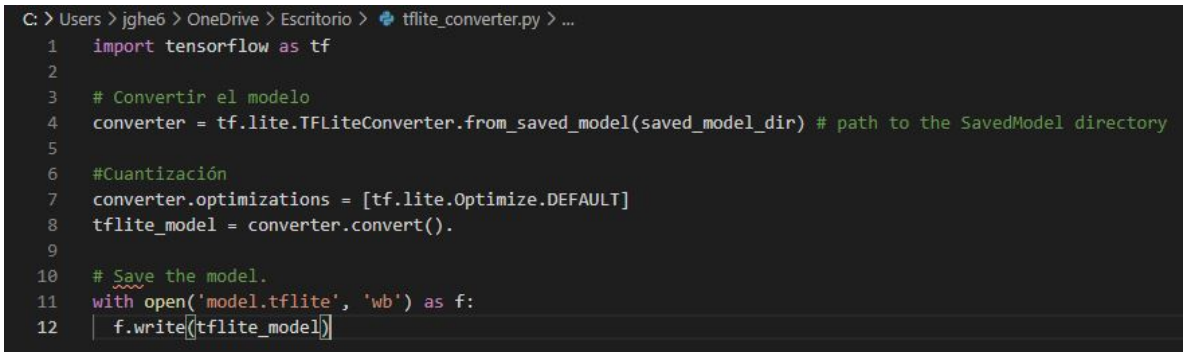
Con esto ya tenemos el gráfico de inferencia convertible a TFlite.

Conversión a TFLite y cuantización: TensorFlow 2 permite la conversión de modelos entrenados con la API de tensorflow a este formato por medio de un paquete de Python: *tf.lite.TFLiteConverter*. Al momento de convertirlo podemos aplicar cuantización al modelo, esta es una técnica de conversión que puede reducir el tamaño del modelo y mejorar la latencia del acelerador de hardware, afectando mínimamente en la precisión del modelo. En la cuantificación los pesos se convierten de 8 bits de precisión a punto flotante y se calculan usando núcleos de punto flotante. Esta conversión se realiza una vez y se almacena en caché para reducir la latencia.

Realizado este procedimiento, ya tenemos el modelo en formato TFLite, el cual tuvo un peso de 3.5 MB aproximadamente.

4.2.3. Clasificación La clasificación de imágenes se realizó usando la librería de python OpenCV, usando técnicas de tratamiento de imágenes. La inferencia dada por el modelo reduce el trabajo de computo que se debe realizar, ya que no se debe

Figura 28. Codigo en python para convertir al formato .tflite

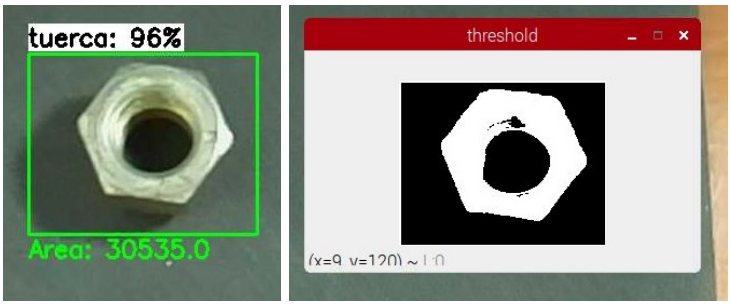


llevar a cabo cálculos para toda la imagen, sino únicamente para las secciones de la misma en donde se han detectado objetos. Para este proceso se siguieron los siguientes pasos:

1. Separación de la sección de imagen con el objeto identificado.
2. Conversión a escala de grises
3. Conversión a imagen binaria
4. Extracción de contornos y cálculo de área

Cada uno de los pasos anteriores es realizado por una función especifica de OpenCV. El primero se logra simplemente recortando la matriz asociada a la sección de imagen donde se detectó un objeto; la conversión a escala de grises mediante la función $cv2.cvtColor$, para luego aplicarle un *threshold* y obtener una imagen binaria, usando la función $cv2.threshold$.

Figura 29. Clasificación



(a) Imagen de objeto (b) Convertida a binaria

El valor del *threshold* varía según la cantidad de luz en el entorno al momento de capturar la foto, por ello cada *threshold* es particular para cada imagen. Se obtiene hallando un promedio de intensidad para cada una con al función $np.mean$ de la librería Numpy.

Las funciones $cv2.findContours$ y $cv2.contourArea$, son las encargadas de obtener el área. La primera recorre la imagen extrayendo todos los contornos encontrados, un contorno es la curva que une todos los puntos continuos que tienen el mismo color o intensidad en una imagen. La segunda función calcula un área relativa a cada contorno en la imagen. Finalmente, nos interesa el contorno con mayor área encontrado en la imagen, que corresponderá con el área del contorno asociado al objeto.

