SISTEMA *IOT* PARA RECONOCIMIENTO DE GESTOS MANUALES USANDO UNA CÁMARA

ANDRÉS FELIPE GÓMEZ RUEDA
EDGAR DAVID NAVARRO BOLAÑOS

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

SISTEMA *IOT* PARA RECONOCIMIENTO DE GESTOS MANUALES USANDO UNA CÁMARA

ANDRÉS FELIPE GÓMEZ RUEDA
EDGAR DAVID NAVARRO BOLAÑOS

Trabajo de Grado para optar al título de
Ingeniero Electrónico

Director
JAIME GUILLERMO BARRERO PEREZ
MAG. en Potencia Eléctrica

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

Dedicado

A mis padres, Alba y Orlando, por su apoyo, sus palabras de aliento y por ser mi mayor motivación desde el inicio de este camino.

A mi novia y amigos, quienes cada semestre estuvieron conmigo e hicieron de la universidad un lugar más bonito y ameno.

ANDRÉS FELIPE GÓMEZ RUEDA

A mis padres, Amayra y Edgar, mi hermana Susana por ser una guía y ejemplo, por sus palabras de apoyo y la ayuda que me brindaron estos años.

A mis amigos, quienes estuvieron conmigo durante cada semestre y me ayudaron a cumplir todos los retos que presentó el mundo universitario.

EDGAR DAVID NAVARRO BOLAÑOS

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

pág.

RESUMEN

TÍTULO: SISTEMA *IOT* PARA RECONOCIMIENTO DE GESTOS MANUALES USANDO UNA CÁMARA [*]

AUTOR: ANDRÉS FELIPE GÓMEZ RUEDA, EDGAR DAVID NAVARRO BOLAÑOS [**]

PALABRAS CLAVE: DETECCIÓN DE GESTOS MANUALES, PROCESAMIENTO DE IMÁGENES, INTERNET DE LAS COSAS, INTELIGENCIA ARTIFICIAL, MICROPROCESADOR.

DESCRIPCIÓN:

En la actualidad la inteligencia artificial (IA) aporta grandes beneficios tecnológicos. Una de las ramas de la inteligencia artificial es la visión artificial. Esta es una combinación de hardware y software que tiene la capacidad de capturar y procesar datos de imágenes, utilizada en aplicaciones como detección de rostros, detección de imágenes para evaluar posibles enfermedades, seguimiento de objetos, entre otros desarrollos que facilitan la toma y control de datos en diversas aplicaciones. En este trabajo de investigación se aplica la detección de gestos manuales para luego utilizar el gesto reconocido y realizar una comunicación a través de internet.

Actualmente en la literatura hay distintos estudios sobre la detección de objetos y este proyecto en particular se enfoca en realizar la detección a través de un microcontrolador, teniendo en cuenta los beneficios que aporta como portabilidad y bajo costo.

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Jaime Barrero, MAG, en Potencia Eléctrica.

ABSTRACT

TITLE: IOT SYSTEM FOR RECOGNITION OF MANUAL GESTURES USING A CAMERA. [*]

AUTHOR: ANDRÉS FELIPE GÓMEZ RUEDA, EDGAR DAVID NAVARRO BOLAÑOS [**]

KEYWORDS: DETECTION OF MANUAL GESTURES, IMAGE PROCESSING, INTERNET OF THINGS, ARTIFICIAL INTELLIGENCE, MICROPROCESSOR.

DESCRIPTION:

Nowadays, artificial intelligence (AI) brings great technological benefits. One of the branches of artificial intelligence is artificial vision. This is a combination of hardware and software that has the ability to capture and process image data, used in applications such as face detection, image detection to evaluate possible diseases, object tracking, among other developments that facilitate the acquisition and control of data in various applications. In this research work, the detection of manual gestures is applied in order to later use the recognized gesture and communicate through the Internet.

Currently in the literature there are different studies on object detection and this particular project focuses on detection through a microcontroller, taking into account the benefits it provides such as portability and low cost.

---

[*] Bachelor Thesis

[**] Faculty of Physical-Mechanical Engineering; School of Electrical, Electronic and Telecommunications Engineering. Director: JAIME BARRERO, MAG, in Electrical Power.

INTRODUCCIÓN

En los últimos años, la interacción máquina-hombre se ha venido desarrollando y ha ganado gran importancia en la industria. La inteligencia artificial (IA) cubre en gran parte este campo. Algunos desarrollos de inteligencia artificial son altamente eficientes llevando a las maquinas a comunicarse fluidamente con los seres humanos e incluso a reemplazarlos en sus labores industriales, sin embargo, aun existen dudas acerca del beneficio que pueda traer un total desarrollo de la IA. Una de las ramas de la inteligencia artificial es el *Machine Learning*, este es el estudio y desarrollo de herramientas informáticas que utilizan la experiencia pasada para tomar decisiones futuras. Ciertos algoritmos de *Machine Learning* pueden ser utilizados para encontrar patrones de datos a través de redes neuronales lo cual se denomina *Deep Learnig*. Estos tipos de algoritmos han demostrado ser sumamente exitosos para resolver determinados tipos de problemas; como por ejemplo, el reconocimiento de imágenes. Para lograr que un sistema realice un reconocimiento eficiente de imágenes se necesita tener una amplia base de datos y una maquina que permita realizar el procesamiento y comparación con los datos previamente almacenados.

Por otro lado están los sistemas *IOT* (Internet de las cosas, por sus siglas en ingles) que permiten transmitir y obtener datos de los procesos en tiempo real desde cualquier sitio a través de internet. Al combinar IA con *IOT* se obtiene un sistema potenciado capaz de cumplir con una gran cantidad de necesidades tecnológicas de la actualidad. Este trabajo de grado tiene como propósito reconocer gestos manuales obtenidos a través de una cámara. Por tal motivo, se propone desarrollar un sistema que por medio de un microprocesador conectado a una cámara, permita hacer el reconocimiento de gestos manuales y utilizar los datos obtenidos para realizar *IOT*.

1. OBJETIVOS

Objetivo general

- Diseñar un sistema *IOT* que reconozca gestos manuales por medio de imágenes e inteligencia artificial (IA).

Objetivos específicos

- Seleccionar y configurar el microprocesador y el lenguaje de programación más adecuado que nos permita realizar el tratamiento de imágenes por medio de Inteligencia Artificial y la transmisión de datos por medio de *IOT*.
- Crear la base de datos que contendrá las plantillas de comparación de gestos manuales como: mostrar una mano abierta, una mano cerrada, el gesto de afirmación, el de amor y paz, entre otros. Con los gestos manuales a ingresar por el usuario.
- Entrenar una red neuronal para que identifique de manera correcta los gestos manuales introducidos al sistema.

2. MARCO TEÓRICO

La detección de objetos adquiere mayor importancia hoy en día en un mundo donde hay cada vez más controles de los procesos y monitoreo de estos. En el presente proyecto de investigación se estudia el estado del arte con el objetivo de ver como se realiza la detección de objetos y las bases teóricas en las que se fundamenta.

2.1. INTELIGENCIA ARTIFICIAL

Aunque muchas personas no lo noten, un gran número de las tecnologías que afectan las actividades que se realizan a diario cuentan con IA (Inteligencia Artificial), desde las redes sociales, *bots* conversacionales usados por empresas para la atención de sus clientes, hasta la detección de fraudes electrónicos y diagnósticos de enfermedades. La IA trata más sobre el proceso, la capacidad de pensamiento súper potente y el análisis de datos que sobre cualquier formato o función en particular. Aunque la visión de IA que se tiene, muchas veces muestra imágenes de robots de aspecto humano de alto funcionamiento que se apoderan del mundo, la IA no pretende reemplazar a los humanos. Su objetivo es mejorar significativamente las capacidades y contribuciones humanas.[1]

Antes de abordar este tema más a fondo y describir las características y funciones que lo componen es necesario establecer una definición formal. Pese a que muchos autores no llegan a un consenso acerca de su definición, se tomará como referencia la siguiente que establece que, la IA implica el estudio, el diseño y la construcción de

---

[1] "What is AI". En: *Oracle.com* (2018).

agentes inteligentes que puedan alcanzar objetivos. Las elecciones que hace una IA deben ser apropiadas a sus limitaciones perceptivas y cognitivas.[2]

La IA abarca distintos campos como son: *Machine Learning*, procesamiento natural del lenguaje, la minería de datos, la robótica, por mencionar algunos. Ahora, entre las ramas del *Machine Learning* se puede mencionar el *Deep Learning*, indispensable en las actividades de reconocimiento de patrones y procesamiento de imágenes.

2.2. MACHINE LEARNING

Machine Learning (*ML*) es una forma de inteligencia artificial que dicen, está puesta para transformar el siglo XXI. Este término abarca muchas áreas, por lo que dar un concepto concreto puede no ser tarea fácil. Si se realiza una revisión del estado del arte, la siguiente se puede considerar una definición acertada. *ML* se refiere a una amplia gama de algoritmos que realizan predicciones inteligentes basadas en un conjunto de datos. Estos conjuntos de datos suelen ser grandes, quizás consistan en millones de puntos de datos únicos. [3]

Machine Learning cuenta con múltiples algoritmos, técnicas y metodologías que pueden ser utilizadas para construir modelos para resolver problemas del mundo real utilizando datos.[4] Estos se pueden clasificar en distintas categorías. A continuación se presenta una de las principales áreas en las que están categorizados, teniendo en cuenta la cantidad de supervisión humana en el proceso de aprendizaje.

---

[2] Wagner A. Welsh S. Bartneck C. Lütge C. “An Introduction to Ethics in Robotics and AI”. En: *What Is AI?* Springer. 2020, págs. 5-16.

[3] Baker MAB Nichols JA Herbert Chan HW. “Machine learning: applications of artificial intelligence to imaging and diagnosis”. En: *Biophysical Reviews* (2019).

[4] Sharma T. Sarkar D. Bali R. “Machine learning basics”. En: *Practical Machine Learning with Python*. 2018, pág. 28.

2.2.1. Aprendizaje Supervisado Los métodos o algoritmos de aprendizaje supervisados incluyen algoritmos de aprendizaje que toman muestras de datos (conocidos como datos de capacitación) y resultados asociados (conocidos como etiquetas o respuestas) con cada muestra de datos durante el proceso de elaboración del modelo de entrenamiento. El objetivo principal es aprender una asociación entre las muestras de datos de entrada "*x*" y sus correspondientes salidas "*y*", basado en múltiples instancias de datos de entrenamiento. Estos métodos se denominan supervisados porque el modelo aprende en muestras de datos en las que las respuestas/etiquetas de salida deseadas ya se conocen de antemano en la fase de entrenamiento.[5] Basado en el tipo de tareas que estos pueden resolver se distinguen dos enfoques principales: Clasificación y Regresión.

2.2.2. Aprendizaje No Supervisado Los métodos de aprendizaje supervisado suelen requerir algunos datos de entrenamiento en los que los resultados que se intentan predecir ya están disponibles en forma de etiquetas discretas o valores continuos. Sin embargo, a menudo no se cuenta con la libertad o ventaja de tener datos de formación pre-etiquetados y aún así se desea extraer características o patrones útiles de nuestros datos. En este escenario, los métodos de *aprendizaje no supervisados* son extremadamente poderosos. Estos métodos se denominan no supervisados porque el modelo o algoritmo intenta aprender estructuras, patrones y relaciones latentes inherentes a partir de datos dados sin ninguna ayuda o supervisión como proporcionar anotaciones en forma de salidas o resultados etiquetados.[5] Estos se pueden dividir en 4 grupos, dependiendo de la tarea que se requiere. *Análisis de grupos*, *Reducción de dimensionalidad*, *Detección de anomalías* y *Reglas de asociación*.

---

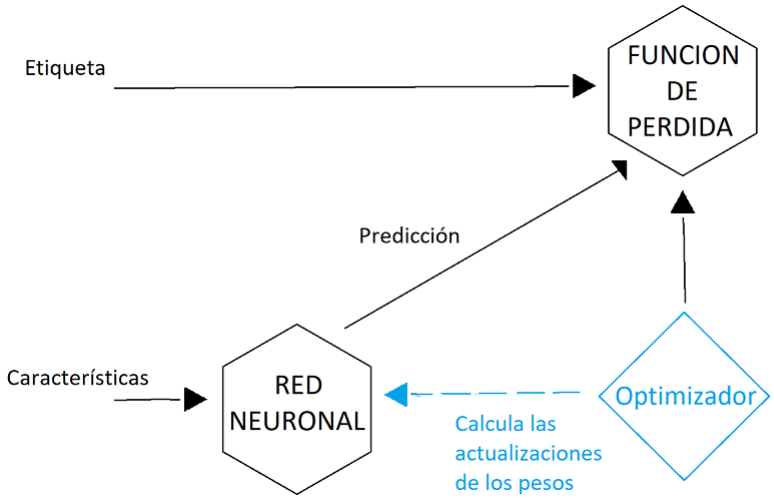
[5] Sharma T. Sarkar D. Bali R. "Machine learning basics". En: *Practical Machine Learning with Python*. 2018, pág. 35.

2.2.3. Aprendizaje semi-supervisado Los métodos de aprendizaje semi-supervisados suelen caer entre los supervisados y los no supervisados. Estos métodos suelen utilizar una gran cantidad de datos de capacitación no etiquetados (que forman el componente de aprendizaje no supervisado) y una pequeña cantidad de datos pre-etiquetados y anotados (que forman el componente de aprendizaje supervisado). Hay múltiples técnicas disponibles en forma de métodos generativos y métodos basados en gráficos.

2.3. APRENDIZAJE PROFUNDO *(DEEP LEARNING)*

Se puede definir el *Deep learning* como un subcampo específico del *Machine learning*, cuya finalidad es acercar la investigación sobre *Machine learning* a su verdadero objetivo de "hacer inteligentes a las máquinas". El término *"deep"* se refiere a que hay varias capas entre la entrada y la salida, permitiendo al algoritmo usar estas múltiples capas de procesamiento, compuestas de múltiples transformaciones lineales y no lineales. En términos prácticos, un enfoque basado en el *Deep learning* trata de construir la inteligencia de la máquina representando los datos como una jerarquía de conceptos en capas, donde cada capa de conceptos se construye a partir de otras capas más simples. Esta arquitectura en capas es en sí misma uno de los componentes centrales de cualquier algoritmo de *Deep learning*.[4]

Figura 1. Representación básica de un modelo de Deep Learning



Fuente: Curso Deep Learning for Computer Vision with Tensor Flow and Keras [figura]. [Consultado: 24 de Diciembre de 2020] Disponible en: https://www.udemy.com/course/deep-learning-for-computer-vision-with-tensor-flow-and-keras/learn/lecture/11159401overview

En la **figura 1** se presenta una interpretación de un modelo de *Deep learning*, en este se tiene una red neuronal la cual recibe varios datos de entrada, y a la vez, estos datos cuentan con distintas características. La idea es que el modelo pueda aprender esas características para hacer una predicción. La diferencia entre las predicciones y los valores reales de las etiquetas produce un error, este error se calcula a través de la función de pérdida. Hay diferentes tipos de función de pérdida. Después de realizar la selección de esta función, se necesita un optimizador, que no es más que un algoritmo que se encarga precisamente de la optimización (encontrar el valor mínimo) de la función de pérdida. El optimizador se encarga de determinar nuevos pesos y calcular un nuevo error, continuando con el proceso hasta que se obtiene un mínimo valor de error o se alcanza un número específico de ciclos. La importancia de las redes neuronales radica en que al estar conformadas por muchas capas ocultas entre las capas de entrada y salida, dichas capas influirán en el

rendimiento y desarrollo predictivo del modelo. Si se usan muy pocos nodos entre las capas, el modelo no será capaz de recoger todos los datos (*underfitting*), teniendo como resultado una baja exactitud o acierto y un pobre desarrollo predictivo. Asimismo, al usar demasiados nodos el modelo tenderá a excederse en los datos (*overfitting*) y no hará una buena generalización de estos. Además de que el requerimiento computacional es mayor, por lo tanto se debe encontrar un equilibrio entre estos dos marcos.[6]

2.4. REDES NEURONALES

Una Red Neural Artificial (RNA) es un modelo computacional y una arquitectura que simula las neuronas biológicas y la forma en que funcionan en nuestro cerebro. Típicamente, una RNA tiene capas de nodos interconectados. Los nodos y sus interconexiones son análogos a la red de neuronas de nuestro cerebro. Una RNA típica tiene una capa de entrada, una capa de salida y al menos una capa oculta entre la entrada y la salida.[4] La importancia de las RNA radica en que pueden resolver problemas como los de visión o aprendizaje, y su procesamiento en paralelo resulta necesario si se quiere lograr respuestas en tiempo real.[7]
Luego de ver la importancia de estas redes neuronales se presentan algunas de las arquitecturas básicas y más conocidas.

2.4.1. Perceptrón En este tipo de arquitectura (**figura 2**) se tienen los datos de entrada y solo una capa de "neuronas", en la que se realiza la suma de estos datos

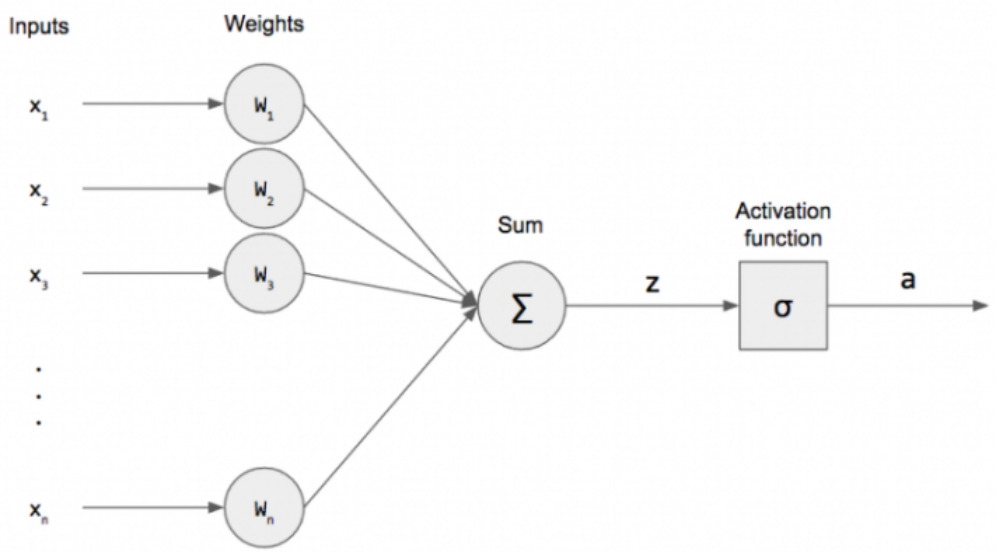
---

6 "Deep learning for computer vision with Tensor Flow and Keras". En: *https://www.udemy.com/course/deep-learning-for-computer-vision-with-tensor-flow-and-keras/* (2020).

7 "Las redes neuronales artificiales para la toma de decisiones". En: *XIX Congreso Internacional de Contaduría, Administración e Informática* (2014).

de entrada de acuerdo a unos pesos que posteriormente pasan por la función de activación para generar los datos de salida que corresponden a la predicción del modelo.

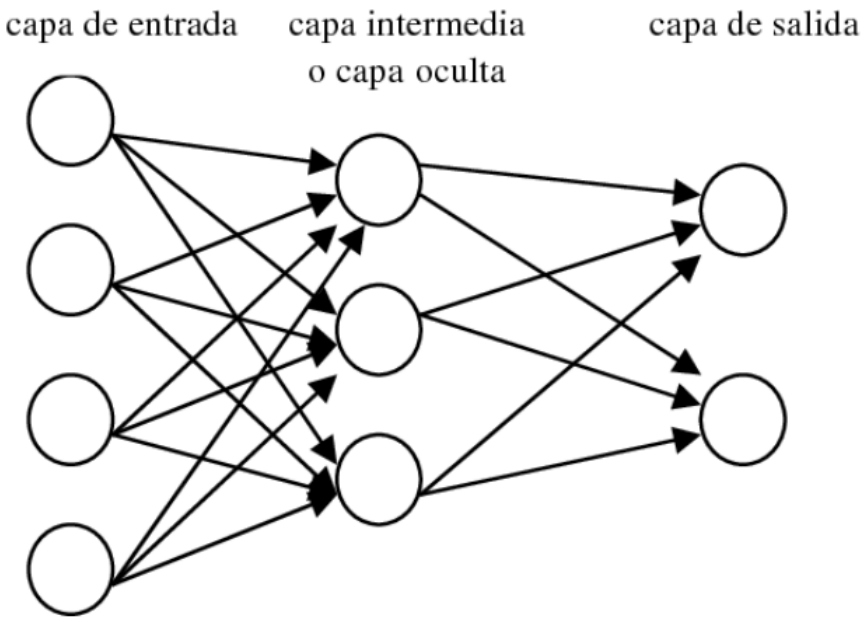
Figura 2. Representación del modelo de Perceptrón



Fuente: Perceptrons: The First Neural Networks [figura]. [Consultado: 26 de Diciembre de 2020] Disponible en: https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/

2.4.2. Perceptrón multicapa Como su nombre lo indica **figura 3**, en este se tienen muchas capas de perceptrón en vez de solo una, es decir, cada capa corresponde a un modelo de perceptrón simple, con lo que se logra tratar problemas de funciones no lineales que es la principal desventaja del perceptrón.

Figura 3. Representación del modelo de Perceptrón multicapa



Fuente: Ejemplo de perceptrón multicapa [figura]. [Consultado: 26 de Diciembre de 2020] Disponible en: https://www.researchgate.net/figure/Figura-3-Ejemplo-de-perceptron-multicapa_fig1_228815505

2.5. REDES NEURONALES CONVOLUCIONALES

Las redes neuronales convolucionales (CNN, por sus siglas en ingles) son un algoritmo de *deep learning* que toma una imagen de entrada y le asigna importancia a diferentes aspectos y características de esta. Una CNN es capaz de capturar las dependencias espacio-temporales en una imagen a través de la aplicación de filtros. Este tipo de arquitectura se ajusta mejor a las bases de datos con imágenes debido a que reduce el número de parámetros involucrados y los pesos de las "neuronas" se pueden reutilizar.[8]

[8] DUSSAN German MOSQUERA Carlos. "Detección y Seguimiento de Múltiples Objetos en Tiempo Real para Vehículos Autónomos". Tesis doct. Universidad Autónoma de Occidente, 2020.

En la actualidad existe un gran número de CNN, siendo su principal uso el analizar imágenes. Sus aplicaciones varían desde reconocimiento en imágenes y video, clasificación de imágenes, análisis de imágenes médicas hasta procesamiento de lenguaje natural.[8] Algunas de las más usadas son:

2.5.1. MobilNet v1 Las redes MobileNet fueron presentadas como modelos eficientes dirigidos a aplicaciones con dispositivos embebidos y dispositivos móviles.[8] Estas redes utilizan un diseño eficiente que implica operaciones de convoluciones separables en profundidad, para formar redes neuronales convolucionales profundas con pesos pequeños.[9] La principal ventaja de la convolución separable profunda es su eficacia computacional, ya que este tipo de convoluciones requieren menos operaciones matemáticas que las convoluciones 2D. Además, gracias a la reducción de operaciones por la implementación de las convoluciones profundas separables, esta versión es con 4 millones de parámetros, 3 veces más rápida que por ejemplo, la red Inception, que tiene 24 millones de parámetros.[8]

2.5.2. MobilNet v2 Los modelos de MobileNet V2 son más rápidos con la misma precisión. En particular, los nuevos modelos usan 2 veces menos operaciones, necesitan un 30 % menos de parámetros y son alrededor de 30-40 % más rápidos en un teléfono con Android que los modelos de MobileNetV1, todo esto mientras logran una mayor precisión.[10] Esto también es gracias a que cuenta con menos parámetros que la versión anterior (v1).

---

[9] V Prabhu N Jacob. "Comparative Review of YOLO MobileNet Versions: A Case Study". En: *International Research Journal of Engineering and Technology* (2020).

[10] "MobileNetV2: The Next Generation of On-Device Computer Vision Networks". En: *https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html* (2018).

2.5.3. YOLO Es una red neuronal convolucional diseñada para la detección de objetos de la cual existen 3 versiones. YOLO trata la detección de imágenes como un problema de regresión en vez de uno de clasificación, y corre una sola red neuronal convolucional para realizar todas sus tareas. Esto trae beneficios tanto en velocidad, debido a que el proceso de convolución se realiza una sola vez en toda la imagen para realizar las predicciones, como en reducción de errores de fondo, ya que mira la imagen de forma entera en vez de por partes y puede sacar conclusiones globales, lo que disminuye errores de falsas predicciones.[8]

2.6. YOLOV2

Solo miras una vez (YOLO, por sus siglas en ingles), es un sistema de detección de objetos en tiempo real. Este sistema consiste en aplicar una única red neuronal a la imagen con los objetos que se quieren detectar, esta red neuronal, divide la imagen en regiones y predice cuadros delimitados, encontrando la mayor probabilidad de detección de uno o varios objetos. Para hacer detección de objetos utilizando YOLO versión 2 se puede usar un modelo previamente entrenado o también se puede entrenar una red neuronal desde cero.[11]

Básicamente, el modelo recibe una imagen la cuál es dividida en una red de celdas de *m x m*. Para cada objeto que se presenta en la imagen, se supone que una celda de la red es responsable de predecirlo. Esta es la celda donde se encuentra el centro del objeto. La última capa utiliza una *función de activación* lineal, el resto de capas usan del tipo *Leaky ReLu*. YOLO predice distintas cajas delimitadoras por cada celda de la cuadrícula. Al momento del entrenamiento lo que se desea es que un 'predictor' de dichas cajas sea responsable por cada objeto, basado en cuál pre-

---

[11] "YOLO: Real-Time Object Detection". En: *https://pjreddie.com/darknet/yolov2/* (2020).

