Construcción y Entrenamiento de una Red Neuronal para la Identificación de Gestos

Daniel Mauricio Avila Rey

Trabajo de Grado para Optar el Título de Ingeniero Electrónico

Director

Jaime Guillermo Barrero Pérez

Magister en potencia eléctrica

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica Electrónica y de Telecomunicaciones

Bucaramanga

2021

Dedicatoria

Dedico este proyecto a todas las personas que me acompañaron a lo largo de mi carrera, que me motivaron en los momentos difíciles y me acompañaron en la celebración de los pequeños y grandes logros que se fueron alcanzando paso a paso. De todas ellas pude aprender y crecer tanto personal como profesionalmente y me han ayudado a convertirme en la persona que soy hoy en día.

Agradecimientos

En primer lugar, quiero agradecer a mis padres y a mi hermano, ya que sin su apoyo incondicional en todos los aspectos no me habría sido posible llegar hasta este punto.

A la Universidad Industrial de Santander, porque en sus instalaciones pude vivir toda mi carrera, en ella conocí a personas invaluables y aprendí lecciones que serán importantes para toda mi vida.

A Jesús, Laura y Nata, mis grandes amigos, quienes estuvieron a mi lado siempre dándome ánimo, motivándome a seguir adelante y apoyándome en los buenos y en los malos momentos.

A mis profesores, quienes me orientaron y pusieron dedicación y su conocimiento durante toda mi formación profesional. En especial a mi director de proyecto, sin el cual no habría sido posible completar esta última etapa.

Finalmente quiero agradecer también a todas esas personas que pasaron por mi vida en estos años de estudio. Porque a pesar de haber compartido con ellos en mayor o menor medida, de todos aprendí algo y aun sin saberlo me ayudaron a ser una mejor versión de mí mismo.

Tabla de contenido

Pág.

Lista de tablas

Lista de figuras

Resumen

Título: Construcción y Entrenamiento de una Red Neuronal para la Identificación de Gestos[*]

Autor: Daniel Mauricio Avila Rey[**]

Palabras Clave: Redes Neuronales Convolucionales, Lenguaje de Señas, Computación On-Edge.

Descripción:

A lo largo de la historia la comunidad sorda ha sido discriminada y aislada de la sociedad, su dificultad para establecer una comunicación con las personas oyentes de su entorno las ha convertido en un grupo vulnerable. Y si bien es cierto que en la actualidad existen métodos de enseñanza e inclusión que han ayudado en gran medida a los sordos, aun hoy el poder comunicarse adecuadamente con otras personas sigue siendo un reto. Es por esta razón que se propuso este proyecto, con el propósito de crear un sistema basado en redes neuronales convolucionales, que se ejecute sobre un sistema embebido y con la capacidad de identificar en tiempo real, gestos propios de la lengua de señas colombiana (LSC). Durante el desarrollo de este proyecto se construyó una base de datos de la LSC con aproximadamente 5300 imágenes divididas en 22 categorías que corresponden con los gestos inmóviles del alfabeto, con esta base de datos se entrenó un modelo basado en las arquitecturas MobileNetV1 y YOLO y se implementó en la tarjeta Sipeed Maix Bit. Obteniendo como resultado un sistema pequeño y económico capaz de captar una imagen, identificar un gesto y asignarle una categoría en el rango de los milisegundos con una precisión superior aceptable.

---

[*] Trabajo de Grado

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

Abstract

Title: Construction and Training of a Neural Network for Gesture Recognition[*]

Author: Daniel Mauricio Avila Rey[**]

Key Words: Convolutional Neural Networks, Edge Computing, Sign Language.

Description:

Since ancient times, deaf people have been mistreated and isolated from their communities due to their struggle to communicate with other people, hence becoming a vulnerable group. Even though nowadays there are many institutes and teaching methods developed specifically for their education and aimed to improve their life conditions, they keep being a vulnerable community who works hard to find equal opportunities. With that in mind was proposed this project, which has as its main goal the implementation of a system capable of performing real-time Colombian sign language (LSC) gesture detection on an embedded system. In order to achieve this goal, this project required the construction of a LSC dataset composed of about 5300 different pictures catalogued into 22 categories corresponding to each immobile alphabet letter hand sign. Using this dataset, a model based on the YOLO and MobileNetV1 architectures was trained and then implemented on a Sipeed Maix Bit board. The result of this implementation is a small and cost-effective system capable of identifying and categorizing a hand gesture within the range of milliseconds keeping an acceptable level of accuracy.

---

[*] Bachelor Thesis

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

Introducción

Históricamente, las personas sordas han sido relegadas socialmente por su condición; su dificultad para aprender una lengua y comunicarse con otros miembros de su entorno hizo que durante siglos le les haya catalogado injustamente como intelectualmente inferiores, negándoseles oportunidades académicas y laborales. No es hasta la historia relativamente reciente que en Europa se empieza a prestar atención a estas personas; si bien inicialmente enfocados en la educación de los hijos sordos de algunos nobles, con el tiempo se fueron abriendo diversas instituciones donde poco a poco se perfeccionaron los métodos de enseñanza y se desarrolló una nueva forma de comunicación basada en gestos articulados con las manos de los hablantes.

A pesar de que en la actualidad existen instituciones especializadas en la educación de la comunidad sorda y que la gran mayoría de los países cuenta con lenguas de señas oficiales, lo cierto es que quienes sufren de algún nivel de discapacidad auditiva aún se enfrentan a muchos retos en sus vidas diarias. Uno de estos desafortunadamente sigue siendo la comunicación con los miembros oyentes de su entorno, principalmente debido a que no se enseña el lenguaje de señas a la gran mayoría de la población.

El desarrollo de herramientas destinadas al apoyo de las personas sordas ha sido estudiado con anterioridad en el mundo, y se ha conseguido construir distintos prototipos, principalmente basados en el reconocimiento de imágenes o en el uso de sensores ubicados en las manos del hablante, que miden la posición de estas y captan los movimientos musculares para diferenciar cada gesto. En Colombia esta es un área de investigación aún en crecimiento, pero a través de la cual se puede hacer un valioso aporte social.

Partiendo de esta problemática, y dado el auge actual de las tecnologías relacionadas con la inteligencia artificial, el reconocimiento de imágenes y la computación *on-edge*, se propone este proyecto con el objetivo de implementar un sistema basado en redes neuronales convolucionales, capaz de reconocer visualmente distintos gestos pertenecientes a la lengua de señas colombiana (LSC) desde un sistema embebido.

Para lograr esta meta se construyó de una base de datos de imágenes de la LSC, con la cual se realizó el entrenamiento de un modelo de clasificación de imágenes, basado en una arquitectura de red neuronal convolucional optimizada para su operación en sistemas con recursos limitados. La selección del modelo se dio tras evaluar 3 alternativas distintas, MobileNetV1, MobileNetV2 y ResNet50; siendo finalmente implementada MobileNetV1 dada su compatibilidad con el sistema embebido. Para hacer reconocimiento en tiempo real se usó la arquitectura YOLO como detector de objetos, usando el modelo seleccionado previamente como extractor de características. La implementación del clasificador se hizo sobre una Sipeed Maix Bit, tras comparar distintas alternativas del mercado buscando un dispositivo de bajo costo con la capacidad de procesar y ejecutar algoritmos de inteligencia artificial sin la necesidad de recurrir a la nube, se concluyó que esta era la mejor alternativa entre las evaluadas. Finalmente se probaron distintos métodos de entrenamiento y conversión del modelo, de los cuales se obtuvo el mejor resultado usando la plataforma MaixHub.

1. Objetivos

1.1 Objetivo general

Programar y entrenar una red neuronal convolucional para la identificación de gestos propios del lenguaje de señas.

1.2 Objetivos específicos

Construir una base de datos de imágenes y clasificarlas en distintas categorías, cada una de estas correspondiendo a un gesto distinto.

Implementar una red neuronal capaz de identificar y categorizar distintos gestos del lenguaje de señas colombiano.

Medir la precisión con la que la red neuronal construida es capaz de categorizar un conjunto de imágenes pertenecientes a las categorías bajo las que fue entrenada.

2. Marco teórico

2.1 Lenguaje de señas

A lo largo de la historia del ser humano su capacidad para usar distintos métodos de comunicación ha demostrado ser una poderosa herramienta, permitiendo incontables avances sociales y tecnológicos en todas las sociedades que han existido. Si bien la comunicación hablada y escrita son las más notables, existe una gran variedad de formas de comunicación distintas que se han desarrollado y perfeccionado con el tiempo como medios complementarios y/o alternativos ante la necesidad de compartir y transmitir información entre personas, a pesar de las limitantes que puedan existir. Tal es el caso de los múltiples lenguajes de señas existentes en el mundo y el enorme impacto que han tenido en la calidad de vida de las comunidades sordas desde que tienen acceso a estos.

A raíz de su condición, durante siglos se catalogó a las personas sordas como sujetos intelectualmente discapacitados debido a su dificultad para comunicarse y consecuentemente para aprender, dada la inexistencia de métodos que les permitieran adquirir el conocimiento que la mayoría de las personas podía obtener a través del habla. De acuerdo con (Cruz, 2008), no fue hasta el siglo XVI, cuando en Europa se empezaron a desarrollar métodos de enseñanza más amigables con las personas no oyentes, que se descubrió que en realidad no poseen ningún tipo de problema de aprendizaje. Y si bien inicialmente este tipo de educación especial era impartida unicamente en los monasterios católicos y dirigida a los hijos de nobles que hubieran nacido con una discapacidad auditiva, estos métodos con el tiempo se fueron acercando a la comunidad en general a través de instituciones especializadas en la educación de gente sorda tal como la escuela

del abad Charles Michel de l'Épée en Francia, allí se empezó a documentar mas detalladamente los distintos signos y gestos mientras que al mismo tiempo se perfeccionaba el lenguaje procurando darle una estructura gramatical similar a la del francés, estos avances llevarían a la que hoy se conoce como lengua de señas francesa de la cual toman base otras lenguas de señas como la americana (ASL), mexicana (LSM) o la colombiana (LSC).

Si bien en la actualidad la mayoría de los países en el mundo cuenta con políticas de inclusión y educación para personas en situación de discapacidad, además de lenguajes de signos oficiales establecidos, la comunicación entre personas sordas y oyentes en muchos casos sigue siendo complicada, principalmente debido al desconocimiento de la lengua de señas por parte del público general quienes no reciben educación de ningún tipo en este aspecto, convirtiendo la comunicación con personas no oyentes en una cuestión de creatividad donde ambas partes deberán idear formas alternativas de dar a entender su mensaje al otro. Es en esta brecha educativa donde puede hacerse uso de la tecnología como un apoyo para las personas sordas, desarrollando dispositivos con la capacidad de reconocer determinados gestos y traducir sus significados de forma que el mensaje pueda transmitirse y comprenderse más fácilmente por personas que desconozcan tal lenguaje.

2.1.1 Lenguaje de señas en Colombia

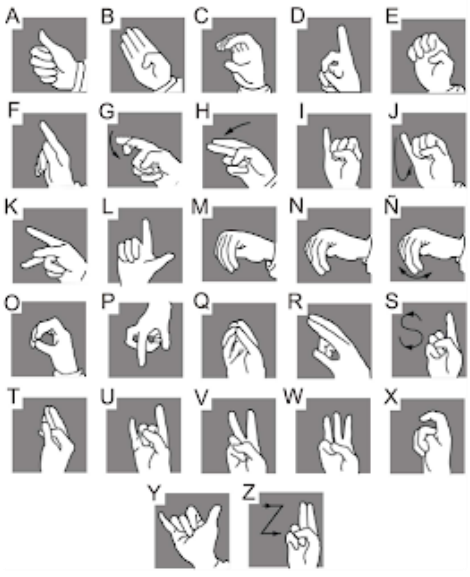
La lengua de señas colombiana (LSC) es el lenguaje oficial de la comunidad no oyente del país, reconocido así desde el año 1996 con la Ley 324. Sin embargo, a partir de los escasos registros escritos existentes, se conoce que su historia se remonta al año 1924 cuando fue fundada en Bogotá la escuela de sordos Nuestra Señora de la Sabiduría, donde empezó a tomar forma la que posteriormente se conocería como LSC.

De acuerdo con (Oviedo, 2013), el desarrollo de la LSC se vio influido principalmente por tres fuentes distintas, en sus inicios tomando elementos de las lenguas de señas francesa gracias a las monjas de La Sabiduría, posteriormente durante los años 50 se empezaron a introducir elementos de la lengua de señas española gracias a inmigrantes educados en España y al mismo tiempo, un amplio número de nacionales que recibieron educación en el país europeo durante esa década. Finalmente, la tercera y última gran influencia que ayudó a conformar la LSC fue lengua de señas estadounidense debido a la presencia de misioneros protestantes en Colombia y oyentes especialistas formados en los Estados Unidos durante los años 70.