4.2.4. aXeleRate: Es un marco de trabajo desarrollado por Dmitry Maslov, el cual permite agilizar el proceso de entrenamiento de redes neuronales para la detección de objetos basado en Yolo, así como la conversión de los mismos para que se ejecuten en plataformas con aceleración de hardware. como el K210, se puede ejecutar de manera local o en la nube con la herramienta *Google Colaboratory* (un servicio en la nube que permite uso gratuito de GPUs y TPUs). Admite la conversión de mo-

delos entrenados a: kmodel, formato TFLite[24]. Se puede acceder a este *framework* desde su reposotorio oficial en github.

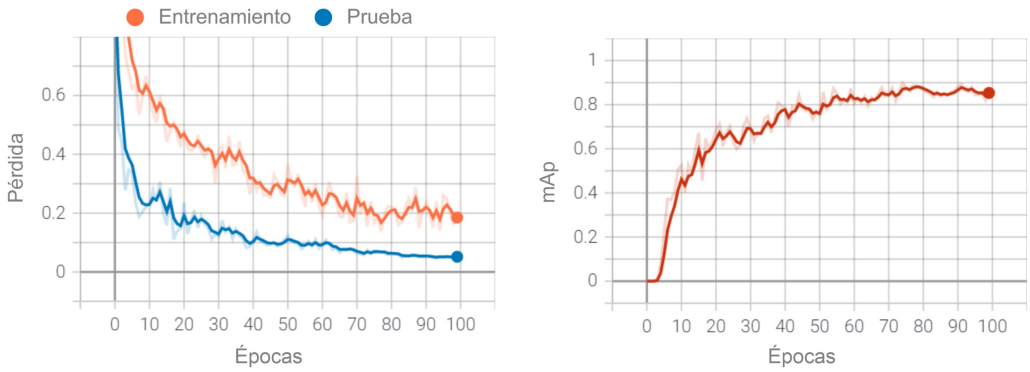
Se optó por usar aXeleRate principalmente por la necesidad de convertir el modelo en el formato adecuado para la tarjeta Maixduino: kmodel. Esta conversión es hecha mediante el conjunto de herramientas que se pueden encontrar en la *Maix Toolbox*, dentro del repositorio oficial de Sipeed, acá se encuentra el compilador *nncase*, mediante el cual se lleva la conversión desde TFLite al formato kmodel. Cabe resaltar que los comandos para instalar este compilador y realizar la conversión, vienen dados para la consola de Linux, sistema operativo en el que también trabaja *Google Colaboratory*, por lo que el entrenamiento se llevará a cabo usando ese servicio en la nube.

La red se entrenó con un tamaño de la capa de entrada de 224x224, que es el tamaño original de la arquitectura elegida en este caso: MobileNet7_5 ya que su estructura esta cuantizada y se adapta a las limitaciones de memoria sin perder su precisión. El entrenamiento hizo 100 épocas con una duración total de 39 minutos. Los resultados obtenido después de este se pueden observar en las figura 30, las cuales se obtuvieron con la herramienta tensorboard, de tensorflow. Se obtiene una **precisión promedio (mAp)** máxima de **0.8708**. La **función de pérdida** en el conjunto de entrenamiento y conjunto de prueba fue de **0.18** y **0.05** respectivamente. La razón por la que se obtiene mas precisión en el conjunto de prueba que en el de entrenamiento es debido a que aunque ambos conjuntos tienen la misma naturaleza (las fotos se capturaron en ambientes iguales), la totalidad del conjunto de datos de prueba contiene imágenes más fácilmente reconocibles, es decir ángulos directos

[24] AXeleRate. *Keras-based framework for AI on the Edge*.

(perpendicular a la mesa) y menores distancias, como sería a nivel práctico en una banda transportadora.

Figura 30. Resultados MobileNet7$_5$



(a) Pérdida vs. épocas

(b) mAp vs. época

5. EJECUCIÓN Y RESULTADOS

5.1. RASPBERRY PI

La ejecución en la tarjeta Raspberry Pi modelo B+ se llevó a cabo instalando y ejecutando el intérprete de tensorflow lite (3.1.2) en un ambiente aislado. Para ejecutar el modelo, se parte del *script* de Edje Electronics, haciendo las modificaciones correspondientes con nuestro modelo, y la clasificación de los objetos. El código, escrito en python, se divide en tres partes principales:

1. Creación de una clase para la transmisión de video.
2. Carga del modelo y transmisión de video.
3. Inferencias y clasificación.
4. Imprimir por pantalla los resultados.