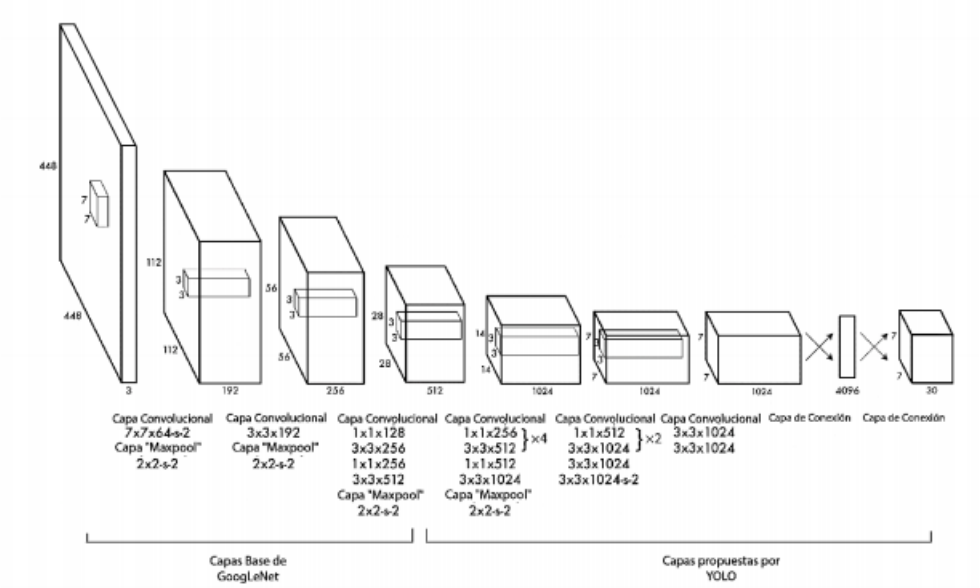
dicción tiene el mayor valor de *IoU (Intersection over union)*, que es una medida de evaluación usada para medir la precisión de un detector de objetos en un determinado conjunto de datos.

En la **figura 4** se observa la forma como está estructurada. La entrada de YOLO está provista de una imagen de tres canales, que se redimensiona a 448x448. La primera conversión es correr la imagen a través de una porción de la arquitectura modificada de *GoogLeNet*. Después de esta conversión, se obtiene los gráficos de características con un tamaño de 14x14x1024. Luego, se aplican dos convoluciones. Después de la segunda convolución, la dimensión disminuye a 7x7x1024. Entonces, se realiza otra convolución. El resultado se utiliza dos veces en una capa totalmente conectada, cambiando a una dimensión de 1470x1 y se transforma en un tensor de 7x7x30. El tensor obtenido se somete a un procedimiento de detección, a cuya salida se obtiene una detección resultante. El tensor es una rejilla de 7x7 en la imagen. 30 valores llevan información sobre la celda: 10 valores describen dos cuadros posibles; 20 valores muestran la relación con cada una de las 20 clases disponibles. Toda esta información es filtrada, y los datos filtrados son mostrados.[12]

---

[12] P Y Yakimov N S Artamonov. “Towards Real-Time Traffic Sign Recognition via YOLO on a Mobile GPU”. En: *The IV International Conference on Information Technology and Nanotechnology* (2018).

Figura 4. Arquitectura de la red neuronal YOLO



Fuente: Detección de equipos de protección personal mediante red convolucional YOLO [figura]. [Consultado: 30 de Diciembre de 2020] Disponible en: http://eii.unex.es/ja2018/actas/JA2018_073.pdf

2.7. ALGORITMO DE RETROPROPAGACIÓN *(BACKPROPAGATION ALGORITHM)*

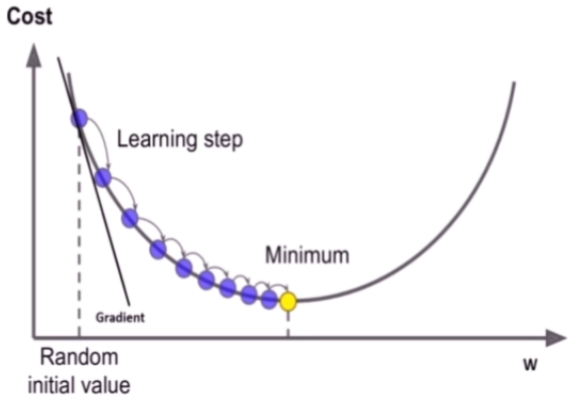
Como ya se mencionó anteriormente, la diferencia entre el valor pronosticado y el valor real produce un error, el cual se quiere que sea el menor posible. Esto se consigue a través del *algoritmo de retropropagación*.

Esta función está basada en los parámetros internos del modelo, es decir en los pesos y el sesgo. Los pesos son modificados usando la *Función de Optimización*, la cual calcula el gradiente de la función de pérdida con respecto a los pesos, y los pesos son modificados en dirección contraria del gradiente calculado[6]. Esta función de retropropagación realiza la actualización de los pesos mediante el método del

Descenso de Gradiente.

2.7.1. Descenso del gradiente En la **figura 5** la curva representa la función costo o error en términos de los pesos (w). El método representa la derivada de esta función costo en cada punto. Iniciando desde un punto aleatorio, se calcula el gradiente con el fin de hallar el valor mínimo de la función, siguiendo así en la dirección opuesta al gradiente. Luego, se calcula un nuevo gradiente en el siguiente punto continuando con el proceso hasta alcanzar el punto mínimo. El movimiento entre los puntos es regulado por la *tasa o razón de aprendizaje* (alfa), esta será la medida de cada paso en la actualización del valor de los pesos. Esta es una representación básica con solo un peso, pero la función costo siempre depende de muchos pesos, por lo tanto lo que se hace es el cálculo de la derivada parcial de dicha función en términos de todos los pesos, y los pesos son modificados en la dirección opuesta del gradiente calculado.

Figura 5. Representación gráfica del método descenso del gradiente



Fuente: Curso Deep Learning for Computer Vision with Tensor Flow and Keras [figura]. [Consultado: 26 de Diciembre de 2020] Disponible en: https://www.udemy.com/course/deep-learning-for-computer-vision-with-tensor-flow-and-keras/learn/lecture/11159401overview

$$w_{i+1} = w_i - \alpha \frac{d}{dw} C(w) \tag{1}$$

En la **ecuación 1** se aprecia la relación entre los pesos, la taza de aprendizaje y la función costo (*C(w)*). El peso actual será igual al peso anterior menos la taza de aprendizaje multiplicada por la derivada de función costo. Esta es una generalización del algoritmo del Descenso del gradiente, el cual presenta algunas variaciones.

1. **Descenso del gradiente por lotes** En este, se tienen en cuenta todos los datos para avanzar un paso. Se toma el promedio de los gradientes de todas las muestras de entrenamiento y luego se usa ese gradiente promedio para la actualización de los parámetros.[13] Esta variante es sensible para regiones en las que se encuentren regiones planas, donde no se aprecia una mejora en el valor de la función costo.

2. **Descenso del gradiente estocástico** Para su desarrollo se realiza una actualización de los parámetros por cada muestra de entrenamiento. Debido a estas actualizaciones frecuentes, los parámetros actualizados tienen una alta variación lo que ocasiona que la función de pérdida fluctúe entre diferentes magnitudes. Esto ayuda a conseguir nuevos y posibles mejoras de mínimos locales. El problema con este método es que debido a las frecuentes actualizaciones y fluctuaciones, dificulta la convergencia a un mínimo exacto.[6]

3. **Descenso del gradiente por mini lotes** Es una mejora a la versión estándar del descenso de gradiente y de la versión estocástica. El método toma lo mejor de las técnicas ya mencionadas y realiza una actualización por cada lote con *n* muestras de entrenamiento en cada uno. Esto reduce la variación en

---

[13] S Patrikar. “Batch, Mini Batch Stochastic Gradient Descent”. En: *https://towardsdata -science.com-batch-mini-batch-stochastic-gradient-descent-7a62ecba642a* (2019).

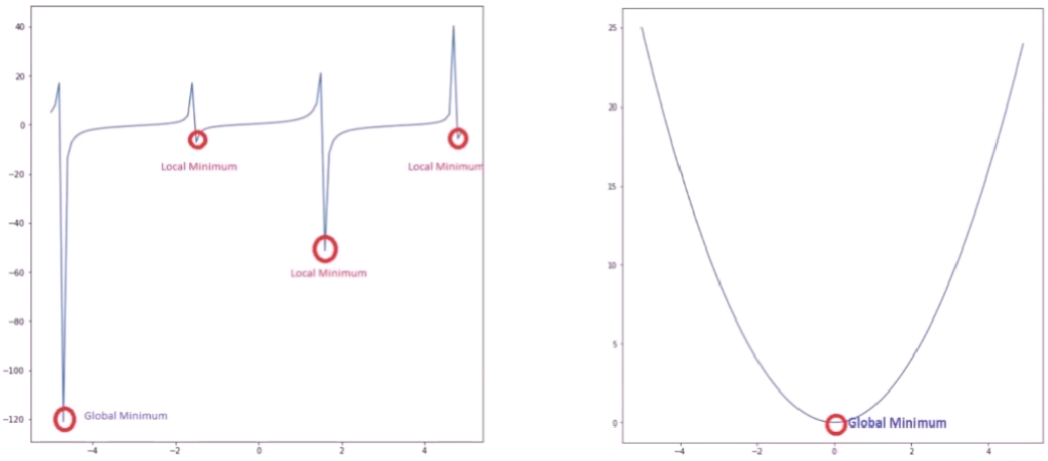
la actualización de los parámetros, lo que puede llevarnos a una mejor y más estable convergencia. Normalmente, este es el algoritmo usado al momento de entrenar una red neuronal.[6]

2.8. FUNCIONES DE PERDIDA O COSTO

Así como hay diferentes tipos de redes neuronales y optimizadores, también hay varias *funciones de pérdida*. Se pueden clasificar en *funciones convexas* y *funciones no convexas*.

- **Funciones no convexas:** En este tipo de funciones se tienen diferentes mínimos locales y solo un mínimo global. La tarea del optimizador es alcanzar el mínimo global, no quedando estancado en un mínimo local. **Figura 6** (izq.)
- **Funciones convexas:** Por el contrario, con estas podemos identificar un único minimo global, el cual es mucho más fácil de obtener. **Figura 6** (der.)

Figura 6. Ejemplos de funciones no convexas y convexas



Fuente: Curso Deep Learning for Computer Vision with Tensor Flow and Keras [figura]. [Consultado: 26 de Diciembre de 2020] Disponible en: https://www.udemy.com/course/deep-learning-for-computer-vision-with-tensor-flow-and-keras/learn/lecture/11159401overview

Se pueden aplicar diferentes tipos de funciones costo dependiendo la situación. Ya sea para problemas de *regresión* en los que lo que se necesita es predecir un valor numérico o problemas de *clasificación* donde el principal objetivo es predecir una clase específica de los datos. La cuestión es que para hacer que el modelo sea mejor al momento de clasificar o predecir, se deben cambiar de alguna manera las variables para los pesos y el sesgo. Ahora, para llevar a cabo esto, se necesita saber como funciona el modelo actualmente, comparando el valor previsto a la salida con el valor deseado.[6] Para realizar esta comparación es necesaria la *función de pérdida*, aquí el por qué de estudiar estas funciones.

2.9. FUNCIONES PARA PROBLEMAS DE REGRESIÓN

Entre las funciones principales de regresión se tienen:

2.9.1. Error de media cuadrática (MSE) Determina la diferencia entre los valores previstos y los valores observados realmente. *n* corresponde al número de muestras.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2}$$

2.9.2. Raíz del error de media cuadrática (RMSE) Corresponde a la raíz cuadrada del *MSE*. El número de muestras se toma como *n*.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

2.9.3. Error medio absoluto (MAE) Con este se calcula el error medio absoluto entre el valor previsto y un valor deseado. *N* corresponde al número de muestras.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2 \tag{4}$$

2.10. FUNCIONES PARA PROBLEMAS DE CLASIFICACIÓN

De las funciones de clasificación más usadas están:

2.10.1. Entropía cruzada Es una medida del rendimiento, una función no convexa y continua que siempre es positiva y si el resultado predictivo del modelo coincide exactamente con el resultado deseado, la función es igual a cero. Las pérdidas en la función aumentan a medida que la probabilidad de predicción diverge del valor real de la etiqueta. Lo que se busca con la optimización es por lo tanto, minimizar el valor de la función para que sea lo más cercano a cero como sea posible.[6] Si se tienen más de dos clases será necesario usar la expresión (5) referente a la función costo, la cual en *Teoría de la información* trata de los bits, bits de información que

son necesarios conocer en valores dudosos de las variables del modelo. Donde *p* es la probabilidad de encontrar una clase específica y *q* la probabilidad de no encontrar esta clase.

$$C(x) = \sum_{j=1}^{n} p(x) log q(x) \tag{5}$$

2.10.2. Divergencia de Kullback-Leibler (KL) Este indicador *(expresión 6)* es muy similar al de *entropía cruzada*, de hecho es relativamente una entropía, porque se centra en el número extra de bits necesarios para codificar los datos. Con el método se mide la diferencia entre las distribuciones de probabilidad *p* y *q*. El valor de esta divergencia nunca será menor a 0, y sólo será igual a 0 si *p* (valor real de la distribución) y *q* (valor previsto) son iguales.[6]

$$KL(P||Q) = \sum_{j=1}^{n} p(x) log \frac{p(x)}{q(x)} \tag{6}$$

2.10.3. Hinge Loss Es una función de pérdida usada para la clasificación de margen máximo, especialmente en *máquinas de vector soporte* (SVM). Esta es además, una función convexa, por lo tanto la mayoría de optimizadores usados en *Machine Learning* funcionan con ella.[14]

2.11. OVERFITTING Y UNDERFITTING

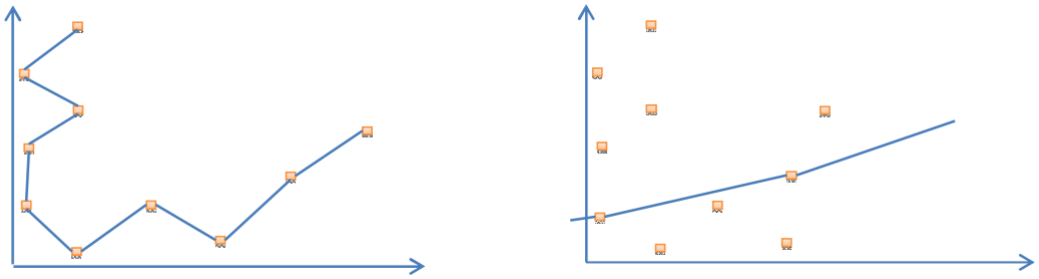
Las principales causas al obtener malos resultados en procesos de *Machine Learning* a menudo suelen ser el *overfitting* o el *underfitting* de los datos. Cuando se entrena un modelo, el objetivo es "hacer encajar" (*fit* en inglés) los datos de entrada entre ellos y con la salida. Tal vez se pueda traducir *overfitting* como "sobreajuste"

---

[14] "Hinge loss". En: *https://handwiki.org/wiki/Hinge_loss* (2020).

y *underfitting* como “subajuste” y hacen referencia al fallo del modelo al generalizar (encajar) los datos que se pretende que adquieran.[15] El *principio de parsimonia*, exige utilizar modelos y procedimientos que contengan todo lo necesario para el modelado pero nada más. Por consiguiente, *Overfitting* (**figura 7** (izq.)) es el uso de modelos o procedimientos que violan esa *parsimonia*, que incluyen más parámetros de los necesarios. Este tipo de modelo tiene un pobre desarrollo predictivo, ya que reacciona de forma exagerada ante el mínimo cambio en los datos de prueba. [16] Por otro lado, *Underfitting* (**figura 7** (der.)) es lo opuesto al *Overfitting*, es decir, el modelo es incapaz de percibir la variabilidad de los datos.[17] En este caso el modelo se entrena con pocos o un tipo de dato, lo que ocasionará que falle en el reconocimiento por falta de muestras al momento del entrenamiento.

Figura 7. Overfitting vs Underfitting



Fuente: OVER-FITTING AND UNDER-FITTING IN SUPERVISED LEARNING]. [Consultado: 26 de Diciembre de 2020] Disponible en: https://www.researchgate.net/Haider-Allamy/Methods-to-Avoid-Over-Fitting-and-Under-Fitting-In-Supervised-Machine-Learning

---

15 “Qué es overfitting y underfitting y cómo solucionarlo”. En: *https://www.aprendemachi-nelearning.com-que-es-overfitting-y-underfitting-y-como-solucionarlo/* (2017).

16 Douglas H. “The problem of Overfitting”. En: *ACS Publications* (2004).

17 Allami H. “Methods to Avoid Over-Fitting and Under-Fitting In Supervised Machine Learning”. En: *Computer Science, Communication Instrumentation Devices* (2014).

Overfitting es el más común de los inconvenientes que se presentan en *Machine Learning*, y para solucionarlo se manejan distintas opciones.[6] Uno de los enfoques principales para tratar este problema es a través de técnicas de *regularización*.

2.12. REGULARIZACION

Es un conjunto de técnicas que se usan para prevenir el *Overfitting*. Se puede definir como cualquier modificación que hacemos a un algoritmo de aprendizaje que tiene como objetivo reducir su error de generalización pero no su error de entrenamiento.[18] La idea es reducir la sumatoria de la *función costo* no penalizada, mas un término de penalización, que puede entenderse como la adición de sesgo y la preferencia de un modelo más simple para reducir la varianza obteniendo pesos más pequeños, y así penalizar los pesos grandes. En *Machine Learning* el objetivo es tener pesos más pequeños.[6]

2.12.1. Lasso, Ridge Regression y ElasticNet Hay distintos tipos de regularización. La regularización *L1* reduce el número de funciones utilizadas en el modelo y aumenta el peso de características que, de otro modo, tenderían a cero los pesos muy reducidos. Además, produce modelos dispersos y reduce la cantidad de ruido en el modelo.[19] Esto lo hace mediante la suma de los valores absolutos de los pesos *(*Ecuación 7). La regularización *L2* produce en general valores de peso más pequeños, lo que estabiliza las ponderaciones cuando hay gran correlación entre

---

[18] A Canziani. “Overfitting and regularization”. En: *https://atcold.github.io/pytorch-Deep-Learning/es/week14/14-3/* (2020).

[19] AWS Documentation. “Amazon Machine Learning, Guía para desarrolladores”. En: *https://docs.aws.amazon.com/es-es/machine-learning/latest/dg/training-parameters .html* (2020).

las funciones.[19] Por su parte, *L2* penaliza mediante la suma de cuadrados de los pesos *(*Ecuación 8), donde *lambda* es un parámetro que controla la intensidad de la penalización. El modelo *L1* penalizado, es conocido como *Lasso*, mientras *L2* penalizado, como *Ridge Regression*. Cuando se usan ambos en un mismo modelo se conoce como: *ElasticNet*.

$$L1 = \lambda \sum_{j=1}^{m} |w_j| \tag{7}$$

$$L2 = \lambda \sum_{j=1}^{m} w_j^2 \tag{8}$$

La función costo para los casos de penalización *Lasso* y *Ridge Regression*, quedan determinadas como:

$$Cost_{ridge} = \sum_{i=1}^{n} (y_i - x_i^T w)^2 + \sum_{j=1}^{m} w_j^2 \tag{9}$$

$$Cost_{lasso} = \sum_{i=1}^{n} (y_i - x_i^T w)^2 + \sum_{j=1}^{m} |w_j| \tag{10}$$

Donde el término asociado a “y” corresponde al valor real, mientras el otro término de la diferencia corresponde al valor previsto, para ambos casos.
Otro enfoque muy usado en *Machine Learning* es el conocido como *Aumento de datos*.

2.12.2. Aumento de Datos *(Data Augmentation)* La tarea de aplicar diferentes transformaciones a los datos disponibles para así sintetizar nuevos datos se conoce

como *Aumento de datos*.[20] Su objetivo es agregar más información acerca de estos para prevenir el problema de *Overfitting*. Algunas de las técnicas más usadas son:

1. **Volteo:** Como su nombre lo indica se usa para voltear imágenes, tanto de forma horizontal como vertical.[6] Esta se aplica en cada dato actual.

2. **Rotación:** Esta puede ser en cualquier sentido y se debe tener en cuenta, que las dimensiones de la imagen pueden cambiar después de aplicar este proceso.[6]

3. **Escala:** La imagen puede ser escalada hacia dentro o hacia afuera.[6]

4. **Recorte:** A diferencia del escalamiento, solo se toma una muestra al azar de una sección de la imagen original. Luego se cambia el tamaño de esta sección, al tamaño de la imagen original.[6]

5. **Traslado:** Involucra el movimiento de la imagen a lo largo del eje *y* o *x*, o incluso ambos.[6]

2.13. FUNCIONES DE ACTIVACIÓN

La finalidad de estas es realizar una transformación del dato que reciben, el cual es la *"red de entrada"*. Con esa transformación se pueden representar problemas no lineales. Es importante el conocimiento de estas funciones y su derivada, ya que la función costo depende de previas funciones de activación en las diferentes capas, por lo tanto si dicha derivada tiende a cero, el gradiente completo de la función costo tenderá a cero y los pesos no serán actualizados.[6] Entonces, estos factores serán importantes al momento de escoger la función de activación más adecuada para la

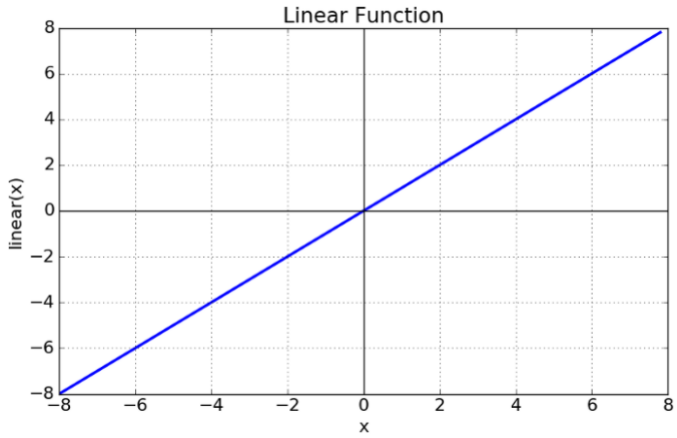
---

[20] M Grochowski A Mikołajczyk. "Data augmentation for improving deep learning in image classification problem". En: *2018 International Interdisciplinary PhD Workshop (IIPhDW)* (2018).

red neuronal a trabajar. Algunas de las principales *funciones de activación* que se encuentran son:

2.13.1. Función Lineal: Esta función (**figura 8**) permite que la entrada sea igual a la salida por lo que si se tiene un red neuronal de varias capas y se aplica una función lineal se dice que es una regresión lineal. Por lo tanto, esta función de activación lineal se usa si a la salida se requiere una regresión lineal y de esta manera a la red neuronal que se le aplica la función va a generar un valor único.[21]

Figura 8. Función lineal



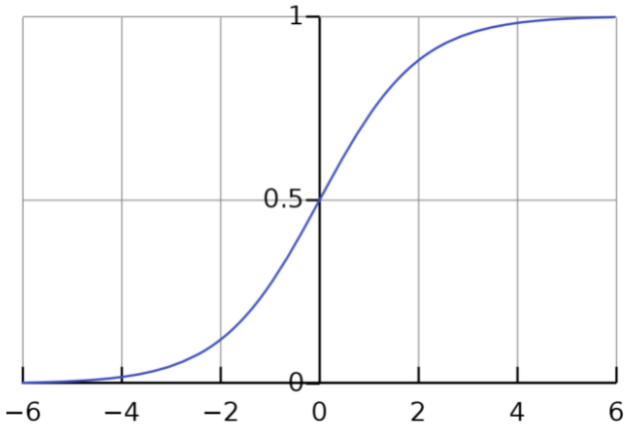
Fuente: Redes neuronales - Bootcamp AI [figura]. [Consultado: 29 de Diciembre de 2020] Disponible en: https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

2.13.2. Función Sigmoide: Es una función muy usada en problemas de clasificación. Es una función no lineal (**figura 9**), cuyo valor de salida se encuentra en un rango de 0 a 1.[6] Si se evalúa la función con valores de entrada muy negativos, es decir x<0 la función será igual a cero, si se evalúa en cero la función dará 0.5 y en

[21] "Redes Neuronales". En: *https://bootcampai.medium.com/redes-neuronales-13349-dd1a5bb: :text=Funci %C3 %B3n %20Linealtext=Por* (2019).

valores altos su valor es aproximadamente a 1. Por lo que esta función se usa en la última capa y se usa para clasificar datos en dos categorías.[21]

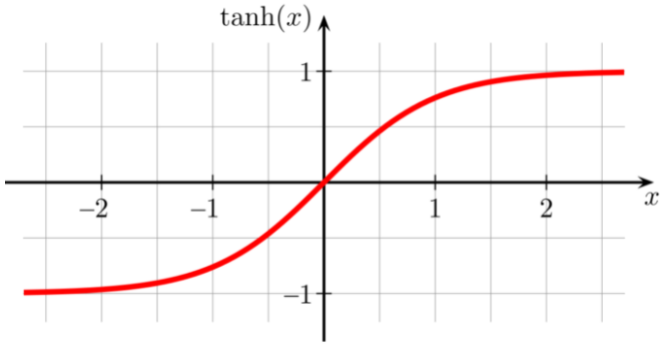
Figura 9. Función sigmoide



Fuente: Redes neuronales - Bootcamp AI [figura]. [Consultado: 29 de Diciembre de 2020] Disponible en: https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

2.13.3. Función tangente hiperbólica Esta función de activación (**figura 10**) llamada tangente hiperbólica tiene un rango de valores de salida entre -1 y 1. Se dice que está función es un escalamiento de la función logística, por lo que a pesar que esta función está centrada tiene un problema similar a la *sigmoide* debido al problema de desaparición del gradiente, que se da cuando en el entrenamiento se genera un error con el algoritmo de propagación hacia atrás y debido a esto el error se va propagando entre las capas, por lo que en cada iteración toma un valor pequeño y la red no puede obtener un buen aprendizaje.[21]

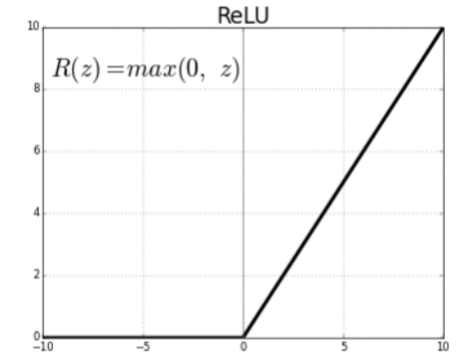
Figura 10. Función tangente hiperbólica



Fuente: Redes neuronales - Bootcamp AI [figura]. [Consultado: 29 de Diciembre de 2020] Disponible en: https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

2.13.4. Función ReLu Está función es la más utilizada debido a que permite el aprendizaje muy rápido en las redes neuronales. Si a esta función (**figura 11**) se le da valores de entrada muy negativos el resultado es cero pero si se le da valores positivos queda igual y además el gradiente de esta función será cero en el segundo cuadrante y uno en el primer cuadrante. Cuando se tiene que la función es igual a cero y su derivada también lo es se genera lo que es la “muerte” de neuronas. Por esta razón la función *ReLu* tiene una variante denominada *Leaky ReLu* que va a prevenir que existan neuronas muertas debido a la pequeña pendiente que existe cuando x<0.[21] La función *ReLu* aporta un gran beneficio y es menor costo computacional que la función *tangente hiperbólica* y *sigmoide*.

