Aplicación de Redes Convolucionales al procesamiento de imágenes en el Data Set de la Fauna Silvestre en Colombia  
Isabella Herrarte López, Gabriel A. Jeannot Viana, Valentina Loaiza Mejia, Andrea Saavedra Viveros

Facultad de Ingeniería, Universidad Autónoma de Occidente  
Cali, Valle del Cuca, Colombia  
[isabella.herrarte@uao.edu.co](mailto:Isabella.herrarte@uao.edu.co)  
[gabriel.jeannot@uao.edu.co](mailto:gabriel.jeannot@uao.edu.co)  
[valentina.loaiza@uao.edu.co](mailto:valentina.loaiza@uao.edu.co)  
[andrea.saavedra@uao.edu.co](mailto:Andrea.saavedra@uao.edu.co)

**Abstract –** The paper's main objective is to describe the training of a Convolutional Neural Network (CNN) that can solve a problem related to the detection of objects through the use of images that are presented as input data (Dataset). To carry out this project, Deep Learning, Keras, and/or TensorFlow tools will be used. The procedure will include information on how the Dataset is conformed, which is focused on Wildlife; the explanation of the images and classes that will be used in the project, which will serve as the basis for a monitoring system. Also, the process of labeling the images for the training of the network, the training of the CNN model used, its validation and the explanation of the architecture of the neural network will be described.

**Keywords –** CNN; Dataset Wildlife; Detection; Keras; Monitoring; Neural Networks; System; Tensorflow.

**Resumen -** El objetivo principal del documento es describir el entrenamiento de una Red Neuronal Convolucional (CNN, por sus siglas en inglés) que pueda solucionar un problema relacionado con la detección de objetos a través del uso de imágenes que son presentados como datos de entrada (Dataset). Para llevar a cabo este proyecto, se utilizarán las herramientas de Deep Learning, Keras y/o TensorFlow. El procedimiento incluirá información sobre cómo está conformado el Dataset, el cual está enfocado en la Fauna Silvestre; además, la explicación de las imágenes y clases que se emplearán en el proyecto, lo cual servirá como base para un sistema de monitoreo. También, se describirá el proceso de etiquetado de las imágenes para el entrenamiento de la red, el entrenamiento del modelo CNN usado, su validación y la explicación de la arquitectura de la red neuronal empleada.

**Palabras Clave -** CNN; Dataset Fauna Silvestre; Detección; Keras; Monitoreo; Redes Neuronales; Sistema; Tensorflow.

# INTRODUCCIÓN

La combinación de la Inteligencia Artificial y la Ingeniería ha permitido solucionar problemas en campos como la medicina, biología animal, seguimiento de patrones de fauna y flora, y el control de modelos. La utilización de la Inteligencia Artificial y el Internet de las Cosas ha adquirido gran importancia en la actualidad, ya que ayuda a diversos sectores a ser más eficientes y productivos. Un ejemplo destacado es la protección de animales en peligro de extinción y la prevención de la caza furtiva en todo el mundo. La aplicación PAWS, desarrollada por el laboratorio Teamcore en el Centro de Inteligencia Artificial de la Universidad del Sur de California, ayuda a los defensores de la vida silvestre a combatir la caza ilegal de animales, considerada la cuarta empresa criminal más rentable del mundo, mediante la optimización de rutas de patrulla. Según NatGeo, la aplicación se utiliza diariamente para controlar la caza furtiva de elefantes africanos, utilizando algoritmos de predicción y toma de decisiones a partir de la información obtenida del monitoreo [1].

El proyecto se enfoca en resolver el problema de la detección de animales silvestres en Colombia que son afectados por el tráfico y la caza ilegal. Para ello, se usará el entrenamiento de una Red Neuronal Convolucional en Google Colab usando Tensorflow2-Keras. El objetivo de este proyecto es que el equipo de trabajo adquiera conocimientos experimentales sobre el uso de las Redes Neuronales Convolucionales en la detección de objetos, a partir de la recolección y etiquetado de imágenes. Esta metodología es útil para implementar sistemas de monitoreo que controlen el estado de salud y el seguimiento de especies en peligro de extinción. Además, esta técnica se puede aplicar para resolver problemas como la predicción de la caza furtiva de animales, el comportamiento de las cadenas tróficas y la dinámica de los biomas, así como la prestación de servicios ecosistémicos.