La creación de una clase para la transmisión de video se hace con el objetivo de obtener un mejor rendimiento (cuadros por segundo), al utilizar un núcleo dedicado para esta. Esta mejora es obtenida gracias a que al utilizar un núcleo exclusivo para esta tarea, estamos disminuyendo la latencia de entrada y salida de las imágenes con las que interactúa nuestro modelo, además aseguramos que el núcleo principal no se bloquee[25]. Para la creación de esta clase se requiere la clase Thread del módulo threading.

---

[25] PyImageSearch. *Increasing Raspberry Pi FPS with Python and OpenCV*.

Figura 31. Clase para transmisión de video

```
class VideoStream:
    """Camera object that controls video streaming from the Picamera"""
    def __init__(self,resolution=(640,480),framerate=30):
        # Initialize the PiCamera and the camera image stream
        self.stream = cv2.VideoCapture(0)
        ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
        ret = self.stream.set(3,resolution[0])
        ret = self.stream.set(4,resolution[1])

        # Read first frame from the stream
        (self.grabbed, self.frame) = self.stream.read()

    # Variable to control when the camera is stopped
        self.stopped = False

    def start(self):
    # Start the thread that reads frames from the video stream
        Thread(target=self.update,args=()).start()
        return self

    def update(self):
        # Keep looping indefinitely until the thread is stopped
        while True:
            # If the camera is stopped, stop the thread
            if self.stopped:
                # Close camera resources
                self.stream.release()
                return

            # Otherwise, grab the next frame from the stream
            (self.grabbed, self.frame) = self.stream.read()

    def read(self):
    # Return the most recent frame
        return self.frame

    def stop(self):
    # Indicate that the camera and thread should be stopped
        self.stopped = True
```

Con la clase definida, se carga el modelo y se inicia la transmisión, como se muestra en la figura 32:

Figura 32. Cargar el modelo

```
81  # Cargar modelo TFLite
82  interpreter = tflite.Interpreter(model_path=PATH_TO_MODEL_DIR)
83  # Cargar labels
84  with open(PATH_TO_LABELS, 'r') as f:
85      labels = [line.strip() for line in f.readlines()]
86
87  interpreter.allocate_tensors()
88  #Cargar detalles del modelo
89  input_details = interpreter.get_input_details()
90  output_details = interpreter.get_output_details()
91
92  height = input_details[0]['shape'][1]
93  width = input_details[0]['shape'][2]
94  floating_model = (input_details[0]['dtype'] == np.float32)
95  input_mean = 127.5
96  input_std = 127.5
97  dim =(100,100)
98  # Iniciar calculo de FPS
99  frame_rate_calc = 1
100 freq = cv2.getTickFrequency()
101 print('Running inference for PiCamera')
102 # Initialize video stream
103 videostream = VideoStream(resolution=(imW,imH),framerate=30).start()
104 time.sleep(1)
```

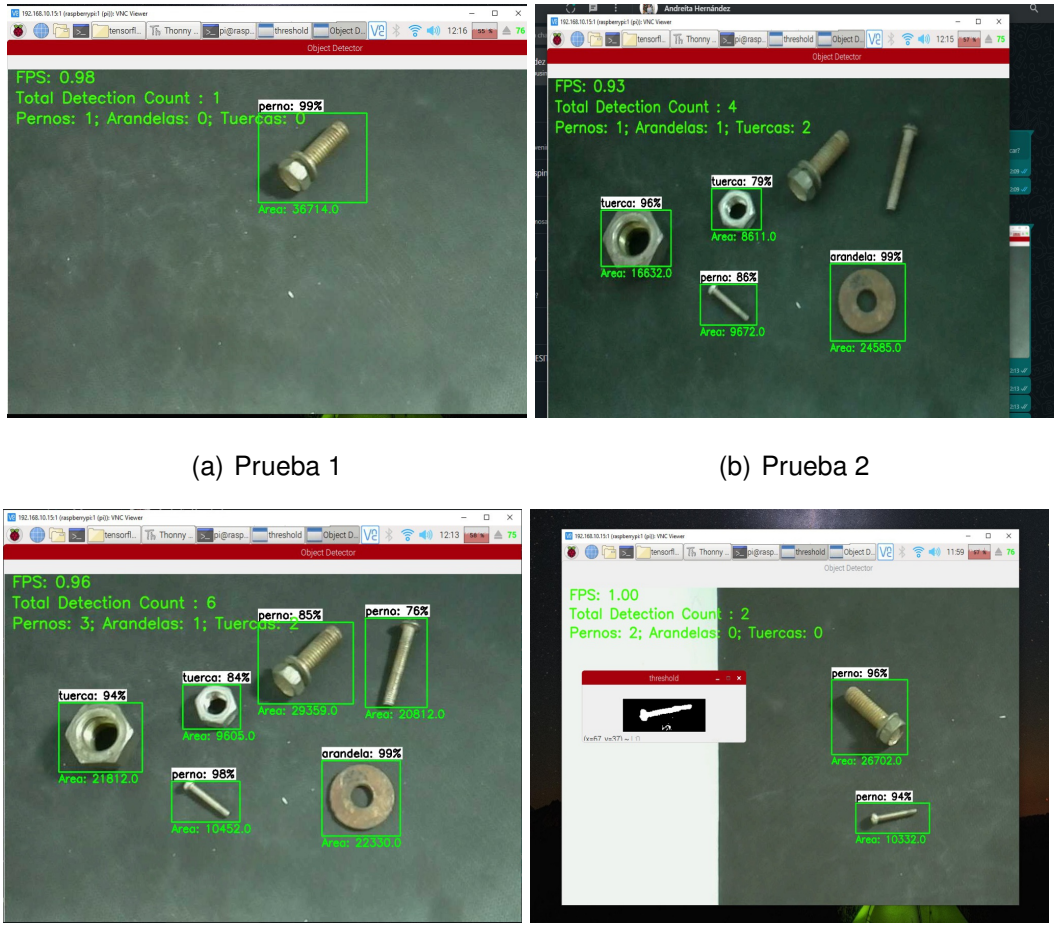
Posteriormente, el código necesario para realizar las inferencias y realizar los cálculos para la clasificación de cada objeto se incluyó dentro de un cilo $whileTrue$. Las setencias necesarias para hacer una inferencia se muestran en la figura 33