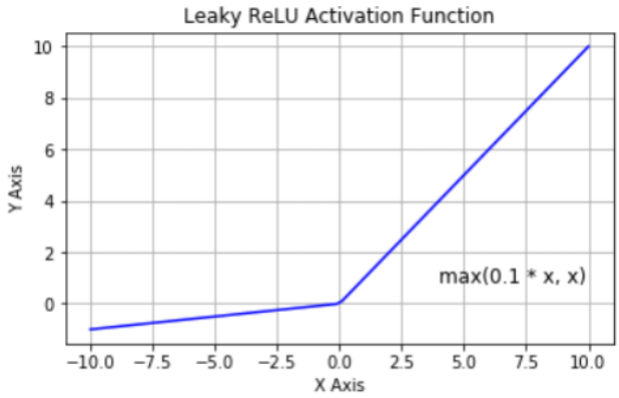
Figura 11. Función ReLu



Fuente: Redes neuronales - Bootcamp AI [figura]. [Consultado: 29 de Diciembre de 2020] Disponible en:
https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

2.13.5. Leaky ReLu Esta resuelve el problema de la *función ReLu*. Este problema puede causar la muerte de muchas neuronas y que estas no respondan haciendo que una gran parte de la red sea pasiva. El objetivo principal con la línea inclinada (**figura 12**) de la función antes del valor 0 de *x*, es permitir que el gradiente sea distinto de cero y se pueda recuperar durante el entrenamiento.

Figura 12. Función Leaky ReLu



Fuente: Redes neuronales - Bootcamp AI [figura]. [Consultado: 29 de Diciembre de 2020] Disponible en: https://www.researchgate.net/figure/Plot-of-the-LeakyReLU-function_fig9_325226633

2.14. MÉTRICAS EN MODELOS DE DETECCIÓN DE OBJETOS

La razón de usar parámetros de medidas es con el fin de evaluar el modelo, y revisar qué tan bien realiza la detección de objetos. Además, permite comparar objetivamente múltiples sistemas de detección o compararlos con un punto de referencia. Aquí se presentan algunas de las principales métricas en tareas de detección de objetos.

2.14.1. Método de evaluación *IoU* Esta métrica es usada para medir la precisión de un detector de objetos frente a un conjunto de datos particular. Para poder aplicar este método es necesario tener las etiquetas que especifican las coordenadas de los objetos en las imágenes y las coordenadas predichas por el detector. Además, el algoritmo premia las predicciones que mejor superpongan los verdaderos valores de los datos.[8]

En la figura 13 se muestran diferentes superposiciones de cuadros delimitadores, y la puntuación que el *IoU* les asigna.

Figura 13. Ejemplo de cálculo de *IoU* para varios cuadros delimitadores.

Fuente: Intersection over Union (IoU) for object detection [figura]. [Consultado: 30 de Diciembre de 2020] Disponible en: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

2.14.2. Media de precisión promedio (mAP) Es el parámetro para medir la precisión de detectores de objetos como el *Faster R-CNN, SSD*, entre otros. Se refiere al promedio de las máximas precisiones en diferentes valores de retorno.[6]

Donde la *precisión* mide la relación entre las detecciones reales de los objetos y el número total de objetos que se predijeron. Y *retorno* calcula la relación entre las detecciones de objetos reales y el número total de objetos. Para calcularlo, se necesita el recuento de los negativos. Dado que cada parte de la imagen en la que no se predijo un objeto se considera un negativo, medir los negativos 'verdaderos' es inútil. Así que sólo se tienen en cuenta los negativos 'falsos', es decir, los objetos que el modelo se ha perdido. Para tener verdaderos positivos y falsos positivos se usa *IoU* con un umbral de 0.5. Si este valor es mayor a 0.5, se considera un verdadero positivo, de lo contrario se considera un falso positivo.[6] Si se definen los términos verdaderos positivo (TP), verdadero negativo (TN), falso positivo (FP) y falso negativo (FN) se pueden obtener las siguientes expresiones para entender mejor los anteriores conceptos.

$$Precisión = \frac{TP}{TP + FP} \tag{11}$$

$$Retorno = \frac{TP}{TP + FN} \tag{12}$$

2.15. aXeleRate

Es un software desarrollado por *Dmitry Maslov*. Este empezó como un programa para entrenar modelos de detección de objetos basados en YOLO v2, para luego exportarlos en formato "*kmodel*" y así poder cargarlos al modulo K210. Además de cumplir este requerimiento, se necesitaba entrenar redes para clasificación de imágenes y programas de conversión adicionales para Raspberry Pi, que fueran compatibles con *TensorFlow lite*. Por lo tanto, el resultado de combinar todos los programas anteriores en un solo paquete fácil de usar y compatible con *Google Colab* dio origen a *aXeleRate*. Esta última característica es muy importante para quienes necesitan correr aplicaciones de clasificación de imágenes, detección de objetos, segmentación, entre otras, en dispositivos de vanguardia. aXeleRate cuenta con una parte de *back-end* que se encarga de extraer características de las imágenes sin procesar. Estas características pasan a la parte de *front-end* donde se extraen en un formato deseado.[22]

2.16. MICROPROCESADORES MAIX

El módulo MAix de la compañía Seeed fue diseñado específicamente para ejecutar inteligencia artificial de vanguardia, ofreciendo un alto desempeño con un bajo consumo de potencia, un precio competitivo en el mercado y la capacidad de conectarse con cualquier dispositivo *ioT*. Por proporcionar soluciones de inteligencia

---

[22] Masolv D. "aXeleRate - Keras-Based Framework for AI on the Edge". En: *https://www.hackster.io/dmitrywat/axelerate-keras-based-framework-for-ai-on-the-edge-96f769* (2020).

artificial de alta calidad y fáciles de implementar, su uso a nivel industrial ha crecido y hoy lo vemos en aplicaciones como mantenimiento predictivo, detección de anomalías, visión artificial, robótica, entre otras.

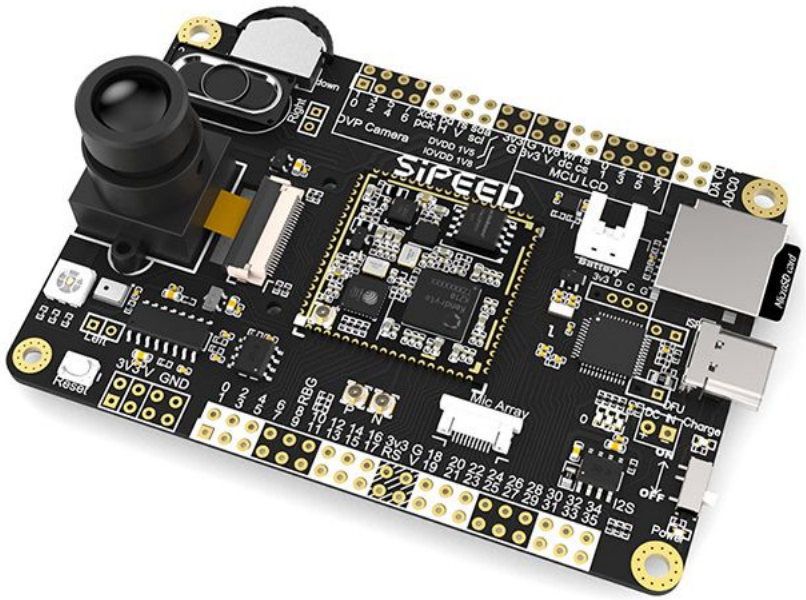
El modulo cuenta con una unidad de procesamiento central(CPU, por sus siglas en ingles) RISC-V de 64 bits de doble núcleo con frecuencia de 400 MHz con capacidad de hasta 800 MHz. También cuenta con un procesador de red neuronal (KPU, por sus siglas en ingles) *Kendryte* K210 con la capacidad de reconocer objetos a 60 fps. Posee un procesador de audio (*APU*, pos sus siglas en ingles) que admite hasta 8 micrófonos con una frecuencia de muestreo de hasta 192 KHz. Tiene la posibilidad de conectar una cámara DVP, ejecutar el algoritmo y mostrarlo en LCD. Tiene 48 GPIO a las que se le puede asignar 255 funciones.[23]

2.16.1. Sipeed MAix Go El kit de desarrollo Sipeed MAixGo con un precio en el mercado de 40.9 Dolares (Consultado el 24 de diciembre de 2020), cuenta con una tarjeta de 88x60 mm, micrófono I2S, altavoz, led RGB, rueda de control, ranura para tarjeta TF, puede ser alimentando con batería de litio o cable USB, tiene una pantalla LCD táctil de 2.8 pulgadas, cámara DVP de lente M12 estándar, puede conectarse por wifi a través de una ESP8285, cuenta con un conector USB tipo C.[23]

---

[23] "Sipeed MAIX GO Suit (MAIX GO + 2.8 inch LCD + ov2640 with M12 lens)". En: *https://www.seeedstudio.com/Sipeed-MAix-GO-Suit-for-RISC-V-AI-IoT-p-2874.html* (2020).

Figura 14. Tarjeta MAix Go



Fuente: Placa de desarrollo Sipeed MAix Go [figura]. [Consultado: 24 de Diciembre de 2020] Disponible en: https://www.seeedstudio.com/Sipeed-MAix-GO-Suit-for-RISC-V-AI-IoT-p-2874.html

MAIX es compatible con el software base de SDK FreeRTOS en C/C++, micropython.

MAIX puede ejecutar modelos de tiny-yolo, mobilenet-v1 y TensorFlow Lite.

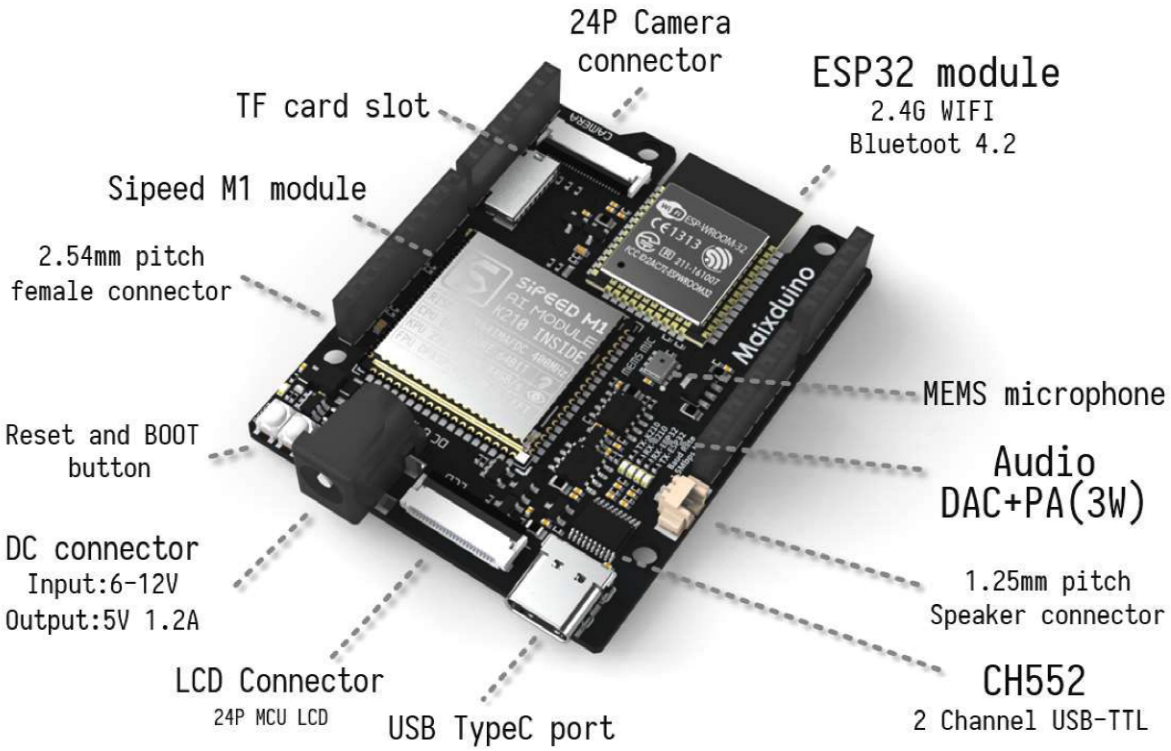
En las especificaciones eléctricas de este kit tenemos:

- Voltaje de suministro entre 4.8 y 5.2 V
- Temperatura de trabajo entre -30℃ y 85℃

- Alimentación externa máxima 600 mA

2.16.2. Sipeed Maixduino El Kit de desarrollo Sipeed Maixduino esta basado en el modulo MAIX para aplicaciones de inteligencia artificial e internet de las cosas, con un precio en la pagina oficial de 23.90 Dolares (Consultado el 24 de diciembre de 2020)

Figura 15. Tarjeta Maixduino



Fuente: Placa de desarrollo Sipeed Maixduino [figura]. [Consultado: 24 de Diciembre de 2020] Disponible en: https://www.seeedstudio.com/Sipeed-Maixduino-Kit-for-RISC-V-AI-IoT-p-4047.html

La tarjeta maixduino cuenta con un modulo ESP32 para realizar conexiones wifi 2.4G y bluetoot 4.2, conector DC de alimentación entre 6 - 12 V, botón de reinicio y de arranque, conector LCD, puerto USB tipo C, conector para cámara de 24P, puerto

para tarjeta TF, micrófono MEMS.[24]

La tarjeta admite los sotfware MaixPy IDE, Arduino, OpenMV IDE y PlatformIO IDE.

Dentro de las principales aplicaciones de esta tarjeta de desarrollo tenemos:

- Desarrollos de hogar inteligentes, como robots de limpieza, altavoces, cerraduras electrónicas
- Aplicaciones en la industria médica como diagnostico, reconocimiento de imágenes médicas, alarmas de emergencia.
- Desarrollo para la industrial en general como clasificaciones inteligente, supervisión de plagas, supervisión de equipos electrónicos.

2.17. INTERNET DE LAS COSAS

El termino internet de las cosas (ioT por sus siglas en ingles) fue acuñado por Kevin Ashton en el año 2009 y habla sobre como es posible que diferentes objetos estén conectados a la web o conectados entre si, va desde los objetos mas básicos conectados a ioT con un sistema de encendido o apagado, hasta objetos mas complejos con sensores que están en tiempo real realizando medidas, procesando información y compartiéndola a la web.[25]

El *ioT* ha sido posible gracias principalmente a dos cosas, la primera es la gran disponibilidad de internet banda ancha que se tiene en este momento, impulsado

---

[24] "Sipeed Maixduino Kit for RISC-V AI + IoT)". En: *https://www.seeedstudio.com/Sipeed-Maixduino-Kit-for-RISC-V-AI-IoT-p-4047.html* (2020).

[25] Adrián V. V. W. Fernando V. E. N. Alberto J. P. L. "Estado del arte de las arquitecturas del internet de las cosas (iot)". En: (2014).

por la demanda de conectividad de los usuarios, la segunda, es la disminución del costo el desarrollo de software y hardware, que se ha visto reflejado en el interés de inversión en *ioT* de los diferentes campos de aplicación de estos sistemas.

Se puede decir que de forma general, las principales aplicaciones de *iot*, están enfocadas en:

- Sociedad

  Desarrollos enfocados en la conectividad de las personas, ciudades, entre otros.[26]

- Medio ambiente

  Desarrollos enfocados en la supervisión, cuidado y aprovechamiento de los recursos naturales, entre otros.[26]

- Industria

  Desarrollos enfocados en el comercio, comunicación de empresas, transacciones financieras, conectividad de autoridades gubernamentales, entre otros.[26]

[26] E. Mohn. “Internet of Things.” En: *Salem Press Encyclopedia of Science.* (2020).

3. DESARROLLO DEL PROYECTO

Para realizar el reconocimiento de gestos manuales, existen varias posibilidades, está la detección de objetos utilizando SSD, *Faster R-CNN*, YOLO entre otras. Pero antes de escoger uno de estos métodos para desarrollar el proyecto, se eligió el microprocesador en el que se iba a implementar el proyecto. *Raspberry Pi*, por su capacidad de procesamiento y la amplia información que se encuentra en la red, era una de las principales opciones, sin embargo, el precio de esta tarjeta de desarrollo de 35 dolares sin cámara (Consultado el 5 de enero del 2021) fue la principal causa para desistir y buscar otra alternativa. Seeed estudio es una compañía que desde el año 2008 se especializa en soluciones de *hardware* para *ioT* e inteligencia artificial a bajo costo, ideal para el desarrollo del proyecto, es por esta razón que se escogió la tarjeta de desarrollo Maixduino **(figura15)** de esta compañía, la cual viene integrada con todo lo que se necesita para hacer detección de objetos con una cámara. También se utilizó la tarjeta de desarrollo MaixGo para comparar el rendimiento de estas dos.

El modulo K210 de las tarjetas de desarrollo *Maix*, procesa redes neuronales en formato *"kmodel"* de hasta 2 MB de tamaño, por lo que se debía escoger una opción de entrenamiento cuyo resultado fuera ligero y se pudiera convertir al formato necesitado. Buscando por la red se encontró *aXeleRate*, un proyecto de un investigador del *MIT* (Instituto Tecnológico de Massachusetts, por sus siglas en ingles) que se especializa en entrenar redes neuronales y convertirlas al formato *"kmodel"* por lo que este sistema se eligió como base para el desarrollo del presente proyecto.

El proyecto se desarrolló en 3 partes:

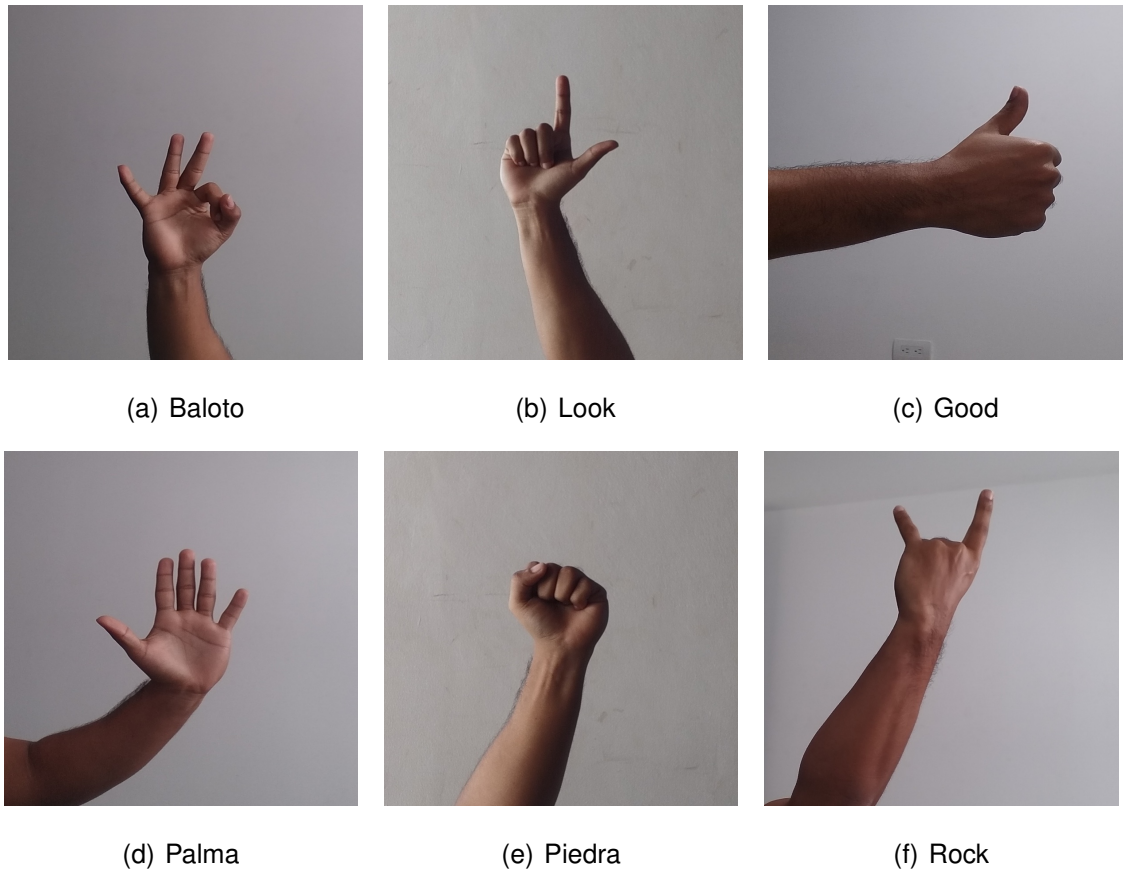
- Creación y configuración de la base de datos
- Entrenamiento de la red neuronal
- Configuración de los microprocesadores *MAIX*

3.1. CREACIÓN Y CONFIGURACIÓN DE LA BASE DE DATOS

Se decidió crear una base de datos propia, con 501 imágenes por cada uno de los gestos manuales o clases que se utilizaron. Las imágenes se tomaron a través de la cámara de los celulares. Se tomaron las fotografías a diferentes horas del día, se vario la luz, la posición del gesto en el cuadro de la cámara y el fondo de las imágenes. Esto para evitar el *Underfitting* porque si se tomaban todas las fotografías con un fondo blanco y con la misma luz, el resultado al entrenar la red neuronal seria muy pobre detección los gestos al variar el ambiente.

