UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA DE INGENIERÍA ELECTRÓNICA

DESARROLLO DE UN DETECTOR DEL AVE HUIRACCHURO EN CULTIVOS MEDIANTE EL RECONOCIMIENTO DE IMÁGENES BASADO EN DEEP LEARNING

Trabajo de titulación previo a la obtención del

Título de Ingeniero Electrónico

AUTOR: Juan Andrés Lapo Molina

TUTOR: Carlos Augusto Cuichán Morales

Quito-Ecuador

2023

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Yo, Juan Andrés Lapo Molina, con documento de identificación N.º 1754088696 manifiesto que:

Soy el autor y responsable del presente trabajo, y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total oparcial el presente trabajo de titulación.

Quito, 31 de julio del año 2023

Atentamente

Juan Andrés Lapo Molina

1754088696

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DETITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Yo, Juan Andrés Lapo Molina, con documento de identificación N.º 1754088696, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del Proyecto Técnico: "Desarrollo de un detector del ave Huiracchuro en cultivos mediante el reconocimiento de imágenes basado en Deep Learning", el cual ha sido desarrollado para optar por el título de: Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemosla entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, 31 de julio del año 2023

Atentamente,

Juan Andrés Lapo Molina

1754088696

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Carlos Augusto Cuichán Morales, con documento de identificación N.º 1714389721, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrolladoel trabajo de titulación: DESARROLLO DE UN DETECTOR DEL AVE HUIRACCHURO EN CULTIVOS MEDIANTE EL RECONOCIMIENTO DE IMÁGENES BASADO EN DEEP LEARNING, realizado por Juan Andrés Lapo Molina, con documento de identificación N.º 1754088696, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 31 de julio del año 2023

Atentamente,

Ing. Carlos Augusto Cuichán Morales MsC.

1714389721

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

Este trabajo de tesis presenta el desarrollo de un sistema de detección en tiempo real del ave Huiracchuro en cultivos utilizando Deep learning. Se utilizo la red neuronal pre-entrenada MobileNetv2, adaptada para detectar específicamente esta especie de ave. La red fue modificada eliminando la capa de clasificación final y añadiendo una nueva capa para identificar la clase “Huiracchuro” al detectar mediante la cámara de un dispositivo móvil.

La implementación se realizó en un aplicativo web, permitiendo a los usuarios acceder a la red y utilizar la cámara de sus dispositivos para la detección en tiempo real. Se empleo NGROK para establecer la comunicación entre el aplicativo web y el servidor local que aloja la red neuronal entenada.

El sistema fue evaluado en pruebas de campo en el distrito metropolitano de Quito, obteniendo resultados prometedores. La red neuronal logro una efectividad superior al 70% al detectar el ave Huiracchuro, siempre que este ocupara al menos un área de 40x40 pixeles en una imagen de 224x224 pixeles.

Este trabajo representa un avance significativo en la detección y reconocimiento de aves en cultivos mediante Deep learning. Su enfoque en tiempo real y la disponibilidad en dispositivos móviles ofrecen una solución practica y accesible para los agricultores. Además, la utilización de la red neuronalMobileNetv2, la modificación de la red con Python y el entrenamiento en Google Colaboratory demuestran la eficacia y la viabilidad de la metodología empleada.

En conclusión, este sistema de detección en tiempo real del ave Huiracchuro en cultivos brinda una herramienta valiosa para la conservación de plantas, la protección de los cultivos y el ingreso económico de los agricultores. Con una efectividad superior al 70%, este sistema demuestra su potencial para su aplicación en campo.

Palabras clave: Huiracchuro, Mobilenetv2, Python, Google Colaboratory, NGROK, identificación en tiempo real.

ABSTRACT

This thesis focuses on the development of a real-time detection system for the Huiracchuro bird in crops using deep learning techniques. The pre-trained MobileNetv2 neural network was utilized, which was modified to specifically detect this bird species. The modification involved removing the final classification layer and adding a new layer to identify the "Huiracchuro" class when it is detected through the camera of a mobile device.

The implementation was carried out in a web application, enabling users to access the network and use their device cameras for real-time detection. NGROK was employed to establish communication between the web application and the local server hosting the trained neural network.

The system was evaluated through field tests conducted in the metropolitan district of Quito, yielding promising results. The neural network achieved an accuracy of over 70% in detecting the Huiracchuro bird, provided that it occupied a minimum area of 40x40 pixels in a 224x224 pixel image.

This work represents a significant advancement in the real-time detection and recognition of birds in crops using deep learning. The real-time focus and availability on mobile devices offer a practical and accessible solution for farmers. Moreover, the utilization of the MobileNetv2 neural network, Python-based network modification, and training on the Google Colaboratory platform demonstrate the effectiveness and feasibility of the methodology employed.

In conclusion, this real-time detection system for the Huiracchuro bird in crops provides a valuable tool for plant conservation, crop protection, and the economic income of farmers. With an accuracy exceeding 70%, this system showcases its potential for field applications.

Keys word: Huiracchuro, Mobilenetv2, Python, Google Colaboratory, NGROK, real time identification.

INTRODUCCIÓN

En el Distrito Metropolitano de Quito, los residentes disponen de terrenos de pequeñas dimensiones destinados al cultivo de diversos productos para el consumo familiar. Entre ellos se encuentran el maíz, el frijol, la zanahoria, la col, las arvejas y algunos árboles frutales.

Entre los desafíos que enfrentan los pequeños productores agrícolas se encuentra el ave Huiracchuro, una especie emblemática de la urbe quiteña. Esta ave, que se alimenta principalmente de granos, representa una de las mayores amenazas para los agricultores, ya que ataca a las plantas de maíz en el momento en que comienzan a dar sus primeros granos.

Por lo tanto, la detección de esta ave se vuelve crucial para preservar tanto las plantas como el ave misma, y garantizar el ingreso económico de los agricultores.

Para la detección de esta ave en particular, se emplea un modelo de red neuronal basado en el aprendizaje profundo (Deep Learning), el cual es modificado y entrenado utilizando el lenguaje de programación Python y el servicio en la nube de Google Colaboratory. Esta red modificada extrae las características del ave utilizando un conjunto de imágenes de entrenamiento y pruebas.

Después de realizar el entrenamiento y las pruebas correspondientes, se lleva el modelo a la fase de pruebas en campo. Para ello, se desarrolla una aplicación web que permite la interacción del usuario y proporciona acceso a la cámara de sus dispositivos electrónicos o celulares.

La conexión entre el dispositivo electrónico y la aplicación web, que se encuentra alojada en un servidor local, se logra gracias a la herramienta NGROK, la cual proporciona una dirección web válida por un tiempo determinado para acceder a la aplicación y poner a prueba la red neuronal.

Las pruebas se realizan en el Barrio Santa Rosa de la parroquia de Puembo y en la parroquia de Puéllaro. Después de llevar a cabo 542 pruebas de campo, se obtiene un resultado con una efectividad mayor al 70%, siempre y cuando el ave ocupe un área igual o mayor a 40x40 píxeles en la imagen de entrada, la cual tiene un tamaño de 224x224 píxeles.

CAPÍTULO 1

ANTECEDENTES

En el presente capítulo se describe el planteamiento del problema junto a sus causas, características, justificación, soluciones y los objetivos a cumplir en el proyecto.

1.1 Planteamiento del problema

En el Distrito Metropolitano de Quito (DMQ) el cual posee un clima variante entre soleado, lluvioso y templado, favorece a la siembra de diferentes cultivos, de los cuales destacan los granos como lo son: el maíz, frejol, arvejas, entre otros. Alimentos propicios para las aves granívoras las cuales pueden provocan una pérdida para el agricultor, está perdida puede ser mínima como también puede resultar en una pérdida total del cultivo. Los pequeños agricultores del DMQ no cuentan con el poder monetario para colocar mallas anti-aves en sus sembríos, por lo cual optan por el uso de fungicidas, pesticidas o incluso al uso de armas, es por eso por lo que es necesario la detección del ave para de esa manera preservar los cultivos y el ingreso económico que percibe el agricultor.

1.2 Justificación

Desarrollar un detector del ave Huiracchuro para la conservación del maíz, que consta de una IA especializada en detección y clasificación de objetos, es un hecho que puede ayudar al dueño del huerto de maíz, para que de esta manera pueda identificar correctamente al ave y de esa manea ahuyentarla y conservar sus sembríos

1.3 Objetivos

1.3.1 Objetivo general

Desarrollo de un sistema inteligente para la detección del ave Huiracchuro en cultivos mediante Deep Learning.

1.3.2 Objetivos específicos

- Indagar sobre modelos de IA para la clasificación de objetos mediante repositorios universitarios y base de datos de artículos académicos.

- Incorporar atributos (morfología y rasgos característicos del ave Huiracchuro) hacia un modelo de Deep Learning para la detección del ave, mediante el servicio de Google Colaboratory.
- Verificar el funcionamiento del sistema inteligente de detección del ave Huiracchuro para su validación desde una página web, mediante pruebas experimentales de campo.

1.4 Metodología

1.4.1 Metodología analítica: Mediante este método se analiza las diferentes variables físicas como lo son: color, forma, tamaño del pico, tamaño del ave, en la plantilla neuronal planteada. Las imágenes por emplearse se obtienen de una base de datos disponible en la web.

1.4.2 Metodología inductiva: Mediante este método se realiza la observación de los cambios dados en la red neuronal al momento de ser modificada con Python y la precisión que está tiene al momento de detectar el ave.

1.4.3 Metodología experimental: Por medio de las pruebas realizadas se verifica el correcto funcionamiento de la red neuronal mediante un pequeño panel de usuario realizado en Python, donde se podrá colocar la imagen que se desea analizar. De la misma forma se van cambiando los parámetros empleados en la detección para determinar los más idóneos y emplearlos en una página web.

CAPÍTULO 2
FUNDAMENTACIÓN TEÓRICA

En el capítulo 2 se tratan conceptos teóricos básicos, basados en la tecnología de Deep Learning basado en IA sobre la cual se establece una solución al problema planteado, se analizan los elementos primarios en los cuales se fundamenta la realización de la investigación.

2.1 Huiracchuro

El ave *Pheuticus chrysogaster* o más conocido como Huiracchuro, es una de las aves más reconocidas en el DMQ, ave de fácil reconocimiento por su canto y sus distinguidos colores. Los machos que miden en promedio 20 cm presentan un color amarillo intenso en gran parte de su cuerpo y alas de color negro con betas blancas, acompañados de un pico grueso de color plata.(*Huiracchuro | Quito, Hábitat Silvestre*, n.d.)

En la figura 2.1 se aprecia al Huiracchuro en su hábitat.

Figura 2.1. Huiracchuro.

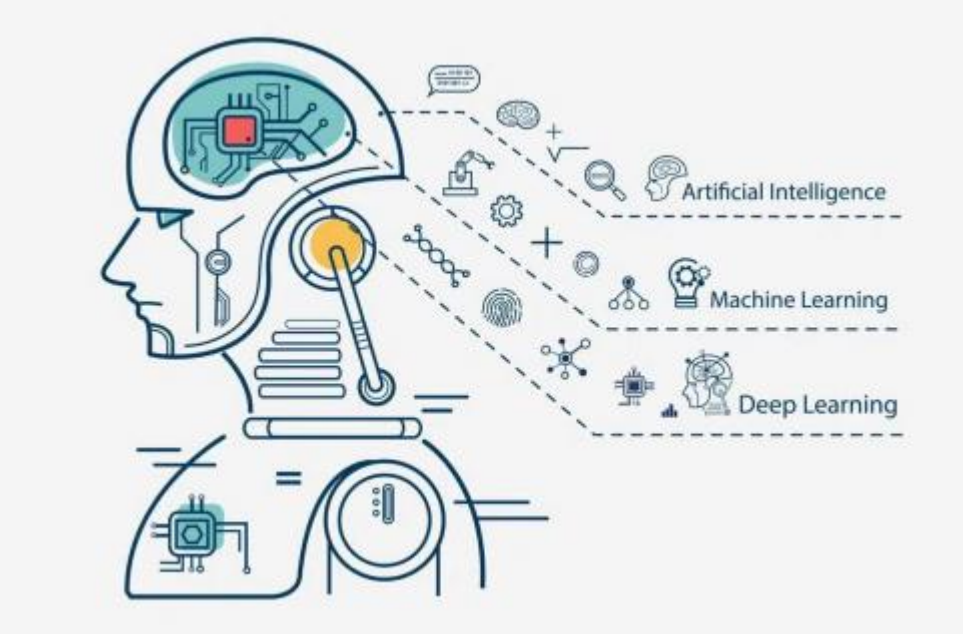
Huiracchuro, Fuente: (iNaturalistEC, 2016)

2.2 Inteligencia Artificial

La inteligencia artificial (IA) ha despertado un gran interés en la actualidad, siendo considerada una de las ramas destacadas en el campo de las ciencias de la computación. Su amplio espectro de aplicaciones ha capturado la atención de numerosos investigadores. Dentro de esta área, la búsqueda de mecanismos que permitan comprender la inteligencia y desarrollar modelos y simulaciones ha motivado a muchos científicos. Entre las diversas ramas de la IA, se encuentran el Machine Learning y el Deep Learning.(Ponce et al., 2014)

En la Figura 2.2 se puede apreciar una representación artística de lo que es IA.