Figura 33. Realizar inferencias

```
#Tomar captura del stream
frame1 = videostream.read()

# Redimensionar acorde al modelo
frame = frame1.copy()
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
frame_resized = cv2.resize(frame_rgb, (width, height))
input_data = np.expand_dims(frame_resized, axis=0)

# Normalizar si es un modelo flotante (Modelo no cuantizado)
if floating_model:
    input_data = (np.float32(input_data) - input_mean) / input_std

# Realizar inferencia
interpreter.set_tensor(input_details[0]['index'],input_data)
interpreter.invoke()

# Capturar resultados Boxes=predicciones, classe=clase predicha, Score=proncentaje de confianza
boxes = interpreter.get_tensor(output_details[0]['index'])[0] # Bounding box coordinates of detected objects
classes = interpreter.get_tensor(output_details[1]['index'])[0] # Class index of detected objects
scores = interpreter.get_tensor(output_details[2]['index'])[0] # Confidence of detected objects
```

Activando el ambiente desde la consola, y especificando la ruta del modelo en formato tf.lite, su mapa de etiquetas, y el umbral de detección; corremos el modelo. Las tomas capturadas con este modelo se presentan en la figura 34.

Figura 34. Modelo de TFLite ejecutandose en Raspberry Pi Modelo B6(RIP)



(a) Prueba 1 (b) Prueba 2

(c) Prueba 3 (d) Prueba 4

La raspberry es capaz de procesar entre 0.9 y 1.2 fotogramas por segundo, un resultado esperado, ya que esta tarjeta tiene un procesador de propósito general de bajo rendimiento. Sin embargo, la clasificación es deseable, puesto que es capaz de detectar nuestros objetos con un nivel de confianza superior al 80 %. Bajo cada cuadro delimitador se puede observar el área relativa a cada objeto, lo cual permite hacer una clasificación entre objetos de tamaño pequeño, mediano o grande.

5.2. SIPEED MAIXDUINO

Para desplegar el modelo en esta tarjeta se utiliza el IDE MaixPy. El modelo se puede cargar a esta de dos maneras: grabándolo en la memoria flash, o por medio de una tarjeta micro SD.

El código se corre en el lenguaje de programación micropython. Micropython es una implementación de Python 3, está especialmente optimizada para ejecutarse en microcontroladores. En general, su sintaxis es parecida a la de python. En la página web https://docs.micropython.org/en/latest/genrst/index.html se pueden encontrar sus diferencias. Así mismo, se puede encontrar sobre las librerías disponibles en la documentación oficial de Sipeed para estas tarjetas.

Para escribir el código en la memoria de la tarjeta, esta se conecta al ordenador por medio de un cable USB tipo C, luego se elige el puerto correspondiente con la tarjeta desde el IDE, y se carga. En la figura 35 se encuentra el código usado para montar el modelo en la tarjeta.