En el año 2006 el Instituto Nacional de Sordos (INSOR) en colaboración con el Ministerio de Educación Nacional publicaron el diccionario básico de la lengua de señas colombiana, en donde se condensan alrededor de 1200 signos distintos de conceptos cotidianos. Constituyendo uno de los primeros pasos en la investigación de esta área, que para dicha época había iniciado hacia tan solo 10 años (Instituto Nacional de Sordos (INSOR), 2006).

Figura 1.

Alfabeto del Lenguaje de Señas Colombiano (LSC).



Nota: Adaptado de *1er. Seminario – Taller en Lengua de Señas Colombiana, 2015*, Fono al Día (http://fonoaldia.unisucre.edu.co/2015/11/1er-seminario-taller-en-lengua-de-senas.html). **(Universidad de Sucre, 2015)**.

Dada la complejidad de la mayoría de estos signos, que en muchos casos no solo requieren de representación manual de los gestos sino de movimientos e interacciones con otras partes del cuerpo del hablante, para este proyecto se decidió trabajar en la identificación de los caracteres estáticos del abecedario de señas colombiano. Al igual que en el español, el alfabeto de la LSC se encuentra compuesto por 27 letras distintas ilustradas en la Figura 1, cada una de estas es descrita por un gesto manual de los cuales 6 son gestos móviles y 21 son estáticos, por lo que son estos ultimos con los cuales se construirá la base de datos y posteriormente se entrenará el modelo seleccionado.

2.2 Clasificación de imágenes

La clasificación de imágenes es una disciplina de la ciencia de datos intrínsecamente relacionada con el aprendizaje profundo, que parte de los conceptos aplicados en el procesamiento digital de imágenes para la extracción de características y los aplica con los métodos de inteligencia artificial. Durante esta sección se dará una breve vista sobre algunos de los conceptos básicos relacionados con el procesamiento digital de imágenes y los fundamentos en los que se basa el funcionamiento de las redes neuronales convolucionales.

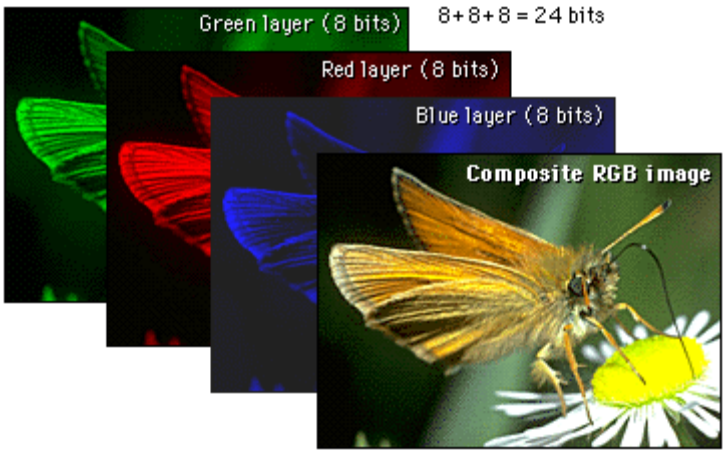
2.2.1 Procesamiento digital de imágenes

Una imagen digital (en representación RGB) es en esencia la combinación de 3 matrices bidimensionales de color (Rojo, Verde y Azul), donde la intensidad del color en cada píxel se define por un valor numérico típicamente entre 0 y 255 en cada capa. A partir de esta base parte el

desarrollo de los métodos de procesamiento digital de imágenes ya que, al representar una figura como un conjunto de valores numéricos se hace posible aplicar operaciones matemáticas sobre cada una de las capas de color, modificando la información contenida en estas y por consiguiente el aspecto visual del conjunto completo. Realizar operaciones sobre una imagen abre un abanico de posibles áreas de investigación ampliamente estudiadas en la actualidad, como por ejemplo lo son los extractores de características que son parte importante de lo que se conoce hoy en día como visión artificial.

Figura 2.

Imagen descompuesta en capas RGB.



Nota: Adaptado de *Yale Style Manual-Graphics for the Web*, por Patrick Lynch y Sarah Horton, 1997, Web Style Guide (https://webstyleguide.com/wsg1/graphics/display_primer.html). (Lynch & Horton, 1997).

Para un ser humano asociar una imagen con un concepto es una tarea desarrollada de forma automática, gracias a la potencia del cerebro solo se requiere de un “vistazo” para conseguirlo, este no es el caso para las máquinas. Dado que estas no pueden apropiarse de los conceptos que

describen el contenido de una imagen, requieren de procesos que descompongan el conjunto visual original en características que logren describirlo, como lo pueden ser el color, los contornos o las texturas.

Estos procesos, comúnmente llamados transformaciones, son operaciones matemáticas que convierten las imágenes en datos que puedan ser comprendidos por una máquina. Uno de los ejemplos más comunes es la magnitud del gradiente (1), la cual permite la detección de bordes al destacar los cambios abruptos en la intensidad del color tanto en dirección horizontal como en vertical, dando como resultado visual una imagen como la mostrada en la Figura 3., en donde se puede observar que se descarta la información menos relevante (áreas con intensidad constante) al mismo tiempo que destacan las zonas de interés a través de los contornos de las formas originales.

$$|\nabla \mathrm{I}(\mathrm{x},\mathrm{y})| = \sqrt{(\frac{\partial I}{\partial x}(x,y))^2 + (\frac{\partial I}{\partial y}(x,y))^2} \quad (1)$$

Figura 3.

Visualización de la magnitud de gradiente de una imagen.

Nota: Izquierda, imagen original. Derecha, resultado del cálculo de la magnitud del gradiente aplicado a cada píxel sobre la imagen original en escala de grises. Adaptado de

CSC320W: Introduction to Visual Computing, por Fernando Flores-Mangas, 2014, Department of Computer Science University of Toronto (https://www.cs.toronto.edu/~mangas/teaching/320/slides/CSC320L05.pdf). (Flores-Mangas, 2014).

2.2.2 Filtrado de imágenes

Si bien el uso de operaciones es efectivo para la identificación de bordes, regiones y otros tipos de características, en muchas ocasiones estas mismas operaciones pueden ser complejas y requerir de una alta capacidad de procesamiento, lo cual es una limitante común a la hora de desarrollar algoritmos relacionados con el procesamiento de imágenes. Como alternativa para abordar este problema se descubrió que es posible hacer uso de la convolución bidimensional (2), operando una imagen de entrada (x) con matrices llamadas filtros o *kernels* (h). Este proceso se puede ilustrar gráficamente tal como se observa en la Figura 4.

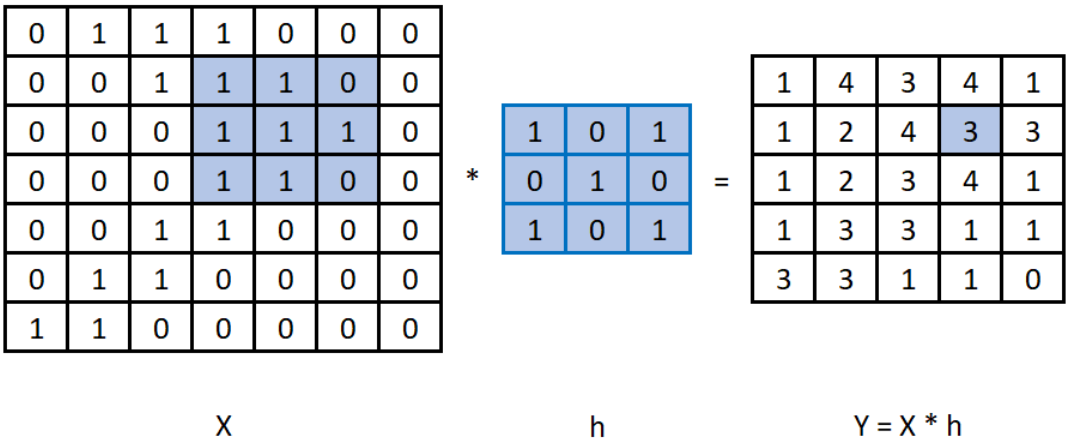
$$y(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[m,n].h[i-m,j-n] \quad (2)$$

En esencia un *kernel* es una matriz cuadrada, usualmente pequeña y de dimensiones impares cuyos valores fueron calculados con el objetivo de obtener determinado resultado al convolucionarse con una imagen. El propósito de hacer filtros pequeños es retener la mayor cantidad de datos posible, evaluando pequeñas regiones de la imagen en cada paso de la convolución, ya que operar con un filtro de mayor tamaño propicia la pérdida de información al dar como resultado una matriz de menor tamaño. Por otro lado, es deseable usar una matriz

cuadrada para que la forma original de la imagen de entrada no se vea afectada, asimismo el hecho de que el kernel tenga dimensiones impares va de la mano con tener un punto central que sirva como referencia durante la convolución.

Figura 4.

Representación gráfica de la convolución 2D.



Nota: El filtro h se desplaza paso a paso, partiendo su centro desde las coordenadas (1,1) a través de la imagen de entrada X, operando h con la región de X que logre abarcar, siendo el resultado de esta operación una posición en la matriz Y.

Este método de procesamiento de imágenes es ampliamente estudiado en la actualidad por su versatilidad ya que es posible conseguir una variedad de resultados distintos con lo que es en esencia una misma operación matemática, de igual forma la convolución es una operación estudiada a profundidad, con implementaciones computacionalmente eficientes al compararse con otros métodos más complejos. Estas ventajas son aplicadas en las Redes Neuronales Convolucionales (CNN) donde se usan kernels para realizar la extracción de características de imágenes. En la actualidad las CNN se han posicionado en la cima de las competencias de visión

artificial por sobre otros modelos de inteligencia artificial, todo esto gracias a su velocidad y precisión a la hora de desarrollar tareas como detección de objetos o clasificación de imágenes.

2.3 Aprendizaje profundo

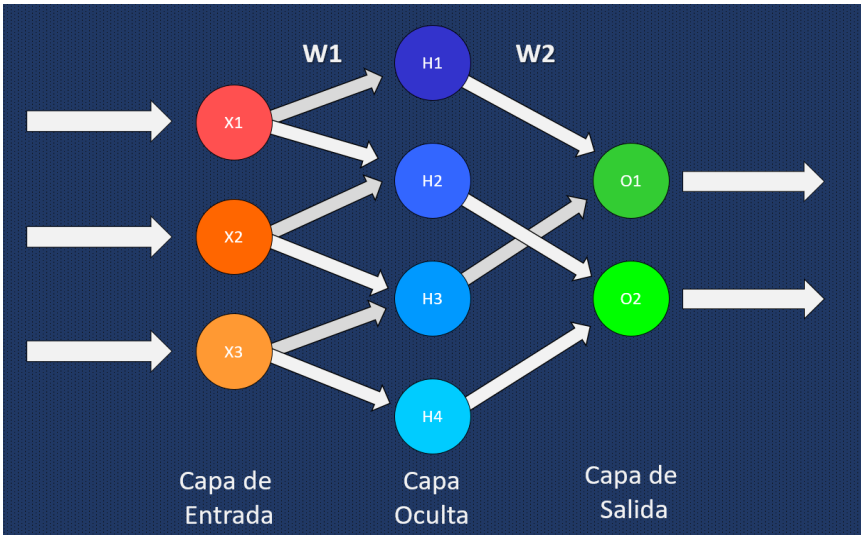
En años recientes la inteligencia artificial (IA) ha tomado fuerza y popularidad tanto en el ámbito investigativo como en el imaginario colectivo, esto se debe a que las nuevas tecnologías, que han permitido mejorar la velocidad y capacidad de los microprocesadores, a su paso han abierto las puertas para el entrenamiento y la ejecución de modelos mucho más complejos que los existentes hace 20 o 30 años cuando empezó a darse más atención a esta área de estudio y con ello abriendo la posibilidad de desarrollar aplicaciones cada vez más cotidianas. Dentro de este "boom" de popularidad ocupa un espacio importante la rama de la IA conocida como aprendizaje profundo o *deep learning* (DL), la cual parte del conocimiento de la forma en que funcionan las neuronas en el cerebro humano para construir redes neuronales artificiales (ANN).

La estructura básica de una red neuronal artificial puede resumirse en 3 capas principales compuestas cada una por nodos que reciben y envían información, tal como muestra la Figura 5. La primera capa es llamada capa de entrada, esta se encarga de recibir los datos que van a ser analizados por la red, cada uno de estos nodos en la entrada puede considerarse como una característica que define al elemento ingresado. A continuación, se encuentra la llamada capa oculta donde sucede todo el procesamiento de la información, se puede definir cada uno de los nodos pertenecientes a esta como una función matemática que transforma los valores que recibe de uno o varios nodos previos y envía los resultados obtenidos al siguiente, en la capa oculta se aplican por primera vez las funciones de activación, añadiendo un factor de no linealidad a la salida de cada neurona. Finalmente, luego de haber sido procesadas, las señales se envían a la capa de

salida, en donde a partir de los valores obtenidos al final del proceso se calculan las probabilidades con las que se decidirá cuál es la respuesta correcta.

Figura 5.