# MARCO TEÓRICO

El proceso de desarrollo abarca varios fundamentos y conceptos que van desde la importación de librerías en Google Collaboratory hasta la validación del modelo. Por lo tanto, el marco teórico actual incluirá los conceptos teóricos y las librerías que se han utilizado en el Notebook de Google Collaboratory. Con relación al Notebook, se mencionarán las librerías que se han usado para codificar.

**Pandas** es un paquete de Python que facilita el trabajo con datos etiquetados o relacionales, proporcionando estructuras de datos rápidas, flexibles y expresivas. En este caso particular, se empleará como una herramienta para cargar el conjunto de datos en formato .csv [2].

**Numpy** es una biblioteca de Python que incluye un objeto de matriz de N dimensiones, junto con funciones avanzadas como la transmisión (broadcasting) y herramientas útiles para el álgebra lineal, la transformada de Fourier y la generación de números aleatorios [3].

**Matplotlib** es una biblioteca en Python que ofrece una amplia variedad de herramientas para la creación de gráficos interactivos, animados y estáticos. Es capaz de generar figuras de alta calidad para su uso en diferentes formatos, tanto para entornos impresos como interactivos, y puede ser utilizada en diversos contextos, desde scripts hasta servidores web y kits de herramientas de interfaz gráfica de usuario [4].

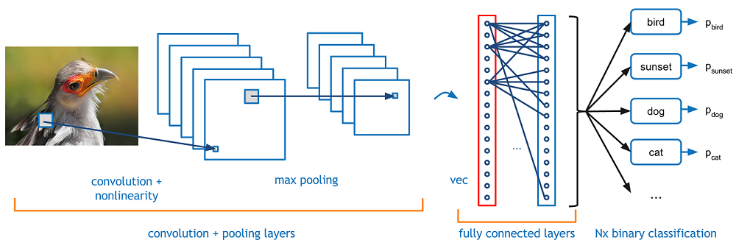
Para cumplir con los requisitos de la guía, se usa como base los modelos pre-entrenados de la API Object Detection de Tensorflow, los cuales se encuentran en el GitHub Tensorflow 2 Detection Model Zoo. Este repositorio contiene una amplia variedad de modelos con diferentes arquitecturas, backbones y tamaños de imágenes de entrada. Para entrenar un modelo personalizado, se emplea un conjunto de imágenes etiquetadas en un archivo TFRecord y un mapa de etiquetas (Label Maps). Estos modelos pre-entrenados permiten realizar una transferencia de aprendizaje y ajustar el modelo a la tarea específica en cuestión.

**Json** sigue la especificación RFC 7159 y ECMA-404, es un formato de intercambio de datos ligero que se basa en la sintaxis literal del objeto JavaScript. Este formato es una alternativa más simple a los módulos de la biblioteca estándar marshal y pickle y ofrece una API familiar a los usuarios. La especificación RFC 7159, que sustituye a la RFC 4627, establece los estándares para el formato JSON [5].

**shutil** proporciona varias operaciones de alto nivel en archivos y colecciones de archivos en Python. En particular, ofrece funciones que facilitan la copia y eliminación de archivos. Sin embargo, para operaciones en archivos individuales, se recomienda también revisar el módulo os [6].