El código usa librerías $sensor$, $image$, lcd, $time$ y kpu de micropython. La librería $sensor$ se utiliza para capturar la imagen desde la cámara, y configurar la: resolución nativa, formato de imagen, tamaño. $Image$ es una librería usada para el tratamiento de imágenes y configurar la interfaz de la misma, en este caso, dibujar los cuadros delimitadores, sus etiquetas y mostrar los cuadros por segundo en pantalla. La librería lcd controla la pantalla LCD conectada a la tarjeta, permitiendo mostrar la imagen con una única linea de código; por último, la librería $time$, que se usa para contar la cantidad de cuadros por segundo que procesa la tarjeta. KPU es el procesador inmerso dentro del módulo Kendryte K210, y el de la librería que permite controlarlo. Esta librería permite especificar la capa de salida de nuestro modelo,

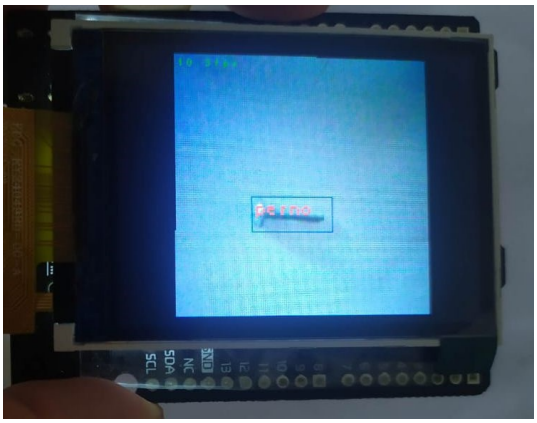
inicializar el modelo con un umbral de detección y realizar las inferencias. La función $kpu.run\_yolo2(task, img)$ dentro del código, devuelve una lista con datos de interés de las inferencias realizadas, estos son: coordenada, etiqueta de objeto, porcentaje de confianza.

Figura 35. Codigo en Maix ID

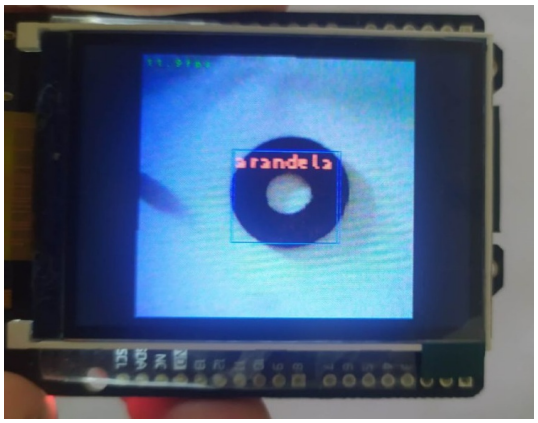
PAT_Detector.py*

```
clock = time.clock()
lcd.init()
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.set_windowing((224,224))
sensor.set_vflip(1)
sensor.run(1)
classes = ["perno","arandela","tuerca"]
task = kpu.load("/sd/YoloDef7740.kmodel")
a = kpu.set_outputs(task, 0, 7, 7, 40)
anchor = (0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828)
a = kpu.init_yolo2(task, 0.5, 0.3, 5, anchor)
a = kpu.memtest()
while(True):
    clock.tick()
    img = sensor.snapshot().rotation_corr(z_rotation=90.0)
    fps =clock.fps()
    a = img.pix_to_ai()
    code = kpu.run_yolo2(task, img)
    Cp = 0
    Ca = 0
    Ct = 0
    if code:
        for i in code:
            if (i.classid()==0):
                Cp = Cp + 1
            if (i.classid()==1):
                Ca = Ca + 1
            if (i.classid()==2):
                Ct = Ct + 1
            a = img.draw_rectangle(i.rect(),color = (0, 0, 255))
            a = img.draw_string(i.x(),i.y(), classes[i.classid()], color=(255,0,0), scale=2, x_spacing=-6)
            a = img.draw_string(2,2, ("%2.1ffps" %(fps)), color=(0,128,0), scale=1)
            a = lcd.display(img)
    else:
            a = img.draw_string(2,2, ("%2.1ffps" %(fps)), color=(0,128,0), scale=1)
            a = lcd.display(img)
    print('Se ha detectado {} perno(s), {} arandela(s), {} tuercas.'.format(Cp,Ca,Ct))
a = kpu.deinit(task)
```