Ejemplo de la arquitectura de una red neuronal artificial.



Nota: Las redes neuronales artificiales intentan replicar las conexiones que existen entre las neuronas cerebrales y la forma en que se transmiten las señales entre ellas.

El aprendizaje de un modelo de IA ocurre durante el proceso de entrenamiento, si bien las técnicas para optimizar este proceso en modelos de inteligencia artificial son un área de estudio en sí mismas, todas tienen un factor en común. Se trata un proceso iterativo en el que, en cada ciclo, también llamado *epoch*, se procesa la totalidad de las muestras disponibles en una base de datos de entrenamiento, se calcula la precisión alcanzada para cada muestra y por el modelo completo a través de una función de costo, y finalmente se aplica propagación inversa para actualizar los valores de los pesos (variables ajustables del modelo), buscando disminuir la diferencia entre la respuesta deseada y la obtenida luego de cada *epoch*.

2.3.1 Función de costo

De acuerdo con la explicación expuesta por (LeCun, Bottou, Orr, & Müller, 1998), durante su entrenamiento un algoritmo de IA procesará cada una de las muestras de entrada e intentará predecir la probabilidad de que pertenezcan a determinada categoría, esto puede definirse con la ecuación (3), donde T^p representa la probabilidad calculada por el algoritmo para la muestra p, dicho valor es dado por la función M que representa al modelo, que depende de las variables Z^p, la muestra de entrada y W los pesos.

$$T^p = M(Z^p, W) \tag{3}$$

Con el objetivo de calcular qué tan alejada se encuentra T^p de la respuesta correcta para la muestra procesada Z^p, se implementa en el algoritmo una función matemática conocida como función de costo. Actualmente existen una gran variedad de funciones optimizadas para cumplir con este propósito. Una idea general de la forma en que trabajan las funciones de costo puede adquirirse comprendiendo una de las más sencillas, como lo es la función del error medio cuadrático.

Luego de cada predicción realizada se aplica la función (4), con la cual se calcula el error para la muestra evaluada (E^p) a partir de la comparación entre la probabilidad calculada por el modelo (T^p) y el valor deseado para dicha respuesta (D^p).

$$E^p = \frac{1}{2}(D^p - T^p)^2 \tag{4}$$

Posteriormente, al finalizar la totalidad de las muestras del *epoch*, se calcula el promedio (5) de todos los errores de las P predicciones realizadas, obteniendo el error total de la iteración (E_{train}), usado como referencia para determinar el desempeño general del modelo y posteriormente realizar los ajustes correspondientes a los pesos de este a través de la propagación inversa.

$$E_{train} = \frac{1}{P}\sum_{p=1}^{P} E^{p} \tag{5}$$

2.3.2 Propagación inversa

Con el uso de las funciones de costo es posible identificar qué tan acertadas son las predicciones de un modelo respecto a la realidad, el propósito del entrenamiento es reducir la brecha entre los resultados obtenidos y la respuesta esperada. Por esta razón se introduce en este punto un concepto conocido como propagación inversa, haciendo uso de esta es posible realimentar al modelo la información obtenida por la función de costo y de esta forma ajustar el valor de los pesos antes del siguiente epoch. Tomando en cuenta estas afirmaciones y retomando la explicación expuesta por (LeCun, Bottou, Orr, & Müller, 1998), sería posible definir el comportamiento de un modelo simple como una acumulación de módulos o capas secuenciales que implementan la función (6), esta representa la operación realizada por la n-ésima capa donde W_n corresponde a los valores ajustables de la capa mientras que X_{n-1} es el valor de entrada de la capa actual que a su vez también es la salida de la capa anterior. En el caso particular de X_0, al igual que en la ecuación (3) toma el valor de Z^p.

$$X_n = F_n(W_n, X_{n-1}) \tag{6}$$

Partiendo del supuesto en que se conoce la derivada parcial de E^p con respecto a X_n, es posible computar las derivadas parciales de E^P con respecto a W_n y X_{n-1} como se indica en las ecuaciones (7) y (8) donde $\frac{\partial F}{\partial X}(W_n, X_{n-1})$ y $\frac{\partial F}{\partial X}(W_n, X_{n-1})$ hacen referencia al Jacobiano de F respecto a W y X respectivamente.

$$\frac{\partial E^P}{\partial W_n} = \frac{\partial F}{\partial W}(W_n, X_{n-1}) \frac{\partial E^P}{\partial X_n} \tag{7}$$

$$\frac{\partial E^P}{\partial X_{n-1}} = \frac{\partial F}{\partial X}(W_n, X_{n-1}) \frac{\partial E^P}{\partial X_n} \tag{8}$$

El concepto de propagación inversa recibe su nombre a partir de la forma en que se usan estas ecuaciones para calcular los gradientes en un modelo, aplicándose a cada capa iniciando desde la N y finalizando en la 1, el cálculo se realiza de esta forma ya que así es posible realizar ajustes sobre los valores de los pesos de cada capa a partir del resultado obtenido por la capa siguiente. El proceso de aprendizaje más sencillo puede definir la forma en que se actualizan los pesos en cada iteración tal como lo describe la función (9), donde t hace referencia a la iteración y dependiendo del procedimiento utilizado el factor η, también conocido como tasa de aprendizaje, puede ser tanto una constante como una variable; la correcta selección de η es de gran importancia

para el entrenamiento, ya que tiene repercusión directa sobre la forma en que se ajusta la precisión del modelo en cada iteración del proceso.

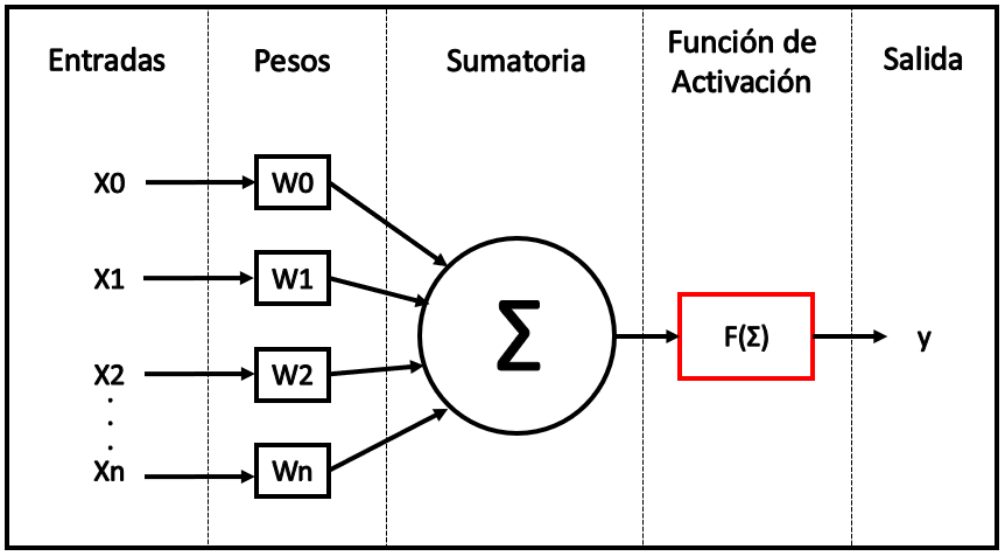
$$W(t) = W(t-1) - \eta \frac{\partial E}{\partial W} \tag{9}$$

2.3.3 Función de activación

En la ecuación (6) se describió la relación entre entrada y salida de un nodo de una red neuronal artificial. Gráficamente se puede representar cada uno de estos nodos, también llamados neuronas, como se ilustra en la Figura 6. Aquí puede observarse que si bien la salida depende directamente de los valores obtenidos a partir de la operación entre entradas y pesos, existe una etapa adicional donde a la sumatoria de los valores de entrada se le aplica una "Función de Activación", el propósito de estas funciones es el de convertir los valores de la sumatoria en valores aptos para la salida del nodo y comparar estos últimos contra un nivel de umbral que dictará si esta neurona debe "activarse" ante determinadas entradas (Jeyanthi & Subadra, 2014), adicionalmente las funciones de activación añaden un factor de no linealidad valioso para aumentar la capacidad de aprendizaje del modelo ante representaciones complejas (David Rasamoelina, Adjailia, & Sincák, 2018).

Figura 6.

Estructura de una neurona artificial.

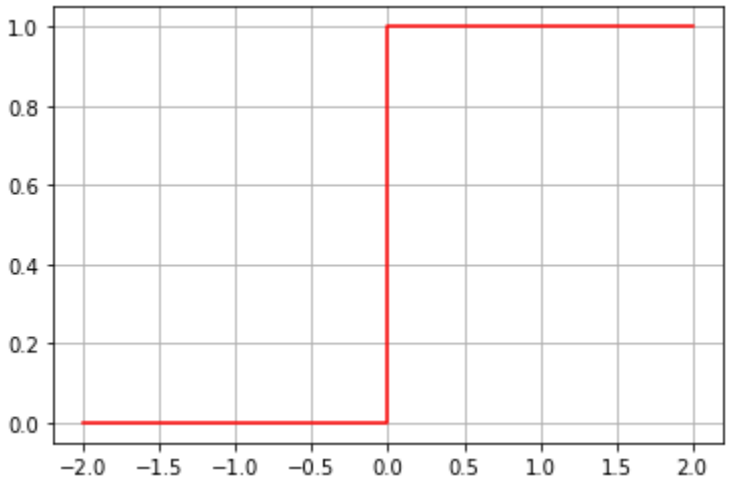


En la actualidad se conoce una gran variedad de funciones de activación que pueden aplicarse en distintas situaciones, adaptándose cada una de mejor o peor forma dependiendo del problema a resolver. Un ejemplo sencillo de esto es el caso de la función escalón (10), la cual se ajusta a los modelos de clasificación binarios, aquellos que cuentan únicamente con 2 categorías para identificar, pero si se usa en un modelo que debe clasificar una cantidad superior de categorías no permitirá al sistema operar correctamente ya que no distingue ningún valor fuera de 0 y 1.

$$y(x) = \begin{cases} 0, si\ x \leq 0 \\ 1, si\ x > 0 \end{cases} \quad (10)$$

Figura 7.

Gráfica de la función escalón



A raíz de esta necesidad se han investigado funciones que permitan a los modelos aprender patrones mucho más complejos, pero que al mismo tiempo sean computacionalmente eficientes. Es así como se llegó a la función sigmoide (11), este tipo de función de activación ganó gran popularidad en las aplicaciones relacionadas con inteligencia artificial cuando Yann LeCun expuso el uso de la tangente hiperbólica como función de activación en su modelo de red neuronal convolucional LeNet, usado en la identificación de dígitos escritos a mano (Wood, s.f.).

$$y(x) = \frac{1}{1 + e^{-x}} \tag{11}$$

Su capacidad para mapear cualquier señal en el rango de (-∞, ∞) a cualquier valor entre 0 y 1 la hacía ideal para el cálculo de probabilidades en la selección de categorías. De igual forma el ser una función derivable cuya derivada puede calcularse de forma relativamente eficiente la

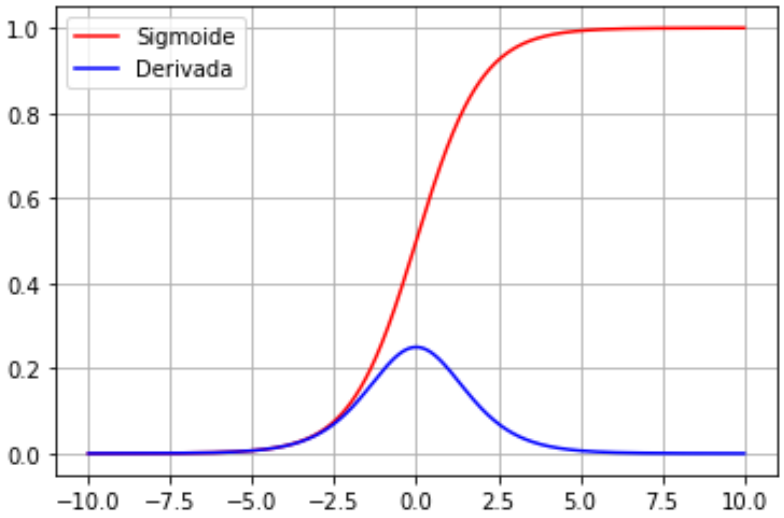
convirtió durante mucho tiempo en la función de activación más usada para los algoritmos cuyo aprendizaje se basaba en el método del descenso del gradiente.

Sin embargo, el hecho de que su derivada tenga un rango efectivo reducido conlleva consigo un problema denominado desvanecimiento del gradiente, el cual limita la capacidad de aprendizaje del modelo a medida que se construyen redes neuronales más profundas y dificulta su optimización debido a que para valores grandes el gradiente tiende a hacerse 0, véase Figura 8. (Squartini, Hussain, & Piazza, 2003).