Figura 2.2. Inteligencia Artificial



Representación artística de Inteligencia Artificial, Fuente: (Hernández, 2022)

2.3 Modelos de identificación

Estos dos modelos de reconocimiento son el aprendizaje automático (ML) y el aprendizaje profundo (DL), que utilizan diferentes tipos de algoritmos, como la regresión lineal o logística, para reconocer patrones en los datos y hacer inferencias que imitan el conocimiento humano. El modelado de IA se trata de crear un proceso de toma de decisiones que consta de tres pasos básicos:

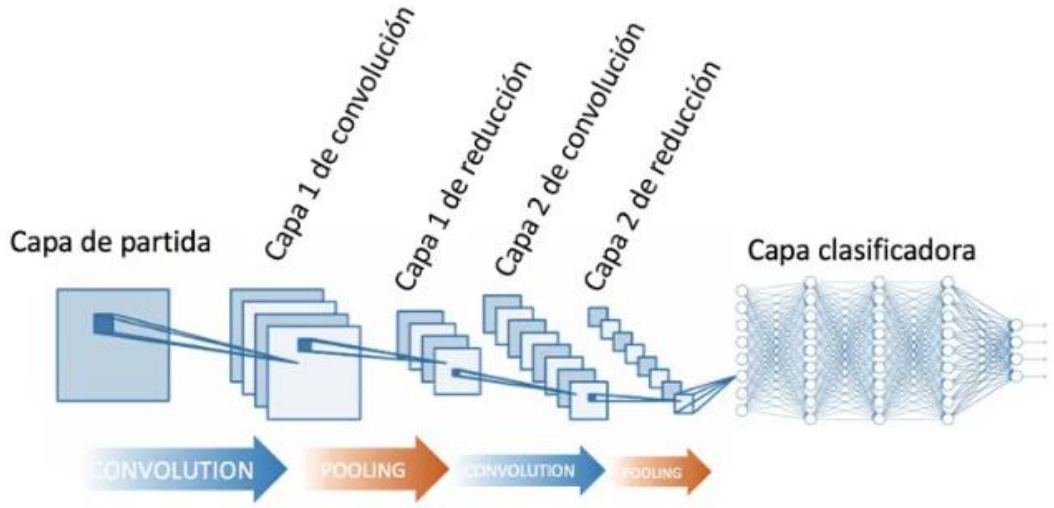
- **Modelos**: el primer paso es crear un modelo de IA usando algoritmos complejos o capas de algoritmos que interpretan datos y toman decisiones basadas en esos datos.
- **Entrenamiento**: entrenar los modelos de IA en grandes cantidades de datos en un ciclo de prueba iterativo y, verifique la precisión de los resultados.

- **Inferencia**: este paso implica la implementación del modelo de IA cuando se usa en el mundo real, donde el modelo extrae conclusiones lógicas de los datos disponibles.

Entre los diferentes modelos de DL y sus algoritmos de aplicación, se puede dividirlos en varias categorías, por ejemplo: CNN (Convolutional Neural Network). ResNet (Residual Network), RNN (Recurrent Neural Network), etc.

En la Figura 2.3 se puede apreciar una representación gráfica de una CNN.

Figura 2.3. Representación gráfica de una CNN.



Red Neuronal Convolucional, Fuente: (Diego Calvo, 2017)

2.4 Deep Learning

La técnica de DL incluye como parte de su procedimiento una extracción de características. En lugar de basarse en la lógica lineal, el aprendizaje profundo se sustenta en teorías acerca de cómo funciona el cerebro humano. El programa está construido por capas anidadas de nodos interconectados entre ellos. Después de cada ciclo, la red aprende y reajusta las conexiones o pesos entre las neuronas para minimizar el error.(Pérez & Hornillo, 2021)

2.5 Servicio de Google Colaboratory

Es un servicio en nube que permite programar y ejecutar código de Python para la creación, modificación o entrenamiento de diferentes tipos de redes neuronales.(Colaboratory - Colaboratory, n.d.). Dicho servicio consta con varias ventajas, entre ellas la más relevante para este proyecto es el Acceso a GPUs (Unidad de Procesamiento Gráfico) sin coste adicional. Lo que permite entrenar redes neuronales en menor tiempo y con mayor eficiencia.

Figura 2.4. Logotipo de Google Colaboratory.

Google Colab con Python, Fuente: (Gustavo Juantorena, 2020)

2.6 Transmisión de datos vía Web

La transmisión de la imagen o en este caso del video en tiempo real se realiza mediante un servidor local HTTP proporcionado por Python. Esto permite desplegar la página web desde el computador usando el comando “localhost: puerto”. Para tener acceso a la página web desde cualquier dispositivo, se utiliza la aplicación NGROK.

NGROK es una aplicación multiplataforma que permite a los desarrolladores migrar servidores de desarrollo locales a internet con el menor esfuerzo. El software permite que su servidor web sea alojado en un subdominio de ngrok.com

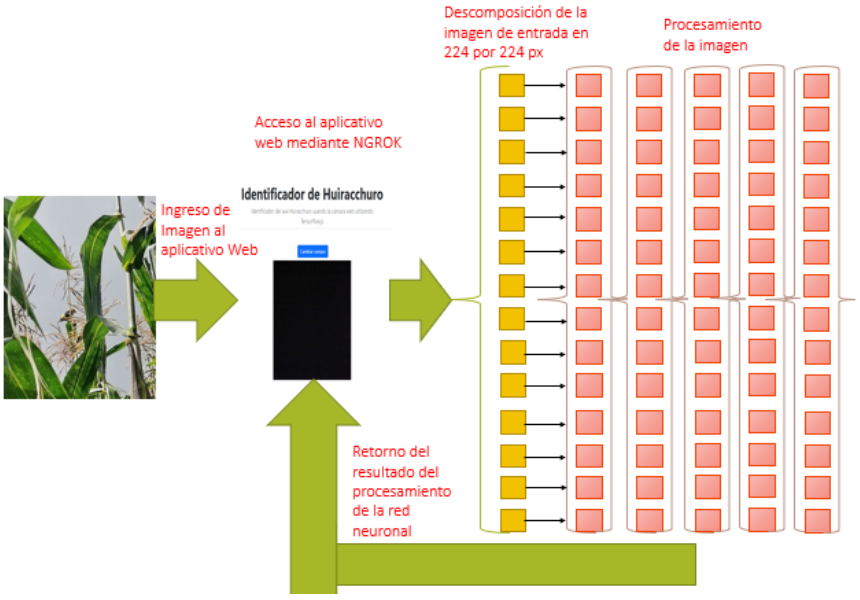
CAPÍTULO 3

DESARROLLO DEL ALGORITMO

En el presente capítulo, se describe el funcionamiento de la aplicación web, se definen las características del ave para su detección, el modelo de red neuronal a implementar. Además, se detalla las modificaciones realizadas en el modelo, su entrenamiento y su validación con una base de datos para la detección del ave Huiracchuro, además de forma detallada el funcionamiento de manera general mediante sus respectivos diagramas de bloques que contienen la comunicación del usuario al servidor que contiene la red neuronal entrenada, la manera en que se recolectan y envían datos, el manejo de la aplicación celular y la alerta del sistema.

A continuación, se muestra en la figura 3.1 un diagrama que ilustra el funcionamiento del aplicativo web. El proceso comienza con la captura de la imagen a través de la cámara de un dispositivo electrónico. Luego, la imagen se envía al aplicativo web a través de su interfaz. A continuación, la imagen se somete a procesamiento mediante una red neuronal. Una vez completado el procesamiento, la red neuronal envía una respuesta al aplicativo web. Por último, el aplicativo web presenta la respuesta recibida, la cual ha sido previamente procesada por el sistema.

Figura 3.1. Diagrama de funcionamiento del aplicativo web.



El diagrama muestra el funcionamiento del aplicativo web, Autor: Andrés Lapo.

3.1 Componentes del sistema

El sistema de detección del ave Huiracchuro se divide en tres partes fundamentales.

3.1.1 Dispositivo Electrónico

El dispositivo electrónico utilizado en este proyecto debe poseer una cámara y conexión a internet. En este caso, se emplea un teléfono celular que se conecta al aplicativo web, permitiendo el acceso a la cámara y la obtención de la imagen de entrada.

3.1.2 NGROK

NGROK facilita la conexión de cualquier dispositivo a nuestro servidor local, donde se encuentra alojado el modelo de red neuronal y la aplicación web. Proporciona una dirección web válida por una hora, permitiendo acceder al servidor y utilizar el sistema desde cualquier dispositivo.

3.1.3 Aplicativo Web

La interfaz de usuario (aplicativo web) es la plataforma a través de la cual el usuario puede interactuar con el sistema. Proporcionando la opción de cambiar la cámara en caso de que el usuario lo considere necesario. Además, se encarga de llamar al modelo de red neuronal previamente entrenado, facilitando la detección del ave Huiracchuro en tiempo real.

3.1.4 Red Neuronal

La red neuronal utilizada en el sistema es una red neuronal previamente entrenada para la detección del ave Huiracchuro. Esta red toma la imagen de entrada y, utilizando las características del ave presentes en la imagen, genera una respuesta positiva si se identifica como un Huiracchuro, o una respuesta negativa en caso contrario. La respuesta generada por la red neuronal es enviada al aplicativo web, donde se muestra al usuario de manera visual. De esta manera, el usuario puede obtener la información sobre la detección del ave Huiracchuro en tiempo real a través de la interfaz del aplicativo web.

3.2 Red Neuronal

En la actualidad existe una gran variedad de redes neuronales especializadas en diversas áreas. En el caso que nos concierne, nos enfocamos en el área de clasificación. A pesar de ello, los repositorios que albergan estas redes cuentan con un extenso catálogo para elegir. Por lo tanto, se utilizarán los siguientes parámetros:

- Red Neuronal Convolucional (CNN)
- Entrenamiento previo con ImageNet.

Las CNN son excelentes para la clasificación de imágenes debido a su capacidad para capturar características relevantes a través de capas convolucionales. Estas redes han demostrado un rendimiento sobresaliente en una variedad de tareas de clasificación de imágenes, como reconocimiento facial, detección de objetos y diagnóstico médico.

ImageNet es un set de datos de imágenes ampliamente utilizado en la investigación y desarrollo de la visión por computadora. Creado en 2009 por investigadores de la Universidad de Stanford y contiene millones de imágenes etiquetadas que abarcan una amplia variedad de categorías.

Las redes neuronales computacionalmente económicas, como MobileNet, están diseñadas para ser eficientes en términos del uso de recursos computacionales, como el tiempo de computación y la memoria requerida.

Esto significa que pueden ejecutarse en dispositivos con recursos limitados, como dispositivos móviles, sistemas integrados o incluso en la nube a un costo menor sin afectar significativamente el rendimiento de la red.

Entre las redes neuronales que cumplen con los parámetros mencionados se encuentran las siguientes opciones altamente recomendadas:

- MobilenetV1, ShuffleNet (1.5)
- ShuffleNet(x2)
- NasNet-A
- MnetV1 + SSDLite
- MnetV2 + SSDLite.

3.2.1 MobileNetV1

Es una arquitectura de red neuronal convolucional diseñada para tareas de clasificación de imágenes en dispositivos con recursos limitados. Su objetivo principal es lograr una alta eficiencia computacional al reducir drásticamente la cantidad de parámetros y la carga computacional en comparación con arquitecturas más profundas y pesadas, utilizando un componente básico denominado curvas resolubles en profundidad, que reemplaza las curvas estándar por curvas resolubles en profundidad y espacialmente. Esto permite modelos con menos pasos y parámetros mientras mantiene un rendimiento de clasificación competitivo.(Howard et al., n.d.-a)

MobileNetV1 se usa ampliamente en aplicaciones de visión por computadora en dispositivos móviles debido a su equilibrio entre eficiencia y precisión.

3.2.2 ShuffleNet (1.5)

Es una arquitectura de red neuronal convolucional conocida por su alta eficiencia y potencia informática legal. Su diseño se basa en la idea de utilizar una operación de "shuffle" para mejorar la eficiencia del aprendizaje profundo utilizando bloques de construcción llamados Shuffle Units, que consisten en una combinación de inversiones de grupo, operaciones de barajado y convoluciones 1x1.