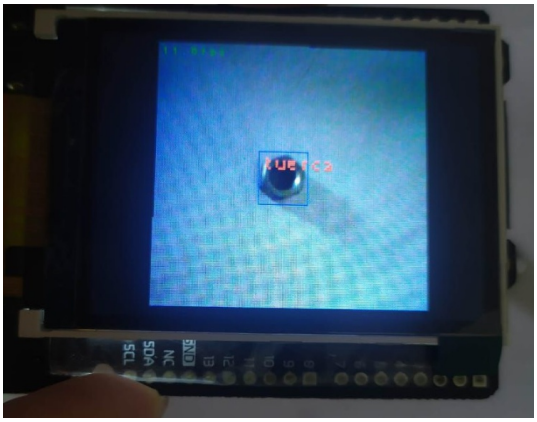
Figura 36. Clasificación



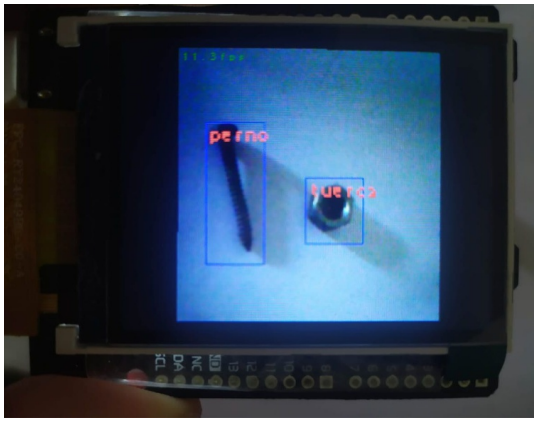
(a) Perno



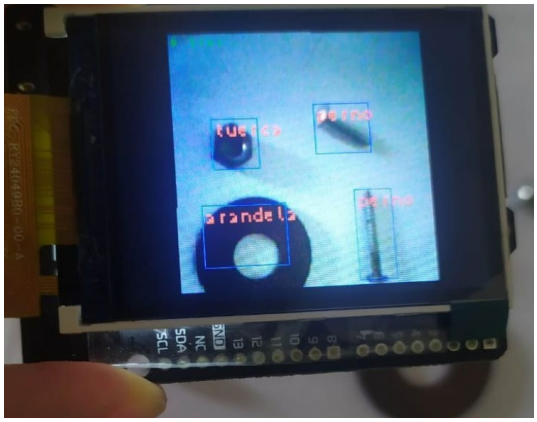
(b) Arandela



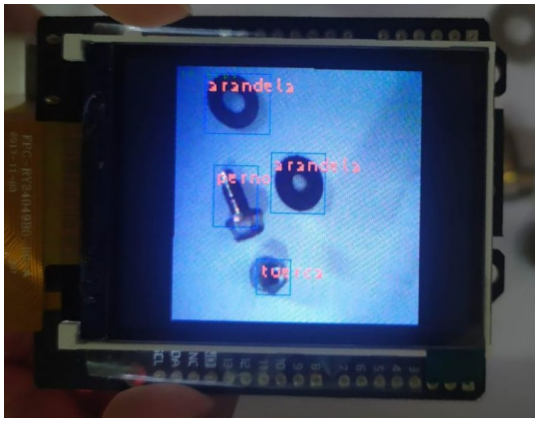
(c) Tuerca



(d) Prueba 1



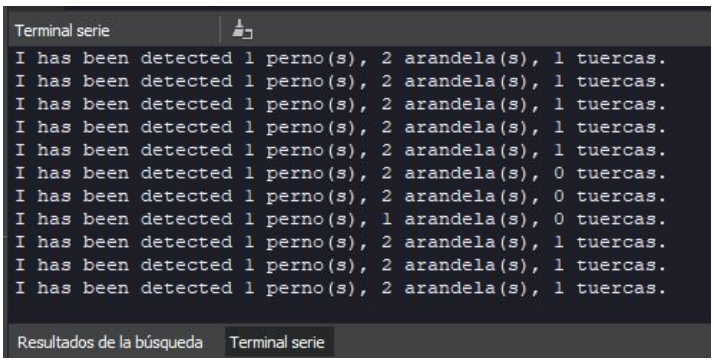
(e) Prueba 2



(f) Prueba 3

En la figura 36 se muestran pruebas realizadas. Siendo capaz de procesar entre 8 a 12 fotogramas por segundo y siendo capaz de detectar sin problemas los objetos. En la terminal del IDE, podremos ver un conteo de cuantos objetos de cada tipo se detectaron (ver figura 37).

Figura 37. Terminal Serie MaixPy IDE



En general, la tarjeta Sipeed Maixduino es capaz de hacer inferencias correctas, sin embargo cuando hay múltiples objetos, es probable que no detecte alguno. El modelo entrenado tiene un tamaño de tan solo 1.8 MB aproximadamente, lo que lo convierte en un modelo bastante liviano, además. como se mencionó previamente, la capa de entrada de 224x224, una baja resolución, que funciona bien mientras los objetos a detectar tenga un tamaño considerable o cubran la mayor parte de la imagen, cosa que no sucede cuando se quieren detectar múltiples objetos simultáneamente. Adicionalmente, cuando se entrenó el modelo con un tamaño de entrada mas grande 240x320, que es el tamaño nativo de las imágenes del conjunto de datos, el modelo presentaba problemas de memoria al momento de cargarse en la memoria.