3.1.1. Clases o gestos manuales: Se seleccionaron 6 gestos para hacer el reconocimiento en vivo con la cámara del microprocesador, los gestos son:

Figura 16. Clases



(a) Baloto (b) Look (c) Good

(d) Palma (e) Piedra (f) Rock

Fuente: Elaboración propia

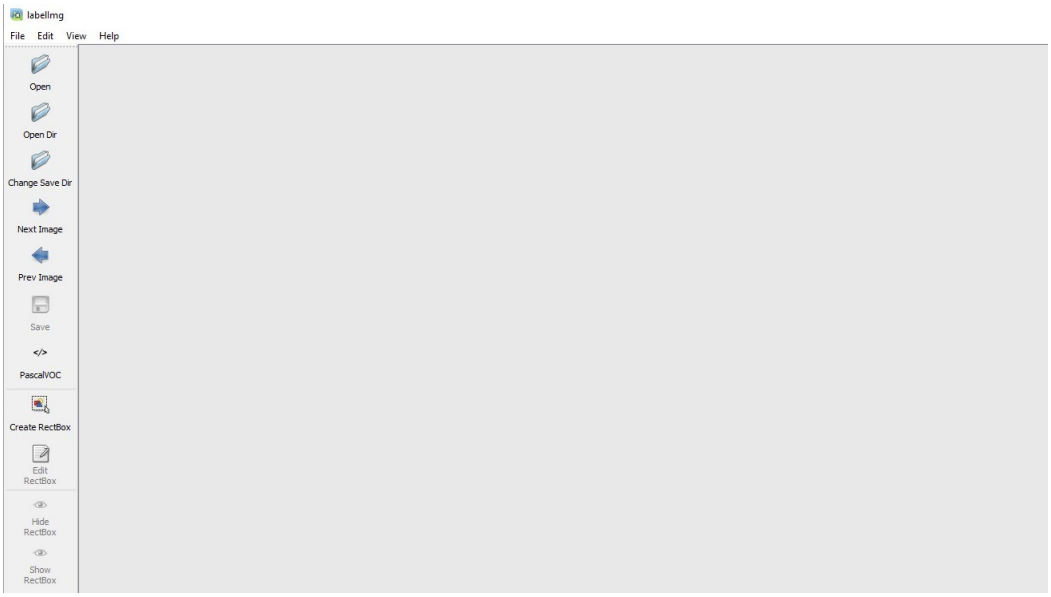
Para cada uno de estos gestos o clases, se tomo el 10 % para validación y el restante para entrenamiento y se separaron en carpetas diferentes.

3.1.2. Etiquetado Una vez que se tuvieron las imágenes, se realizó el etiquetado con ayuda de la herramienta *labelImg*.[27]

---

[27] "LabelImg". En: *https://github.com/tzutalin/labelImg* (2021).

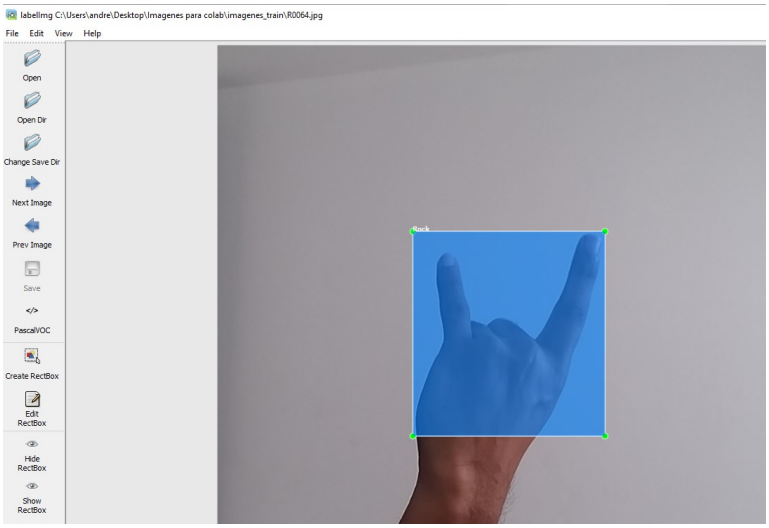
Figura 17. LabelImg



Fuente: Elaboración propia

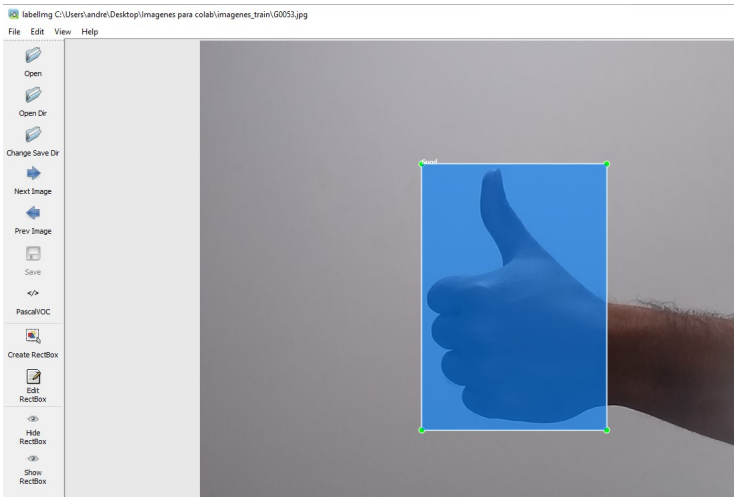
Se utilizó esta herramienta en la opción de etiquetado PascalVoc para generar un archivo de extensión “xml”, se cargó la carpeta donde se encontraban todas las imágenes y se etiquetó de la siguiente manera:

Figura 18. Etiqueta clase Rock



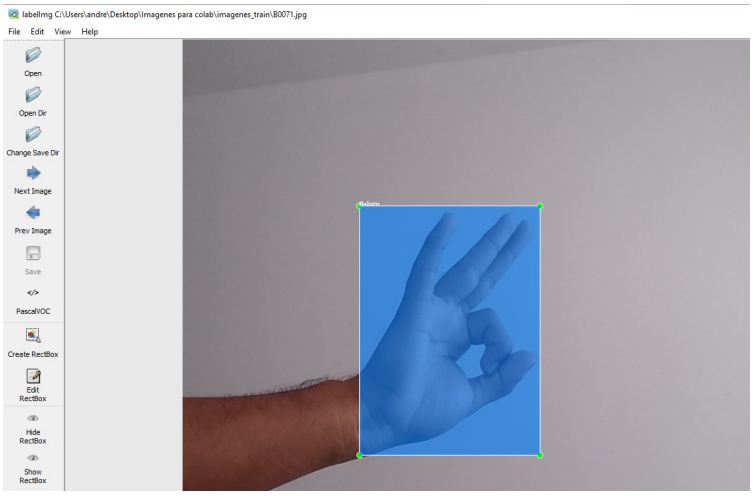
Fuente: Elaboración propia

Figura 19. Etiqueta clase Good



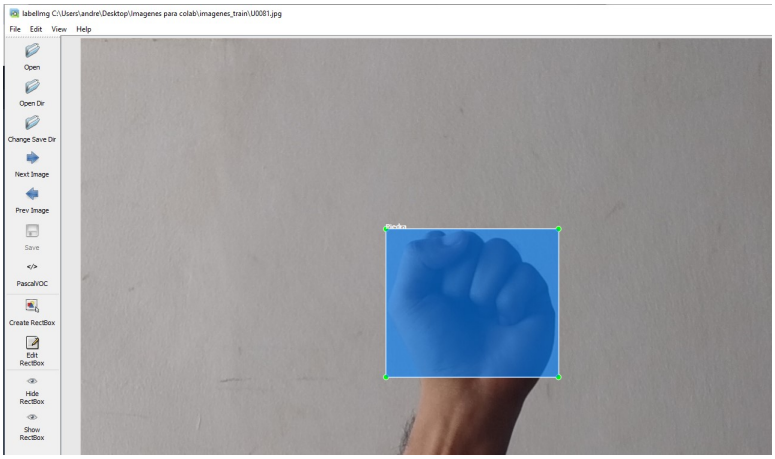
Fuente: Elaboración propia

Figura 20. Etiqueta clase Baloto



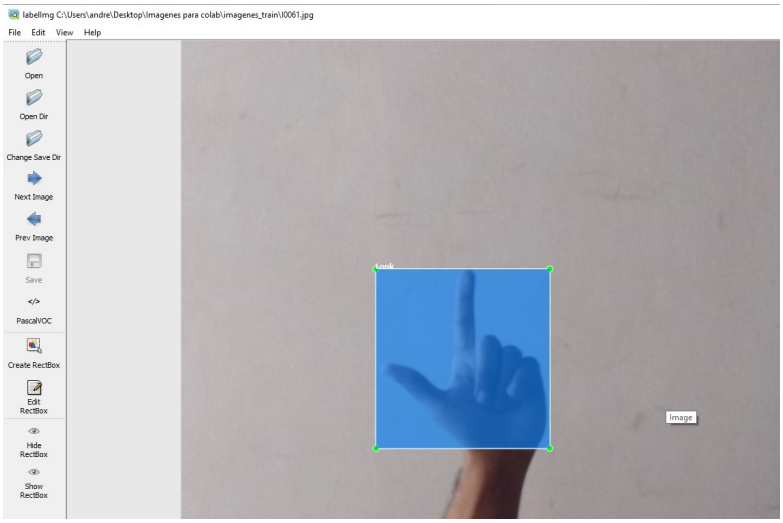
Fuente: Elaboración propia

Figura 21. Etiqueta clase Piedra



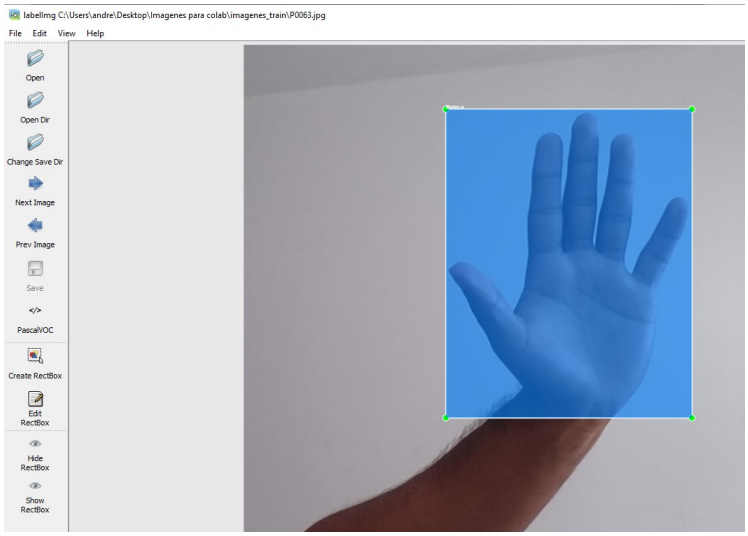
Fuente: Elaboración propia

Figura 22. Etiqueta clase Look



Fuente: Elaboración propia

Figura 23. Etiqueta clase Palma



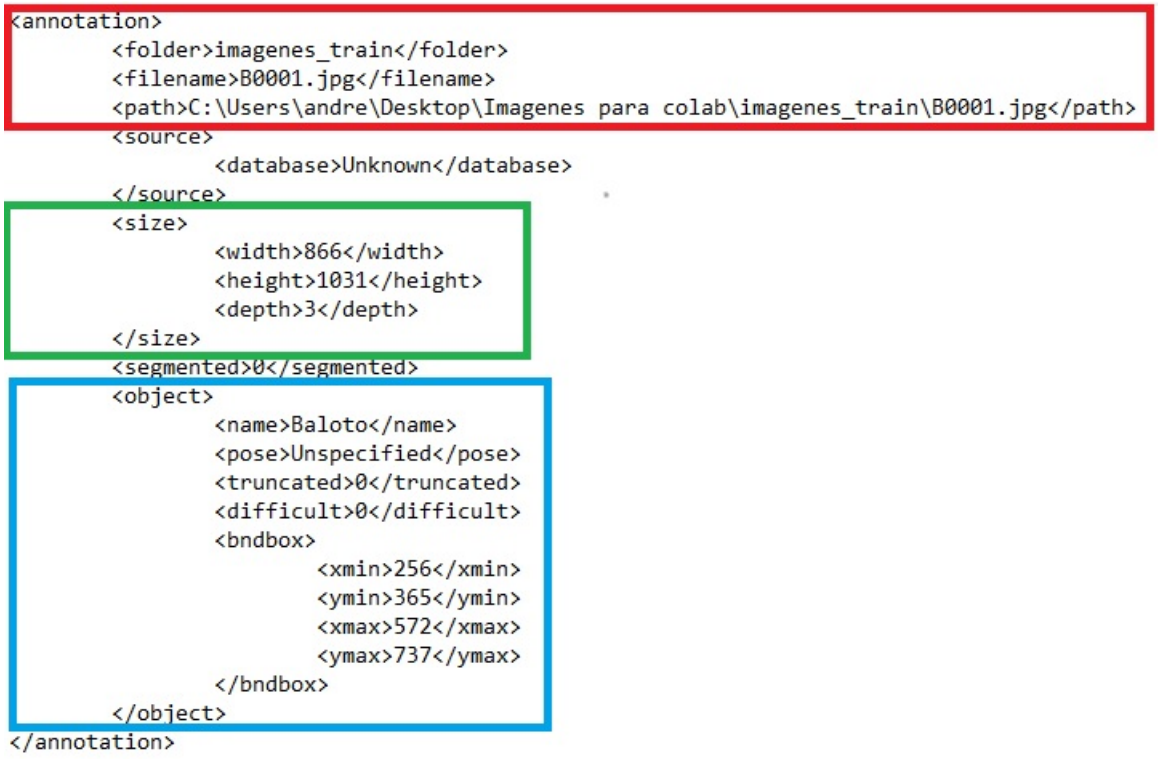
Fuente: Elaboración propia

Para cada una de las etiquetas se seleccionó el gesto completo que se quiere identificar con la red neuronal, algunas imágenes tenían 2 gestos o mas de una misma clase pero no se mezclaron clases en una misma imagen. Este paso se repitió para

todas las imágenes de entrenamiento y verificación y las etiquetas resultantes se guardaron en carpetas separadas.

En la **figura 24**, se tiene el archivo de etiqueta generado por el software LabelImg. En la primera parte en rojo, está el nombre de la carpeta donde se encuentra guardada la imagen que se etiquetó y el nombre de esta. En verde, está el tamaño de la imagen. En azul esta el nombre de la etiqueta generada y la posición de esta.

Figura 24. Archivo generado por LabelImg



Fuente: Elaboración propia

3.2. ENTRENAMIENTO DE LA RED NEURONAL

Para el entrenamiento se decidió construir la red neuronal desde cero pues de esta forma la red resultante es mas liviana que si se hace *transfer learning* que consiste en utilizar los pesos de una red neuronal ya existente. Se utilizaron las arquitecturas MobileNet7_5, MobileNet5_0, MobileNet2_5 para comprobar cual de estas da un mejor desempeño.

Google Colab ofrece un entorno interactivo en linea llamado *notebook colab* el cual cuenta con acceso gratuito a una unidad de procesamiento gráfico (*GPU*, por sus siglas en ingles) ideal para realizar el entrenamiento de las redes neuronales ya que no se necesita contar con una maquina potente. Además, *Colab* cuenta con la posibilidad de conectarse con *Google Drive* para cargar o guardar imágenes, modelos, resultados, entre otras cosas.

Para el entrenamiento se utilizo una *notebook colab*[28] ya creada que hace parte de aXeleRate y se configuró para realizar el entrenamiento:

En la **figura 25** se tiene el código para hacer la conexión con el *Google Drive*, donde previamente se han subido todas las imágenes y etiquetas en carpetas separadas de entrenamiento y validación.

---

[28] "PASCAL-VOC Detection model Training and Inference". En: *https://colab.research.google.com/github/AIWintermuteAI/aXeleRate/blob/master/resources /aXeleRate_pascal20_detector.ipynb* (2021).

Figura 25. Conexion con *Google Drive*

```
[1]  from google.colab import drive
     drive.mount('/content/drive')
```

Fuente: PASCAL-VOC *Detection model Training and Inference*. [Consultado 11 de enero del 2021] Disponible en: https://colab.research.google.com/github/AIWintermuteAI/aXeleRate/blob/master/resources/aXeleRate_pascal20_detector.ipynb

En la **figura 26** se define primero la versión a utilizar para la librería imgaug la cual corresponde a la 0.4 que es con la que aXeleRAte funciona sin ningún problema, luego se copia al colab el repositorio de aXeleRate, se crea una carpeta llamada axelerate y se importan ahí las carpetas setup_training y setup_inference.

Figura 26. librerias

```
#we need imgaug 0.4 for image augmentations to work properly
!pip uninstall -y imgaug && pip uninstall -y albumentations && pip install imgaug==0.4
!git clone https://github.com/AIWintermuteAI/aXeleRate.git
import sys
sys.path.append('/content/aXeleRate')
from axelerate import setup_training, setup_inference
```

Fuente: PASCAL-VOC Detection model Training and Inference. [Consultado 11 de enero del 2021] Disponible en: https://colab.research.google.com/github/AIWintermuteAI/aXeleRate/blob/master/resources/aXeleRate_pascal20_detector.ipynb

En la **figura 27** se están visualizando las imágenes de verificación con sus respectivas etiquetas previas al entrenamiento, en el tamaño de 224x224 píxeles el cual es el tamaño con el que se entrenan imágenes en la arquitectura MobileNet. En num_imgs se puede definir la cantidad de imágenes que se quieren mostrar.

Figura 27. Verificacion

Fuente: PASCAL-VOC Detection model Training and Inference. [Consultado 11 de enero del 2021] Disponible en: https://colab.research.google.com/github/AIWintermuteAI/aXeleRate/blob/master/resources/aXeleRate_pascal20_detector.ipynb

En la **figura 28** se tiene el modelo ya configurado que necesita aXeleRate para realizar el entrenamiento, este está constituido por :

- *"model"* :
    - "type" : El tipo de red neuronal que se quiere entrenar, puede ser para detección de objetos (*Detector*), para clasificación (*Classifier*) o para segmentación (*SegNet*).[28]
    - "architecture": La arquitectura con la que se quiere construir la red, para detección de objetos están: Tiny Yolo, MobileNet1_0, MobileNet7_5, MobileNet5_0, MobileNet2_5, SqueezeNet, NASNetMobile, DenseNet121, ResNet50.[28]
    - "input_size": El tamaño al que se van a convertir las imágenes para el entrenamiento, los modelos MobileNet entrenan con un tamaño de 224x224 píxeles.
    - "anchors": Anclajes del modelo YOLO.
    - "labels": El nombre de las etiquetas que previamente se le han puesto a las imágenes utilizando la herramienta de labelImg.

- “coord_scale” : Determina cuánto penalizar la posición incorrecta y las predicciones de tamaño del cuadro de detección.[28]
- “class_scale” : Determina cuánto penalizar la predicción de clase incorrecta.[28]
- “object_scale” : Determina cuánto penalizar la predicción incorrecta de la confianza de los predictores de objetos.[28]
- “no_object_scale” : determina cuánto penalizar la predicción incorrecta de la confianza de los predictores que no son objetos.[28]

- “weights” :
  - “full” : Si se quiere entrenar la red utilizando como base otra red ya entrenada, aquí se pone esa red, de lo contrario, si se quiere entrenar la red neuronal desde cero, se deja vacío.

- “train” :
  - “actual_epoch” : La cantidad de ciclos que se quiere entrenar el conjunto de datos completo.
  - “train_image_folder” : Ubicación de las imágenes de entrenamiento.
  - “train_annot_folder” : Ubicación de las etiquetas de las imágenes de entrenamiento.
  - “train_times” : Las veces que se quiere realizar el entreno.
  - “valid_image_folder” : Ubicación de las imágenes de verificación.
  - “valid_annot_folder” : Ubicación de las etiquetas de las imágenes de validación.
  - “valid_times” : Las veces que se quiere realizar la validación.

- “valid_metric” : El método para comprobar la precisión de la red neuronal.
- “batch_size” : La cantidad de imágenes que se quieren procesar en un lote de entrenamiento. No puede superar a la cantidad de imágenes de verificación.
- “learning_rate” : La tasa de aprendizaje.
- “saved_folder” : El sitio donde se quiere guardar los resultados del entrenamiento.
- “augumentation” : Si se quiere utilizar la técnica para aumentar la base de datos.
- “is_only_detect” :

- “converter” : El tipo de formato que se quiere convertir la red neuronal entrenada.

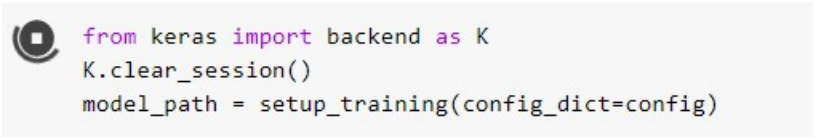
Figura 28. Configuración para entrenamiento de las *CNN* desde cero

```
config = {
        "model":{
            "type":                 "Detector",
            "architecture":         "MobileNet5_0",
            "input_size":           224,
            "anchors":              [0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828],
            "labels":               ["Baloto", "Good", "Look", "Palma", "Rock", "Piedra"],
            "coord_scale" :     1.0,
            "class_scale" :     1.0,
            "object_scale" :    5.0,
            "no_object_scale" :   1.0
        },
        "weights" : {
            "full":           "",
            "backend":          "imagenet"
        },
        "train" : {
            "actual_epoch":         50,
            "train_image_folder":   "drive/MyDrive/Desarrollos/axelerate/data/imagenes_train",
            "train_annot_folder":   "drive/MyDrive/Desarrollos/axelerate/data/label_train",
            "train_times":          1,
            "valid_image_folder":   "drive/MyDrive/Desarrollos/axelerate/data/imagenes_test",
            "valid_annot_folder":   "drive/MyDrive/Desarrollos/axelerate/data/label_test",
            "valid_times":          1,
            "valid_metric":         "mAP",
            "batch_size":           7,
            "learning_rate":        1e-4,
            "saved_folder":       F"/content/drive/MyDrive/Desarrollos/axelerate/output/MobileNet5_0_sinpeace",
            "first_trainable_layer": "",
            "augumentation":        True,
            "is_only_detect" :    False
        },
        "converter" : {
            "type":           ["k210","tflite"]
        }
    }
```

Fuente: PASCAL-VOC Detection model Training and Inference. [Consultado 11 de enero del 2021] Disponible en: https://colab.research.google.com/github/AIWintermuteAI/aXeleRate/blob/master/resources/aXeleRate_pascal20_detector.ipynb

Una vez que se tuvo el archivo de configuración listo se dio inicio al entrenamiento (**figura 29**) y la conversión de la red neuronal al modelo de extensión “kmodel”.

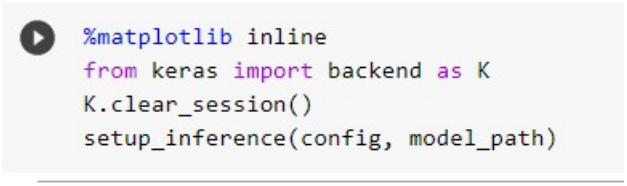
Figura 29. Entrenamiento



Fuente: PASCAL-VOC Detection model Training and Inference. [Consultado 11 de enero del 2021] Disponible en: https://colab.research.google.com/github/AIWintermuteAI/aXeleRate/blob/master/resources/aXeleRate_pascal20_detector.ipynb

Si se quiere hacer la verificación de la red neuronal ya entrenada se utiliza el código de la **figura 30** para someter las imágenes de verificación a la red y mirar el resultado de la detección de objetos.

Figura 30. Verificacion



Fuente: PASCAL-VOC Detection model Training and Inference. [Consultado 11 de enero del 2021] Disponible en: https://colab.research.google.com/github/AIWintermuteAI/aXeleRate/blob/master/resources/aXeleRate_pascal20_detector.ipynb

Al finalizar el entrenamiento, se obtuvo una red neuronal ligera en formato “kmodel” lista para ser cargada a las tarjetas de desarrollo.

Este proceso se repitió para los 3 modelos de arquitecturas MobileNet.

3.3. CONFIGURACIÓN MICROPROCESADORES MAIX

Para la configuración de las tarjetas Maixduino y MaixGo se realizó el mismo proceso.

3.3.1. Carga de firmware y modelo entrenado Utilizando la aplicación kflash_guiV1.6[29], se cargó a la memoria flash del modulo k210 en la ubicación 0x00000 un archivo micropython firmware como se ve en la **figura 31**, descargado de la pagina oficial de Sipeed[30], se utilizó la versión maixpy_v0.6.2_8_g78fac709.bin para la tarjeta Maixduino y la versión maixpy_v0.5.0_125_gd4bdb25.bin para la MaixGo porque son las que no presentaron errores para esta aplicación. Además, se debe especificar en *"board"* la tarjeta a la que se le va a cargar el archivo, en *"port"* el puerto *USB* al que se ha conectado la misma.

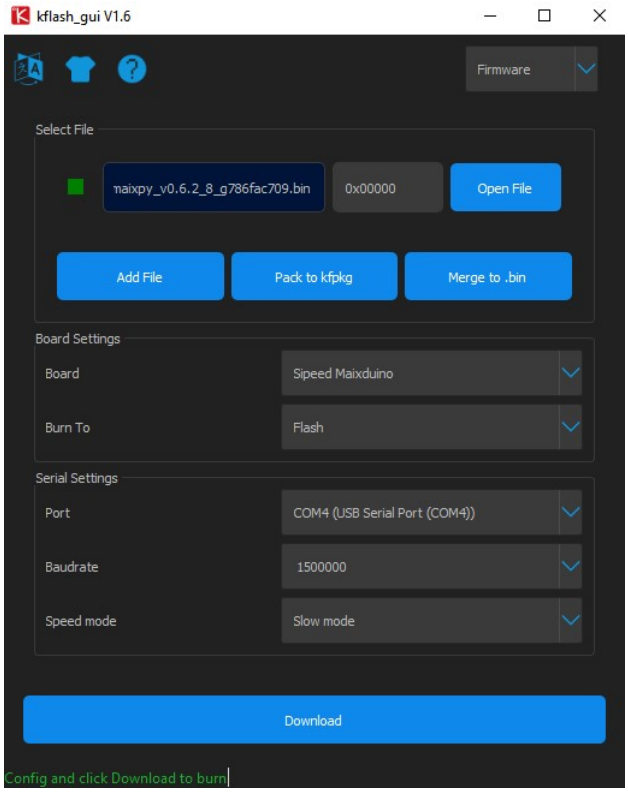
Para cargar el modelo entrenado se tienen 2 opciones. La primera opción es cargar el modelo a la memoria flash 0x300000 utilizando la aplicación kflash_guiV1.6[29]. La segunda forma es cargar el modelo a una memoria SD y luego introducirla a las tarjetas de desarrollo y desde allí correr el modelo.