Gracias al notable incremento en la capacidad de computación que se ha visto en las últimas décadas, la tendencia en el desarrollo de arquitecturas de redes neuronales ha sido la creación de modelos más profundos con la capacidad de aprender patrones más complejos, haciendo del desvanecimiento del gradiente un problema cada vez más relevante, por lo que ha sido necesario buscar funciones de activación alternativas que no sufrieran tan gravemente de dicha limitante.

Figura 8.

Gráfica de la función sigmoide logística y su derivada.



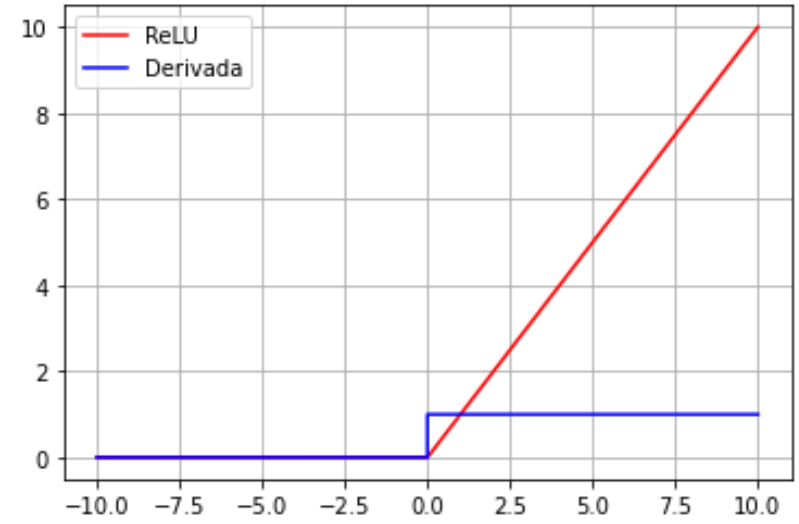
La función ReLU (Unidad Lineal Rectificada), definida por la ecuación (12), es la función de activación más usada en la actualidad en redes neuronales convolucionales, esta popularidad la ha ganado gracias a su desempeño superior durante la etapa de entrenamiento respecto funciones de activación clásicas como la sigmoide logística o la tangente hiperbólica (David Rasamoelina, Adjailia, & Sincák, 2018), siendo ReLU capaz de acelerar la convergencia del modelo y llegando a mejores soluciones (Kaiming, Xiangyu, Shaiqing, & Jian, 2015). De acuerdo con la explicación dada por (Glorot, Bordes, & Bengio, 2011), otra de sus grandes ventajas es que permite añadir un factor de dispersión (sparsity en inglés) gracias a que, a diferencia de las sigmoides, ReLU asigna un valor de 0 real a todos los valores que se encuentren por debajo de su umbral de activación, esto representa una ventaja en algunos aspectos como los listados a continuación:

- Permite "desenredar" la información. A diferencia de las representaciones dispersas, en las representaciones de datos densas es difícil aislar los distintos factores para estudiar las causas de sus variaciones dado que cualquier cambio en la entrada modificará la mayor parte de los elementos del vector representativo de datos.
- Representación eficiente de tamaño variable. Al desactivar totalmente las neuronas que no superan el umbral le facilita al modelo controlar la dimensionalidad de la representación para una entrada dada.

$$y(x) = \max(0, x) \tag{12}$$

Figura 9.

Gráfica de la función ReLU y su derivada.



.

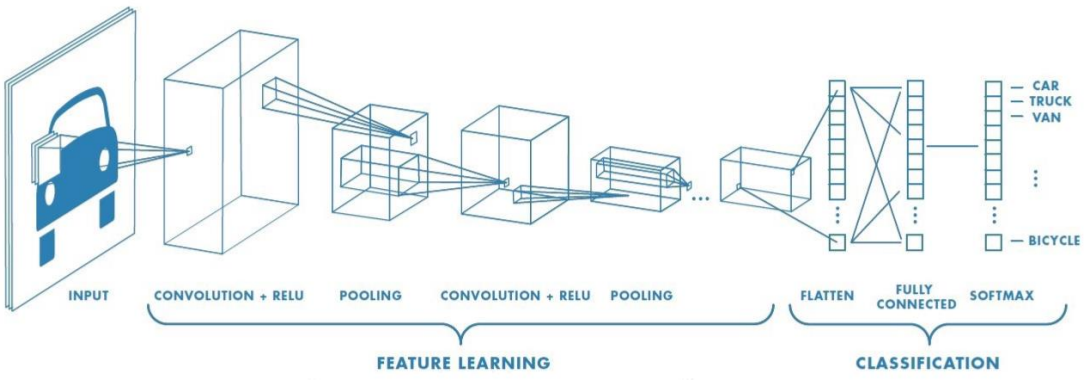
Si bien el uso de la función ReLU presenta grandes ventajas respecto a las anteriormente populares sigmoides, esta función también tiene sus desventajas. Por un lado, forzar demasiada dispersión afectará la precisión de las predicciones ya que desactivar una gran cantidad de neuronas al mismo tiempo reduce la capacidad efectiva del modelo (Glorot, Bordes, & Bengio, 2011). Por otro lado, el hecho de que todos los valores negativos sean reducidos a 0 implica una pérdida de información significativa (David Rasamoelina, Adjailia, & Sincák, 2018). Adicionalmente como se observa en la Figura 9., dado que el gradiente es 0 cuando la neurona se encuentra inactiva, al usar algoritmos de optimización basados en el gradiente existe la posibilidad de que no se actualicen los pesos de una neurona que no se haya activado inicialmente (Maas, Hannun, & Ng, 2013).

2.3.4 Redes neuronales convolucionales

Habiendo presentado las bases del funcionamiento de las redes neuronales y el procesamiento de imágenes es posible describir las redes neuronales convolucionales como la fusión de estas dos áreas de investigación. Siguiendo el diagrama expuesto en la Figura 10., una red neuronal convolucional (CNN) consta de una serie de convoluciones que funcionan como extractores de características, descomponiendo las imágenes de entrada en múltiples capas. Luego de haber sido entrenado el modelo, cada una de estas capas de convolución corresponde a una neurona artificial especializada en detectar una característica en particular.

Figura 10.

Diagrama de una red neuronal convolucional.

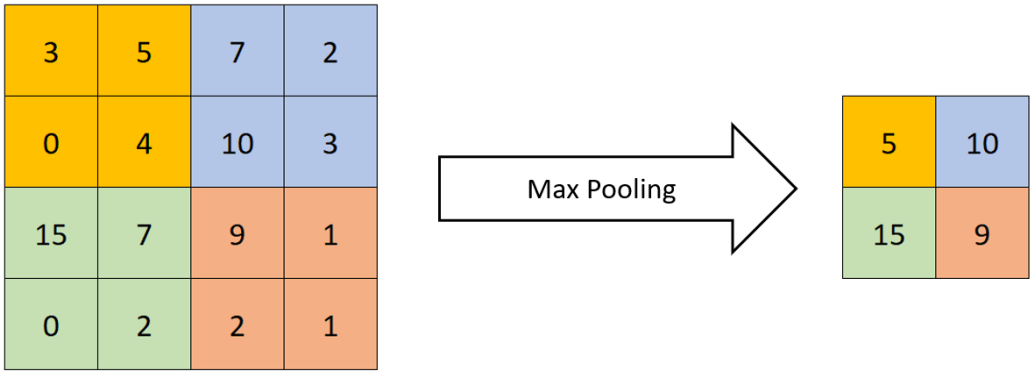


Nota. Se ilustra el proceso de una imagen desde que ingresa a la red neuronal hasta que el algoritmo de clasificación decide a cuál categoría pertenece. Adaptado de *A Comprehensive Guide to Convolutional Neural Networks*, por Sumit Saha, 2018, Towards Data Science (https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53). (Saha, 2018).

En las CNN también se introducen conceptos de nuevas capas como el *pooling*, cuyo proceso se ilustra en la Figura 11., este se encarga de reducir el tamaño del mapa de características obtenido luego de una convolución, esto es de utilidad ya que los métodos de pooling buscan retener la información más relevante del mapa de características al mismo tiempo que descarta los puntos con menor relevancia, de esta forma se reduce la cantidad de datos que debe procesar el modelo sin perder las características identificadas a través de la convolución.

Figura 11.

Operación max pooling.



Nota. En este ejemplo se realiza la operación de pooling con una ventana de 2x2 pixeles y un paso de 2 pixeles.

También se introducen capas de aplanado o *flatten* las cuales convierten los mapas de características, entregados por la etapa de procesamiento de la imagen, en un único vector de características que sirve como entrada para las capas densas. En estas últimas cada una de sus neuronas corresponde con una característica extraída de la imagen ingresada al modelo; en esta etapa los valores del vector de entrada se operan con los pesos de cada neurona y se les aplica una

función de activación para determinar cuáles tienen un valor significativo, acercando al algoritmo a dar una respuesta para la muestra analizada.

La capa final del modelo, llamada *Softmax*, aplica la función (13) del mismo nombre, la cual se encarga de calcular la probabilidad de que la respuesta del modelo pertenezca a cada una de las categorías para las que fue entrenado.

$$\sigma(Z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{13}$$

En esta función Z corresponde al vector de entrada entregado por las capas previas del modelo, z_i corresponde a la posición i en el vector de entrada, y K al total de categorías para las que se entrenó el modelo.

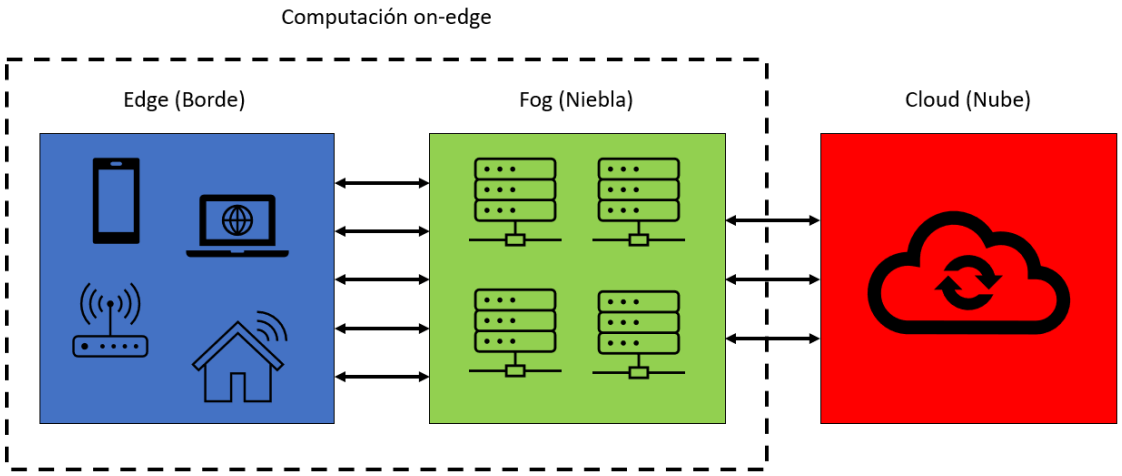
2.4 Computación *on-edge*

Debido a la gran cantidad de operaciones que deben realizar para cumplir con su función, tradicionalmente los modelos de inteligencia artificial han requerido de una capacidad de cómputo superior para su entrenamiento y posterior uso. Con el paso de los años se han dado grandes avances tecnológicos a través de los cuales, entre otras cosas, ha sido posible incrementar el poder de procesamiento de los equipos electrónicos, permitiendo el desarrollo de procesos más complejos y que al mismo tiempo estos puedan verse envueltos cada vez más en la resolución de situaciones cotidianas. De forma paralela, el auge actual de las tecnologías IoT (Internet de las cosas), con las cuales es posible mantener múltiples dispositivos interconectados y al mismo tiempo con conexión a la red, ha aliviado en parte la necesidad de "fuerza bruta" requerida para acceder a la inteligencia artificial desde dispositivos menos robustos, ya que al contar con acceso

a la nube no es necesario procesar grandes cantidades de datos o realizar operaciones complejas en el dispositivo, la información es cargada en grandes servidores donde se procesa para posteriormente retornar los resultados al punto inicial. Si bien esta tecnología ha sido útil para muchas aplicaciones distintas también tiene desventajas significativas que no pueden ignorarse, como los costos de acceso a los servidores, la latencia superior a que tendría un equipo que ejecuta las operaciones por sí mismo, las limitaciones en la conectividad a internet en algunas zonas o la seguridad de la información que se está enviando.

Figura 12.

Computación on-edge.



Nota. La computación on-edge abarca no sólo los dispositivos finales sino también el procesamiento realizado en servidores intermedios con la nube llamados fog o niebla.

La computación *On-Edge*, representada en la Figura 12., es uno de los campos de estudio que más atención está recibiendo actualmente en el área de las tecnologías de la información, dentro de las que se encuentra la inteligencia artificial. Ya que se presenta como una alternativa

para solventar algunos de los problemas vistos comúnmente en las aplicaciones que hacen uso de la nube. La idea detrás de esta relativamente nueva tendencia es contar con dispositivos que puedan procesar total o parcialmente los datos, esto trae ventajas para la seguridad al descentralizar la información, una disminución en el tiempo de respuesta del sistema al tratarse de un sistema local y un costo operacional inferior, entre otras. Esto es posible gracias a que cada vez es posible conseguir una mayor capacidad de procesamiento en dispositivos más pequeños que pueden ajustarse a una mayor variedad de aplicaciones.