Estos Shuffle Units le permiten que las características se mezclen entre diferentes grupos, lo que fomenta el intercambio de información y la representación rica en el modelo con menos parámetros. La arquitectura ShuffleNet (1.5) ha demostrado un rendimiento excelente en las tareas de clasificación de imágenes y requiere menos recursos computacionales que otras arquitecturas más pesadas.(Zhang et al., n.d.)

Su eficiencia y precisión hacen de ShuffleNet (1.5) una opción popular para aplicaciones de visión artificial en dispositivos con recursos limitados

3.2.3 ShuffleNet (x2)

Es una versión mejorada de ShuffleNet diseñada para mejorar aún más la eficiencia y el rendimiento. La arquitectura utiliza estrategias de agrupación y barajado de canales para reducir la carga computacional y la cantidad de parámetros sin comprometer la precisión de la clasificación.(Ma et al., n.d.)

ShuffleNet (x2) logra esto al combinar un controlador ShuffleNet más profundo y una relación de compresión más alta. Además, utiliza operaciones de agrupación de canales para agrupar canales de entrada para una mejor interacción y un intercambio de información más eficiente entre ellos. Esto da como resultado modelos más eficientes y compactados que se pueden implementar en dispositivos con recursos limitados sin perder rendimiento en las tareas de clasificación de imágenes.

3.2.4 MobileNetV1+SSDLite

Es una combinación de dos tecnologías, la red neuronal MobilenetV1 y Single Shot MultiBox Detecto (SSD) el cual es un algoritmo popular para la detección de objetos en imágenes. Utilizando MobilenetV1 como la base para la extracción de características de la imagen, que se utiliza luego en el algoritmo SSDLite para realizar la detección de varios objetos en tiempo real. Esta combinación es ampliamente utilizada en sistemas de seguridad, vigilancia, conducción autónoma, reconocimiento facial, entre otros.(Howard et al., n.d.-b)

3.2.5 MobileNetV2+SSDLite

MobileNetV2 SSDLite es una combinación poderosa y eficiente de arquitecturas de redes neuronales convolucionales para aplicaciones de detección de objetos en tiempo real. Conocido por su poder de procesamiento rápido y su bajo costo computacional, lo que lo hace ideal para dispositivos con recursos limitados, como los dispositivos móviles.(Sandler et al., n.d.)

Esta combinación permite la detección precisa y en tiempo real de múltiples objetos en imágenes y video, lo que la hace ideal para aplicaciones de seguridad, vigilancia, realidad aumentada y reconocimiento de objetos en entornos móviles.

En Tabla 1, se muestra la comparación de las redes neuronales descritas, todas especializadas en la clasificación de imágenes con la base de datos ImageNet. El propósito de esta comparación es evaluar la eficiencia de cada arquitectura neuronal en la solución de la tarea de identificación.

Las métricas que se muestran en Tabla 3.1 son:

- Top1: Representa el porcentaje de imágenes de ImageNet que el modelo clasifica correctamente.
- Params: Cantidad de parámetros que pueden ser entrenados en cada arquitectura de red neuronal.
- MAdd: Numero de operaciones requeridas para realizar un paso hacia adelante o a través de la red neuronal.
- CPU: Eficiencia computacional.

Tabla 3.1. Rendimiento en Mobilenetv2

Network	Top 1	Params	MAdd	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet(1.5)	71.5	3.4M	292M	-
ShuffleNet(x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MNet V1 + SSDLite	**72.0**	**3.4M**	**300M**	**75ms**
MNet V2 + SSDLite	**74.7**	6.9M	585M	**143ms**

Tabla comparativa de MobilNetv2 con otras redes neuronales, Fuente:MobileNetV2, Sandler M

En conclusión, la tabla muestra el gran rendimiento en la clasificación de imágenes y el bajo costo de cómputo que posee MobileNet v2 frente a otras redes neuronales, incluso su misma versión anterior.

3.3 MobileNetv2

Considerando que el objetivo principal es la detección del ave Huiracchuro en tiempo real y no en la localización precisa de objetos, la red a utilizar es MobilenetV2.

Al ser una red neuronal eficiente y precisa que puede realizar la tarea de clasificación de imágenes. Evitando así la sobrecarga computacional y la complejidad adicional que implica la detección y localización precisa ofrecida por MobileNetV1 + SSDLite y MobileNetV2 + SSDLite.

La arquitectura de Mobilenetv2 permite el ingreso de una imagen de 224px por 224px a color (a tres canales), dando lugar al análisis en profundidad de cada pixel, extrayendo y reconociendo líneas, curvas, patrones, colores para poder clasificar una imagen en una las 1001 categorías que posee.

La red consta de 32 filtros, seguidos de 19 cuellos de botella y finalmente capas de salida en la cual se encuentra la capa de clasificación.(Sandler et al., 2018)

Para finales didácticos se divide la red neuronal en tres grandes partes:

- Las 16 primeras capaz de la red para el procesamiento de la imagen
- Los 19 cuellos de botella residuales
- Capas de salida de la red neuronal

Los primeros 32 filtros se encuentran contenidos en la primera capa de convolución

llamada 2D Convolution la cual se conecta con las capas Batch Normalization, Clipped ReLU, 2D Grouped Convolution, Expanded Convolution, las cuales se repiten y conectan.

La Figura 3.2 muestra la disposición de las 16 primeras capas antes de continuar con los 19 filtros de botella residuales.

Figura 3.2. Disposición de las capas iniciales de mobilenetv2.

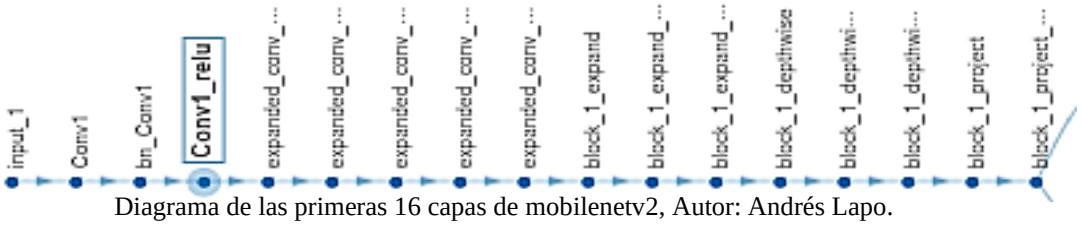


Diagrama de las primeras 16 capas de mobilenetv2, Autor: Andrés Lapo.

- Capa de convolución 2D: Se aplica un filtro de convolución deslizante a lo largo de las entradas verticales y horizontales mientras se calculan los pesos y el producto de los puntos de entrada, y finalmente se agrega un término de sesgo(*2-D Convolutional Layer - MATLAB*, n.d.). En esta capa se encuentran los 32 filtros cada uno con tamaño de 3x3, dando así paso a la primera red neuronal. En la Figura 3.3 se muestra la convolución de una imagen en la paleta de color rojo, este proceso se repite para cada canal de color RGB que componen las imágenes digitales, esta se desplaza de forma horizontal y vertical por toda la imagen. La Figura 3.4 muestra la finalización de la convolución construyendo una nueva matriz.

Figura 3.3. Imagen en proceso de convolución.

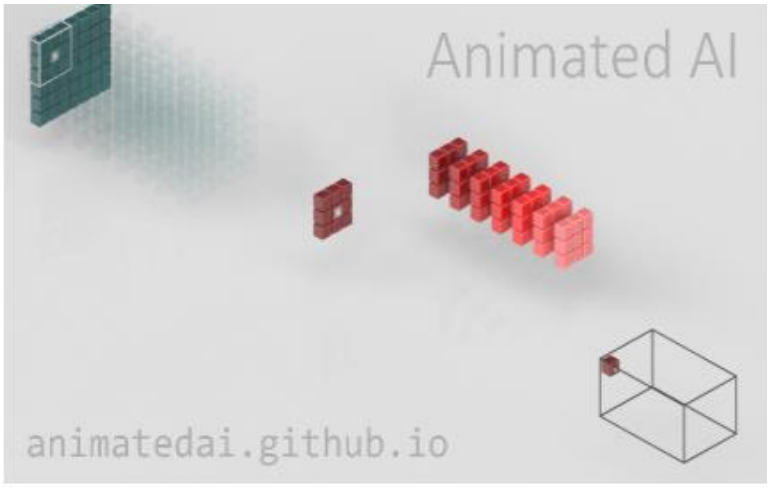


Imagen de la convolución de una imagen en una red neuronal, Fuente: Animated AI.

Figura 3.4. Imagen de la finalización del proceso de convolución.

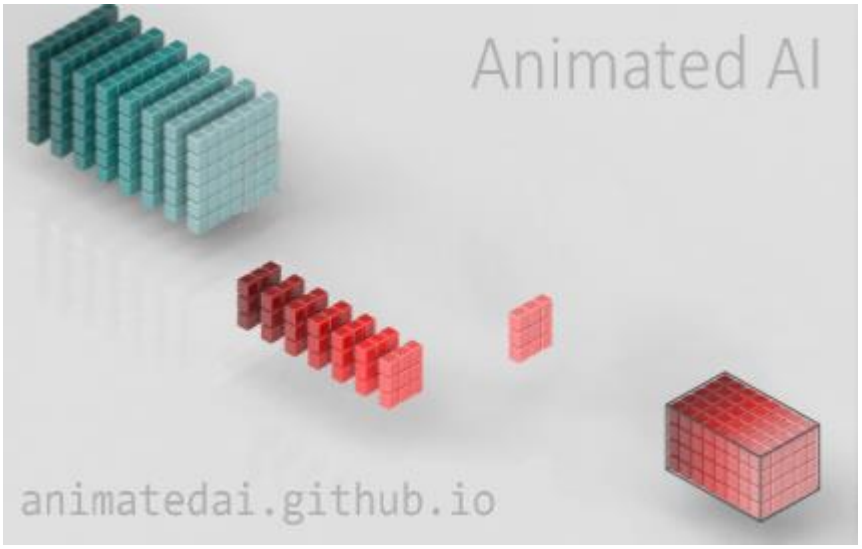
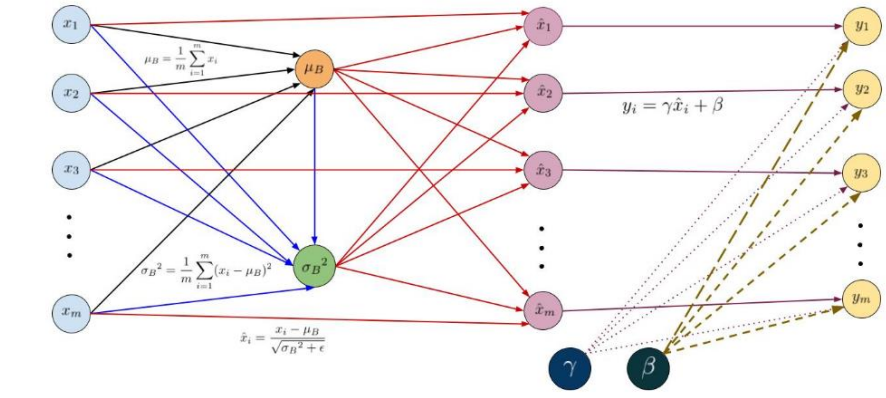


Imagen de la finalización de convolución, Fuente: Animated AI.

- Batch Normalization: El enfoque de Serge Iofe y Christian Szegedy para entrenar redes neuronales artificiales al reducir la cantidad de épocas requeridas para entrenar y estabilizar la normalización de otras capas de redes neuronales. El efecto de la normalización por lotes es evidente porque reduce el efecto del sesgo de covarianza interna que se produce cuando se entrena la red al variar la entrada y las distribuciones correspondientes de cada capa de la red neuronal(Santurkar et al., n.d.). En pocas palabras, batch normalization mitiga el cambio en las conexiones existentes entre las diferentes capas de la red neuronal con cada entrenamiento que se produce.

En la Figura 3.5 se muestra la ecuación de bacht normalization aplicada en la conexión de entrada con la capa siguiente.

Figura 3.5. Aplicación de la ecuación de batch normalization.

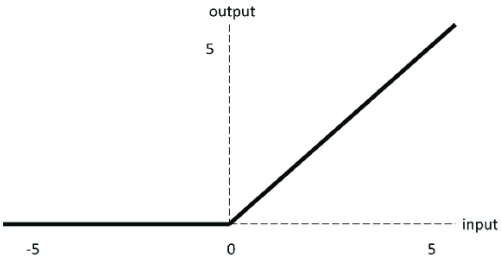


Ecuación batch normalization en conexión de redes, Fuente: Animated AI.

- Clipped ReLU layer: La función de activación ReLU transforma los valores introducidos anulando los valores negativos y dejando los valores positivos(Lecun et al., 2015) que no pasen el techo del umbral. Que en esta red es 6, esto crea conexiones más fuertes entre las diferentes capas de la red al descartar valores posibles y no absolutos.