---

29 "kflash_gui". En: *https://github.com/sipeed/kflash_gui* (2020).

30 "Sipeed firmware". En: *https://cn.dl.sipeed.com/shareURL/MAIX/MaixPy/release /master/ maixpy_v0.6.2_8_g786fac709* (2021).

Figura 31. Kflash



Fuente: Elaboración propia

3.3.2. Programación de las tarjetas MAIX Para la programación de las tarjetas se uso el software Maixpy IDE versión 0.2.5[31]. Este programa utiliza el lenguaje de programación micropython. Se programaron las tarjetas para hacer la detección de los gestos manuales y la conexión ioT. En el sistema ioT se está realizando la conexión y desconexión de la red wifi, se esta obteniendo el ping y la direccion IP de la pagina www.uis.edu.co. El código se desarrolló basado en los ejemplos[32] del

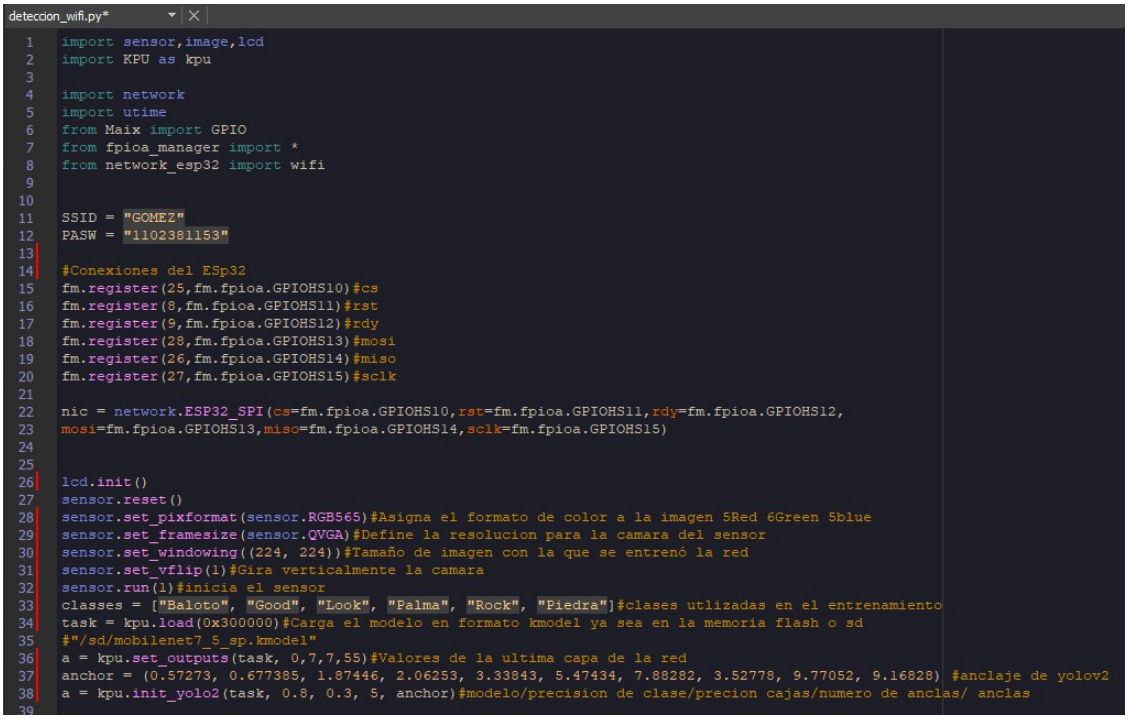
---

31 "Maixpy IDE V0.2.5". En: *https://dl.sipeed.com/shareURL/MAIX/MaixPy/ide/v0.2.5* (2020).

32 "MaixPy Scripts". En: *https://github.com/sipeed/MaixPy_scripts* (2021).

mismo software Maixpy IDE y es el siguiente:

Figura 32. Código para detección de gestos e ioT



```
deteccion_wifi.py*
1  import sensor,image,lcd
2  import KPU as kpu
3
4  import network
5  import utime
6  from Maix import GPIO
7  from fpioa_manager import *
8  from network_esp32 import wifi
9
10
11 SSID = "GOMEZ"
12 PASW = "1102381153"
13
14 #Conexiones del ESp32
15 fm.register(25,fm.fpioa.GPIOHS10)#cs
16 fm.register(8,fm.fpioa.GPIOHS11)#rst
17 fm.register(9,fm.fpioa.GPIOHS12)#rdy
18 fm.register(28,fm.fpioa.GPIOHS13)#mosi
19 fm.register(26,fm.fpioa.GPIOHS14)#miso
20 fm.register(27,fm.fpioa.GPIOHS15)#sclk
21
22 nic = network.ESP32_SPI(cs=fm.fpioa.GPIOHS10,rst=fm.fpioa.GPIOHS11,rdy=fm.fpioa.GPIOHS12,
23 mosi=fm.fpioa.GPIOHS13,miso=fm.fpioa.GPIOHS14,sclk=fm.fpioa.GPIOHS15)
24
25
26 lcd.init()
27 sensor.reset()
28 sensor.set_pixformat(sensor.RGB565)#Asigna el formato de color a la imagen 5Red 6Green 5blue
29 sensor.set_framesize(sensor.QVGA)#Define la resolucion para la camara del sensor
30 sensor.set_windowing((224, 224))#Tamaño de imagen con la que se entrenó la red
31 sensor.set_vflip(1)#Gira verticalmente la camara
32 sensor.run(1)#inicia el sensor
33 classes = ["Baloto", "Good", "Look", "Palma", "Rock", "Piedra"]#clases utlizadas en el entrenamiento
34 task = kpu.load(0x300000)#Carga el modelo en formato kmodel ya sea en la memoria flash o sd
35 #"/sd/mobilenet7_5_sp.kmodel"
36 a = kpu.set_outputs(task, 0,7,7,55)#Valores de la ultima capa de la red
37 anchor = (0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828) #anclaje de yolov2
38 a = kpu.init_yolo2(task, 0.8, 0.3, 5, anchor)#modelo/precision de clase/precion cajas/numero de anclas/ anclas
39
```

Fuente: Elaboración propia.

Figura 33. Código para detección de gestos e ioT

```
while(True):
    img = sensor.snapshot().rotation_corr(z_rotation=90.0)#Captura la imagen
    a = img.pix_to_ai()
    code = kpu.run_yolo2(task, img)#ejecuta la red neuronal con la imagen tomada
    if code:
        for i in code: #Muestra en la lcd la respuesta
            a=img.draw_rectangle(i.rect(),color = (0, 255, 0))
            a = img.draw_string(i.x(),i.y(), classes[i.classid()], color=(255,0,0), scale=3)
            etiqueta=classes[i.classid()]#obtene la case

#CLASE GOOD
            if etiqueta == 'Good':
                print("Escaeno de redes wifi iniciado")
#Hace el escaneo de redes wifi disponible
                enc_str = ["OPEN", "", "WPA PSK", "WPA2 PSK", "WPA/WPA2 PSK", "", "", ""]
                aps = nic.scan()
                for ap in aps:
                    print("SSID:{:^20}, ENC:{:>5} , RSSI:{:^20}".format(ap[0], enc_str[ap[1]], ap[2]))

#CLASE BALOTO
            if etiqueta == 'Baloto':
                print("Conexion a red Wifi iniciada")
                nic.connect("GOMEZ", "1102381153")#Se conecta a una red wifi

                if nic.isconnected() == False:
                    for i in range(5):
                        try:
                            nic.reset()
                            print('try AT connect wifi...')
                            nic.connect(SSID, PASW)
                            if nic.isconnected():
                                break
                        except Exception as e:
                            print(e)
                print('Network state:', nic.isconnected(), nic.ifconfig())

#CLASE ROCK
            if etiqueta == 'Rock':
                print("Desconexión de red Wifi")
                nic.disconnect()#Se desconecta a una red wifi
                print('Network state:', nic.isconnected(), nic.ifconfig())

```

Fuente: Elaboración propia.

Figura 34. Código para detección de gestos e ioT

```
#CLASE PIEDRA
            if etiqueta == 'Piedra':
                print("Obtenteniendo ping de www.uis.edu.co ...")
                nic.connect("GOMEZ", "1102381153")#Se conecta a una red wifi
                print('Network state:', nic.isconnected(), nic.ifconfig())
                if nic.isconnected() == False:
                    for i in range(5):
                        try:
                            nic.reset()
                            print('try AT connect wifi...')
                            nic.connect(SSID, PASW)
                            if nic.isconnected():
                                break
                        except Exception as e:
                            print(e)
                print("ping_uis.edu.co:", nic.ping("uis.edu.co"), "ms")#Obtiene el ping de una pag web
                nic.disconnect()

#CLASE LOCK
            if etiqueta == 'Look':
#Obtiene la direccion IP de un sitio WEB
                print("Obteniendo IP de sitio web www.uis.edu.co")

                def enable_esp32():
                    from network_esp32 import wifi
                    if wifi.isconnected() == False:
                        for i in range(5):
                            try:
                                # Running within 3 seconds of power-up can cause an SD load error
                                # wifi.reset(is_hard=False)
                                wifi.reset(is_hard=True)
                                print('try AT connect wifi...')
                                wifi.connect(SSID, PASW)
                                if wifi.isconnected():
                                    break
                            except Exception as e:
                                print(e)
                        print('network state:', wifi.isconnected(), wifi.ifconfig())
```

Fuente: Elaboración propia.

Figura 35. Código para detección de gestos e ioT

```
enable_esp32()

def enable_espat():
    from network_espat import wifi
    if wifi.isconnected() == False:
        for i in range(5):
            try:
                # Running within 3 seconds of power-up can cause an SD load error
                # wifi.reset(is_hard=False)
                wifi.reset()
                print('try AT connect wifi...')
                wifi.connect(SSID, PASW)
                if wifi.isconnected():
                    break
            except Exception as e:
                print(e)
    print('network state:', wifi.isconnected(), wifi.ifconfig())

#enable_espat()

# from network_w5k import wlan

try:
    import usocket as socket
except:
    import socket

TestHttps = False

def main(use_stream=True):
    s = socket.socket()
    s.settimeout(1)
    host = "www.uis.edu.co"
    if TestHttps:
        ai = socket.getaddrinfo(host, 443)
    else:
        ai = socket.getaddrinfo(host, 80)
    print("Address infos:", ai)
    addr = ai[0][-1]
    for i in range(5):
        try:
            print("Connect address:", addr)
            s.connect(addr)
```

Fuente: Elaboración propia.

Figura 36. Código para detección de gestos e ioT

```
                            if TestHttps: # ssl
                                try:
                                    import ussl as ssl
                                except:
                                    import ssl
                                tmp = ssl.wrapsocket(s, server_hostname=host)
                                tmp.write(b"GET / HTTP/1.1\r\n\r\n")
                            else:
                                s.write(b"GET / HTTP/1.1\r\n\r\n")
                            data = (s.readline('\r\n'))
                            print(data)
                            with open('test.txt', 'wb') as f:
                                f.write(data)

                        except Exception as e:
                          print(e)

                    s.close()

                main()

            print(etiqueta)
        a = lcd.display(img)

    else:
        a = lcd.display(img)
a = kpu.deinit(task)
```

Fuente: Elaboración propia.

En el anterior código se observa como a la detección de algunos gestos se le asigno una tarea. A la clase *"Good"* se le asignó el escaneo de las redes wifi cercanas a la tarjeta de desarrollo, a la clase *"Baloto"* se le dio la tarea de conectarse a la red wifi, a la clase *"Rock"* la desconexión a la red wifi, a la clase *"Piedra"* la obtención del ping y finalmente a la clase *"Look"* la obtención de la IP de una pagina web.

Para el correcto funcionamiento del código se debe primero cargar el archivo llamado "network_esp32" que se encuentra en los ejemplos[32] utilizando la ruta: herramientas, *transfer file to board*.

El *notebook Colab*, las imágenes, los códigos y todo lo que se necesita para desarrollar este proyecto se puede encontrar en el repositorio GitHub junto con vídeos explicativos que hacen mas interactivo y fluido el aprendizaje. **(https://github.com/AndresFelipeGomez/Deteccion-de-gestos-manuales-con-Maixduino-y-MaixGo)**.

4. RESULTADOS

Se entrenaron 3 redes neuronales con las arquitecturas MobileNet5_0, MobileNet7_5 y MobileNet2_5, estas redes fueron entrenadas con las mismas imágenes y con los mismos parámetros que podemos observar en la **figura 28** donde se resalta, un *Epoch* de 50 y *batch size* de 7. Ahora se evidenciaran los resultados obtenidos por cada red, con las imágenes de verificación y con la detección de gestos manuales en vivo.

4.1. MOBILENET5_0

En el entrenamiento de la red neuronal hecha con la arquitectura MobileNet5_0 el mejor *Epoch* en el que se obtuvo el mayor mAP fue el 46 como se ve en la **figura 37**, en este se obtuvo un mAP=0.9886 donde el mejor desempeño fue para la clase “Palma” con 0.9963

Figura 37. Mejor desempeño de entrenamiento MobileNet5_0

```
Epoch 46/50
386/386 [==============================] - 70s 178ms/step - loss: 0.0905 - val_loss: 0.0651


Baloto 0.9925
Good 0.9935
Look 0.9800
Palma 0.9963
Rock 0.9821
Piedra 0.9874
mAP: 0.9886
mAP improved from 0.9801165522465056 to 0.9886396937692455, saving model to /content/drive/MyDrive
Epoch 00045: Learning rate is 1.7788294643315884e-06.
```

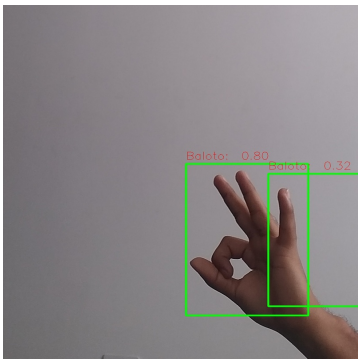
Fuente: Elaboración propia

4.1.1. Detección de imágenes de verificación Se hizo la prueba de la red neuronal sometiendo las imágenes de verificación que corresponden a 50 de cada clase

y el resultado fue le siguiente:

Para esta red neuronal en la clase “Baloto” se tuvo un solo fallo (Figura 38) en la detección del gesto, este fallo corresponde a una doble detección por parte de la red y se dice que es un falso positivo.

Figura 38. Fallos en la detección clase Baloto MobileNet5_0

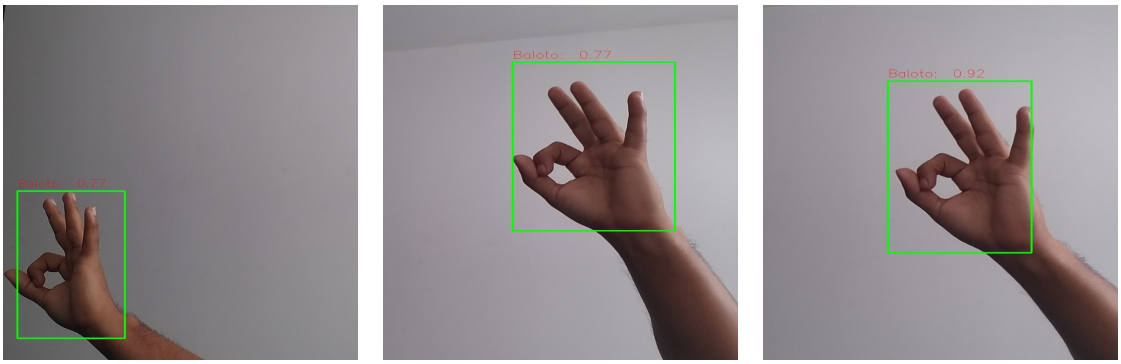


(a) Doble clase

Fuente: Elaboración propia

Algunas de las 49 detecciones correctas de la clase “Baloto” se muestran en la **figura 39**.

Figura 39. Algunos aciertos en la detección clase Baloto MobileNet5_0

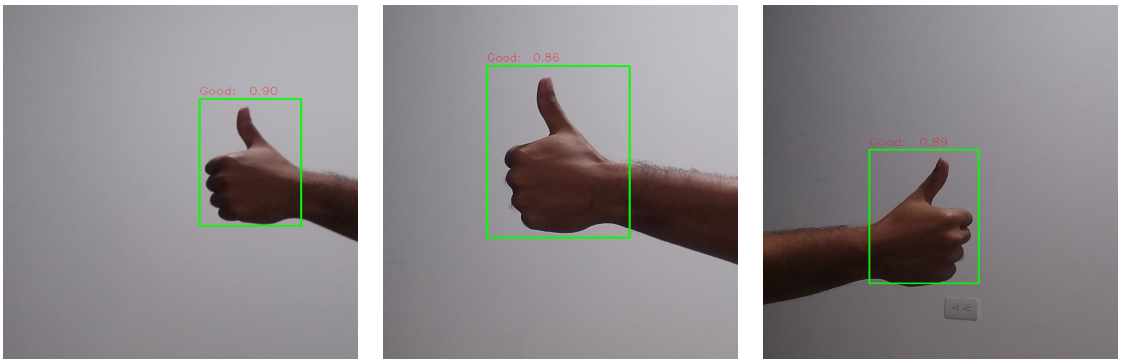


(a) Probabilidad 0.77 (b) Probabilidad 0.77 (c) Probabilidad 0.92

Fuente: Elaboración propia

Para la clase “Good” no hubo fallos en la detección y algunas imágenes son presentadas en la **figura 40**.

Figura 40. Algunos aciertos en la detección clase Good MobileNet5_0



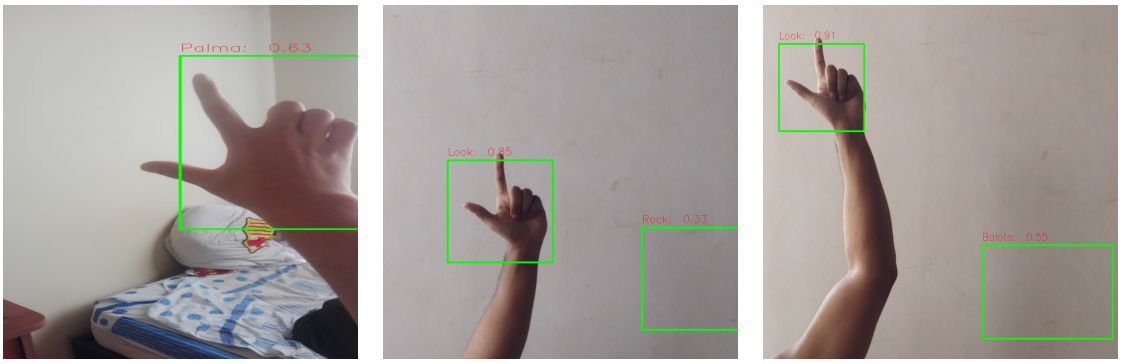
(a) Probabilidad 0.90 (b) Probabilidad 0.86 (c) Probabilidad 0.89

Fuente: Elaboración propia

La clase “Look” tuvo 3 fallos en la detección como se muestra en la **figura 41**, el fallo **41(a)**, corresponde a un error en la clase detectada, el **42(b)**, aunque se detecta

correctamente la clase “Look” aparece una detección incorrecta de otra clase o falso positivo y en el **43(c)** pasó lo mismo.

Figura 41. Fallos en la detección clase Look MobileNet5_0

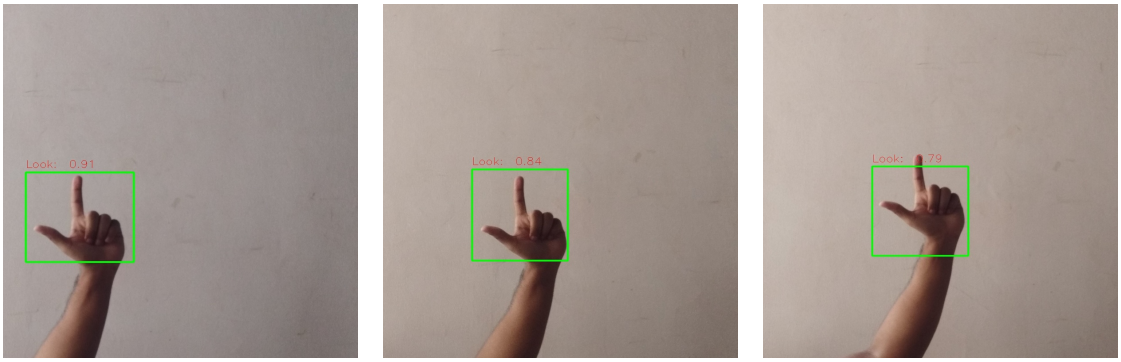


(a) Clase incorrecta (b) Doble Clase (c) Doble clase

Fuente: Elaboración propia

Algunos aciertos en la detección de la clase “Look” son presentados en la **figura 42**.

Figura 42. Algunos aciertos en la detección clase Look MobileNet5_0



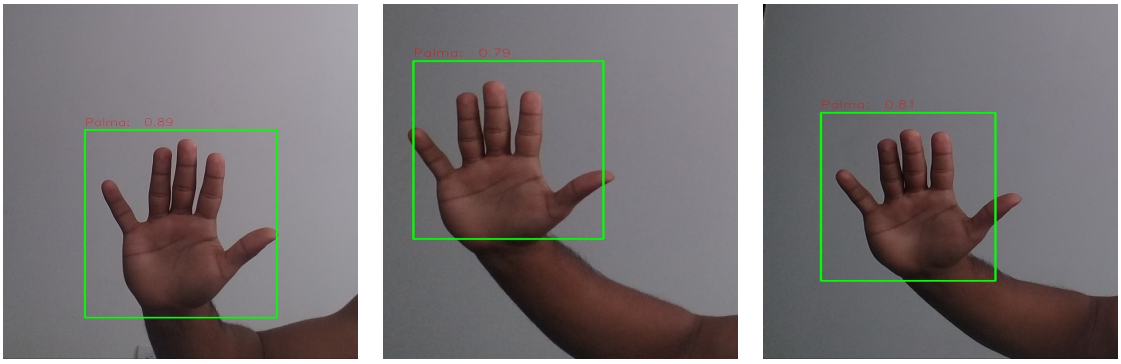
(a) Probabilidad 0.91 (b) Probabilidad 0.84 (c) Probabilidad 0.79

Fuente: Elaboración propia

La clase “Plama” no tuvo errores en la detección y algunos aciertos son presentados

en la **figura 43**.

Figura 43. Algunos aciertos clase Palma MobileNet5_0

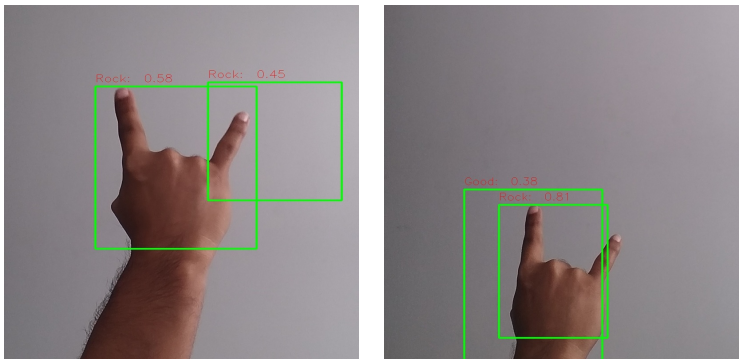


(a) Probabilidad 0.89 (b) Probabilidad 0.79 (c) Probabilidad 0.81

Fuente: Elaboración propia

La clase “Rock” tuvo 2 fallas en su detección y son presentadas en la **figura 44**.

Figura 44. Fallas en la detección clase Rock MobileNet5_0

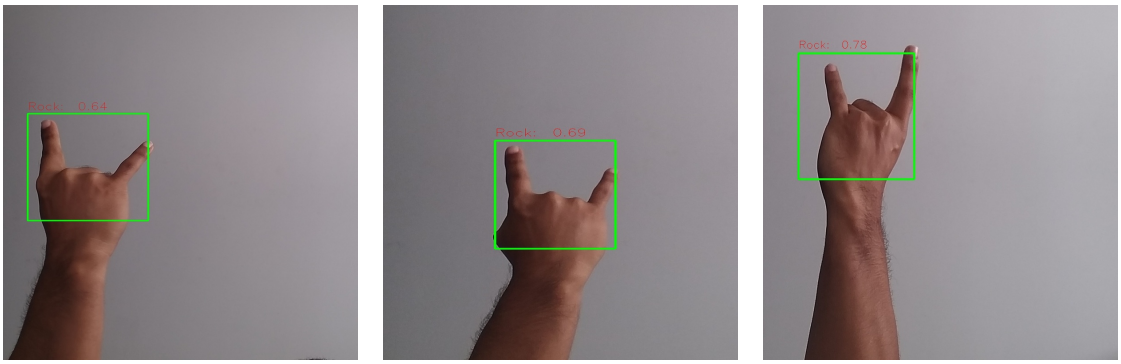


(a) Doble clase detectada (b) Fallo clase detectada

Fuente: Elaboración propia

Algunos de los 48 aciertos en la detección de la clase Rock son presentados en la **figura 45**.