3. Desarrollo del proyecto

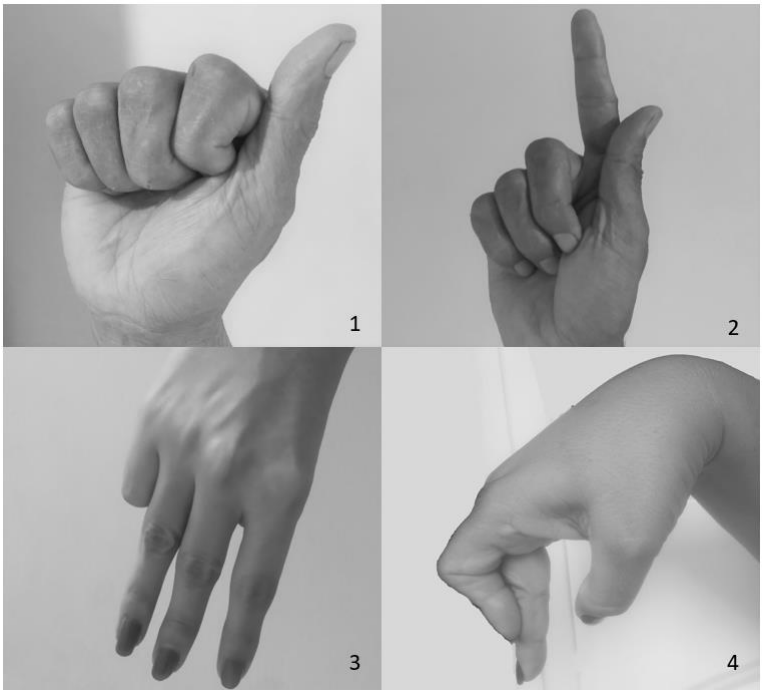
En este capítulo se describe el paso a paso seguido para el desarrollo del proyecto, pasando por las consideraciones tenidas en cuenta para la construcción de la base de datos, la selección del sistema embebido y la arquitectura de red neuronal convolucional a utilizar y finalmente el entrenamiento y la implementación del modelo seleccionado.

3.1 Base de datos

Para el entrenamiento del modelo se construyó una base de datos con aproximadamente 5500 imágenes separadas en 22 categorías distintas, disponibles aquí. Cada categoría corresponde a uno de los signos inmóviles del lenguaje de señas colombiano y está compuesta por alrededor de 250 imágenes como las que se muestran en la Figura 13. Las imágenes fueron capturadas en su totalidad con la cámara de un celular Xiaomi Redmi 5 plus con una resolución de 4000x3000 pixeles, se usó como modelos a 14 personas distintas, 6 mujeres y 8 hombres, con edades entre 10 y 50 años.

Figura 13.

Ejemplos de los gestos capturados para la base de datos.



Nota. Los gestos tomados como ejemplo para la imagen corresponden a las letras 1. A, 2. F, 3. M y 4. P.

Adicionalmente, con el objetivo de reducir las posibles fuentes de ruido que pudieran afectar la precisión alcanzada por el modelo durante el entrenamiento de la red neuronal, se optó por editar manualmente cada imagen siguiendo los pasos listados a continuación:

Recorte de imagen: Dado que la cámara con la que fueron tomadas las fotos las almacena con una resolución de 4000x3000 pixeles, y que en la mayoría de ellas se encontraba un amplio espacio que no correspondía con el objetivo principal, se optó por recortar cada imagen, eliminando las secciones irrelevantes y así permitiendo que el foco principal sea la mano que realiza el gesto.

Transformación a escala de grises: Ya que el propósito de la red neuronal a entrenar es identificar las figuras formadas por las manos de cada persona, y para esta finalidad el color no es una característica prioritaria, se decidió convertir las imágenes a escala de grises para facilitar algunos de los pasos siguiente en el proceso de edición.

Eliminación de elementos de fondo: En algunas de las imágenes se encontraban elementos en el fondo, como por ejemplo sombras de la propia mano, que podrían dar espacio a errores al momento de la identificación del gesto. Por lo que se eliminaron procurando dejar un fondo plano en el que destacara únicamente la señal realizada por la persona fotografiada.

Aumento del contraste entre la mano y el fondo: Dado que las condiciones de iluminación bajo las cuales se realizaron las capturas no pudieron mantenerse constantes para todas, se decidió modificar el nivel de sombras, contraste e iluminación para destacar la mano respecto al fondo en los casos que fuera necesario.

Ajuste de tamaño: Teniendo en cuenta que el primer paso de la edición consistió en recortar cada imagen de forma individual, dejando tamaños inconsistentes entre unas y otras, se utilizó un script de Python con el que se ajustaron todas las capturas tomadas, reescalando sus dimensiones a 250x250 pixeles. De esta forma se garantiza que todas las imágenes sean simétricas y posean el mismo tamaño para el proceso de convolución. Al mismo tiempo se reduce el espacio necesario en memoria para procesar la base de datos durante el entrenamiento del modelo.

3.2 Selección del sistema embebido

Cuando se trata de sistemas embebidos, en la actualidad es posible encontrar en el mercado una variedad de opciones distintas. Y gracias a la creciente popularidad de las tecnologías de procesamiento *on-edge*, cada vez una mayor cantidad de tarjetas soporta la ejecución de procesos relacionados con la inteligencia artificial como los algoritmos de clasificación en tiempo real,

existiendo incluso sistemas de tamaño reducido con la capacidad de entrenar por sí solos redes neuronales convolucionales.

Si bien existen múltiples opciones con la capacidad de trabajar con redes neuronales convolucionales, hay factores importantes para tener en cuenta en la selección de un dispositivo que se ajuste a los objetivos de este proyecto, siendo uno de los más importantes el económico; como ya se mencionó anteriormente existen dispositivos, como por ejemplo la Jetson Nano de NVIDIA, que poseen una gran potencia de procesamiento, pero al mismo tiempo tienen un costo elevado que las poco atractivas para el desarrollo de este proyecto. Con el objetivo de decidir cuál es la opción más apta se realizó una comparación entre múltiples opciones disponibles en el mercado.

Tabla 1.

Comparación entre sistemas embebidos para procesamiento de redes neuronales disponibles en el mercado.

Nombre	**Especificaciones**							**Precio USD**
	Procesador	**Velocidad**	**Voltaje**	**Flash**	**RAM**	**Cam**	**LCD**	
Sparkfun Edge - Apollo3 Blue	32-bit ARM Cortex-M4F	48MHz	3,6 V	1MB	384KB	C	X	18,69
Jetson Nano	ARM A57	1,43 GHz	5V	16GB	4GB	C	X	123,75
Coral Dev Board	NXP i.MX 8M SoC	1,3GHz	5V	8GB	1GB	MiPi CSI 2	X	177,23
Sipeed Maix bit	Kendryte k210	400MHz	5V	Micro SD	8MB	FPC	8bit MCU 24P	24,21
Raspberry Pi 4	ARM CORTEX A72	1.5 GHz	5V	Micro SD	1,2 o 4 GB	MPI CSI	MIPI DSI Display	35

32F746G DISCO-VERY	STM32F746NGH6 ARM CORTEX	216MHz	3.3-5 V	1MB	340kB	C	4,3" Color LCD-TFT touch	56,25
Adafruit Edge Badge	ATSAMD51J19	120MHz	3,3V	2MB	192kB	X	1,8" TFT Display	35,95

Nota. Las celdas marcadas con la letra C hacen referencia a que el dispositivo cuenta con un conector destinado a la función indicada, mientras que las marcadas con una X indican que no cuenta con esta opción.

A partir de la información recopilada en la Tabla 1. se concluyó que, por su balance entre un precio accesible y sus especificaciones, principalmente la potencia de su núcleo diseñado específicamente para el procesamiento de redes neuronales, la alternativa que más se ajusta a los objetivos de este proyecto es la Sipeed Maix Bit. A continuación, se profundizará más en los detalles de esta.

3.2.1 Sipeed Maix Bit

De acuerdo con la descripción provista por el proveedor la Sipeed Maix Bit es un dispositivo desarrollado por Seeed Studio con el propósito de ser capaz de ejecutar, con alto rendimiento, algoritmos de inteligencia artificial en una placa de tamaño reducido con bajo consumo de energía a un costo accesible (Seeed Studio, 2021). En la Tabla 2.

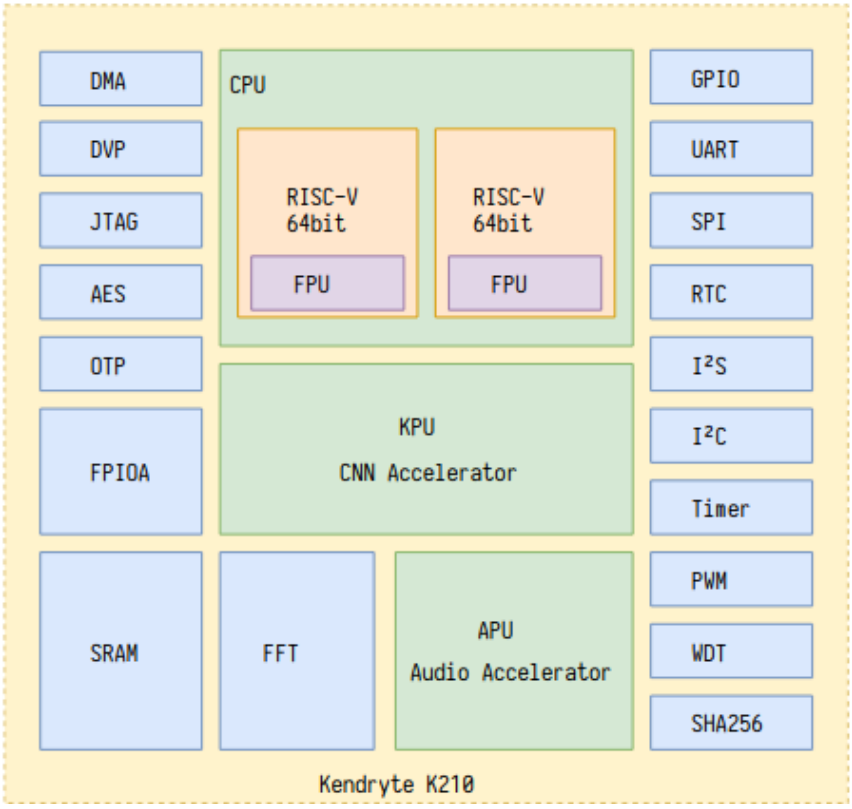
*Características de la Sipeed Maix Bit.*se sintetiza de forma más detallada las características de esta tarjeta.

Esta tarjeta se construye sobre el núcleo Kendryte K210 (Canaan Inc., 2019) (Figura 14.

Diagrama de bloques del núcleo Kendryte K210.), el cual cuenta con 8MB de memoria RAM; bajo condiciones normales opera alrededor de los 400MHz, mientras que bajo *overclocking* es capaz de alcanzar una velocidad de 800MHz. Gracias a esto y su procesador de redes neuronales (KPU) integrado se encuentra en la capacidad de ejecutar 0.5 Tera Operaciones por Segundo (TOPS), pudiendo realizar reconocimiento de imágenes en tiempo real a 60 cuadros por segundo en calidad VGA.

Figura 14.

Diagrama de bloques del núcleo Kendryte K210.



Nota. Adaptado de K210 Datasheet (p. 8), por Canaan Inc., 2019. (Canaan Inc., 2019).

Adicionalmente como complemento a sus capacidades para la visión artificial, el procesador Kendryte K210 cuenta también con un procesador de audio con 8 canales de entrada, con la capacidad muestrear señales a 192kHz y con una unidad de aceleración de transformada rápida de Fourier (FFT). Esto le permite a la Sipeed Maix Bit ser apta para aplicaciones de escucha artificial e incluso aplicaciones que combinen tanto visión como escucha de forma simultánea.

Tabla 2.

Características de la Sipeed Maix Bit.

Procesamiento	
Procesador	Kendryte K210
Velocidad	400MHz
IA	
Acelerador	KPU
Memoria	
Flash	MicroSD
RAM	8MB
Interfaces	
Cámara MPI CSI	
LCD MCU	
USB-C	
Micrófono MEMS	
48 GPIO	
Alimentación	
Voltaje	4,8 - 5,2 V
Corriente	600 mA
Precio [USD]	
Digikey	14,36
Seedstudio	12,9

Dado que el foco principal de la Maix Bit es la inteligencia artificial, este equipo no cuenta con gran variedad de opciones para su conectividad con otros dispositivos. Sin embargo, posee

conectores FPC para cámara MPI CSI y pantalla LCD MCU de 8 bits, además de un micrófono MEMS integrado, también cuenta también con 48 pines GPIO.