6. CONCLUSIONES Y RECOMENDACIONES

Hoy en día, existen herramientas como lenguajes de alto nivel y *frameworks*, que permiten introducirse rápidamente en el *deep learning*, detección de objetos. En esta tesis se ponen en práctica Tensorflow API y aXeleRate. Lo presentado a lo largo de este documento, permite confirmar que ambos ambientes funcionan de manera óptima al momento de entrenar el modelo, y ambos permiten ser entrenados de manera local o en la nube mediante la herramienta*Google Colaboratory*, sin embargo, para entrenar localmente es necesario contar con una tarjeta de gráficos Nvidia, por lo que *Google Colaboratory* se presenta como una opción mas económica y práctica.

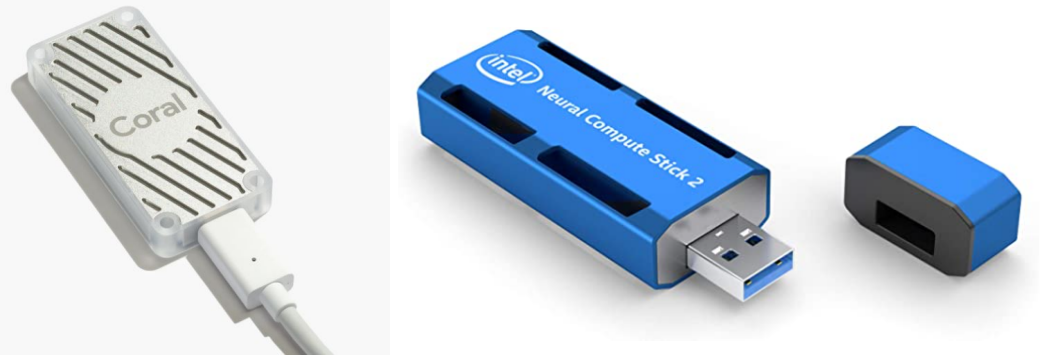
En cuanto a las tarjetas de desarrollo, como la Sipeed Maixduino es una tarjeta de origen chino relativamente nueva no ofrece buena documentación, este trabajo es un aporte a la comunidad interesada en trabajar en esta área, haciendo mas fácil el trabajo con esta tarjeta y volviéndola igual de competitiva que raspberry en el campo de las redes neuronales.

El rendimiento es considerablemente diferente en ambas tarjetas, la maixduino es capaz de procesar hasta 10 veces mas rápido que la raspberry, por un menor precio, esto gracias al modulo acelerador de hardware integrado con esta, sin embargo, el formato que acepta (kmodel), es muy liviano, esto sumado a la resolución que permite, hace que sea difícil para la tarjeta identificar objetos de tamaño reducido, a diferencia de la raspberry, que puede detectar múltiples objetos a la vez con un nivel de confianza alto. En general, en aplicaciones donde esté presente la detección de objetos, se debe buscar en las arquitecturas el equilibrio adecuado entre precisión, donde se destacan las arquitecturas como FasterRCNN, o velocidad; en donde so-

bresalen las arquitecturas Yolo; según las necesidades dadas para la aplicación o proyecto que se esté desarrollando.

Es posible obtener mas fotogramas por segundo aún con Raspberry, por ejemplo en el modelo Raspberry Pi 4, de 4GB RAM, donde al tener mas potencia de computo, se espera una mejora en la cantidad de frames por segundo procesados. Incluso, dado el caso de la necesidad un modelo para una aplicación específica, por ejemplo visión artificial en un coche autónomo, donde el tiempo de predicción importa bastante, 2 FPS no serán suficientes, por lo que se puede hacer uso de dispositivos como el acelerador USB de Google Coral o el *neural compute stick* los cuales son dispositivos USB que incorporan un co-procesador TPU, permitiendo inferencias de mucha mayor velocidad en una amplia variedad de sistemas: Debian Linux, macOS, Windows 10, y Raspberry Pi OS; todo esto simplemente conectadose a un puerto USB. Su principal desventaja es el precio, ya que este hardware se consigue desde $60 USD ($210000 COP aproximadamente).