La Figura 3.6 muestra la gráfica de la función de activación ReLU.

Figura 3.6. Función ReLU



Función de activación ReLU con umbral 5, Fuente: ResearchGate

- 2D Grouped Convolutional Layer: La capa de convolución por agrupación divide el resultado de la convolución anterior en grupos, aplicando filtros en cada uno de ellos. La aplicación de esta forma de convolución es disminuir el tiempo de procesamiento de la red neuronal, ya que puede lograr ser 9 veces más rápida que otras formas más tradicionales de convolución.(*2-D Grouped Convolutional Layer - MATLAB*, n.d.)

Para esta red neuronal, la capa de Groupped Convolution posee 32 grupos cada uno con un filtro de tamaño 3x3 en un solo canal.

La Figura 3.7 muestra gráficamente el funcionamiento de esta capa de convolución separando la matriz resultante de la primera convolución en dos grupos y aplicando los filtros necesarios.

Figura 3.7. Separación en dos grupos.

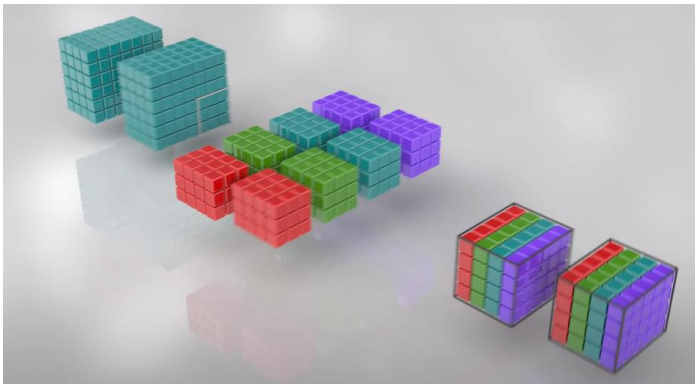


Imagen del funcionamiento de la capa Groupped Convolution, Fuente: Animated AI

En resumen, estas primeras capaz de la red neuronal aplican 32 filtros con un tamaño de matriz de 3x3, extrayendo asi características más puntuales en cada píxel de imagen que ingresa a la red neuronal. Para pasar a la segunda parte de la estructura que son 19 cuellos de botella lineales y residuales invertidos (Inverted Residual and Linear Bottlenecks).

En la Figura 3.8 se observa la estructura del cuello de botella implementado en Mobilenetv2.

Figura 3.8. Estructura del cuello de botella.

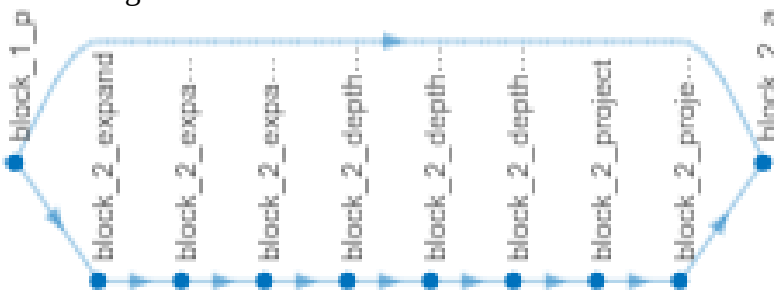


Diagrama de la estructura de cuello de botella, Autor: Andrés Lapo.

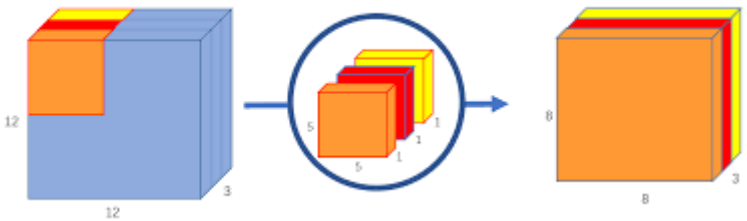
La última capa que se aplica a la imagen de entrada antes de ingresar al cuello de botella es la capa de Batch Normalization.

El cuello está compuesto por diferentes capas de los cuales son tres los componentes principales.

- Convolución en profundidad (Depthwise Convolution): Esta es una técnica utilizada en redes neuronales convolucionales para reducir el costo computacional y la cantidad de parámetros. En lugar de aplicar la convolución tradicional a cada canal de entrada, la convolución de profundidad aplica un filtro separado a cada canal separado. Esto significa que cada canal de entrada tiene su propio filtro y las salidas de cada filtro se combinan para formar la salida final. Este método reduce la cantidad de cálculos requeridos y la cantidad de parámetros mientras mantiene una rica representación de la función.(Sandler et al., 2018).

En la figura 3.9 se observa el funcionamiento de una convolución en profundidad.

Figura 3.9. Convolución en profundidad.

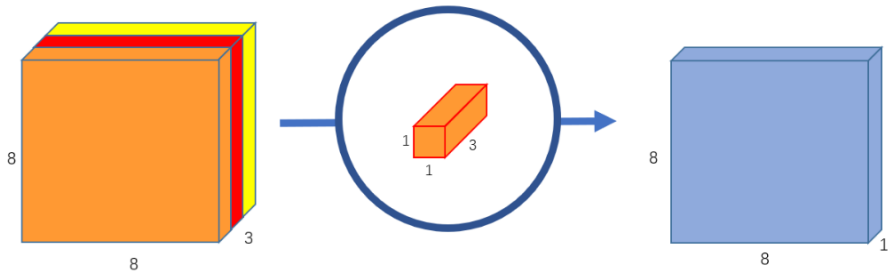


Convolución en profundidad, Fuente: Chi-Feng Wang.

- Convolución de punto (Pointwise Convolution): es una operación básica en la que se realiza una convolución 1x1 para combinar características en diferentes canales. Esta operación se utiliza para ajustar la dimensionalidad de los datos, es decir, para cambiar el número de canales manteniendo la resolución espacial.(Sandler et al., 2018).

En la figura 3.10 se observa el funcionamiento de una convolución de punto.

Figura 3.10. Convolución de punto.



Convolución de punto, Fuente: Chi-Feng Wang.

- Conexión de salto (Skip Connection): es una conexión adicional en una red neuronal que le permite guardar información importante y mejorar el flujo de información. Consiste en añadir la salida de la capa anterior a la capa siguiente, saltándose algunas capas intermedias. Esta conexión directa permite que la información viaje rápidamente por la red, evitando que la información se dañe o se pierda durante el proceso de aprendizaje. Saltar es particularmente útil en modelos profundos porque ayuda a resolver el problema del decaimiento del gradiente y permite que la red aprenda representaciones más profundas y complejas.(He et al., 2015)

La combinación de convolución en profundidad y convolución de punto da como resultado lo que se conoce como Expanded Convolution permitiendo reducir el costo computacional y el número de parámetros del modelo al aplicar la convolución en profundidad antes de la convolución de punto.

Esta convolución expandida (Expanded Convolution) se conecta con otras capas como lo son la capa ReLU, Batch Normalization, etc.

Los cuellos de botella en MobileNetV2 están destinados a reducir la carga computacional y la cantidad de parámetros del modelo mientras se mantiene un rendimiento y una precisión aceptables en las tareas de clasificación de imágenes. Estos cuellos de botella son bloques de capas que incluyen una combinación de

operaciones de convolución para comprimir información y reducir el tamaño de los datos. El uso de cuellos de botella en MobileNetV2 se basa en la suposición de que la mayoría de las características de la imagen se pueden representar de manera eficiente utilizando dimensiones espaciales de baja resolución. Por lo tanto, los cuellos de botella son los responsables de reducir la resolución espacial de los datos a través de operaciones de convolución con tamaños de filtro más pequeños, lo que reduce en gran medida la carga computacional.

La tercera parte de la arquitectura MobileNetV2 incluye las capas de salida responsables de generar la salida final de la red neuronal después de superar los cuellos de botella. Estas clases están diseñadas para detectar o clasificar objetos según el propósito de la red. Las capas de salida consisten en operaciones de convolución global, seguidas de capas totalmente conectadas y una capa de activación final, como la función softmax. Estas últimas capas forman el elemento esencial de la red que produce pronósticos precisos y representativos. A medida que la información pasa a través de las capas de salida, se agrega y realiza una detección o clasificación de objetos eficiente.

En la Figura 3.11 se observa la estructura de salida implementada en Mobilenetv2.

Figura 3.11. Disposición de las capas finales de mobilenetv2.

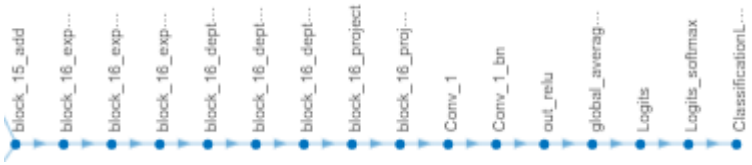
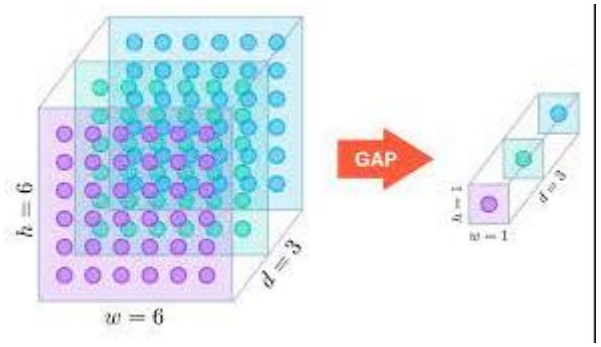


Diagrama de la estructura de salida, Autor: Andrés Lapo

La capa de convolución global desempeña un papel crucial al capturar características contextuales a nivel global en toda la imagen. Esta capa realiza una convolución especial conocida como Global Average Pooling (GAP) que, en lugar de aplicar convoluciones en ventanas deslizantes como las capas anteriores, la capa de convolución global considera la información de todos los píxeles de la imagen.(Sandler et al., 2018)

En la Figura 3.12 se observa el funcionamiento de GAP

Figura 3.12. Funcionamiento de GAP.



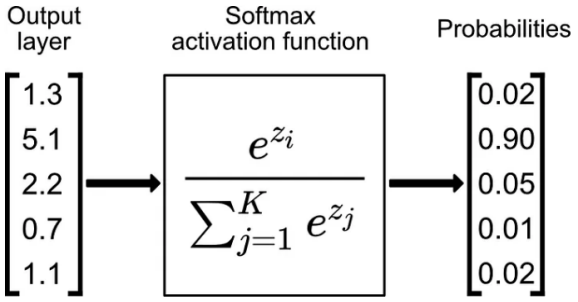
Funcionamiento de GAP, Fuente: Alexis Cook.

El proceso de convolución global implica promediar los valores de activación en cada mapa de características de entrada. Esto reduce el tamaño de los mapas de características, convirtiéndolos en un solo vector de características. Al calcular un promedio global en lugar de usar una ventana deslizante, obtiene un cálculo más general que captura información contextual global. Este enfoque ayuda a capturar características de alto nivel y relaciones duraderas en la imagen. La convolución global es una forma eficiente de resumir la información espacial de una imagen en un vector de características compacto. Reducir el tamaño reduce la carga computacional y aumenta la eficiencia de la red.

La función softmax es una función utilizada comúnmente en las capas de salida de las redes neuronales. La función softmax se utiliza para convertir las salidas lineales de una red en una distribución de probabilidades, asignando una probabilidad a cada clase o categoría de salida(Sandler et al., 2018).

En la Figura 3.13 se representa la función de activación Softmax en una red neuronal.

Figura 3.13. Función de activación Softmax.



Función de activación softmax, Fuente: Dario Radecic.

La función Softmax toma un vector de valores como entrada y realiza dos pasos principales. Primero calcula el exponente de cada valor en el vector, lo que garantiza que todos los valores no sean negativos. Luego, normalice los exponentes dividiendo

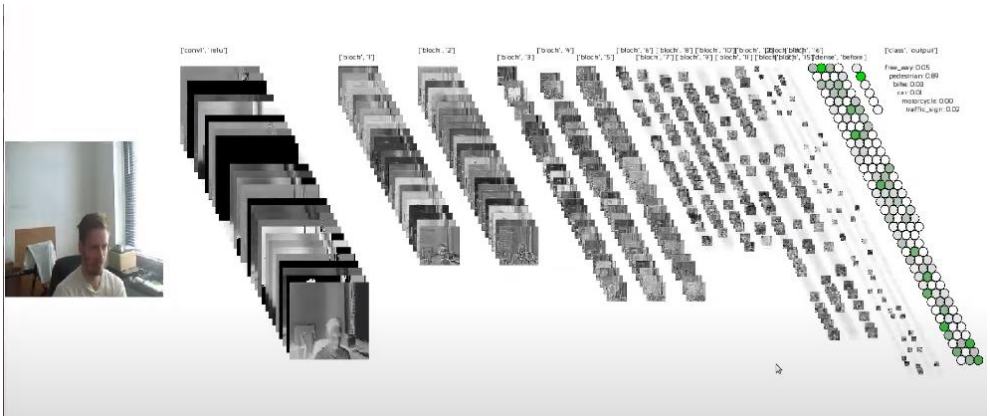
por la suma de todos los exponentes para que la probabilidad de que el resultado resultante sume sea igual a 1.