Respecto a su programación, existen distintas herramientas que pueden usarse para este propósito. La Maix Bit soporta el kit de desarrollo de software (SDK) FreeRTOS basado en C, también fue desarrollado un *port* de MicroPyton para el chip Kendryte K210, llamado MaixPy y finalmente existen también librerías disponibles para las distintas tarjetas desarrolladas por Seeed Studio que pueden descargarse desde el IDE de Arduino para compilar y cargar programas a la tarjeta.

3.3 Selección del modelo de red neuronal

Para la selección del modelo a implementar se partió de dos puntos clave. En primer lugar, se desea desarrollar una aplicación que pueda hacer reconocimiento de gestos en tiempo real, mientras que, en segundo lugar, es importante tener presentes las limitaciones de hardware (principalmente en cuanto a la memoria RAM) del dispositivo seleccionado, por lo que el modelo a entrenar debe ser ligero y eficiente. Teniendo esto en mente y dado que Seeed Studio destaca la compatibilidad de la arquitectura MobileNetV1 (Howard, y otros, 2017), con sus productos (Seeed Studio, 2021), se decidió hacer pruebas sobre esta para determinar su precisión entrenándose sobre la base de datos de la LSC. Junto con MobileNetV1 se decidió también evaluar también las arquitecturas MobileNetV2 (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018) y ResNet50 (He, Zhang, Ren, & Sun, 2015) con el fin de obtener una comparativa en sus desempeños.

3.3.1 Entrenamiento preliminar

La evaluación de los modelos consistió en entrenar cada una de las arquitecturas seleccionadas y comparar sus resultados prestando especial atención a su precisión y al tamaño de

cada uno, de esta forma se determinó cuál posee mejor desempeño y su viabilidad a la hora de cargarlos al sistema embebido.

Para entrenamiento de los modelos a evaluar en este proyecto existe un factor determinante que no se puede obviar, este es el tamaño de la base de datos. Debido a que el entrenamiento se realizará con un *data set* con una cantidad reducida de muestras (250 imágenes por categoría), es muy probable que no se alcancen los resultados deseados si se decide entrenar una red neuronal desde 0, dando pie a problemas como sobre ajuste, al no haber una gran variedad de muestras distintas, o sencillamente no alcanzando una precisión aceptable. Es por esto por lo que se decide utilizar el método de transferencia de conocimiento, en el cual se hace uso de un modelo con pesos inicializados producto de un entrenamiento previo en una base de datos más robusta donde la red haya aprendido a extraer adecuadamente información general, como bordes, formas básicas o texturas, en sus primeras capas. De esta forma el entrenamiento con la base de datos final se llevó a cabo únicamente sobre las ultimas capas de la red, que se ajustarán para identificar las características específicas de las nuevas categorías a clasificar. Mejorando así la precisión del modelo y reduciendo el tiempo de entrenamiento.

Para preparar los *data set* de entrenamiento y validación se utilizó un *script* de Python encargado de separar aleatoriamente cada una de las categorías en dos grupos con una distribución de las muestras de 80% para el set de entrenamiento y 20% para el de validación. Adicionalmente, durante la carga de la base de datos, y con la finalidad de reducir la probabilidad de que se presentara un sobre ajuste en el modelo, se realizó un aumento de datos sobre el *set* de entrenamiento aplicando las operaciones mostradas en la Tabla 3.

Tabla 3.

Operaciones de aumento de datos.

Operación	Valor
Rotación	25
Zoom	0,1
Desplazamiento Horizontal	0,1
Desplazamiento Vertical	0,1
Inversión Horizontal	True
Transvección	0,2

Los modelos utilizados como base para el entrenamiento son los almacenados en la librería Keras de Python, se cargaron con los pesos obtenidos luego de haber sido entrenados con la base de datos ImageNet (Stanford Vision Lab, 2020). Se ajustaron eliminando las últimas capas de cada uno, encargadas de hacer las predicciones para las categorías originales; a cambio de estas se inicializaron nuevas capas de predicción sin entrenar, respetando la arquitectura de cada modelo.

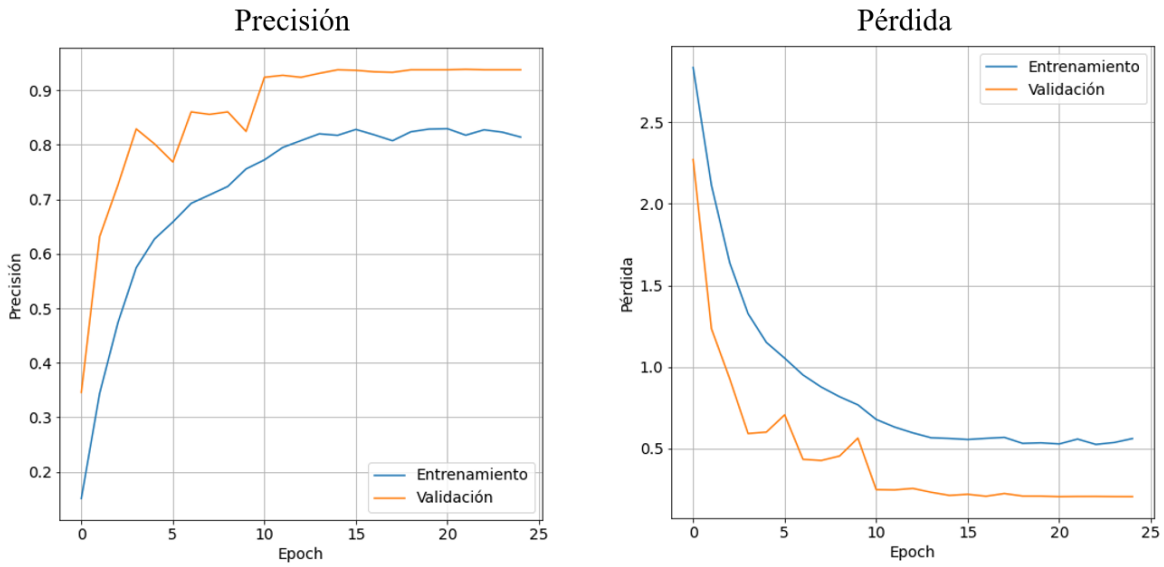
Entrando propiamente en el proceso de entrenamiento se decidió trabajar con un tamaño de *batch* de 16 dado el bajo número de muestras, las iteraciones se llevarían a cabo durante 50 *epochs* como máximo, para los 3 modelos se usó la entropía cruzada como función de costo y como optimizadores Adam para ResNet50 y SGD para MobileNet, tanto para su primera como para su segunda versión, con ratio de aprendizaje inicial de $1x10^{-3}$ para todos los casos.

Usando herramientas de la librería Keras se definieron dos condiciones para los entrenamientos. La primera de estas se encarga de reducir el ratio de aprendizaje en un 25% en caso de no encontrar un aumento superior a $1x10^{-8}$ en la precisión del set de validación durante 3 epochs consecutivos. La segunda, detiene por completo el entrenamiento si se presenta un

crecimiento en la precisión del set de validación inferior a $5x10^{-3}$ en los últimos 10 epochs, y retoma los pesos calculados durante el epoch con mejor precisión. El objetivo de esto es el de maximizar la precisión alcanzada durante el entrenamiento y al mismo tiempo que se previene un sobre ajuste causado por entrenamiento excesivo. Los resultados del entrenamiento pueden verse en las figuras Figura 15., Figura 16. y Figura 17.

Figura 15.

Entrenamiento del modelo MobileNetV1.



Con MobileNetV1, como puede observarse en la Figura 15., se presentó una diferencia significativa entre el resultado de sus sets de entrenamiento y validación, alcanzando un 81.4% en el primero y un 93.8% en el segundo, este comportamiento puede deberse a que el conjunto o *set* de entrenamiento, por haber sido sometido a operaciones de aumento de datos y contar con una capa de *dropout* añadida antes de la clasificación, contaba con muestras que resultaron ser más difíciles de identificar que las usadas durante la validación. El modelo guardado en formato H5

tiene un peso de 21.4 MB, al ser convertido a formato TFLITE se redujo a 13.6 MB. El script de su entrenamiento puede encontrarse aquí.

Con MobileNetV2 (Figura 16.

Entrenamiento del modelo MobileNetV2.) puede observarse que se redujo notablemente la diferencia entre la precisión durante el entrenamiento (95.3%) y la validación (96.5%) luego de entrenarse durante 30 epochs, 5 más de los que requirió MobileNetV1 para estabilizarse. Adicionalmente cabe destacar que MobileNetV2 cuenta con 2.5 millones de parámetros, 1 millón menos que su predecesora, lo cual permite que los modelos generados con esta arquitectura sean más ligeros; ocupando en formato H5 17.9 MB y en 9.95 MB en formato TFLITE. El script de su entrenamiento puede encontrarse aquí.

Figura 16.

Entrenamiento del modelo MobileNetV2.

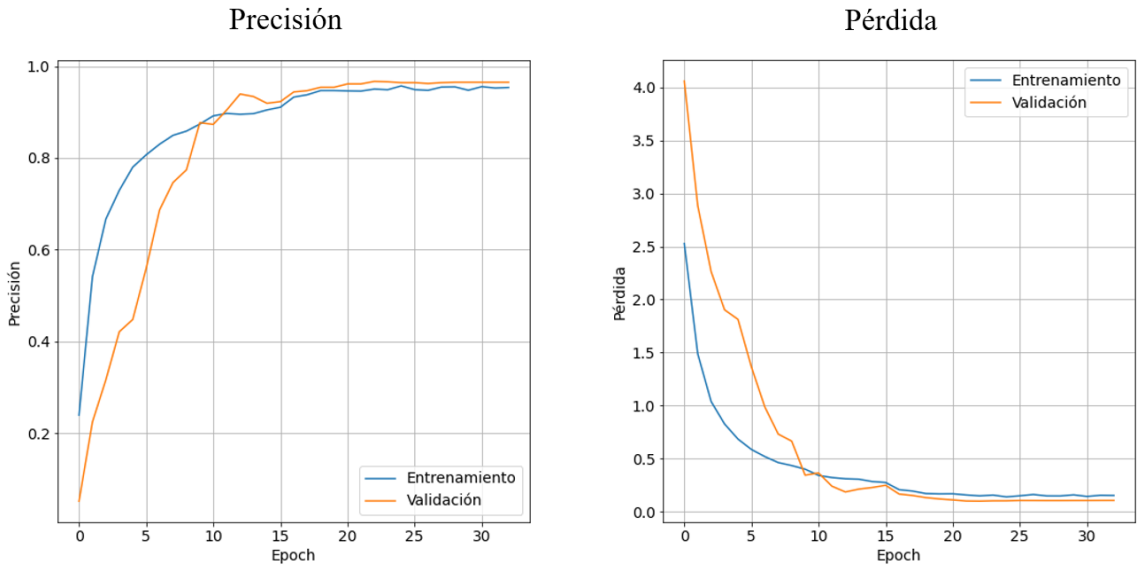
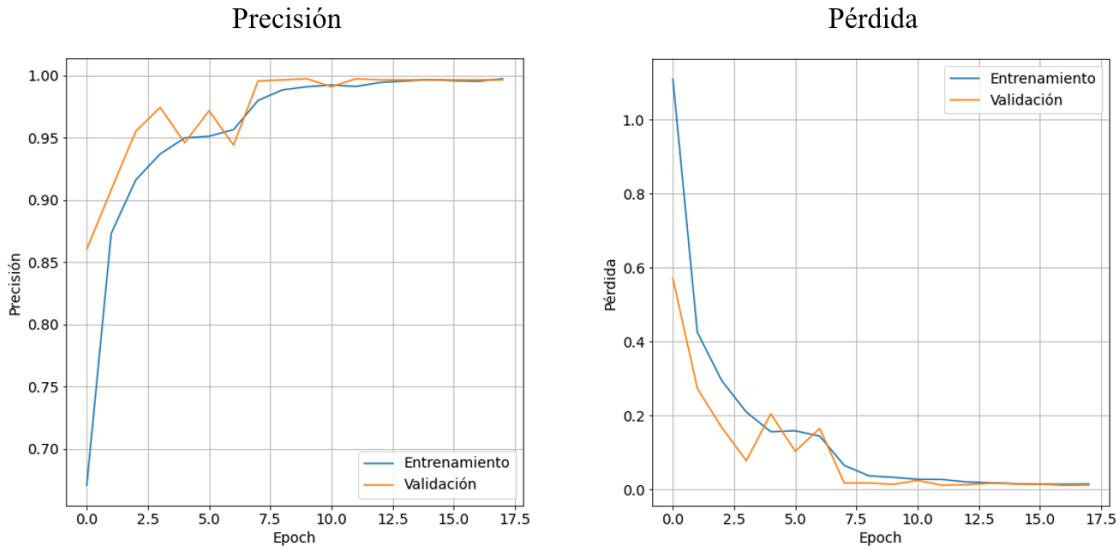


Figura 17.

Entrenamiento del modelo ResNet50.