Figura 45. Algunos aciertos en la detección clase Rock MobileNet5_0

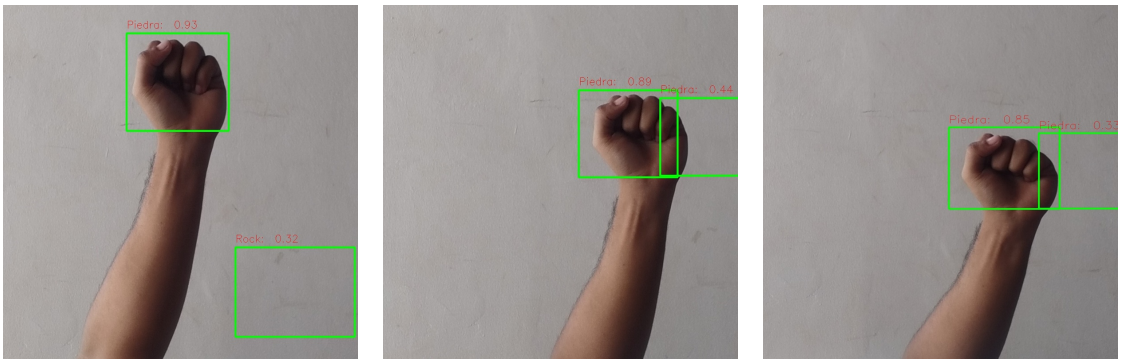


(a) Probabilidad 0.64 (b) Probabilidad 0.69 (c) Probabilidad 0.78

Fuente: Elaboración propia

La clase “Piedra” tuvo 3 errores en la detección y son mostrados en la **figura 46**, en las tres figuras se tiene una detección correcta y un falso positivo.

Figura 46. Fallos en la detección clase Piedra MobileNet5_0

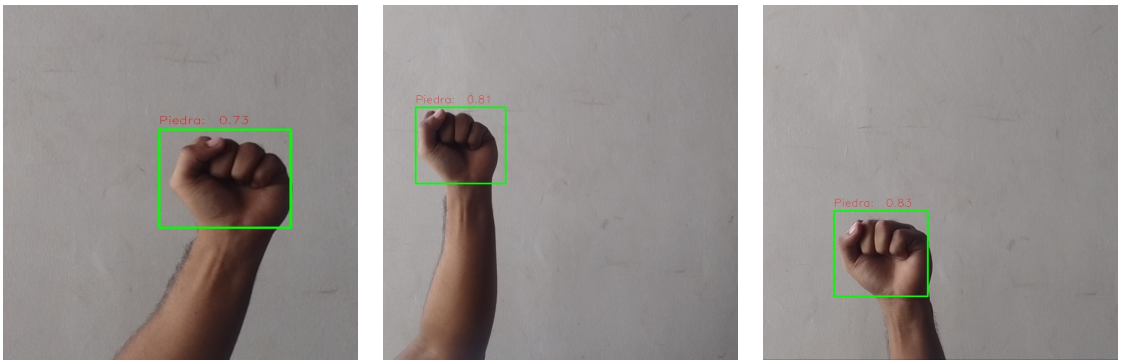


(a) Detección de clase incorrecta (b) Doble detección (c) Doble detección

Fuente: Elaboración propia

Algunos aciertos en la detección de la clase piedra son mostrados en la Figura 47

Figura 47. Algunos aciertos en la detección clase Piedra MobileNet5_0

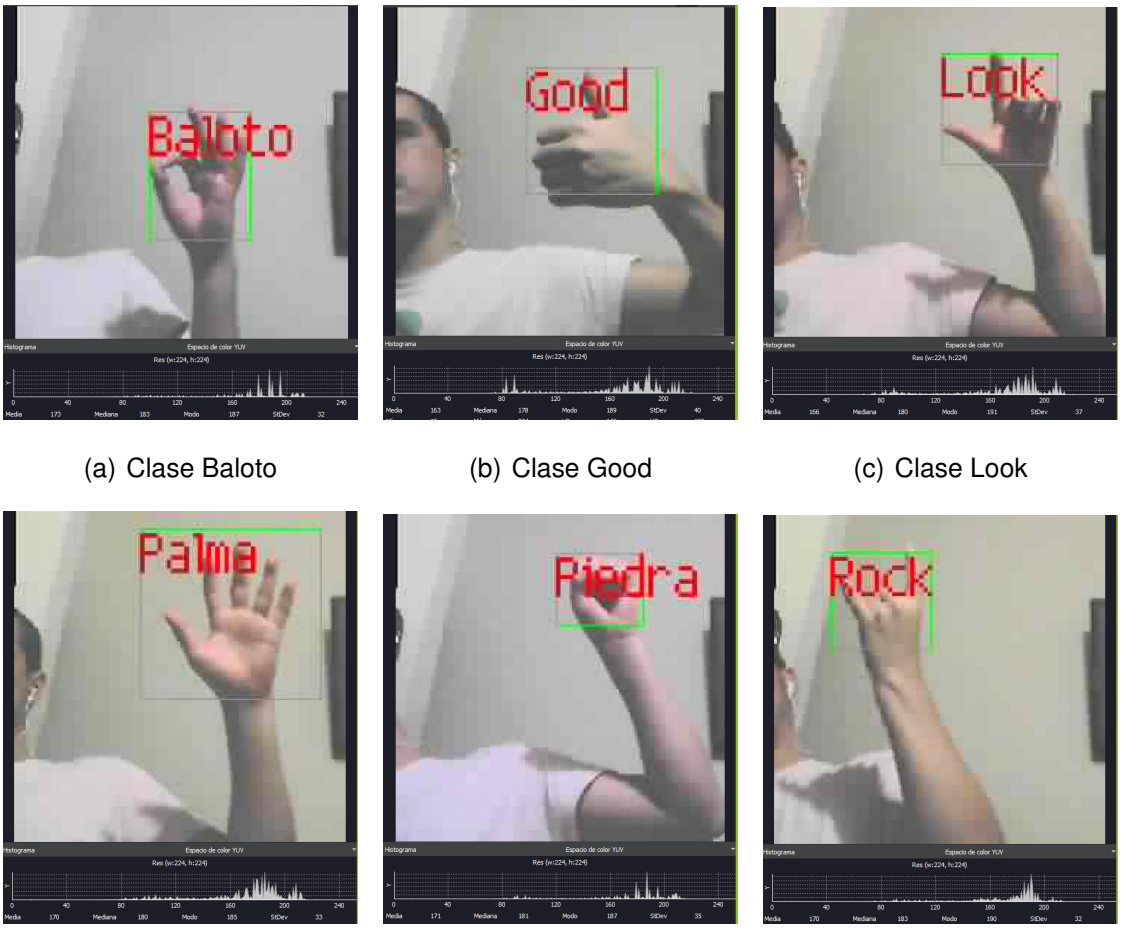


(a) Probabilidad 0.73 (b) Probabilidad 0.81 (c) Probabilidad 0.83

Fuente: Elaboración propia

4.1.2. Detección de gestos manuales en vivo Para esta red neuronal, en vivo se tuvieron muy buenos resultados. En el código de la **figura 32** en la linea 38 especifica que la detección de los gestos se hace con una precisión mayor a 0.8, teniendo en cuenta esto se tuvieron los resultados en vivo mostrados en la **figura 48** para la tarjeta Maixduino y **figura 49** para la tarjeta MaixGo. La detección con la tarjeta MaixGo puesto que esta tiene una mejor cámara *VGA* con un lente M12 se hace de una manera mas rápida, reconoce los gestos en seguida son mostrados a la cámara y presenta mas estabilidad al variar la luz del ambiente.

Figura 48. Detección de gestos con arquitectura MobileNet5_0 en vivo Maixduino

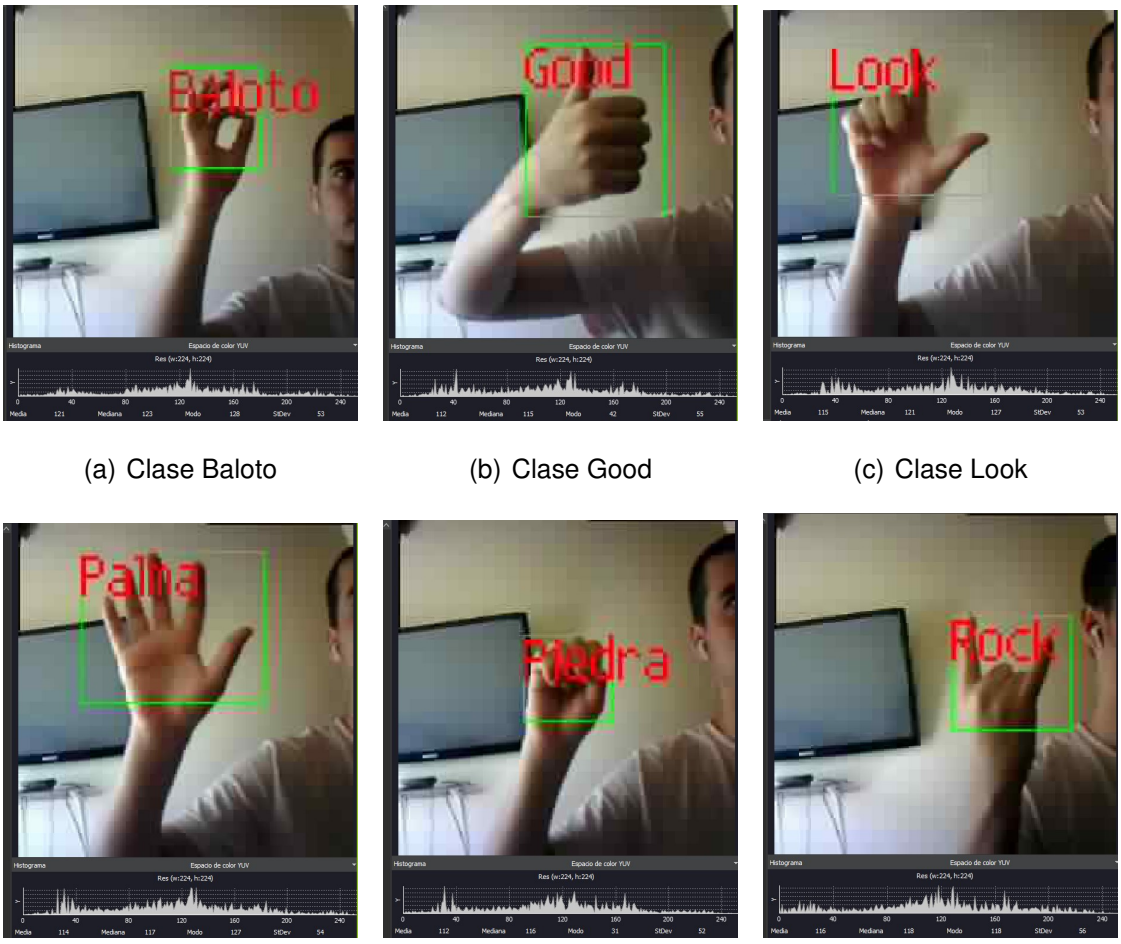


(a) Clase Baloto (b) Clase Good (c) Clase Look

(d) Clase Palma (e) Clase Piedra (f) Clase Rock

Fuente: Elaboración propia

Figura 49. Detección de gestos con arquitectura MobileNet5_0 en vivo MaixGo



(a) Clase Baloto (b) Clase Good (c) Clase Look

(d) Clase Palma (e) Clase Piedra (f) Clase Rock

Fuente: Elaboración propia

4.2. MOBILENET7_5

En el entrenamiento de la red neuronal hecha con la arquitectura MobileNet7_5 el *Epoch* en el que se obtuvo el máximo mAP posible fue el 32 como se ve en la **figura 37**, en este se obtuvo un mAP=1.0 puesto que todas las clases llegaron a su punto óptimo, por esta razón, esta arquitectura se escogió como la mejor de las tres

porque en pocos *Epochs* pudo llegar a su mejor desempeño.

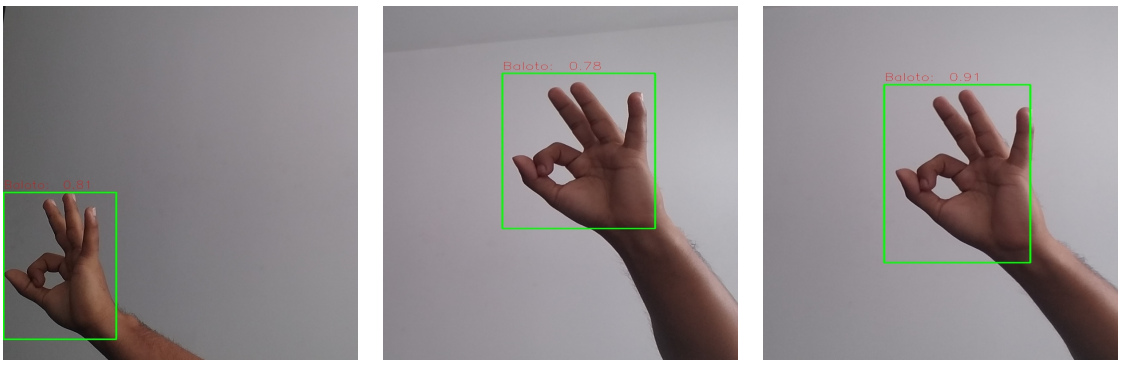
Figura 50. Mejor mAP en entrenamiento MobileNet7_5

```
Epoch 32/50
386/386 [==============================] - 70s 179ms/step - loss: 0.0749 - val_loss: 0.0698


Baloto 1.0000
Good 1.0000
Look 1.0000
Palma 1.0000
Rock 1.0000
Piedra 1.0000
mAP: 1.0000
mAP improved from 0.9974489795918368 to 1.0, saving model to /content/drive/MyDrive/Desarrollos,
Epoch 00031: Learning rate is 3.203769142399339e-05.
```

Fuente: Elaboración propia

4.2.1. Detección de imágenes de verificación Por haber obtenido un mAP de 1 en el entrenamiento, esta red neuronal no presentó ninguno error en la detección de gestos manuales en las 300 imágenes de verificación de las 6 clases. Algunas de estas imágenes son mostradas a continuación.

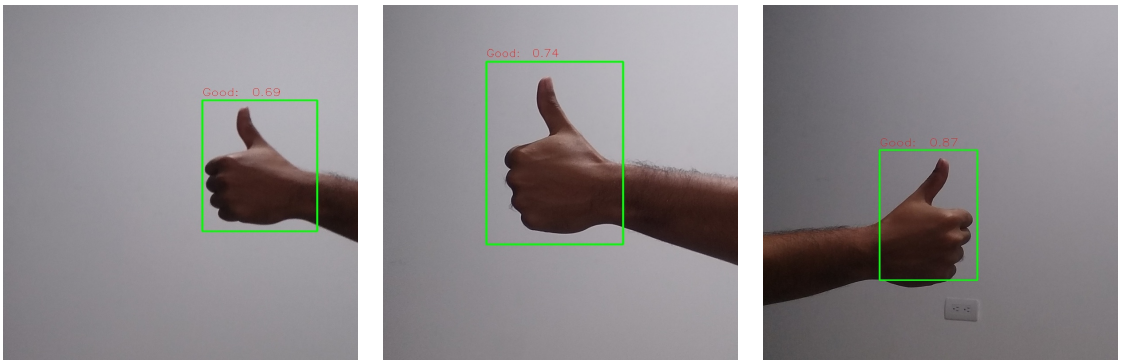
Figura 51. Algunos aciertos en la detección clase Baloto MobileNet7_5



(a) Probabilidad 0.81 (b) Probabilidad 0.78 (c) Probabilidad 0.91

Fuente: Elaboración propia

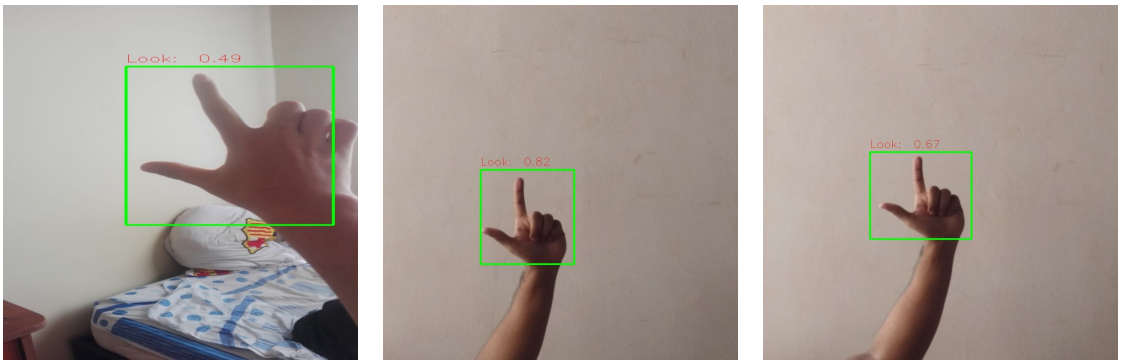
Figura 52. Algunos aciertos en la detección clase Good MobileNet7_5



(a) Probabilidad 0.69 (b) Probabilidad 0.74 (c) Probabilidad 0.87

Fuente: Elaboración propia

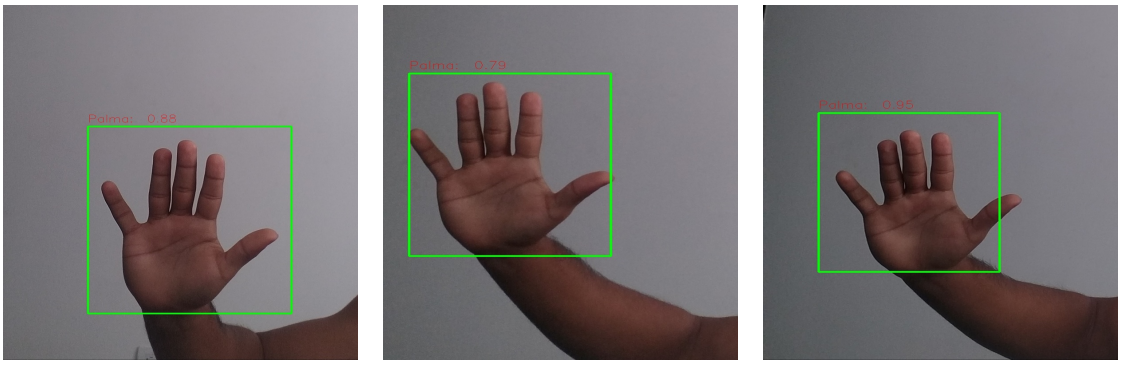
Figura 53. Algunos aciertos en la detección clase Look MobileNet7_5



(a) Probabilidad 0.49 (b) Probabilidad 0.82 (c) Probabilidad 0.67

Fuente: Elaboración propia

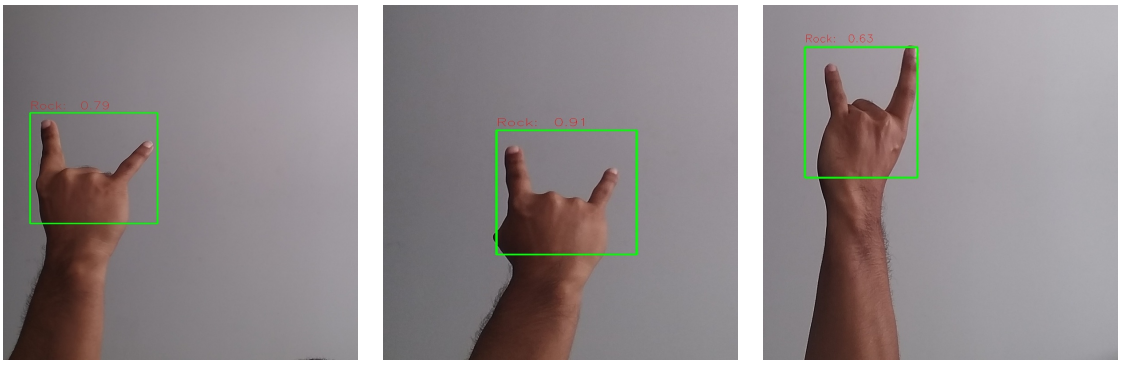
Figura 54. Algunos aciertos en la detección clase Palma MobileNet7_5



(a) Probabilidad 0.88 (b) Probabilidad 0.79 (c) Probabilidad 0.95

Fuente: Elaboración propia

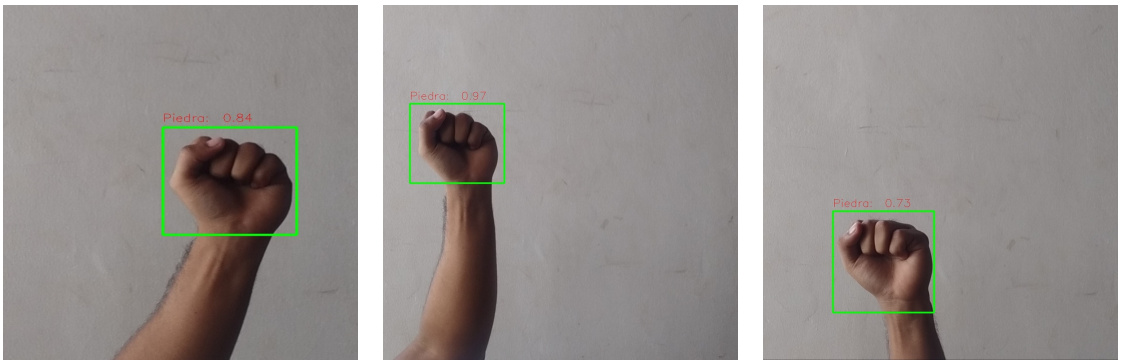
Figura 55. Algunos aciertos en la detección clase Rock MobileNet7_5



(a) Probabilidad 0.79 (b) Probabilidad 0.91 (c) Probabilidad 0.63

Fuente: Elaboración propia

Figura 56. Algunos aciertos en la detección clase Piedra MobileNet7_5



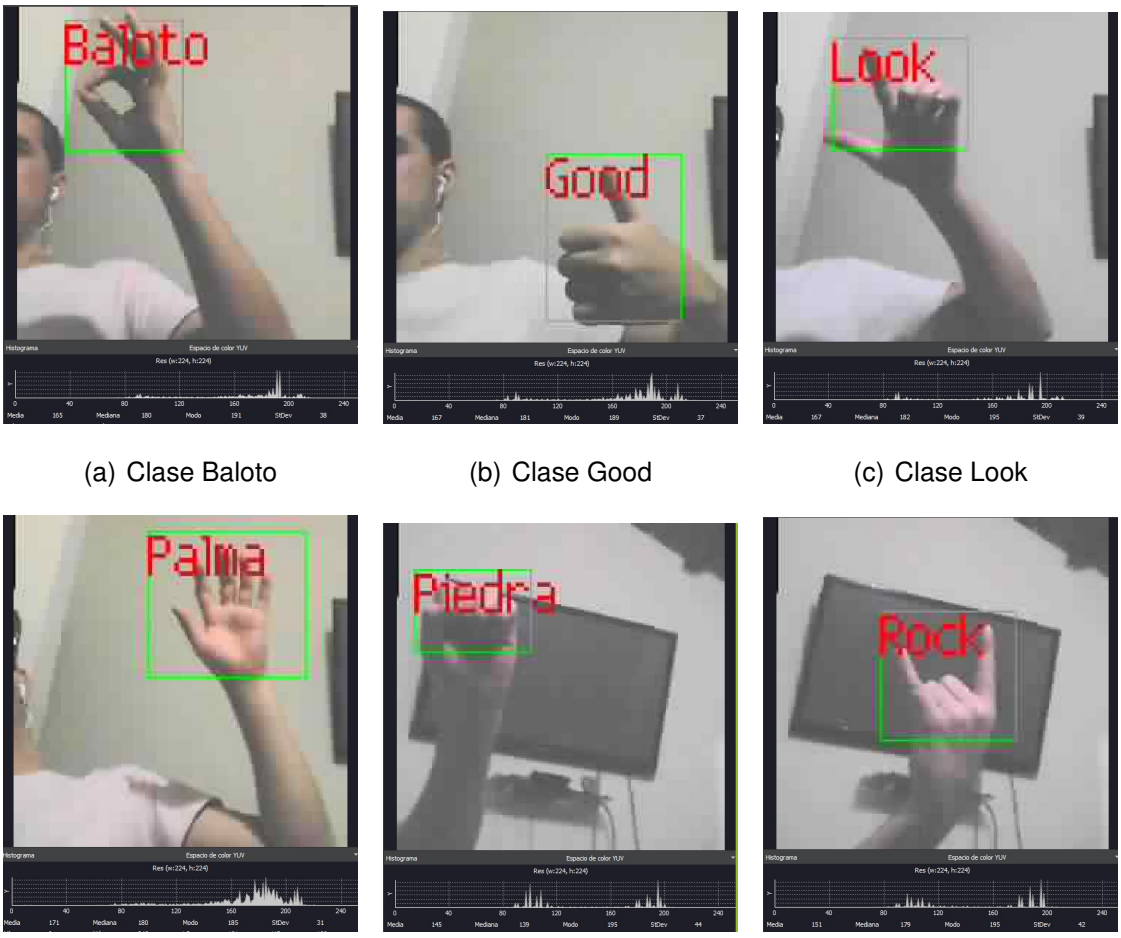
(a) Probabilidad 0.84 (b) Probabilidad 0.97 (c) Probabilidad 0.73

Fuente: Elaboración propia

4.2.2. Detección de gestos en vivo

1. La detección en vivo para la red neuronal creada a partir de la arquitectura MobileNet7_5, presenta gran estabilidad y el mejor desempeño de las 3 arquitecturas, puesto que reconoce los gestos manuales, sin cometer ninguno error tanto en la tarjeta de desarrollo Maixduino, como en la tarjeta MAixGo.