Esta normalización garantiza que la salida de la red se interprete como probabilística, lo que facilita la interpretación de los resultados. La clase con la probabilidad más alta de la función Softmax se considera el último predictor de la red.

La función softmax es especialmente útil para problemas de clasificación de clases múltiples cuando necesita obtener la distribución de probabilidad para cada clase de salida. Esto permite tomar decisiones con base en las probabilidades relativas de cada clase y proporciona resultados más fáciles de entender y más relevantes para la toma de decisiones.

En la figura 3.14 se aprecia la visualización de la arquitectura de Mobilenetv2.

Figura 3.14. Visualización de la red Mobilenetv2.



Visualización de Mobilenetv2, Fuente: Divis Václav.

3.4 Modificaciones en Mobilenetv2

En este numeral se describirán las modificaciones realizadas en la red neuronal Mobilenetv2 para incluir la clasificación Huiracchuro. El objetivo fue adaptar la red preentrenada para lograr una detección precisa y confiable de esta especie de ave.

La red neuronal recibe como entrada una imagen de 224px por 224px en 3 canales, para posteriormente clasificarla en alguna de las 1,001 clases que posee.

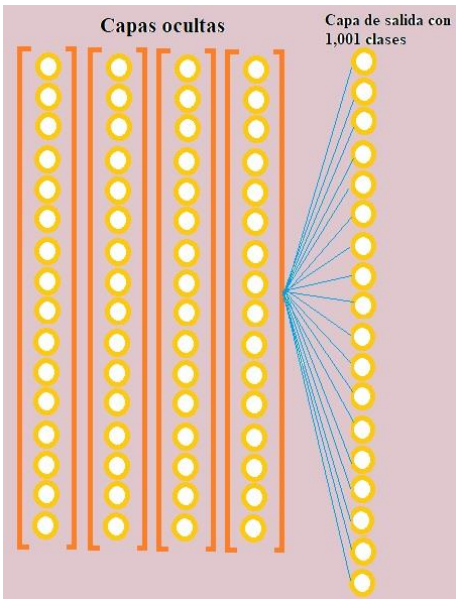
Las modificaciones realizadas a está red son:

- Eliminación de la capa de salida: Como se mencionó con anterioridad, la red posee una capa de salida con 1001 categorías de clasificación para imágenes, en las cuales no se encuentra la clasificación Huiracchuro. La eliminación de la capa de categorización permite crear una nueva capa de clasificación, la cual contara con pesos y sesgos asignados automáticamente.

- Aumento de una nueva capa densa con la función de activación Softmax: La nueva capa de salida de la red neuronal, la cual posee la clasificación Huiracchuro, será entrena y evaluada con una base de datos específica para modificar sus pesos y sesgos, esto se detalla con mayor profundidad en el apartado de Google Colaboratory.
- Congelamiento de los pesos y sesgos de las capas anteriores: La red neuronal que se aplica fue entrenada con el dataset ImageNet, al eliminar la última capa original de la red y cambiarla por una nueva se necesita establecer los pesos y sesgos de esta nueva conexión. Por ello es importante el congelamiento de las capas anteriores para que no pierdan el entrenamiento previo y solo se entrene esta nueva capa añadida.
- Aprendizaje y entrenamiento: Se establece los pesos y sesgos entre las conexiones de la penúltima capa de la red con la nueva capa añadida, para una mejor detección.

La Figura 3.15 muestra de manera artística la representación de la red neuronal MobileNetv2 con su capa de salida.

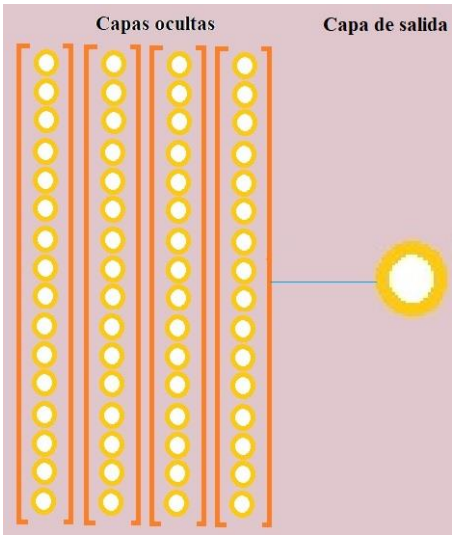
Figura 3.15. Representación de la red neuronal MobileNetv2



El diagrama muestra de manera artística la red neuronal a emplearse, Autor: Andrés Lapo.

La Figura 3.16 muestra el reemplazo de las 1,001 neuronas de salida por una sola, la cual se activa si el ave fue detectada.

Figura 3.16. Eliminación de la última capa y reemplazo por una salida



El diagrama muestra de manera artística el cambio de la capa de salida, Autor: Andrés Lapo.

3.5 Características del ave para su detección

Las características extraídas para la identificación del ave se extraen en las capas de entrada y en los cuellos de botella de la arquitectura de Mobilenetv2. Estas capas han sido entrenadas durante un extenso periodo de tiempo utilizando ImageNet, lo que ha permitido establecer de manera sólida los pesos, sesgos y conexiones correspondientes. Algunas de las características tomadas por la red son:

- Forma del pico
- Tamaño del ave
- Tamaño del ojo
- Colores

3.6 Aprendizaje y entrenamiento

3.6.1 Datos para entrenamiento y pruebas

Para la obtención de una base de datos para el entrenamiento y pruebas se utiliza la extensión de Google Download All Images, la cual permite descargar varias imágenes en un documento .zip, las imágenes se obtienen de la página web Ecuador.inaturalist.org. La cual tienen una gran cantidad de imágenes de varias aves, entre las cuales se encuentra el ave Huiracchuro.

La Figura 3.17 muestra un ejemplo del funcionamiento de la extensión Download All Images.

Figura 3.17. Ejemplo de uso de la extensión Download All Images

La imagen muestra cómo se descarga varias imágenes en un documento zip, para después ser utilizado,

Fuente: Download All Images.

Una vez obtenido el archivo .zip se limpia los datos contenidos en la carpeta descomprimida. Los criterios para que una imagen sea considerada para ser usada en la red o ser eliminada son:

- La imagen debe ser un archivo PNG, JPG
- Solo debe haber un ave en la imagen
- La imagen no debe tener corrupción visible, entiéndase con esto que la imagen no debe líneas de colores, ruido o distorsiones
- La imagen no debe tener texto que influya al momento de extraer las características del ave

3.6.2 Google Colaboratory

La modificación y programación de la red neuronal se realiza utilizando la página web Google Colaboratory (GC), la cual permite utilizar servidores de Google con GPU para un mejor entrenamiento y menor tiempo de procesado en la red. Como primer punto es la creación de las carpetas que contienen las imágenes de la base de datos clasificadas de la siguiente manera:

- Cercanas: El ave ocupa el 50% o más de la imagen.
- Medianas: El ave ocupa más del 25% y menos del 50% de la imagen.
- Lejanas: El ave ocupa menos del 25% de la imagen

Este elegido bajo la métrica de la cantidad de espacio que ocupa el ave en la imagen, estás carpetas son creadas en GC para el almacenamiento de las imágenes que son usadas para entrenamiento y prueba de la red neuronal. Para ello se hace uso de la librería TensorFlow, la cual permite importar un modelo de red neuronal almacenado en TensorFlow Hub.

Se procede a realizar un conteo de las imágenes existentes en cada una de las carpetas, una vez obtenida la cantidad de imágenes contenidas en las carpetas anteriormente mencionadas se realiza la creación de la carpeta dataset en la que se moverán las imágenes en una cantidad específica, esta cantidad será tomada como el número menor de imágenes que contenga alguna de las carpetas originales, para de esa forma entrenar a la red con la misma cantidad de imágenes en cada categoría.

Del total de imágenes contenidas en la carpeta dataset se destina el 20% a pruebas, mientras que el 80% destinado al entrenamiento de la red. Las imágenes tomadas para el entrenamiento y pruebas de la red son tomadas totalmente al azar de la carpeta dataset.

Con el set de datos terminado, se procede a llamar a la red neuronal, la cual se encuentra en el repositorio TensorFlow Hub para poder ser modificada, entrenada y utilizada en pruebas posteriores.

A continuación, se inhibe los parámetros de la red, pesos y sesgos, previamente entrenados y se adhiere una nueva capa de salida que contiene la clasificación Huiracchuro junto con la función de activación Softmax.

Como siguiente paso la compilación de la red, se añaden los parámetros de optimización, perdida y métricas para su posterior validación.

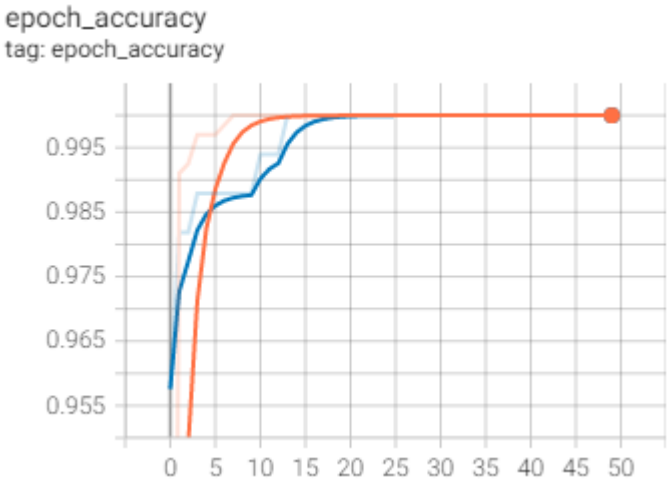
El entrenamiento de la red se realiza por diferentes épocas 25, 50 y 100, guardando los datos de perdida, precisión. Dan como resultado que la red con mejor clasificación y menor perdida es la que se entrena solo por 50 épocas.

Cada periodo de ajuste de ida y vuelta para reducir la perdida se denomina época.

El resultado del entrenamiento y las pruebas de la red neuronal son graficadas con la herramienta TensorBoard.

La Figura 3.18 muestra el resultado de la precisión de la red en entrenamiento (curvatura de color naranja) y pruebas para un entrenamiento (curvatura de color azul) de 50 épocas.

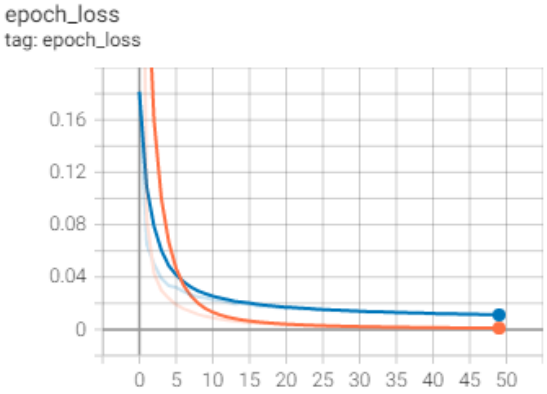
Figura 3.18. Precisión de la red en pruebas y entrenamiento.



Precisión de la red en los datos de prueba y entrenamiento, Autor: Andrés Lapo

La Figura 3.19 muestra el resultado de perdida de datos de la red en entrenamiento (curvatura de color naranja) y pruebas para un entrenamiento de 50 épocas.

Figura 3.19. Perdida de la red en pruebas y entrenamiento.

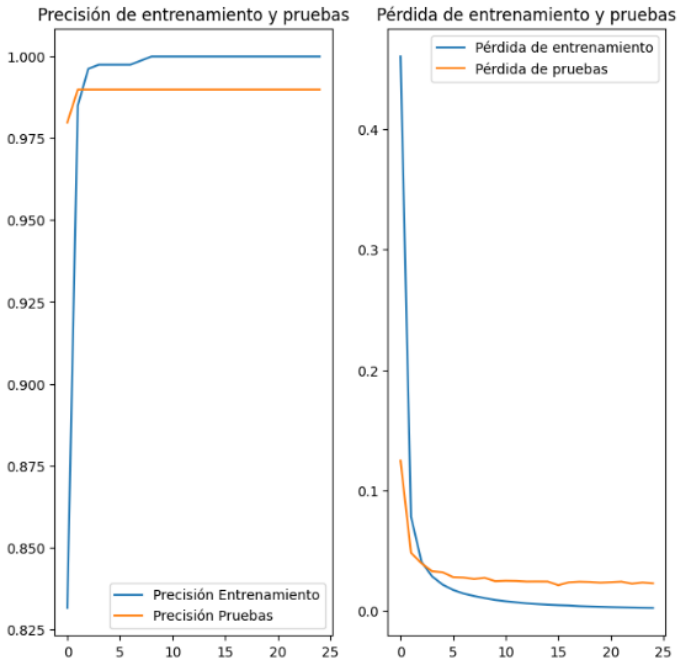


Perdida de la red en los datos de prueba y en el entrenamiento, Autor: Andrés Lapo

La nueva red neuronal modificada es almacenada en un archivo tipo h5, para posteriormente ser convertido en un archivo de tipo Jason (JS) para su descarga y uso en otras plataformas o programas.