Finalmente requiriendo tan solo 17 epochs para su estabilización, ResNet50 consiguió tanto para su set de entrenamiento como como para el de validación una precisión del 99.65%, posicionándose notablemente por encima de los resultados obtenidos con los otros dos modelos evaluados. Sin embargo, con un total de 24 millones de parámetros es también el modelo más pesado de todos, ocupando 134 MB en formato H5 y 94 MB al convertirse a TFLITE; por lo que, a pesar de su precisión superior, es una alternativa inviable implementar en la Sipeed Maix Bit, que no cuenta con la memoria suficiente. El script de su entrenamiento puede encontrarse aquí.

3.3.1 Reconocimiento en tiempo real

Habiendo obtenido resultados preliminares del desempeño para cada arquitectura de red neuronal convolucional, el siguiente paso consistió en hallar una forma en la que se pudiera aplicar alguno de los modelos seleccionados para hacer reconocimiento en tiempo real.

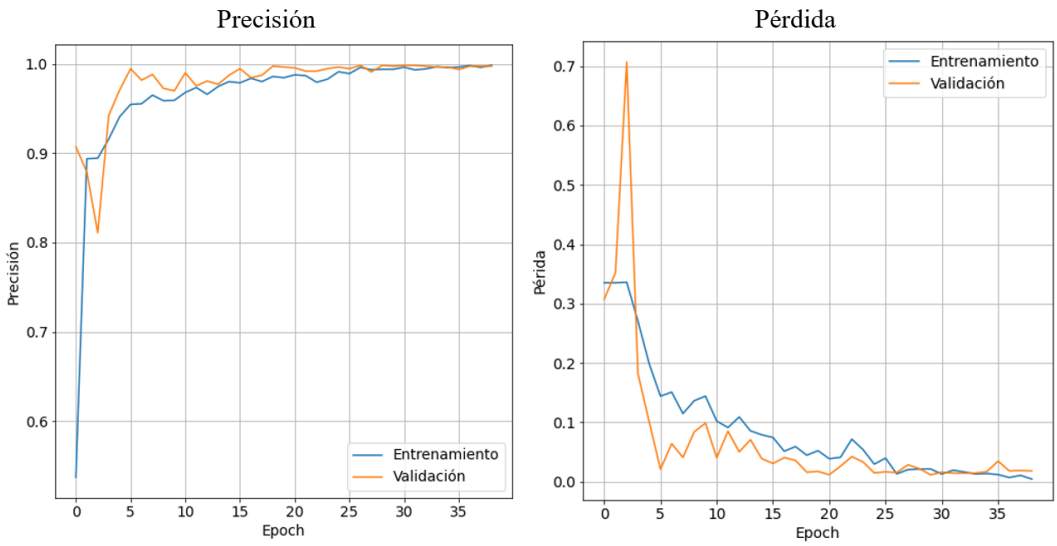
De acuerdo con la información entregada por el proveedor en su página (Seeed Studio, 2021), la Sipeed Maix Bit es compatible con la arquitectura YOLO (You Only Look Once) (Redmon, Divvala, Girshick, & Farhadi, 2016), uno de los modelos más populares en aplicaciones que realizan reconocimiento de objetos en tiempo real. Dado que esta arquitectura se encuentra soportada por Seeed Studio para su implementación en sus dispositivos, se decidió investigar más al respecto de la forma en que se entrena y cómo puede interactuar con los modelos evaluados previamente.

Si bien es posible entrenar YOLO por otros métodos, se encontró que específicamente para aplicaciones relacionadas con el núcleo Kendryte K210 existen dos herramientas que facilitan este proceso.

aXeleRate (Maslov, 2019), es un *framework* desarrollado por Dimitry Maslov con el objetivo de entrenar redes para detección de objetos basadas en la arquitectura YOLOv2 que puedan implementarse en un chip K210. Esta herramienta permite cierto nivel de personalización a la hora de definir el entrenamiento, por lo que es posible seleccionar un modelo de red neuronal como extractor de características para YOLO; dentro de los modelos compatibles con aXeleRate se encuentran MobileNetV1 y ResNet50. Dado que, como se concluyó anteriormente, ResNet50 no es una alternativa viable para la Sipeed Maix Bit, se realizó un entrenamiento de prueba de YOLOv2 con MobileNetV1 como extractor de características.

Figura 18.

Entrenamiento de YOLO usando aXeleRate.



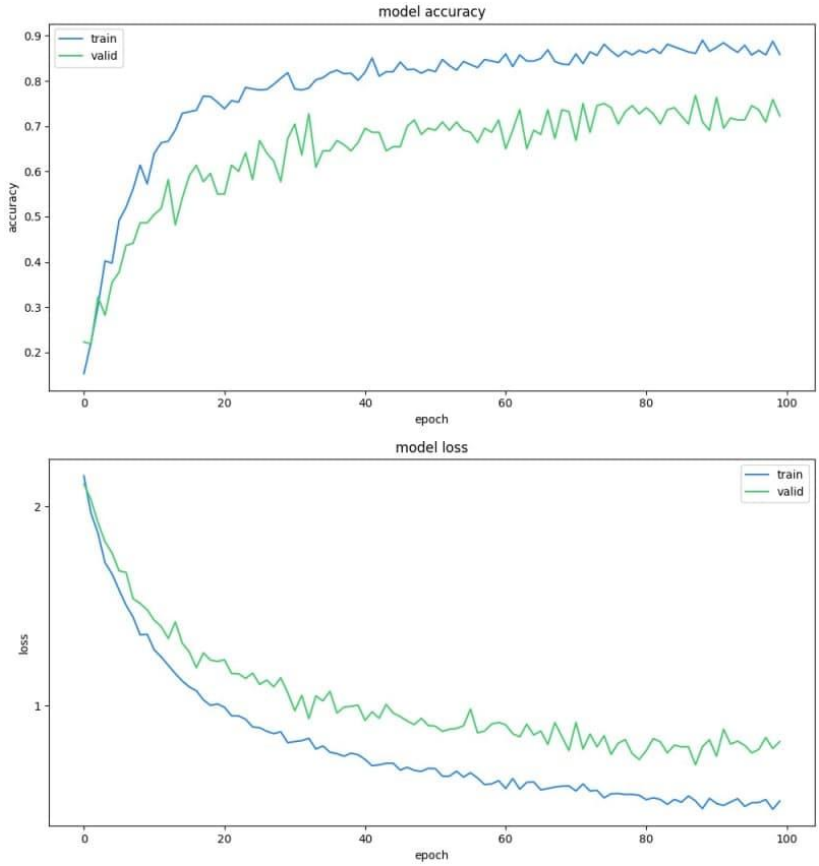
Como parámetros de este entrenamiento se definió que, se iteraría por un máximo de 50 epochs, con un *batch size* de 16, tomando un entrenamiento de la red con ImageNet como pesos base para realizar transferencia de conocimiento y se iniciaría el proceso con un ratio de aprendizaje de $1x10^{-3}$. Se ilustra en la Figura 18. los resultados obtenidos a partir de esta prueba, donde tras 35 epochs el modelo alcanzó una precisión del 99.5%, adicionalmente su tamaño es comparable al de MobileNetV1 y MobileNetV2, ocupando 37.5 MB en formato H5, disminuyendo a 12.3 MB al convertirse a formato TFLITE y finalmente 3.2 MB en formato KMODEL. El script de este entrenamiento puede encontrarse aquí.

La segunda herramienta de entrenamiento es MaixHub, una página web desarrollada por Sipeed para el entrenamiento de modelos compatibles con los dispositivos de la familia del chip K210, si bien no permite una mayor personalización en los parámetros del entrenamiento que se

realiza, se presenta como una alternativa de fácil uso con la que se pueden obtener resultados rápidos.

Figura 19.

Precisión del modelo entrenado en la plataforma MaixHub.



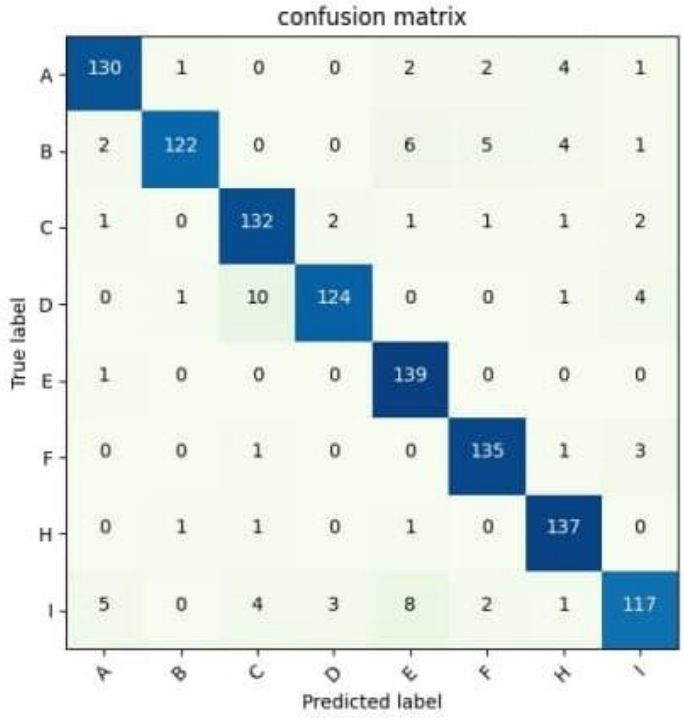
Esta plataforma entrena por defecto todos los modelos con la arquitectura YOLO y realiza de forma automática la conversión a formato KMODEL (formato específico para el Kendryte K210), por lo que las métricas que genera se calculan sobre el modelo listo para ser cargado en la tarjeta. Para su uso debe seleccionarse el tipo de modelo a entrenar, clasificador o detector de objetos, y cargar la base de datos con las imágenes para el entrenamiento.

En la Figura 19. se muestran las gráficas generadas por la plataforma MaixHub, en estas puede observarse que el modelo se entrenó por 100 *epochs*, logrando una precisión cercana al 86%

para el conjunto de entrenamiento, mientras que para el de validación llegó alrededor del 71%, con un peso de 1.8 MB en formato KMODEL.

Figura 20.

Matriz de confusión producto del entrenamiento de algunas categorías de la base de datos.



En la matriz de confusión de la Figura 20.

Matriz de confusión producto del entrenamiento de algunas categorías de la base de datos. se destaca que, a pesar de que la precisión mostrada por la gráfica de la Figura 19. se encuentra alrededor del 70%, durante la prueba de inferencia realizada con el set de validación se logró un promedio de 92.2% de precisión en la detección de las letras entre la A y la I.

3.5 Implementación en el sistema embebido

La implementación de un algoritmo entrenado en el sistema embebido puede sintetizarse en 3 pasos. Primero se debe cargar MicroPython en la tarjeta, el cual permitirá la lectura y

ejecución del modelo, posteriormente se debe realizar la conversión del modelo a formato KMODEL ya que este es el único que puede procesar la Sipeed Maix Bit y para finalizar se encuentra la carga del modelo en la memoria de la tarjeta.

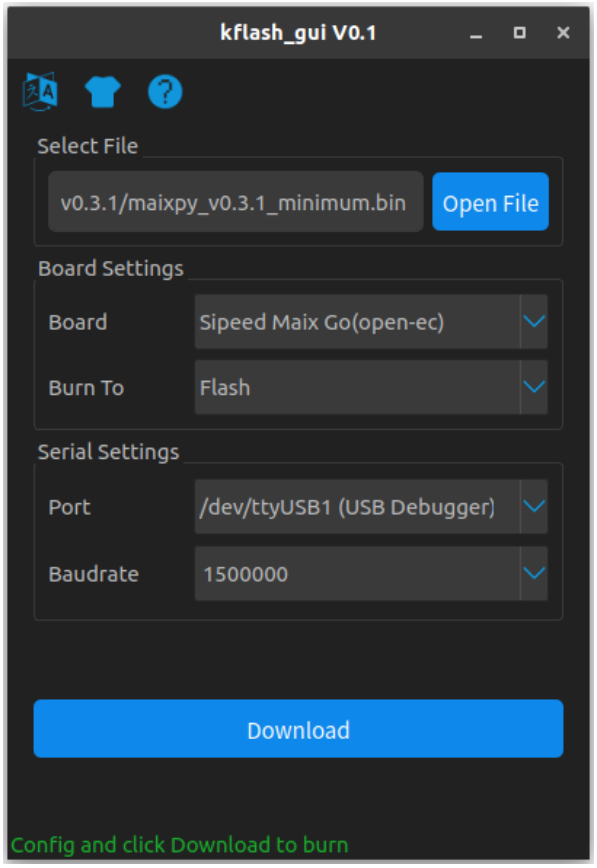
3.5.1 Carga del *firmware*

La Sipeed Maix Bit opera con un firmware basado en MicroPython llamado MaixPy, el cual cuenta con herramientas que le permiten cumplir con una gran variedad de tareas. Sin embargo, dados los limitados recursos con los que cuenta la tarjeta, Sipeed ofrece en su plataforma MaixHub una herramienta para compilar MaixPy únicamente con las características que desee el usuario; de esta forma es posible optimizar la cantidad de memoria utilizada por el firmware, permitiendo así que se puedan cargar posteriormente modelos de mayor tamaño.