Figura 57. Detección de gestos con arquitectura MobileNet7_5 en vivo Maixduino



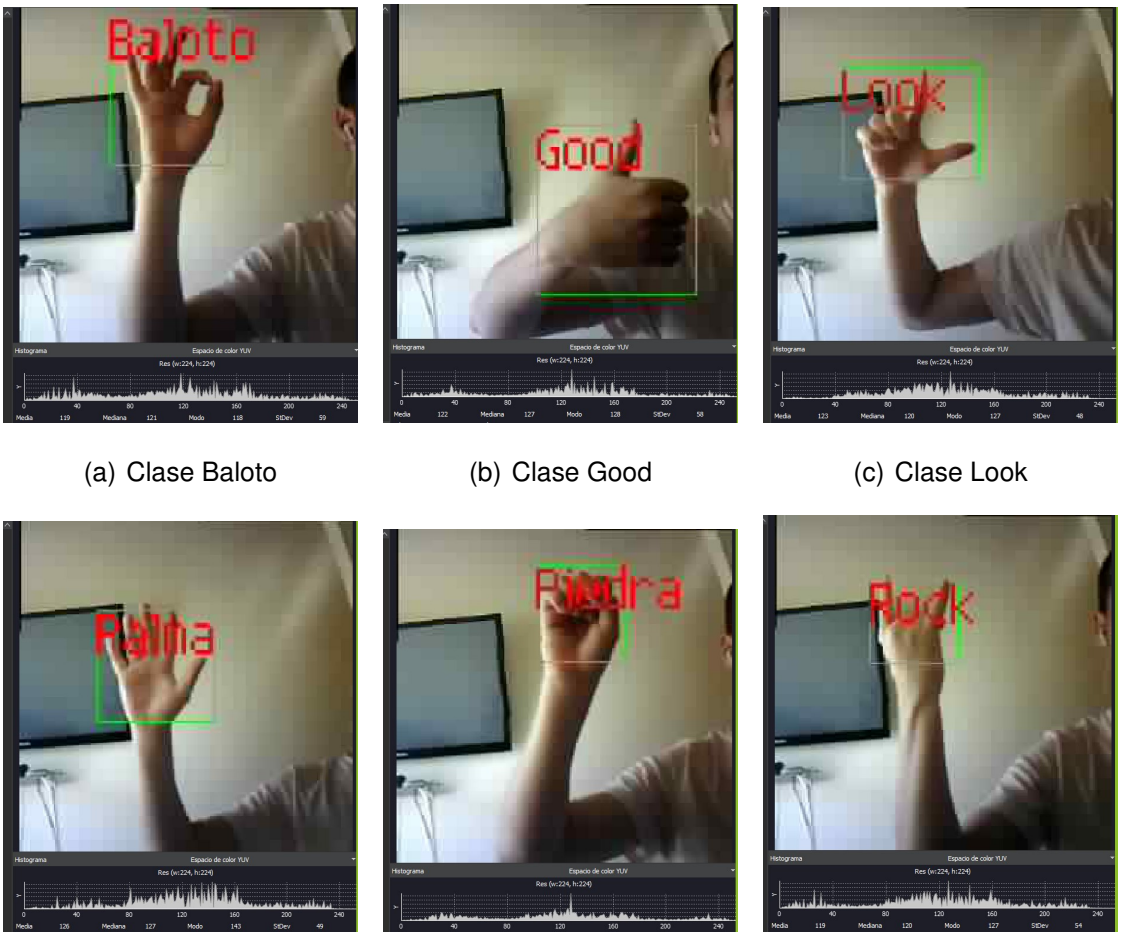
(a) Clase Baloto (b) Clase Good (c) Clase Look

(d) Clase Palma (e) Clase Piedra (f) Clase Rock

Fuente: Elaboración propia

2. Para la tarjeta **MaixGo** también se obtuvieron resultados muy buenos, al utilizar esta tarjeta con un mejor lente, la detección de los gestos se hace mas estable antes los cambios de luz, haciendo que esta combinación de la red neuronal entrenada con la arquitectura MobileNEt7_5 y MAixGo sea la mejor de todas. Algunos de estos resultados se pueden apreciar en la **figura 58**.

Figura 58. Detección de gestos con arquitectura MobileNet7_5 en vivo MaixGo



(a) Clase Baloto (b) Clase Good (c) Clase Look

(d) Clase Palma (e) Clase Piedra (f) Clase Rock

Fuente: Elaboración propia

4.3. MOBILENET2_5

En el entrenamiento de la red neuronal hecha con la arquitectura MobileNet2_5 el *Epoch* en el que se obtuvo el mayor mAP fue el 41 como se ve en la **figura 59**, en este se obtuvo un mAP=0.8616 donde el mejor desempeño fue para la clase “Baloto” con 0.9910. Esta arquitectura tuvo el mAP mas bajo, por esta razón de las

3 arquitecturas, esta tuvo el desempeño mas bajo en la detección con las imágenes de verificación y en vivo.

Figura 59. Mejor mAP en entrenamiento MobileNet2_5

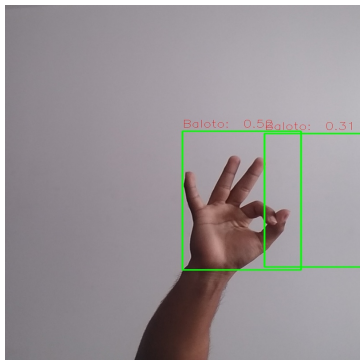
```
Epoch 41/50
386/386 [==============================] - 69s 177ms/step - loss: 0.2089 - val_loss: 0.0805


Baloto 0.9910
Good 0.8641
Look 0.9641
Palma 0.6537
Rock 0.8369
Piedra 0.8599
mAP: 0.8616
mAP improved from 0.8599802553297057 to 0.8616116366164056, saving model to /content/drive/I
Epoch 00040: Learning rate is 8.782822764196258e-06.
```

Fuente: Elaboración propia

4.3.1. Detección de imágenes de verificación En la detección con las imágenes de verificación, se obtuvo una considerable cantidad de errores, siendo la clase “Palma” la que mas tuvo.

En la clase “Baloto” se obtuvo 1 error como se muestra en la **figura 60**, donde hay una doble detección de una misma clase o tambien llamado, falso positivo.

Figura 60. Fallos en la detección clase Baloto MobileNet2_5

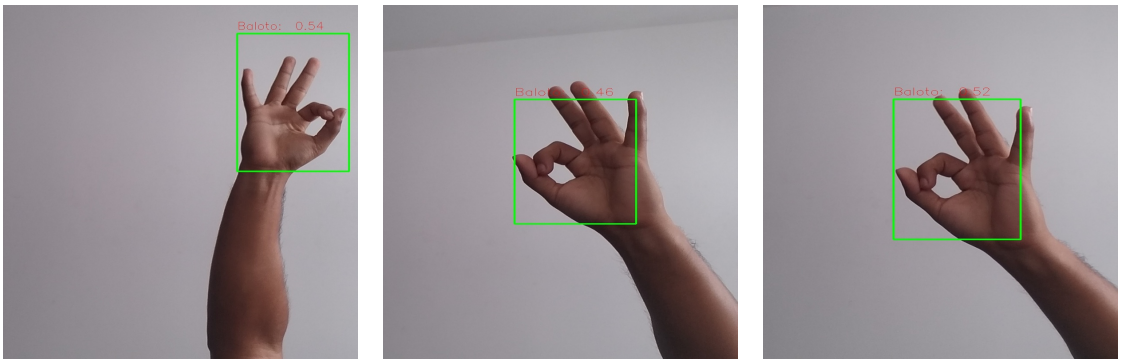


(a) Doble detección

Fuente: Elaboración propia

Algunos de los 49 aciertos en la detección de la clase “Baloto” son presentados en la **figura 61**.

Figura 61. Algunos aciertos en la detección clase Baloto MobileNet2_5



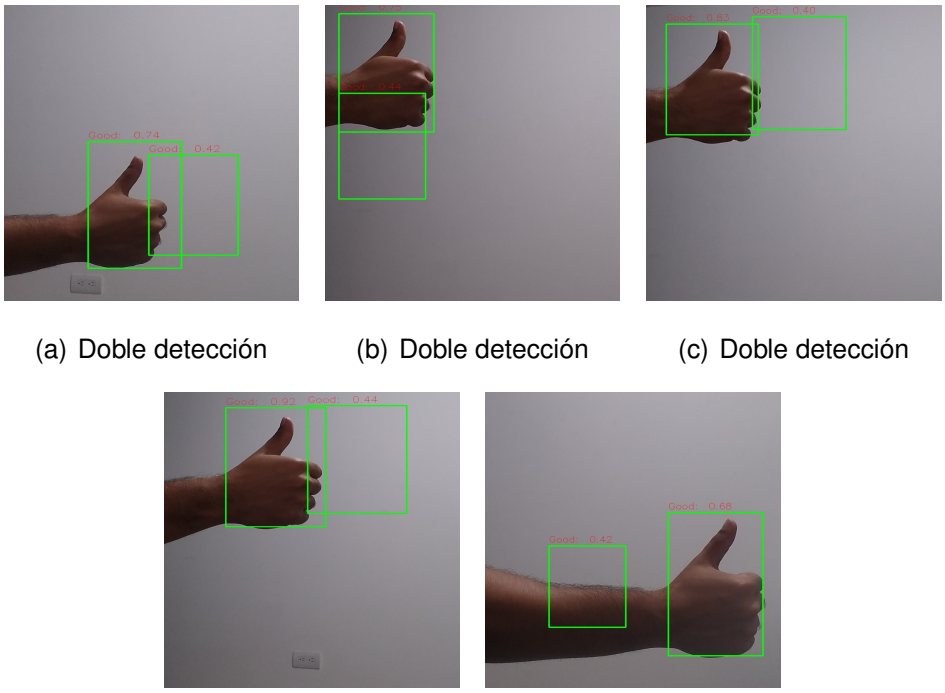
(a) Probabilidad 0.54 (b) Probabilidad 0.46 (c) Probabilidad 0.52

Fuente: Elaboración propia

La red neuronal tuvo varios fallos en la detección de la clase “Good”, estos errores son presentados en la **figura 62**, donde se muestran errores de doble detección de

la misma clase.

Figura 62. Fallos en la detección clase Good MobileNet2_5



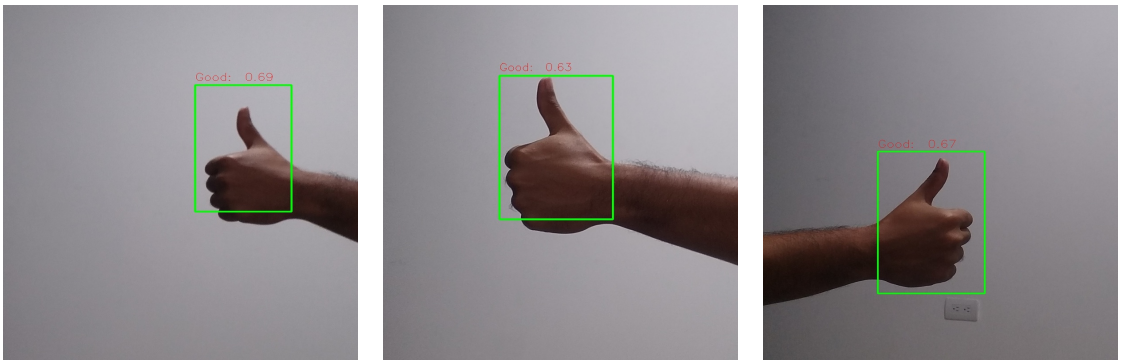
(a) Doble detección (b) Doble detección (c) Doble detección

(d) Doble detección (e) Detección incorrecta

Fuente: Elaboración propia

Algunos de los aciertos de la red neuronal en la detección de la clase “Good” son mostrados en la **figura 63**.

Figura 63. Algunos aciertos en la detección clase Good MobileNet2_5

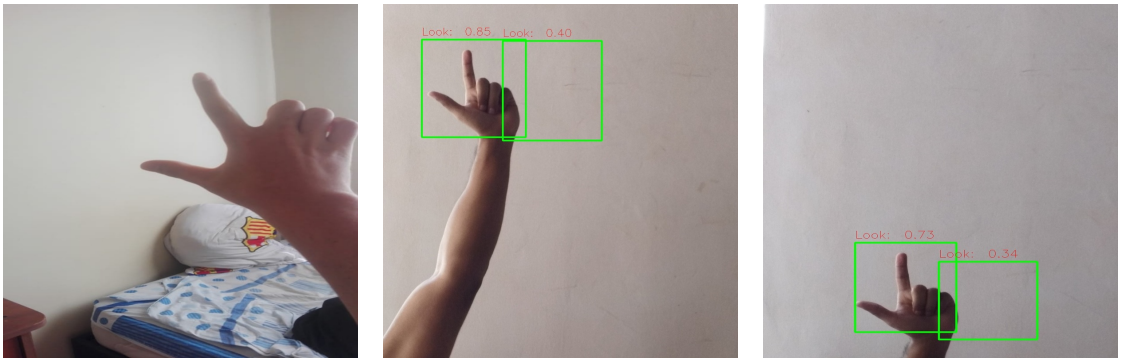


(a) Probabilidad 0.69 (b) Probabilidad 0.63 (c) Probabilidad 0.67

Fuente: Elaboración propia

En la clase “Look” la red neuronal tuvo 3 fallos en la detección de la clase, estos son mostrados en la **figura 64**, allí se ve que en la **figura 64(a)**, no hubo detección de ninguna clase, en la **figura 64(b)** y **figura 64(c)** hubo una doble detección de la misma clase.

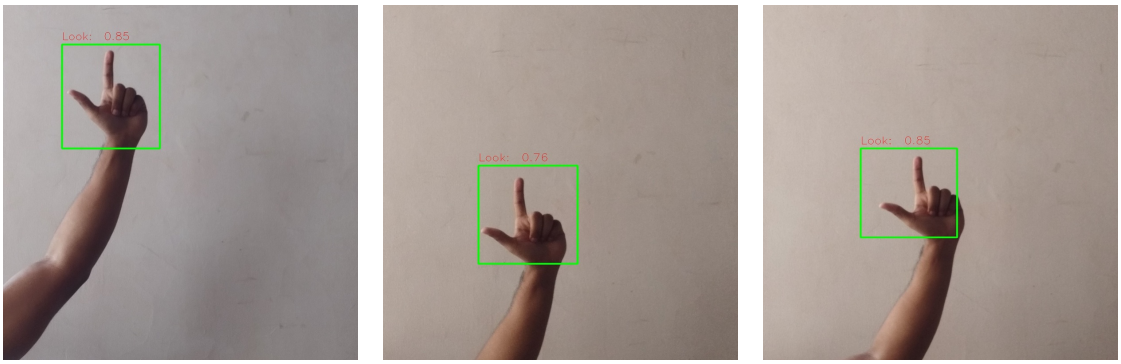
Figura 64. Fallos en la detección clase Look MobileNet2_5



(a) No hubo detección (b) Doble detección (c) Doble detección

Fuente: Elaboración propia

Figura 65. Algunos aciertos en la detección clase Look MobileNet2_5

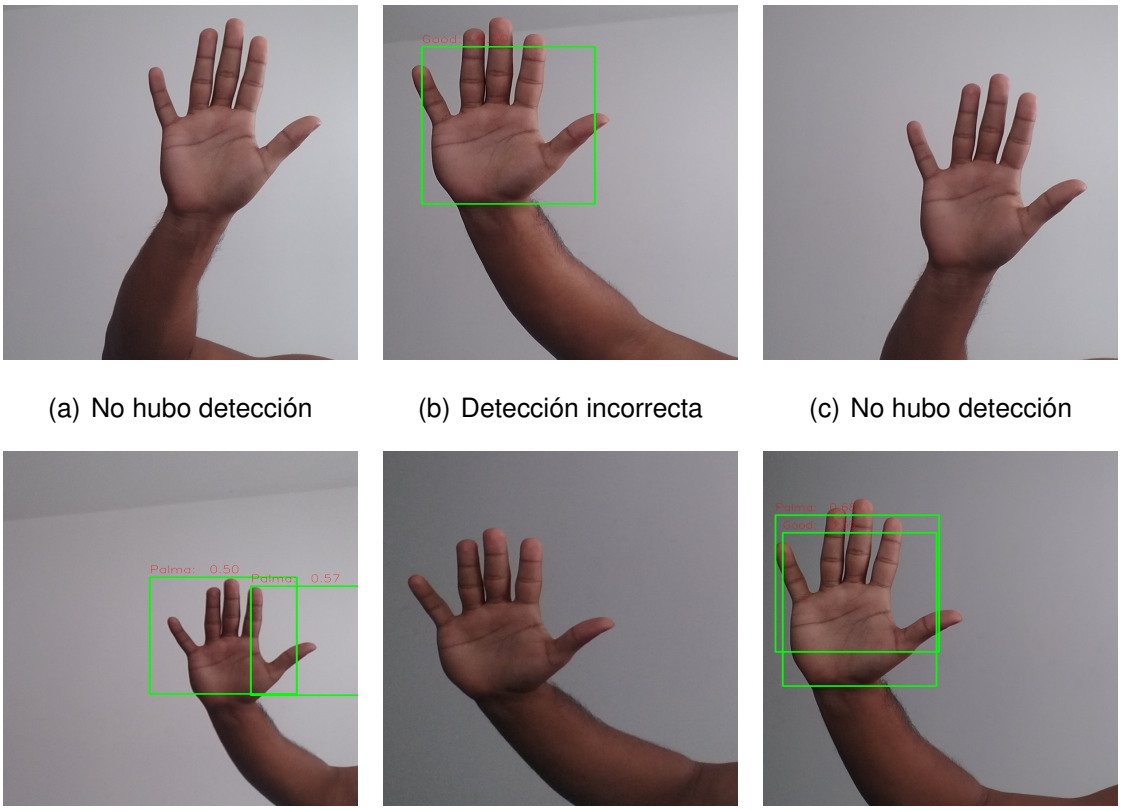


(a) Probabilidad 0.85 (b) Probabilidad 0.76 (c) Probabilidad 0.85

Fuente: Elaboración propia

La clase “Palma” con 18 fue la que presentó mas errores en la detección por parte de la red neuronal entrenada con la arquitectura MobileNet2_5, algunas de estas fallas son mostradas en la **figura 66**, donde hay errores de no detección, detecciones incorrectas y dobles detecciones de la misma clase.

Figura 66. Algunos Fallos en la detección clase Palma MobileNet2_5

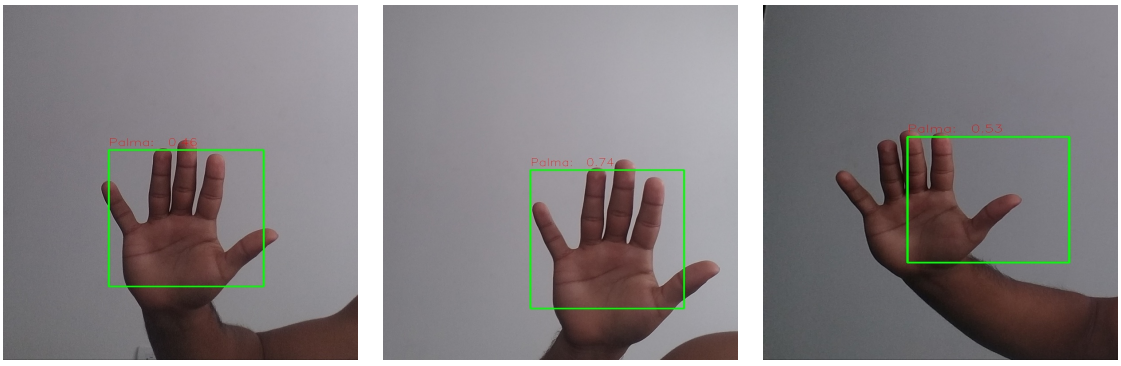


(a) No hubo detección (b) Detección incorrecta (c) No hubo detección

(d) Doble Detección (e) No hubo detección (f) Detección incorrecta

Fuente: Elaboración propia

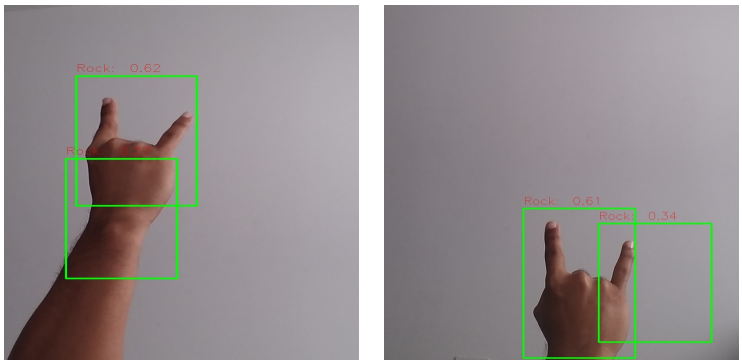
Figura 67. Algunos aciertos en la detección clase Palma MobileNet2_5



(a) Probabilidad 0.46 (b) Probabilidad 0.74 (c) Probabilidad 0.53

Fuente: Elaboración propia

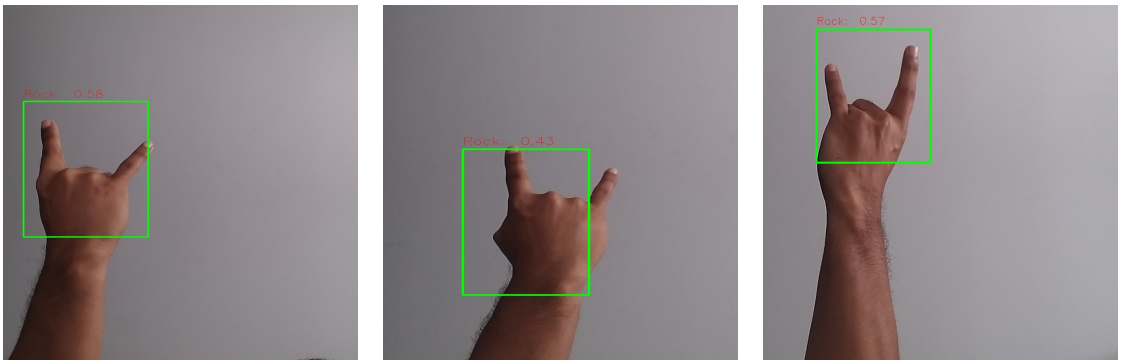
Figura 68. Fallos en la detección clase Rock MobileNet2_5



(a) Detección incorrecta (b) Detección incorrecta

Fuente: Elaboración propia

Figura 69. Algunos aciertos en la detección clase Rock MobileNet2_5

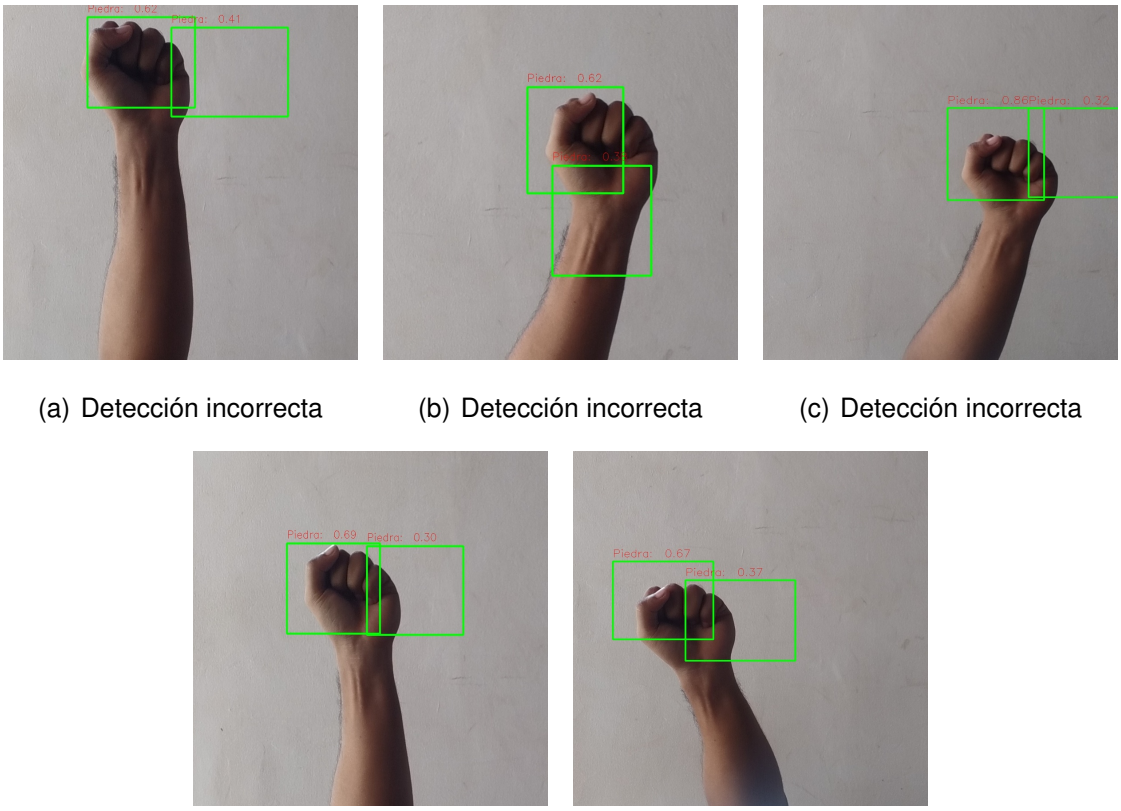


(a) Probabilidad 0.58 (b) Probabilidad 0.43 (c) Probabilidad 0.57

Fuente: Elaboración propia

La detección de la clase “Piedra” entregó 5 errores de doble detección del mismo gesto **figura 70**