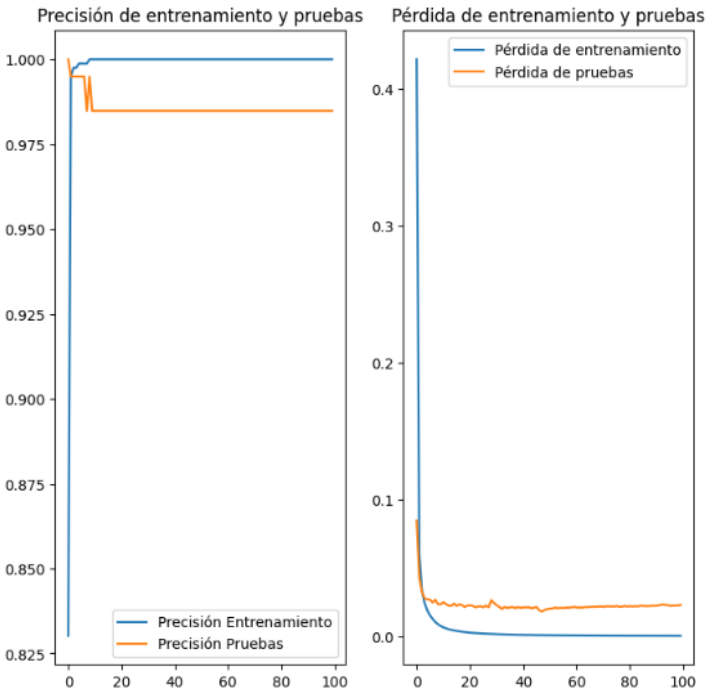
En la figura 3.20 y 3.21 se muestra el resultado de aprendizaje y perdida para el entrenamiento de la red por solo 25 épocas. Como resultado se obtiene que la red con un entrenamiento de 25 épocas es bajo porque no logra aprender de manera correcta las características del ave, mientras que el entrenamiento de 100 épocas se da un sobre ajuste, se aprende los datos de entrenamiento, pero para los datos de prueba con los datos de prueba el error aumenta.

Figura 3.20. Precisión y perdida de la red en pruebas y entrenamiento con 25 épocas



Gráfica de resultado de entrenamiento de la red por 25 épocas, Autor: Andrés Lapo

Figura 3.21. Precisión y perdida de la red en pruebas y entrenamiento con 100 épocas



Gráfica de resultado de entrenamiento de la red por 100 épocas, Autor: Andrés Lapo.

3.7 Aplicativo Web

El aplicativo web consta de varias partes explicadas en la figura 3.22:

Figura 3.22. Vista general del aplicativo web

Figura descriptiva de las partes del aplicativo web, Autor: Andrés Lapo

El aplicativo genera una pequeña ventana emergente pidiendo el acceso a la cámara del dispositivo.

Una vez concedido el permiso, el aplicativo toma la imagen de entrada, lo redimensiona a un tamaño de 224px x224px para posteriormente enviar toda esta información hacia la red neuronal.

La red neuronal se encarga del procesamiento de la imagen y origina un array de valores, los cuales son interpretados por el aplicativo web, en el cual mediante una sentencia condicional genera un mensaje, el cual puede ser “Huiracchuro Detectado” o “Indefinido”. Tanto el aplicativo como la red neuronal se encuentran en un servidor local, el cual no tiene salida a internet y por ende no es accesible desde cualquier dispositivo externo al del servidor local.

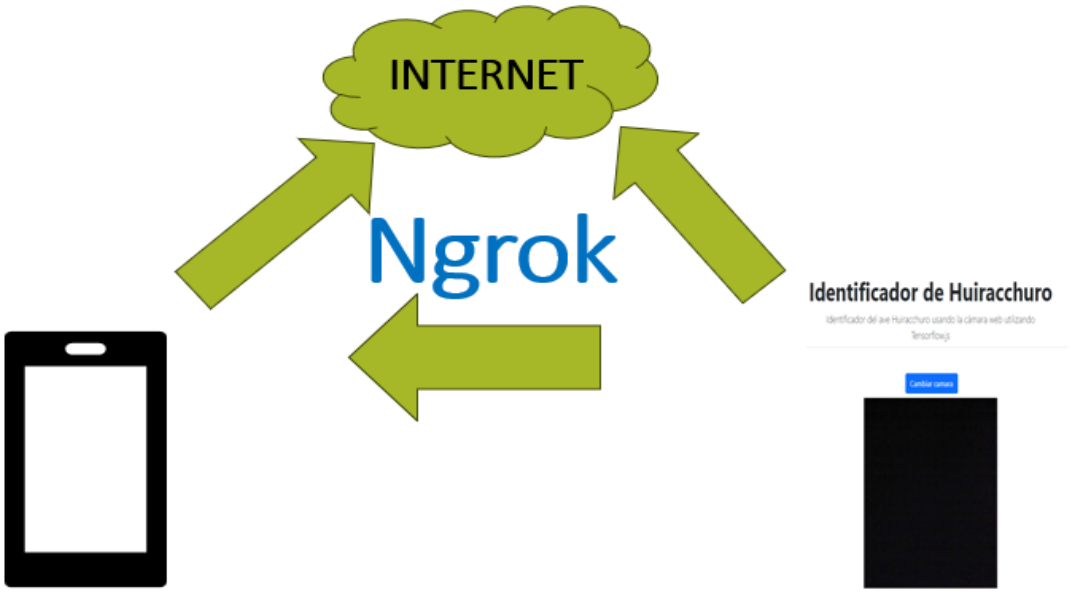
La solución a la problemática de acceso al aplicativo y él envió de datos por internet se logra a través de la herramienta gratuita Ngrok, la cual al ser ejecutada entrega un enlace seguro (link) para su acceso y de esa manera poder acceder al aplicativo web desde cualquier dispositivo.

La herramienta Ngrok cuenta con una versión gratuita y una versión de paga, en este

proyecto se utilizó la versión gratuita que consta con la limitación de que cada enlace entregado tiene un tiempo de duración de una hora, pero no tiene límite de generación del enlace. Eso permite utilizar el enlace por la hora permitida, pero a la vez generar otro de manera inmediata una vez se cumpla el tiempo de uso del primer enlace.

En la figura 3.23 se recrea de manera artística el uso del aplicativo web desde cualquier dispositivo gracias a la conexión que ofrece Ngrok.

Figura 3.23. Salida a internet por Ngrok



Conexión de cualquier dispositivo hacia el sistema proporcionado por la herramienta Ngrok, Autor: Andrés Lapo.

CAPÍTULO 4

PRUEBAS Y RESULTADOS

4.1 PRUEBAS

El entorno de pruebas son las huertas y cultivos orgánicos del Barrio Santa Rosa en la Parroquia de Puembo y en el Barrio Coyagal de la Parroquia de Puéllaro, entre estos cultivos se encuentran girasoles, mandarinas, fresas, maíz, etc.

Las pruebas experimentales se llevan a cabo en franjas horarias específicas que han sido cuidadosamente seleccionadas. Dichas franjas corresponden a las primeras horas de la mañana y las últimas horas de la tarde. Con mayor precisión, las pruebas se realizan en el intervalo de tiempo comprendido entre las 5:00 a.m. y las 7:00 a.m., así como entre las 5:00 p.m. y las 6:30 p.m. Estos horarios han sido elegidos con el fin de asegurar condiciones óptimas y consistentes para el desarrollo de las pruebas en el marco de esta investigación.

Las pruebas se llevan a cabo desde un teléfono Huawei P40LITE con una cámara trasera de 48 megapíxeles (MP) con focal de 1.8, cámara gran angular de 8 MP con focal 2.4, cámara macro de 2MP con focal de 2.4 y una cámara de profundidad de 2MP con focal 2.4. La cámara que se emplea tiene una resolución de 48 MP aplicando un Zoom digital de x6 que es el máximo permitido en la cámara. El celular se conecta a la página web construida con anterioridad, la cual muestra el mensaje “Huiracchuro Detectado” cada vez que el ave pase por la cámara o sea enfocada.

Figura 4.1. Barrio Santa Rosa

Barrio Santa Rosa de Puembo, Autor: Google Earth.

Figura 4.2. Barrio Coyagal de Puéllaro

Barrio Coyagal de la parroquia de Puéllaro, Autor: Google Earth.

4.2 RESULTADOS OBTENIDOS

Después de realizar 542 pruebas se realiza una limpieza de los datos tratados, utilizando 119 imágenes en las que el ave ocupa un área mayor o igual a 40x40 px, la efectividad de la red neuronal es del 73.94%.

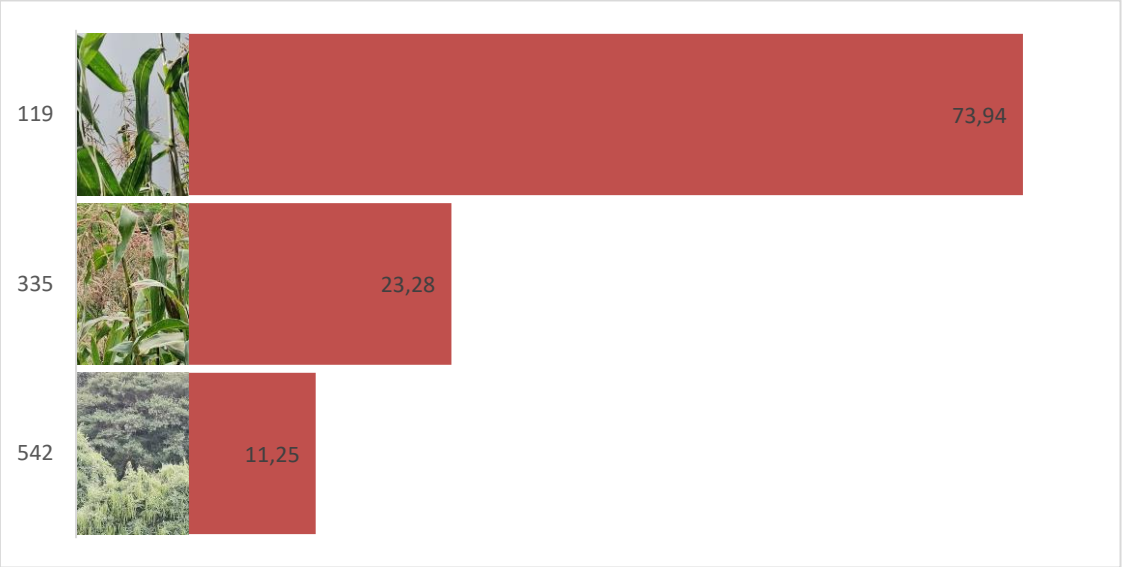
Cuando se añaden 216 imágenes adicionales, el conjunto de datos se compone de un total de 335 imágenes. En estas imágenes, se incluyen aquellas donde el ave ocupa un área de 20px x 20px. Al analizar los resultados, se observa que la efectividad de la red neuronal alcanza un valor del 23.28%.

Por concluir, se emplearon un total de 542 imágenes en las cuales el ave ocupa un área inferior a la mencionada anteriormente. Tras realizar el análisis correspondiente, se obtuvo una efectividad de la red neuronal del 11.25%. Es importante destacar que todas las imágenes utilizadas tenían un tamaño de entrada de 224px x 224px, de acuerdo con la configuración establecida para la red neuronal.

Estos resultados evidencian la capacidad y limitaciones de la red neuronal en la tarea de detección y clasificación de aves en imágenes de tamaño predefinido.

Estos resultados se pueden observar en la figura 4.1 en la cual del lado izquierdo está el número de imágenes utilizadas, en el centro un ejemplo de las imágenes utilizadas y en el lado derecho el porcentaje de efectividad de la red según las imágenes.