Ya que para esta aplicación únicamente se requiere utilizar la funcionalidad de procesamiento de redes neuronales se compiló la versión mínima del firmware con IDE, la cual incluye la compatibilidad con la cámara y pantalla LCD del dispositivo. La carga de este en la tarjeta se realiza con la herramienta Kflash V1.6 tal como se indica en la documentación de MaixPy (Sipeed, 2018). En la Figura 21. se puede observar la interfaz de Kflash, para realizar la carga en la tarjeta se debe seleccionar el archivo a utilizar, la referencia de la tarjeta, la memoria en la que se va a cargar el archivo (flash o SD), el puerto al que se encuentra conectada la tarjeta al computador y finalmente el baud rate con el que se van a transmitir los datos; al seleccionar la opción *Download* Kflash se encargará de realizar la instalación del *firmware* de forma automática.

Figura 21.

Interfaz de la herramienta KFLASH.



Nota. Adaptado de "Kflash_Gui", por Sipeed, 2019, GitHub (https://github.com/sipeed/kflash_gui) (Sipeed, 2019).

3.5.2 Conversión del modelo

Como se indicó previamente, la Sipeed Maix Bit únicamente puede procesar modelos de red neuronal en su formato particular, por lo que para los casos en los que no se entrene el modelo a utilizar a través de MaixHub Sipeed provee una herramienta llamada Maix Toolbox (Sipeed, 2019), compuesta de múltiples scripts basados en NNCASE un compilador para aceleradores de IA desarrollado por Kendryte (Kendryte, 201), para realizar conversiones de un modelo entre distintos formatos desde H5 propio de Keras hasta KMODEL.

Desde la perspectiva del usuario la conversión es un proceso sencillo que consiste en hacer un llamado uno a uno, desde el CMD de Ubuntu, a los scripts requeridos para llegar al formato deseado, es importante destacar que se requiere de una muestra de la base de datos para realizar la cuantización del modelo. Partiendo de un entrenamiento realizado Keras se debe seguir los pasos indicados en la Figura 22. para transformar el modelo a formato KMODEL.

Figura 22.

Proceso de conversión de un modelo de Keras a formato KMODEL.

Es importante destacar en este punto que la transformación del modelo consiste un proceso de cuantización y compresión, lo que implica que la precisión lograda originalmente durante los entrenamientos de prueba se puede ver afectada de forma notoria; esto puede notarse con claridad en la Figura 19., donde se observa una disminución notable en la precisión del modelo, ya que esta gráfica corresponde al rendimiento calculado por MaixHub para el modelo en formato KMODEL.

3.5.3 Carga del modelo a la tarjeta

Teniendo la Sipeed Maix Bit con el framework correcto instalado y habiendo convertido el modelo entrenado al formato compatible con la tarjeta, solo resta realizar la carga de este último para su ejecución. Este paso puede realizarse de dos formas distintas, subiendo el modelo a través de Kflash al igual que se hizo para la instalación del firmware, indicando una dirección de memoria con la cual posteriormente se pueda hacer el llamado del modelo para su ejecución (típicamente se usa 0x300000), lo cual almacena el modelo en la memoria FLASH del dispositivo. Por otro lado, la alternativa usada en este proyecto fue cargar el modelo manualmente en una tarjeta SD,

en cuyo caso el llamado al modelo se realizó indicando la ruta donde fue almacenado dentro de la SD en lugar de una dirección de memoria.

MaixPy cuenta con una variedad de scripts de ejemplo que pueden ser usados como base para la ejecución del modelo, uno de estos ejemplos fue adaptado para el clasificador de lenguaje de señas. Este script puede verse aquí.

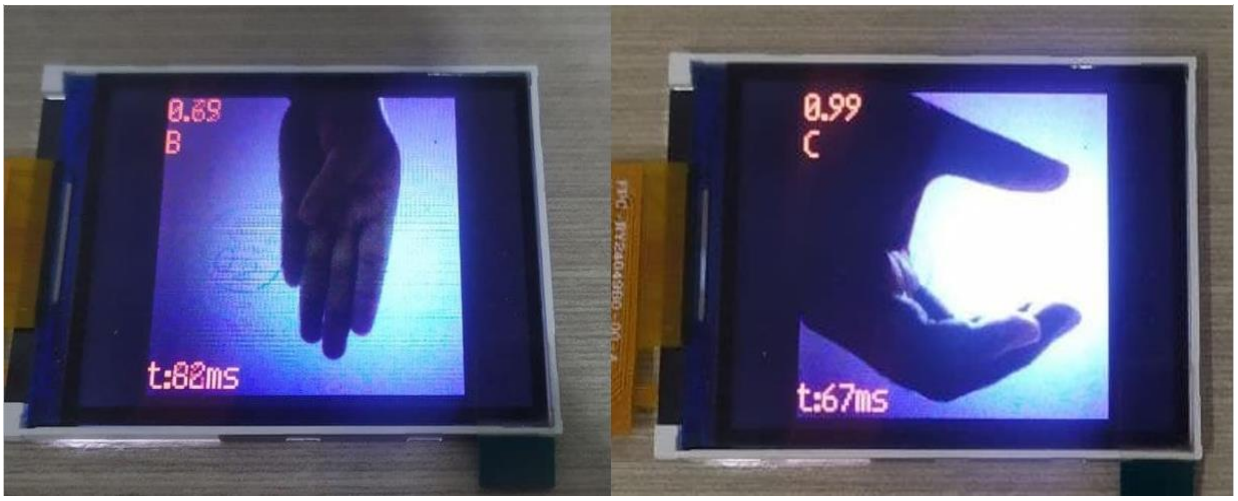
4. Resultados

Debido a las limitaciones de memoria que sufre la tarjeta finalmente se implementó el modelo entrenado a través de MaixHub, ya que no fue posible cargar en la tarjeta un modelo con un peso superior a 2 MB. Luego convertirse a KMODEL, el modelo de aXeleRate ocupaba 3.2 MB mientras que el de MaixHub tan solo 1.8 MB.

En la Figura 23. pueden observarse un par de ejemplos de los resultados generados al detectar en tiempo real gestos realizados frente a la cámara de la Sipeed Maix Bit. Si bien la precisión con la que predice la categoría puede variar dependiendo de las condiciones bajo las que la cámara detecta la mano y la forma en que la mano replique el gesto, el sistema logra diferenciar correctamente y de forma consistente las distintas categorías para las que fue entrenado.

Figura 23.

Capturas de las predicciones realizadas por el modelo de red neuronal cargado en el sistema embebido.



Se destaca también que los tiempos de respuesta para cada detección se encuentran en el orden de los milisegundos, el cual es un rango aceptable para esta aplicación ya que podrán reconocerse distintos gestos de forma fluida a medida que el usuario vaya articulando distintas letras con sus manos.

5. Conclusiones

La construcción una base de datos y su posterior liberación en un repositorio en línea constituye un aporte a futuros proyectos de investigación en el área de la inteligencia artificial, relacionados con la LSC.

Se demostró que es posible entrenar con esta base de datos, modelos capaces de reconocer gestos de la LSC, con un nivel de precisión superior al 90%.

Gracias a su núcleo Kendryte K210, la Sipeed Maix Bit es un dispositivo potente con la capacidad procesar redes neuronales convolucionales además de realizar detección y clasificación en tiempo real, posicionándose como una de las alternativas de bajo costo más poderosas disponibles en el mercado.

6. Recomendaciones y trabajo futuro

Con miras a mejorar los resultados obtenidos en este proyecto se deberá explorar nuevas alternativas en cuanto a arquitecturas y métodos de entrenamiento implementados, así como otros dispositivos, como por ejemplo la Maix-II, de forma que se pueda comparar sus desempeños y así encaminar futuros desarrollos hacia soluciones cada vez más optimas.

El complementar la base de datos es de vital importancia para mejorar su desempeño, añadiendo una mayor variedad de imágenes dentro de cada categoría. De esta forma podrá depender en menor medida de entrenamientos previos con otras bases de datos para alcanzar la precisión deseada.

Referencias Bibliográficas

Canaan Inc. (2019). K210 Datasheet. Obtenido de https://s3.cn-north-1.amazonaws.com.cn/dl.kendryte.com/documents/kendryte_datasheet_20181011163248_en.pdf

Cruz, M. (2008). Gramática de la lengua de señas mexicana. *(Tesis doctoral).* Colegio de México, México, D.F.

David Rasamoelina, A., Adjailia, F., & Sincák, P. (2018). A Review of Activation Function for Artificial Neural Network. *IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*. doi:10.1109/IECBES.2018.8626714

Flores-Mangas, F. (2014). *CSC320W: Introduction to Visual Computing.* Obtenido de Department of Computer Science University of Toronto: https://www.cs.toronto.edu/~mangas/teaching/320/slides/CSC320L05.pdf

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. doi:arXiv:1512.03385v1

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., . . . Adam, H. (2017). MobielNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. doi:arXiv:1704.04861v1

Instituto Nacional de Sordos (INSOR). (2006). *Diccionario Básico de la Lengua de Señas Colombiana.* Bogotá.

Jeyanthi, S., & Subadra, M. (2014). Implementation of Single Neuron Using Various Activation Functions With FPGA. *IEEE International Conference on Advanced Communication Control and Computing Technologies* .

Kaiming, H., Xiangyu, Z., Shaiqing, R., & Jian, S. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *2015 IEEE International Conference on Computer Vision*. doi:10.1109/ICCV.2015.123

Kendryte. (1 de Abril de 201). *NNCASE*. Obtenido de GitHub: https://github.com/kendryte/nncase

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (1998). Efficient BackProp. *Springer, 7700*. doi:https://doi.org/10.1007/978-3-642-35289-8_3

Lynch, P., & Horton, S. (1997). *Yale Style Manual-Graphics for the Web*. Recuperado el 21 de 03 de 2021, de Web Style Guide: https://webstyleguide.com/wsg1/graphics/display_primer.html

Maas, A., Hannun, A., & Ng, A. (2013). Rectifier Nonlinearities Improve Neural Network Acoustic Models. *ICML*.

Maslov, D. (28 de Septiembre de 2019). *aXeleRate*. Obtenido de GitHub: https://github.com/AIWintermuteAI/aXeleRate

Oviedo, A. (09 de 2013). *Cultura Sorda*. Obtenido de https://cultura-sorda.org/colombia-atlas-sordo/

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. doi:arXiv:1506.02640v5

Saha, S. (15 de 12 de 2018). *Towards Data Science*. Obtenido de A comprenhensive guide to convolutional neural networks - the ELI5 way: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetsV2: Inverted Residuals and Linear Bottlenecks. *The IEEE Conference on Computer Vision and Pattern Recognition*. doi:arXiv:1801.04381v4

Seeed Studio. (2021). *Seeed Studio*. Obtenido de Sipeed MAix BiT for RISC-V AI+IOT: https://www.seeedstudio.com/Sipeed-MAix-BiT-for-RISC-V-AI-IoT-p-2872.html

Sipeed. (2018). *Update MaixPy firmware*. Obtenido de Sipeed Wiki: https://wiki.sipeed.com/soft/maixpy/en/get_started/upgrade_maixpy_firmware.html

Sipeed. (5 de Mayo de 2019). *Kflash Gui*. Obtenido de Github: https://github.com/sipeed/kflash_gui

Sipeed. (31 de Mayo de 2019). *Maix Toolbox*. Obtenido de GitHub: https://github.com/sipeed/Maix_Toolbox

Squartini, S., Hussain, A., & Piazza, F. (2003). Preprocessing Based Solution for the Vanishing Gradient Problem in Recurrent Neural Networks. *Proceedings of the 2003 International Symposium on Circuits and Systems*. doi:10.1109/ISCAS.2003.1206412.

Stanford Vision Lab. (2020). *ImageNet*. Obtenido de https://image-net.org/

TeX - LaTeX Stack Exchange. (2020). Obtenido de https://tex.stackexchange.com/questions/522118/visualizing-matrix-convolution

Universidad de Sucre. (17 de 11 de 2015). *1er Seminario - taller en lengua de señas colombiana*. Recuperado el 27 de 03 de 2021, de Fono Al Día: http://fonoaldia.unisucre.edu.co/2015/11/1er-seminario-taller-en-lengua-de-senas.html

Wang, C.-F. (13 de Agosto de 2018). *A Basic Introduction to Separable Convolutions*. Recuperado el 29 de Junio de 2021, de Towards Data Science: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

Wang, X., Han, Y., Leung, V., Niyato, D., Yan, X., & Chen, X. (2020). Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Communications surveys and tutorials*, 869-904. doi:10.1109/COMST.2020.2970550

Wood, T. (s.f.). *DeepAI*. Recuperado el 20 de 05 de 2021, de https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function