Figura 70. Fallos en la detección clase Piedra MobileNet2_5



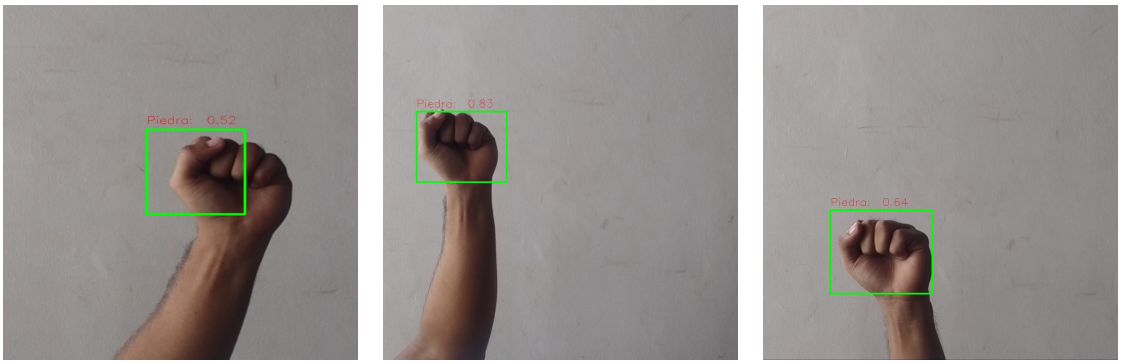
(a) Detección incorrecta (b) Detección incorrecta (c) Detección incorrecta

(d) Detección incorrecta (e) Doble detección

Fuente: Elaboración propia

Algunos de los aciertos en la detección de la clase “Piedra” son mostrados en la **figura 71**

Figura 71. Algunos aciertos en la detección clase Piedra MobileNet2_5



(a) Probabilidad 0.52 (b) Probabilidad 0.83 (c) Probabilidad 0.64

Fuente: Elaboración propia

4.3.2. Detección de gestos en vivo

1. El reconocimiento en vivo para la tarjeta **maixduino** es muy inestable, la detección de objetos falla constantemente y por tener programada la tarjeta para hacer reconocimientos con precisión mayor a 0.8 no siempre reconoce los gestos, por lo que esta red es la que menor rendimiento entrega en vivo.

Figura 72. Detección de gestos con arquitectura MobileNet2_5 en vivo Maixduino

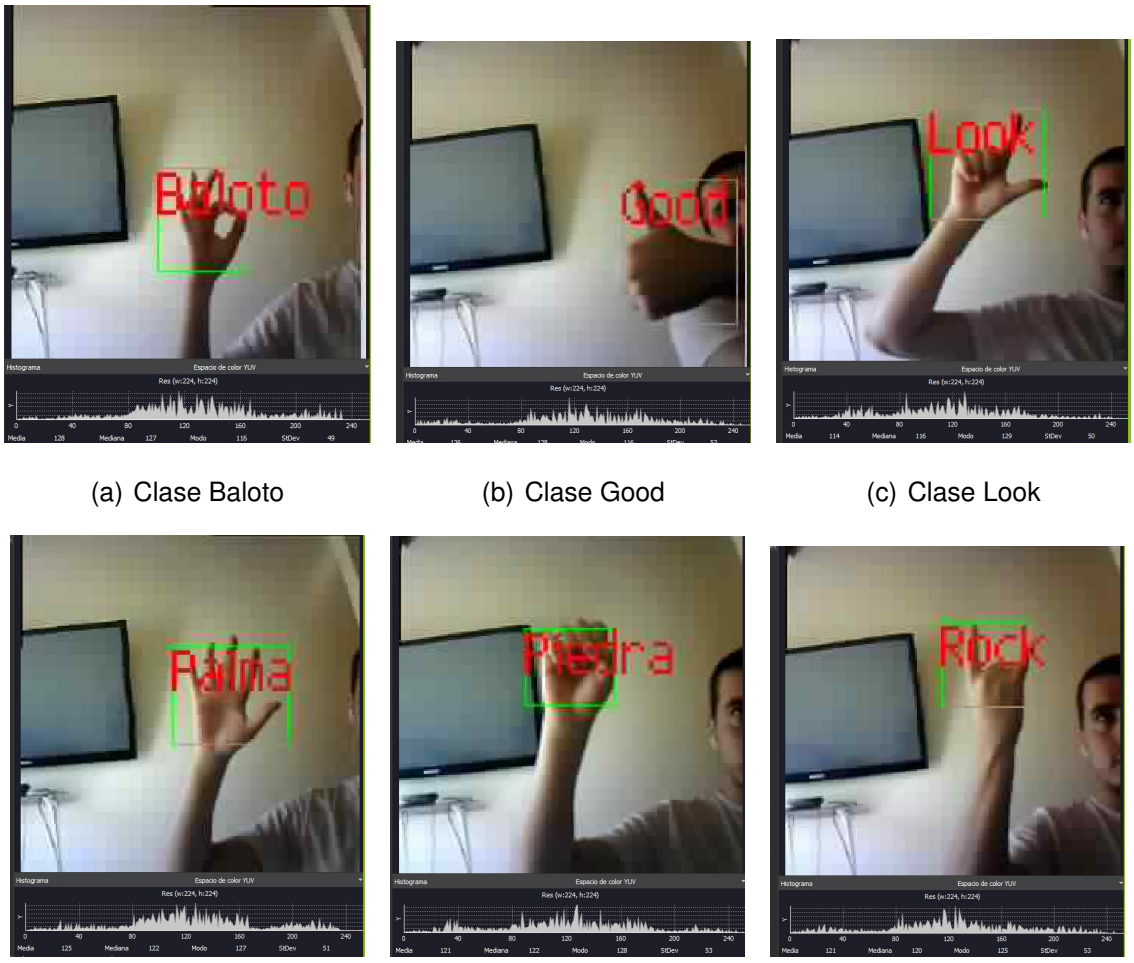
(a) Clase Baloto (b) Clase Good (c) Clase Look

(d) Clase Palma (e) Clase Piedra (f) Clase Rock

Fuente: Elaboración propia

2. Para la tarjeta **MaixGo** se obtuvieron mejores resultados, puesto que esta tarjeta cuenta con una mejor cámara con un lente M12 *VGA*, lo que hizo que la red neuronal tuviera una mejor detección en vivo (**figura 73**).

Figura 73. Detección de gestos con arquitectura MobileNet2_5 en vivo MaixGo



(a) Clase Baloto (b) Clase Good (c) Clase Look

(d) Clase Palma (e) Clase Piedra (f) Clase Rock

Fuente: Elaboración propia

La **tabla 1** presenta el resumen de los resultados de la detección de gestos manuales con las 3 redes neuronales convolucionales en las imágenes de verificación, donde A corresponde a los aciertos en la detección y F a los fallos en la misma, recordando que las imágenes de verificación son 50 por clase.

Tabla 1. Resumen de resultados en la detección de gestos manuales con las imágenes de verificación.

ARQUITECTURA		CLASE BALOTO	CLASE GOOD	CLASE LOOK	CLASE PALMA	CLASE ROCK	CLASE PIEDRA
MOBILNET 7.5	A	50	50	50	50	50	50
	F	0	0	0	0	0	0
MOBILNET 5.0	A	49	50	47	50	48	47
	F	1	0	3	0	2	3
MOBILNET 2.5	A	49	42	47	32	48	45
	F	1	8	3	18	2	5

Fuente: Elaboración propia

4.4. *DESARROLLO IOT*

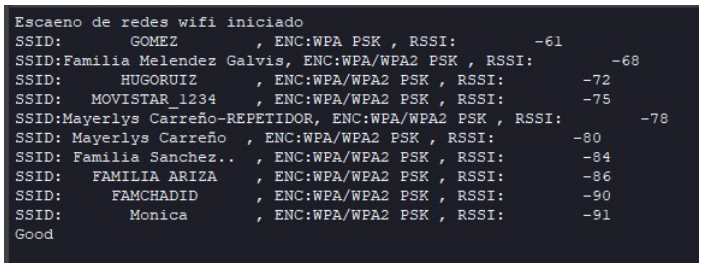
El sistema *IOT* se implementó para la tarjeta Maixduino, pues fue capaz de hacer *IOT* y procesar el modelo realizado con la arquitectura MobileNet5_0, mientras que la tarjeta MaixGo no. La tarjeta MaixGo fue capaz de realizar *IOT* y procesar el modelo de la arquitectura MobileNet2_5 pero este fue el que menor desempeño tuvo, entonces la tarjeta MaixGo fue descartada para esta implementación.

La Maixduino cuenta con un modulo esp32 incorporado para *IOT*. Al hacer la detección de gestos en vivo se pueden cargar los 3 modelos entrenados a la tarjeta SD y correr estos desde ahí, pero al momento de intentar cargar un código con mayor

requerimiento de procesamiento como el de *IOT*, la tarjeta solamente permite correr modelos de redes neuronales que se encuentren almacenados en su memoria flash. De los 3 modelos, las arquitecturas MobileNet2_5 y MobileNet5_0 pudieron correr desde la memoria flash debido a su tamaño de tan solo 253 KB y 874 KB respectivamente. La red MobileNet7_5 a pesar de ser la que mejores resultados entregó en la detección de gestos manuales no pudo ser utilizada desde la memoria flash debido a su mañaño de 1.878 KB.

A continuación se muestra el resultado al utilizar la red neuronal entrenada con la arquitectura MobileNet5_0 puesto que esta fue la que entrego el 2 mejor desempeño tanto en la detección con las imágenes de verificación como en vivo.
En la **figura 74** se tiene el escaneo de las redes wifi cercanas, alli se puede apreciar el nombre de la red, el tipo de encriptación y el nivel de potencia de las señales wifi cercanas. Esta tarea fue asignada como se puede ver en la parte final, a la detección de la clase “Good”.

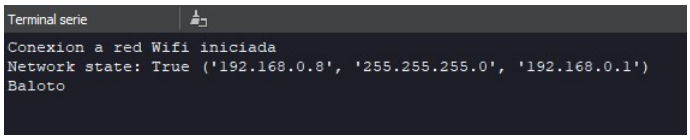
Figura 74. Escaneo de redes wifi



Fuente: Elaboración propia

En la **figura 75** está el resultado que se obtiene al hacer la conexión a la red wifi, allí se puede ver el estado en “True” que corresponde a una conexión efectiva, seguido de la dirección IP, la mascara de subred y finalmente el servidor *DNS*. Esto está asignado a la detección de la clase “Baloto”.

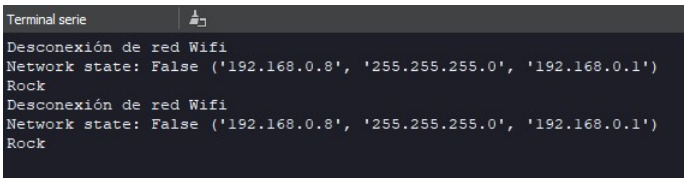
Figura 75. Conexión a una red wifi



Fuente: Elaboración propia

A la detección de la clase “Rock” se le asignó la desconexión de la red wifi (**figura 76**).

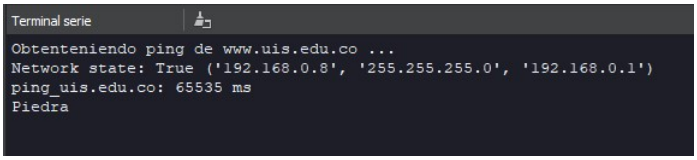
Figura 76. Desconexión a la red wifi



Fuente: Elaboración propia

A la clase “piedra” se le dio la tarea de obtener el *ping* de la pagina www.uis.edu.co como se puede ver en la **figura 77**.

Figura 77. Ping www.uis.edu.co



Fuente: Elaboración propia

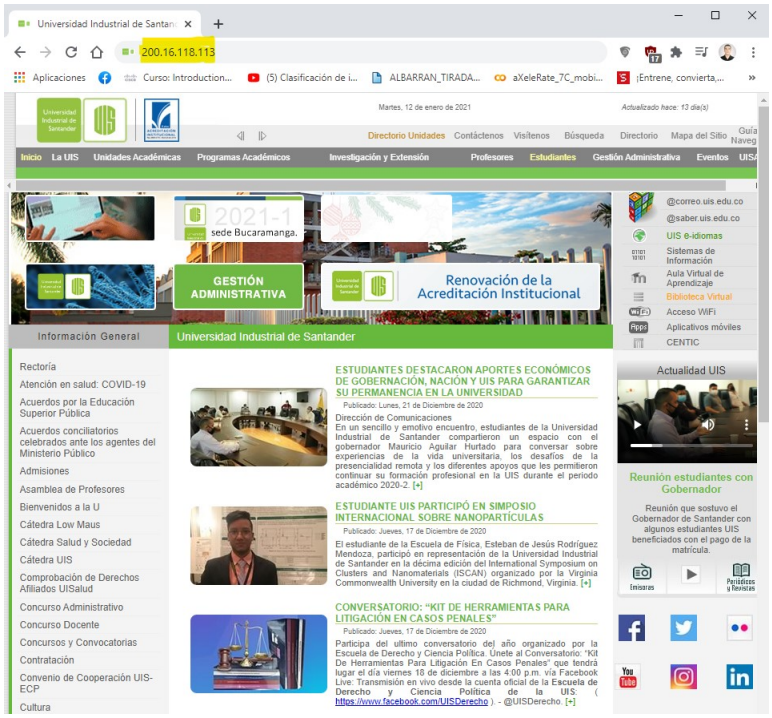
Por último, a la detección de la clase “Look” se relacionó con la obtencion de la dirección IP de una pagina en especifico en este caso de la pagina de la Universidad Industrial de Santander (**figura 78**).

Figura 78. Dirección IP de www.uis.edu.co

```
Obteniendo IP de sitio web www.uis.edu.co
Use Hareware SPI for other maixduino
[esp32_spi] use hard spi(1)
hard spi
esp32 set hard spi clk:9159090
ESP32_SPI firmware version: 1.4.0
try AT connect wifi...
network state: True ('192.168.0.8', '255.255.255.0', '192.168.0.1')
Address infos: [(2, 1, 0, '', ('200.16.118.113', 80))]
Connect address: ('200.16.118.113', 80)
b'HTTP/1.1 400 Bad Request\r\n'
Connect address: ('200.16.118.113', 80)
b''
Connect address: ('200.16.118.113', 80)
b'HTTP/1.1 400 Bad Request\r\n'
Connect address: ('200.16.118.113', 80)
b'HTTP/1.1 400 Bad Request\r\n'
Connect address: ('200.16.118.113', 80)
b'HTTP/1.1 400 Bad Request\r\n'
Look
```

Fuente: Elaboración propia

Figura 79. www.uis.edu.co



Fuente: Elaboración propia

5. CONCLUSIONES Y RECOMENDACIONES

Para la creación de la base de datos, las imágenes deben ser tomadas con diferentes fondos, variando la luz, el ángulo y la posición del gesto manual, para evitar que la red neuronal convolucional se adapte solo a ciertas características especificas y no reaccione positivamente al exponer el modelo a un ambiente con características diferentes.

Se debe crear un mínimo de 500 imágenes por clase para poder obtener redes neuronales con altas probabilidades de detección de objetos.

Usar el sistema de regularización de aumento de datos *(Data Augmentation)* al momento de entrenar la red, ayudó a que se obtuviera una mayor precisión en la detección de los gestos manuales en vivo, evitando el *overfitting*, además es recomendable para entrenamientos con bases de datos pequeñas.

Cuando se esté realizando el etiquetado de las clases, se debe seleccionar todo el gesto manual o la característica principal que diferencia una clase de otra.

aXeleRate simplifica la conversión de modelos a formato “kmodel” el cual es el que recibe el módulo K210.

La utilización de *Google Collab* para el entrenamiento de redes neuronales convolucionales por parte de aXeleRate, permite a quiénes quieran desarrollar proyectos de inteligencia artificial y no cuenten con una máquina potente, realizarlos de manera eficiente.

Para escoger el firmware correcto, depende de la aplicación que se quiera hacer y de la tarjeta de desarrollo en la que se desee implementar.

Para seleccionar el firmware con el que se trabajó las tarjetas, se escogió una versión reciente más no la última lanzada, puesto que esta versión suele estar en desarrollo y puede presentar problemas.

Las tarjetas de desarrollo Maixduino y MaixGo permiten correr redes neuronales convolucionales de hasta 1 MB desde su memoria flash y hasta 2 MB desde una tarjeta SD.

De las 3 redes neuronales convolucionales entrenadas, la que presentó un mayor mAP en 50 *epoch* fue la realizada con la arquitectura MobilNet7_5, que entregó un mAP igual a 1.

A pesar de que la red neuronal convolucional entrenada con la arquitectura MobilNet7_5 presentó mejores resultados tanto en la detección de los gestos manuales con las imágenes de verificación como en vivo, no fue posible utilizarla para el sistema ioT, debido a que este solo admite modelos cargados desde su memoria flash y este modelo tenía un peso mayor a 1 MB.

No fue posible enviar datos a través de internet debido al temprano desarrollo y la poca información que pudimos encontrar acerca de estas tarjetas.

El desarrollo de aplicaciones de IA a través de las tarjetas Maixduino y MaixGo trae ventajas desde el punto de vista económico respecto a otras tarjetas del mercado, ya que con una poca inversión se puede tener un sistema con el que se realicen

programas de inteligencia artificial.

Al ser las tarjetas Maix un sistema relativamente nuevo, se encuentra poca información de algunas de sus características como el wifi, es por eso que este trabajo ayuda a que se avance en el desarrollo de mayor implementación de estas tarjetas.

La tarjeta MaixGo al tener una mejor cámara (VGA) que la maixduino (QVGA), permite que la detección de gestos en vivo sea superior en cuánto a rapidez de reconocer gestos y acertar en su precisión.

BIBLIOGRAFÍA

A Mikołajczyk, M Grochowski. “Data augmentation for improving deep learning in image classification problem”. En: *2018 International Interdisciplinary PhD Workshop (IIPhDW)* (2018) (vid. pág. 37).

Alberto J. P. L., Adrián V. V. W. Fernando V. E. N. “Estado del arte de las arquitecturas del internet de las cosas (iot)”. En: (2014) (vid. pág. 48).

Bartneck C. Lütge C., Wagner A. Welsh S. “An Introduction to Ethics in Robotics and AI”. En: *What Is AI?* Springer. 2020, págs. 5-16 (vid. pág. 17).

Canziani, A. “Overfitting and regularization”. En: *https://atcold.github.io/pytorch-Deep-Learning/es/week14/14-3/* (2020) (vid. pág. 35).

D., Masolv. “aXeleRate - Keras-Based Framework for AI on the Edge”. En: *https://www.hackster.io/dmitrywat/axelerate-keras-based-framework-for-ai-on-the-edge-96f769* (2020) (vid. pág. 44).

“Deep learning for computer vision with Tensor Flow and Keras”. En: *https://www.udemy.com/course/deep-learning-for-computer-vision-with-tensor-flow-and-keras/* (2020) (vid. págs. 21, 27, 29-33, 35, 37, 38, 43).

Documentation, AWS. “Amazon Machine Learning, Guía para desarrolladores”. En: *https://docs.aws.amazon.com/es-es/machine-learning/latest/dg/training-parameters.html* (2020) (vid. págs. 35, 36).

H, Allami. “Methods to Avoid Over-Fitting and Under-Fitting In Supervised Machine Learning”. En: *Computer Science, Communication Instrumentation Devices* (2014) (vid. pág. 34).

H, Douglas. “The problem of Overfitting”. En: *ACS Publications* (2004) (vid. pág. 34).

“Hinge loss”. En: *https://handwiki.org/wiki/Hinge_loss* (2020) (vid. pág. 33).

“kflash_gui”. En: *https://github.com/sipeed/kflash_gui* (2020) (vid. pág. 65).

“LabelImg”. En: *https://github.com/tzutalin/labelImg* (2021) (vid. pág. 52).

“Las redes neuronales artificiales para la toma de decisiones”. En: *XIX Congreso Internacional de Contaduría, Administración e Informática* (2014) (vid. pág. 21).

“Maixpy IDE V0.2.5”. En: *https://dl.sipeed.com/shareURL/MAIX/MaixPy/ide /v0.2.5* (2020) (vid. pág. 66).

“MaixPy Scripts”. En: *https://github.com/sipeed/MaixPy_scripts* (2021) (vid. págs. 66, 71).

“MobileNetV2: The Next Generation of On-Device Computer Vision Networks”. En: *https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html* (2018) (vid. pág. 24).

Mohn, E. “Internet of Things.” En: *Salem Press Encyclopedia of Science.* (2020) (vid. pág. 49).

MOSQUERA Carlos, DUSSAN German. “Detección y Seguimiento de Múltiples Objetos en Tiempo Real para Vehículos Autónomos”. Tesis doct. Universidad Autónoma de Occidente, 2020 (vid. págs. 23-25, 42).

N Jacob, V Prabhu. “Comparative Review of YOLO MobileNet Versions: A Case Study”. En: *International Research Journal of Engineering and Technology* (2020) (vid. pág. 24).

N S Artamonov, P Y Yakimov. “Towards Real-Time Traffic Sign Recognition via YOLO on a Mobile GPU”. En: *The IV International Conference on Information Technology and Nanotechnology* (2018) (vid. pág. 26).

Nichols JA Herbert Chan HW, Baker MAB. “Machine learning: applications of artificial intelligence to imaging and diagnosis”. En: *Biophysical Reviews* (2019) (vid. pág. 17).

“PASCAL-VOC Detection model Training and Inference”. En: *https://colab.research.google.com/github/AIWintermuteAI/aXeleRate/blob/master/resources /aXeleRate_pascal20_detector.ipynb* (2021) (vid. págs. 58, 60, 61).

Patrikar, S. “Batch, Mini Batch Stochastic Gradient Descent”. En: *https://towardsdata-science.com-batch-mini-batch-stochastic-gradient-descent-7a62ecba642a* (2019) (vid. pág. 29).

“Qué es overfitting y underfitting y cómo solucionarlo”. En: *https://www.aprendemachi-nelearning.com-que-es-overfitting-y-underfitting-y-como-solucionarlo/* (2017) (vid. pág. 34).

“Redes Neuronales”. En: *https://bootcampai.medium.com/redes-neuronales-13349-dd1a5bb: :text=Funci%C3%B3n%20Linealtext=Por* (2019) (vid. págs. 38-40).

Sarkar D. Bali R., Sharma T. “Machine learning basics”. En: *Practical Machine Learning with Python*. 2018, pág. 28 (vid. págs. 17, 19, 21).

Sarkar D. Bali R., Sharma T. “Machine learning basics”. En: *Practical Machine Learning with Python*. 2018, pág. 35 (vid. pág. 18).

“Sipeed firmware”. En: *https://cn.dl.sipeed.com/shareURL/MAIX/MaixPy/release /master/*
maixpy_v0.6.2_8_g786fac709 (2021) (vid. pág. 65).

“Sipeed MAIX GO Suit (MAIX GO + 2.8 inch LCD + ov2640 with M12 lens)”. En: *https://www.seeedstudio.com/Sipeed-MAix-GO-Suit-for-RISC-V-AI-IoT-p-2874.html* (2020) (vid. pág. 45).

“Sipeed Maixduino Kit for RISC-V AI + IoT)”. En: *https://www.seeedstudio.com/Sipeed-Maixduino-Kit-for-RISC-V-AI-IoT-p-4047.html* (2020) (vid. pág. 48).

“What is AI”. En: *Oracle.com* (2018) (vid. pág. 16).

“YOLO: Real-Time Object Detection”. En: *https://pjreddie.com/darknet/yolov2/* (2020) (vid. pág. 25).