Figura 4.3. Porcentaje de efectividad

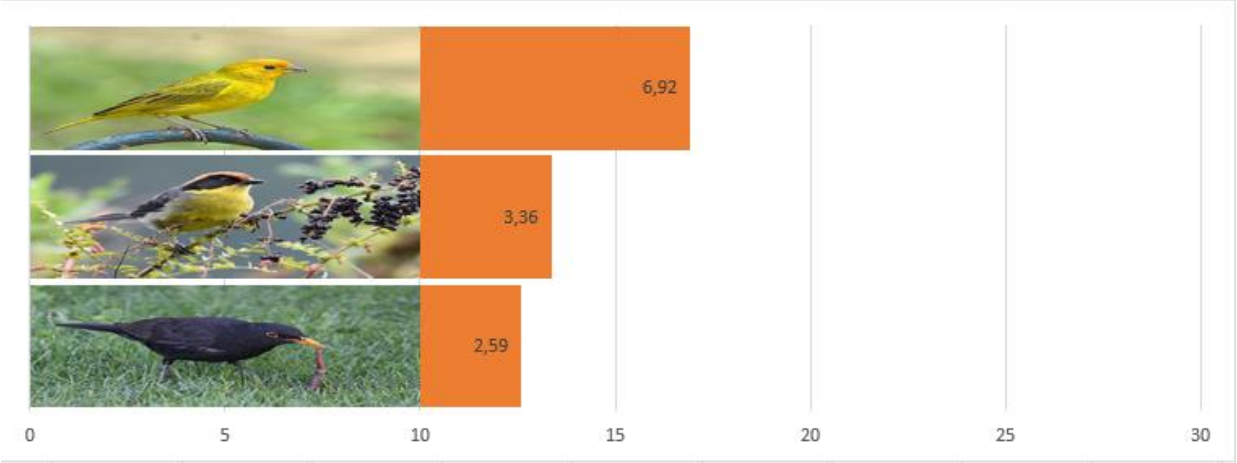


El gráfico muestra el porcentaje de efectividad y la cantidad de imágenes, Autor: Andrés Lapo

Se lleva a cabo pruebas de falsos positivos para evaluar el error de la red al momento de utilizar una imagen que no corresponde al ave Huiracchuro, para esto se utiliza un set de datos con tres tipos de aves, el ave Mirlo, Monja Pechiamarilla y Sicalis Flaveola. Estas aves fueron escogidas por sus colores, forma del pico, tamaño del ave o su alimentación. Estas imágenes fueron filtradas por su tipo de archivo para ser aceptados, los archivos deben ser JPG o PNG, el tamaño de la imagen no exceda los 1280px x 960px, que solo se vea un ave a la vez en la imagen y que la imagen no tenga corrupción, ruido o texto que influya en su clasificación.

Dando como resultados que de 231 imágenes el 12.98% son clasificadas como ave Huiracchuro, al evaluar las imágenes clasificadas como Huiracchuro se observa que de estas tres aves la que más es aceptada como un Huiracchuro es el ave Sicalis Flaveola por su color amarillo y por la forma de su pico porque su alimentación se basa en frutos al igual que el Huiracchuro. Por estas características, la capa de función de activación Softmax genera un porcentaje de confiabilidad mayor a comparación de las otras dos aves mencionadas.

Figura 4.4. Porcentaje de falsos positivos



El gráfico muestra el porcentaje falsos positivos, muestra de las aves que dan falsos positivos,

Autor: Andrés Lapo

4.3 CONCLUSIONES

- La adaptación de la red Mobilenetv2 para la detección del ave Huiracchuro demuestra su aplicabilidad en tareas de reconocimiento de objetos específicos. Esta estrategia puede ser utilizada en otros proyectos de detección de especies o clasificación de objetos similares.
- La implementación de la aplicación web que permite la clasificación en tiempo real proporciona una herramienta práctica y accesible para identificar el ave Huiracchuro. Esto tiene aplicaciones en campos como la observación de especies, la observación de aves y la educación ambiental.
- La efectividad del 73.94% en la detección del ave Huiracchuro cuando ocupa más de 40x40 píxeles en la imagen demuestra la capacidad de la red para identificar con precisión el ave cuando es prominente en la imagen. Esto es un logro significativo y muestra el potencial de la red en la tarea de detección.
- La capacidad de la red para reconocer el ave Huiracchuro incluso en diferentes poses y ángulos indica una cierta robustez en la detección de la especie. Esto sugiere que el modelo puede generalizar y reconocer características distintivas del ave en diversas situaciones.

4.4 RECOMENDACIONES

- Evaluar su desempeño en la detección de otras especies de aves. Esto permitiría determinar si el modelo puede generalizar y adaptarse a diferentes especies, lo cual tendría aplicaciones más amplias en la identificación de aves en general.
- Realizar pruebas de implementación en diferentes dispositivos móviles que posean una cámara integrada. Estos dispositivos pueden tener diferentes sistemas operativos, como Android e iOS. Esta evaluación permitirá verificar la compatibilidad y rendimiento de la aplicación en diversos entornos.
- Investigar la integración de la red Mobilenetv2 en estos autónomos. Esto abrirá la posibilidad de realizar tareas de detección y clasificación del ave Huiracchuro u otros objetos en tiempo real desde el aire, lo cual puede ser de gran utilidad en estudios de monitoreo de especies o investigaciones de conservación.

BIBLIOGRAFÍA

2-D convolutional layer - MATLAB. (n.d.). Retrieved June 23, 2023, from https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.convolution2dlayer.html

2-D grouped convolutional layer - MATLAB. (n.d.). Retrieved June 23, 2023, from https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.groupedconvolution2dlayer.html

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. http://image-net.org/challenges/LSVRC/2015/

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., & Andreetto, M. (n.d.-a). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., & Andreetto, M. (n.d.-b). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*.

Huiracchuro | Quito, hábitat silvestre. (n.d.). Retrieved November 8, 2022, from https://quitohabitatsilvestre.wordpress.com/2012/07/02/huiracchuro-2/

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (n.d.). *ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design*.

Pérez, M., & Hornillo, S. (2021). *Autor: Marta Pérez Ortiz de Landaluce Tutor: Susana Hornillo Mellado*.

Ponce, J. C., Torres, A., Sprock, A. S., & Casali, A. (2014). *Inteligencia Artificial*. https://doi.org/10.13140/2.1.3720.0960

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (n.d.). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. http://arxiv.org/abs/1801.04381

Santurkar, S., Tsipras, D., Ilyas, A., & Mit, A. M. ˛ A. (n.d.). *How Does Batch Normalization Help Optimization?*

Te damos la bienvenida a Colaboratory - Colaboratory. (n.d.). Retrieved December 6, 2022, from https://colab.research.google.com/?hl=es

Zhang, X., Zhou, X., & Lin, M. (n.d.). *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*.

BIBLIOGRAFÍA DE FIGURAS

Duran, M. (2016). Foto 39782757, (c) Marcoduran, Todos los derechos reservados, Subido Por Marcoduran · inaturalist ecuador. Retrieved from https://ecuador.inaturalist.org/photos/39782757

Hernández, por N., 3.0, por E., López, por L. G., & Pajuelo, por L. (2023). Retrieved from https://www.educaciontrespuntocero.com/tecnologia/poencial-ia-en-educacion/

Calvo, D. (2018). Retrieved from https://www.diegocalvo.es/red-neuronal-convolucional/

Juantorena, G. (2020). Retrieved from https://gjuantorena.medium.com/como-empezar-a-analizar-datos-con-python-usando-google-colab-1e3cf68cba

Animated Ai. (n.d.). https://animatedai.github.io/

Zaffnet. (n.d.). Training Deep Neural Networks with batch normalization. Retrieved from https://zaffnet.github.io/batch-normalization

Sultan, H. H., Salem, N. M., & Al-Atabany, W. (2019). Multi-Classification of Brain Tumor Images Using Deep Neural Network. *IEEE Access*, *7*, 69215–69225. https://doi.org/10.1109/ACCESS.2019.2919122

Wang, C.-F. (2018). Retrieved from https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

Cook, A. (2017). Retrieved from https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/

Radecic, D. (2022). Retrieved from https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60

Václay, D. (2020). Retrieved from https://www.youtube.com/watch?v=MCjjoBj_njs&ab_channel=Divi%C5%A1V%C3%A1clav

Company, G. (2021). Retrieved from https://chrome.google.com/webstore/detail/download-all-images/ifipmflagepipjokmbdecpmjbibjnakm

ANEXOS

ANEXO 1. Modificación y programación de la red neuronal Mobilenetv2

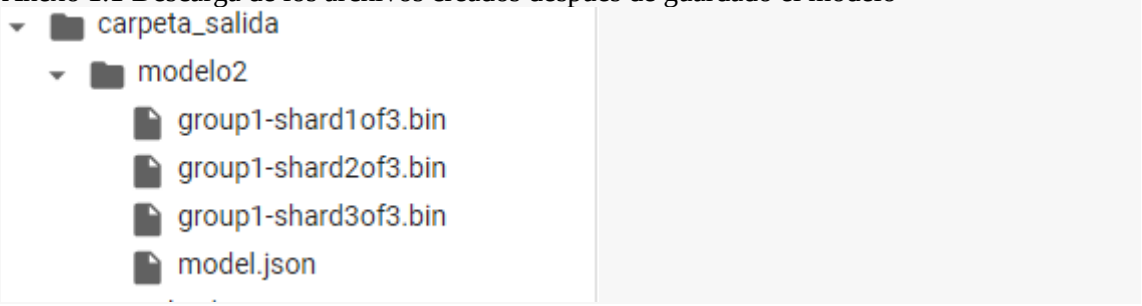
```
#Crear las carpetas para subir las imagenes
!mkdir cercanas
!mkdir media
!mkdir lejana
#Entrar en cada carpeta y descomprimir el archivo zip
%cd cercanas
!unzip cercanas.zip
%cd ..

%cd media
!unzip Media.zip
%cd ..

%cd lejana
!unzip Lejana.zip
%cd ..
#Borrar los archivo ZIP
!rm -rf /content/cercanas/cercanas.zip
!rm -rf /content/media/Media.zip
!rm -rf /content/lejana/Lejana.zip
#Mostrar cuantas imagenes tengo de cada categoria
!ls /content/media/Media | wc -l #333
!ls /content/cercanas/cercanas | wc -l #278
!ls /content/lejana/Lejana | wc -l #364
#Mostrar algunas imagenes con pyplot
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

plt.figure(figsize=(15,15))

carpeta = '/content/cercanas/cercanas'
imagenes = os.listdir(carpeta)

for i, nombreimg in enumerate(imagenes[:25]):
  plt.subplot(5,5,i+1)
  imagen = mpimg.imread(carpeta + '/' + nombreimg)
  plt.imshow(imagen)
#Crear carpetas para hacer el set de datos

!mkdir dataset
!mkdir dataset/cercanas
!mkdir dataset/media
!mkdir dataset/lejana
#Copiar imagenes que subimos a carpetas del dataset
```

```
#Limitar para que todos tengan la misma cantidad de imagenes
#maximo 278 (el num. menor de imagenes que subi)

import shutil
carpeta_fuente = '/content/cercanas/cercanas'
carpeta_destino = '/content/dataset/cercanas'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
  if i < 278:
    #Copia de la carpeta fuente a la destino
    shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
'/' + nombreimg)
carpeta_fuente = '/content/media/Media'
carpeta_destino = '/content/dataset/media'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
  if i < 278:
    #Copia de la carpeta fuente a la destino
    shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
'/' + nombreimg)
carpeta_fuente = '/content/lejana/Lejana'
carpeta_destino = '/content/dataset/lejana'

imagenes = os.listdir(carpeta_fuente)

for i, nombreimg in enumerate(imagenes):
  if i < 278:
    #Copia de la carpeta fuente a la destino
    shutil.copy(carpeta_fuente + '/' + nombreimg, carpeta_destino +
'/' + nombreimg)
#Mostrar cuantas imagenes tengo de cada categoria en el dataset
!ls /content/dataset/cercana | wc -l
!ls /content/dataset/media | wc -l
!ls /content/dataset/lejana | wc -l
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

#Crear el dataset generador
datagen = ImageDataGenerator(
    rescale=1. / 255,
    validation_split=0.2 #20% para pruebas
)

#Generadores para sets de entrenamiento y pruebas
```

```
data_gen_entrenamiento =
datagen.flow_from_directory('/content/dataset',
target_size=(224,224),
                                                    batch_size=32,
shuffle=True, subset='training')
data_gen_pruebas = datagen.flow_from_directory('/content/dataset',
target_size=(224,224),
                                                    batch_size=32,
shuffle=True, subset='validation')

#Imprimir 10 imagenes del generador de entrenamiento
for imagen, etiqueta in data_gen_entrenamiento:
  for i in range(10):
    plt.subplot(2,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(imagen[i])
  break
plt.show()
#Descarga del modelo de red neuronal a utilizar
import tensorflow as tf
import tensorflow_hub as hub

url = "https://tfhub.dev/google/tf2-
preview/mobilenet_v2/feature_vector/4"
mobilenetv2 = hub.KerasLayer(url, input_shape=(224,224,3))
#Congelar el modelo descargado
mobilenetv2.trainable = False
modelo = tf.keras.Sequential([
    mobilenetv2,
    tf.keras.layers.Dense(3, activation='softmax')
])
#Compilacion de la red neuronal
modelo.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
#Entrenar el modelo
EPOCAS = 50

historial = modelo.fit(
    data_gen_entrenamiento, epochs=EPOCAS, batch_size=32,
    validation_data=data_gen_pruebas
)
#Graficas de precisión
acc = historial.history['accuracy']
val_acc = historial.history['val_accuracy']
```

```
loss = historial.history['loss']
val_loss = historial.history['val_loss']

rango_epocas = range(50)

plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.plot(rango_epocas, acc, label='Precisión Entrenamiento')
plt.plot(rango_epocas, val_acc, label='Precisión Pruebas')
plt.legend(loc='lower right')
plt.title('Precisión de entrenamiento y pruebas')

plt.subplot(1,2,2)
plt.plot(rango_epocas, loss, label='Pérdida de entrenamiento')
plt.plot(rango_epocas, val_loss, label='Pérdida de pruebas')
plt.legend(loc='upper right')
plt.title('Pérdida de entrenamiento y pruebas')
plt.show()
#Crear la carpeta para exportarla a TF Serving
!mkdir -p carpeta_salida/modelo2
#Guardar el modelo en formato SavedModel
modelo.save('carpeta_salida/modelo2')
```