Figura 38. Aceleradores USB



(a) Acelerador USB de Google Coral

(b) *Neural Compute Stick*

BIBLIOGRAFÍA

Asociación Cluster de Industrias de Medio Ambiente de Euskadi. Aclima. *Tecnología e industria 4.0: la sostenibilidad en la cuarta Era Industrial*. Inf. téc. Madrid: Fundación Conama (Congreso Nacional del Medio Ambiente), 2018 (vid. pág. 13).

AXeleRate. *Keras-based framework for AI on the Edge* (vid. pág. 69).

Bonet Cruz, Isis y col. "Redes neuronales recurrentes para el analisis de secuencias". En: (2007) (vid. pág. 28).

Bryson, Joanna J. "La última década y el futuro del impacto de la IA en la sociedad | OpenMind". En: *OpenMindBBVA* () (vid. pág. 12).

C., Diana. *¿Qué Es El Protocolo SSH Y Cómo Funciona?* (Vid. pág. 46).

Cardillo, Emanuele y Alina Caddemi. "Feasibility Study to Preserve the Health of an Industry 4.0 Worker: A Radar System for Monitoring the Sitting-Time". En: *2019 IEEE International Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2019 - Proceedings*. Institute of Electrical y Electronics Engineers Inc., 2019, págs. 254-258. DOI: 10.1109/METROI4.2019.8792905 (vid. pág. 12).

Conti, Maurice. "The incredible inventions of intuitive AI". En: *TED Conferences*. 2016 (vid. pág. 18).

DEA. "Procesamiento digital de imágenes". En: *Perfiles Educativos* 72 (1996) (vid. pág. 22).

Eremia, Mircea, Lucian Toma y Mihai Sanduleac. “The Smart City Concept in the 21st Century”. En: *Procedia Engineering*. Vol. 181. Elsevier Ltd, 2017, págs. 12-19. DOI: 10.1016/j.proeng.2017.02.357 (vid. pág. 13).

Halfacree, Gareth. *THE OFFICIAL Raspberry Pi Beginner’s Guide How to use your new computer*. 2018 (vid. pág. 43).

Hardy, Thomas. *Revista de la Universidad Bolivariana Volumen*. Inf. téc. (vid. pág. 18).

IBM. *What are Neural Networks? | IBM* (vid. pág. 24).

Lenin Marcel Cadena Castro, Jonathan Alejandro Heredia López. “Sistema inteligente con visión artificial para el reconocimiento de piezas mecánicas en el robot NAO”. B.S. thesis. Universidad Politécnica Salesiana, 2018 (vid. pág. 16).

Luis Piñuel Moreno, Francisco Igual Peña. “ACELERACIÓN DE AI EN DISPOSITIVOS DE BAJO CONSUMO”. B.S. thesis. Universidad Complutense de Madrid, 2020 (vid. pág. 16).

Martinez, Jose. *Regularización Lasso L1, Ridge L2 y ElasticNet - IArtificial.net* (vid. pág. 35).

ML. *What Is Deep Learning? | How It Works, Techniques & Applications - MATLAB & Simulink*. 2019 (vid. pág. 21).

OpenCV. *About OpenCV*. 2019 (vid. pág. 42).

PwC. *Industria 4.0: ¿cómo puede beneficiar al medio ambiente?* 2018 (vid. pág. 13).

PyImageSearch. *Increasing Raspberry Pi FPS with Python and OpenCV* (vid. pág. 71).

Redmon, Joseph y col. “You only look once: Unified, real-time object detection”. En: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2016-Decem. 2016, págs. 779-788. DOI: 10.1109/CVPR.2016.91. arXiv: 1506.02640 (vid. pág. 40).

Sandler, Mark y col. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. En: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2018, págs. 4510-4520. DOI: 10.1109/CVPR.2018.00474. arXiv: 1801.04381 (vid. pág. 63).

Shamim, Saqib y col. “How firms in emerging economies can learn industry 4.0 by extracting knowledge from their foreign partners? A view point from strategic management perspective”. En: *International Conference on Advanced Mechatronic Systems, ICAMechS*. Vol. 2019-Augus. IEEE Computer Society, 2019, págs. 390-395. DOI: 10.1109/ICAMechS.2019.8861622 (vid. pág. 12).

Siemens. *La Inteligencia Artificial en la industria | Industria | Siemens Global* (vid. págs. 10, 11).

Sipeed. *Sipeed Maixduino Kit for RISC-V AI + IoT* (vid. pág. 50).

Valderas, Antonio José Toro. “Implementación de redes neuronales en Raspberry Pi 3 con Movidius Neural Compute Stick”. B.S. thesis. Universidad de Sevilla, 2020 (vid. pág. 17).

Vázquez, JA Serrano. *Raspberry Pi 3 Model B+*. 2019 (vid. pág. 44).