Anexo 1.1 Descarga de los archivos creados después de guardado el modelo



Carpeta que contiene al modelo ya entrenado en Google Colab, Autor: Andrés Lapo

ANEXO 2. Programación del aplicativo web

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Huiracchuro</title>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASj
```

```
C" crossorigin="anonymous">

  <style>
   #resultado {
     font-weight: bold;
     font-size: 6rem;
     text-align: center;
   }
  </style>
 </head>
 <body>

  <main>
   <div class="px-4 py-2 my-2 text-center border-bottom">
     <h1 class="display-5 fw-bold">Identificador de Huiracchuro</h1>
     <div class="col-lg-6 mx-auto">
       <p class="lead mb-0">Identificador del ave Huiracchuro usando la
c&aacute;mara web utilizando Tensorflow.js</p>
     </div>
   </div>

   <div class="b-example-divider"></div>

   <div class="container mt-5">
     <div class="row">
       <div class="col-12 col-md-4 offset-md-4 text-center">
         <video id="video" playsinline autoplay style="width: 1px;"></video>
         <button class="btn btn-primary mb-2" id="cambiar-camara"
onclick="cambiarCamara();">Cambiar camara</button>
         <canvas id="canvas" width="400" height="400" style="max-width:
100%;"></canvas>
         <canvas id="otrocanvas" width="150" height="150" style="display:
none"></canvas>
         <div id="resultado"></div>
       </div>
     </div>
   </div>

   <div class="b-example-divider"></div>
   <div class="b-example-divider mb-0"></div>
```

```
    </main>

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxV
XM" crossorigin="anonymous"></script>

    <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js"></script>

  <script type="text/javascript">

    var tamano = 400;
    var video = document.getElementById("video");
    var canvas = document.getElementById("canvas");
    var otrocanvas = document.getElementById("otrocanvas");
    var ctx = canvas.getContext("2d");
    var currentStream = null;
    var facingMode = "user";

    var modelo = null;

    (async() => {
      console.log("Cargando modelo...");
      modelo = await tf.loadLayersModel("model.json");
      console.log("Modelo cargado");
    })();

    window.onload = function() {
      mostrarCamara();
    }

    function mostrarCamara() {
      var opciones = {
        audio: false,
        video: {
          width: tamano, height: tamano
        }
      }
```

```
 if (navigator.mediaDevices.getUserMedia) {
  navigator.mediaDevices.getUserMedia(opciones)
     .then(function(stream) {
       currentStream = stream;
       video.srcObject = currentStream;
       procesarCamara();
       predecir();
     })
     .catch(function(err) {
       alert("No se pudo utilizar la camara");
       console.log(err);
       alert(err);
     })
 } else {
  alert("No existe la funcion getUserMedia");
 }
}

function cambiarCamara() {
    if (currentStream) {
        currentStream.getTracks().forEach(track => {
            track.stop();
        });
    }

    facingMode = facingMode == "user" ? "environment" : "user";

    var opciones = {
        audio: false,
        video: {
            facingMode: facingMode, width: tamano, height: tamano
        }
    };



    navigator.mediaDevices.getUserMedia(opciones)
        .then(function(stream) {
            currentStream = stream;
```

```
            video.srcObject = currentStream;
        })
        .catch(function(err) {
            console.log("Hubo un error", err);
        })
  }

function procesarCamara() {
  ctx.drawImage(video, 0, 0, tamano, tamano, 0, 0, tamano, tamano);
  setTimeout(procesarCamara, 20);
}

function predecir() {
  if (modelo != null) {
    resample_single(canvas, 100, 100, otrocanvas);

    //Predicción
    var ctx2 = otrocanvas.getContext("2d");
    var imgData = ctx2.getImageData(0,0, 100, 100);

    var arr = [];
    var arr100 = [];

    for (var p=0; p < imgData.data.length; p+= 4) {
      var rojo = imgData.data[p] / 255;
      var verde = imgData.data[p+1] / 255;
      var azul = imgData.data[p+2] / 255;

      var gris = (rojo+verde+azul)/3;

      arr100.push([gris]);
      if (arr100.length == 100) {
        arr.push(arr100);
        arr100 = [];
      }
    }
    arr = [arr];

    var tensor = tf.tensor4d(arr);
```

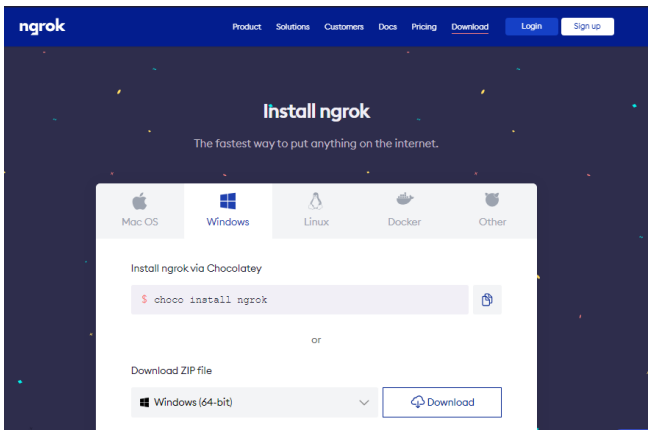
```
  var resultado = modelo.predict(tensor).dataSync();

  var respuesta;
  if (resultado < .7) {
    respuesta = "Huiracchuro Detectado";
  } else {
    respuesta = "Indefinido";
  }
  document.getElementById("resultado").innerHTML = respuesta;

 }
 console.log(resultado)

 setTimeout(predecir, 150);
}
 function resample_single(canvas, width, height, resize_canvas) {
    var width_source = canvas.width;
    var height_source = canvas.height;
    width = Math.round(width);
    height = Math.round(height);

    var ratio_w = width_source / width;
    var ratio_h = height_source / height;
    var ratio_w_half = Math.ceil(ratio_w / 2);
    var ratio_h_half = Math.ceil(ratio_h / 2);

    var ctx = canvas.getContext("2d");
    var ctx2 = resize_canvas.getContext("2d");
    var img = ctx.getImageData(0, 0, width_source, height_source);
    var img2 = ctx2.createImageData(width, height);
    var data = img.data;
    var data2 = img2.data;

    for (var j = 0; j < height; j++) {
       for (var i = 0; i < width; i++) {
          var x2 = (i + j * width) * 4;
          var weight = 0;
          var weights = 0;
          var weights_alpha = 0;
```

```
        var gx_r = 0;
        var gx_g = 0;
        var gx_b = 0;
        var gx_a = 0;
        var center_y = (j + 0.5) * ratio_h;
        var yy_start = Math.floor(j * ratio_h);
        var yy_stop = Math.ceil((j + 1) * ratio_h);
        for (var yy = yy_start; yy < yy_stop; yy++) {
          var dy = Math.abs(center_y - (yy + 0.5)) / ratio_h_half;
          var center_x = (i + 0.5) * ratio_w;
          var w0 = dy * dy;
          var xx_start = Math.floor(i * ratio_w);
          var xx_stop = Math.ceil((i + 1) * ratio_w);
          for (var xx = xx_start; xx < xx_stop; xx++) {
            var dx = Math.abs(center_x - (xx + 0.5)) / ratio_w_half;
            var w = Math.sqrt(w0 + dx * dx);
            if (w >= 1) {
              continue;
            }
            weight = 2 * w * w * w - 3 * w * w + 1;
            var pos_x = 4 * (xx + yy * width_source);
            //alpha
            gx_a += weight * data[pos_x + 3];
            weights_alpha += weight;
            //colors
            if (data[pos_x + 3] < 255)
              weight = weight * data[pos_x + 3] / 250;
            gx_r += weight * data[pos_x];
            gx_g += weight * data[pos_x + 1];
            gx_b += weight * data[pos_x + 2];
            weights += weight;
          }
        }
        data2[x2] = gx_r / weights;
        data2[x2 + 1] = gx_g / weights;
        data2[x2 + 2] = gx_b / weights;
        data2[x2 + 3] = gx_a / weights_alpha;
      }
    }
```

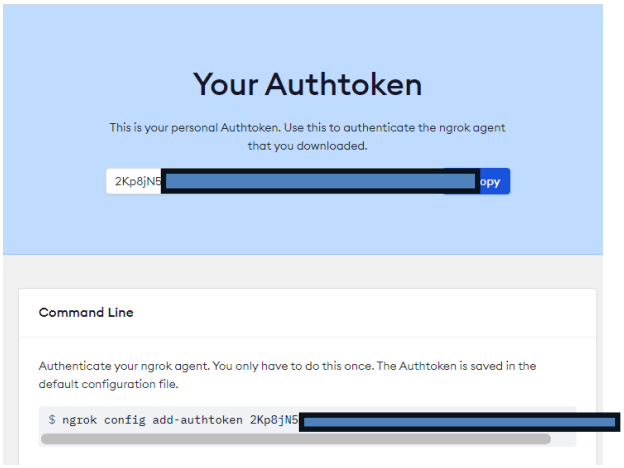
```
        ctx2.putImageData(img2, 0, 0);
    }
  </script>
  </body>
</html>
```

ANEXO 3. Manual de uso de NGROK

Anexo 3.1 Descarga de la herramienta de Ngrok para el sistema operativo necesario.



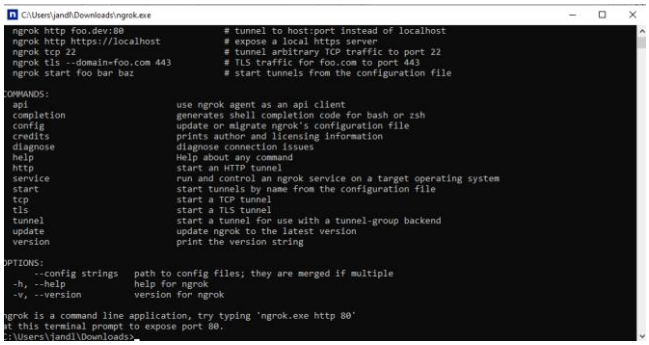
Anexo 3.2 Creación de una cuenta en Ngrok.



Anexo 3.3 Se ejecuta el aplicativo web en un puerto del computador desde el comand shell.

```
C:\Windows\system32\cmd.exe - python  -m http.server 5000
Microsoft Windows [Versión 10.0.19045.3086]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\jandl>cd Desktop

C:\Users\jandl\Desktop>cd Huiracchuro

C:\Users\jandl\Desktop\Huiracchuro>python -m http.server 5000
Serving HTTP on :: port 5000 (http://[::]:5000/) ...
```

Anexo 3.4 Al momento de crear la cuenta, se generará un token exclusivo que será utilizado para la autenticación y el acceso a la aplicación.



Anexo 3.5 Se ejecuta el comando ngrok http numeroDePuerto

```
ngrok is a command line application, try typing 'ngrok.exe http 80'
at this terminal prompt to expose port 80.
C:\Users\jandl\Downloads>ngrok http 5000
```

Anexo 3.6 Dirección web entregada por Ngrok

```
ngrok

Announcing ngrok's Kubernetes Ingress Controller: https://ngrok.com/s/k8s-ingress

Session Status                online
Account                       Andrés (Plan: Free)
Update                        update available (version 3.3.1, Ctrl-U to update)
Version                       3.2.2
Region                        South America (sa)
Latency                       -
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://4645-186-42-1-206.ngrok-free.app -> http://localhost:5000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```