**TFRecord** es un formato de archivo personalizado de TensorFlow que se usa para almacenar grandes conjuntos de datos orientados a registros en un formato binario. Esta estructura de archivo es ampliamente usada en aplicaciones de TensorFlow para entrenar modelos de aprendizaje automático. Los TFRecord son útiles porque permiten la combinación de múltiples conjuntos de datos y se integran perfectamente con la funcionalidad de importación y preprocesamiento de datos proporcionada por la biblioteca. Esto es especialmente importante para conjuntos de datos grandes que no caben en la memoria, ya que solo se cargan los datos necesarios para el procesamiento en ese momento. Por otro lado, el Label Map es un archivo de texto plano con extensión pbtxt que mapea las etiquetas de cada clase con los identificadores correspondientes. Este archivo es necesario para procesos de entrenamiento y predicción y se utiliza junto con los TFRecord [7].

Las **Redes Neuronales Convolucionales** (CNN por sus siglas en inglés) son el enfoque principal del presente documento, especialmente en el procesamiento de imágenes. Estas redes se dividen en dos bloques: el primero, compuesto por capas convolucionales y de pooling, cuyo objetivo es identificar patrones visuales; mientras que el segundo bloque tiene como objetivo la clasificación de los datos que reciben. Esto se logra mediante el entrenamiento con datos, que ajusta los pesos y los sesgos intrínsecos de la red para mejorar su desempeño y reducir su función de pérdida (loss) o error de precisión. Además, una de las ventajas de estas arquitecturas es que se ejecutan de forma concurrente, lo que permite que varias neuronas procesen al mismo tiempo y reduzcan significativamente el tiempo de cómputo. En general, las redes neuronales ofrecen grandes resultados, lo que justifica la inversión continua en su estudio y desarrollo. En términos estructurales, las CNN realizan la convolución y el pooling, y luego se utiliza el proceso de aplanamiento (flattening) para convertir la matriz en un vector, lo que lo hace computacionalmente viable y adecuado para ser introducido como entrada a las capas densas de las redes neuronales [8].



*Ilustración 1 Arquitectura típica de una red neuronal convolucional*

## Redes presentadas en Github

Se hizo una búsqueda analítica en la plataforma de GitHub con el fin de analizar las diferentes posibilidades de selección de redes preentrenadas. En esta plataforma se encuentra disponible una colección de modelos de detección previamente entrenados con el dataset COCO 2017 [9].

Según la referencia [10], COCO es un conjunto de datos de detección, segmentación y subtitulación de objetos a gran escala con características que incluyen:

* Segmentación de objetos
* Reconocimiento en contexto
* Segmentación de cosas superpixeladas
* 330K imágenes (>200K etiquetadas)
* 1,5 millones de instancias de objetos
* 80 categorías de objetos
* 91 categorías de objetos
* 5 leyendas por imagen
* 250.000 personas con puntos clave

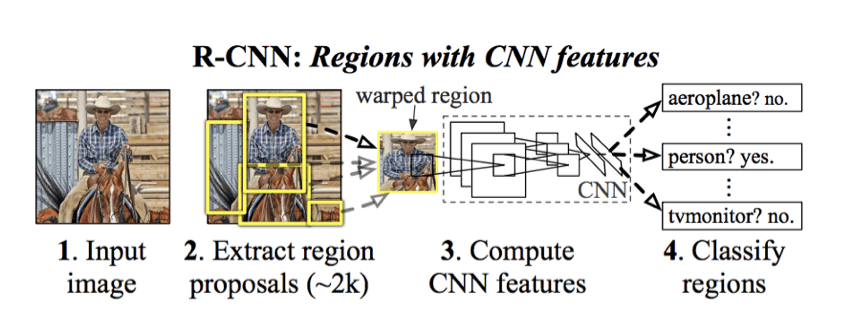
Las tareas principales de la Visión por Computadora, que buscan resolver problemas del mundo real mediante la extracción de información a partir de imágenes, son la **Clasificación, Detección y Segmentación**, y se llevan a cabo con algoritmos de codificación.

En este proyecto se enfoca en el concepto de **Detección de Objetos**, que implica clasificar los objetos en una imagen y determinar su ubicación a través de bounding boxes. Estas cajas delimitadoras son utilizadas para entrenar una Red Neuronal, que devuelve la localización y la probabilidad de pertenencia del objeto detectado a una clase determinada.

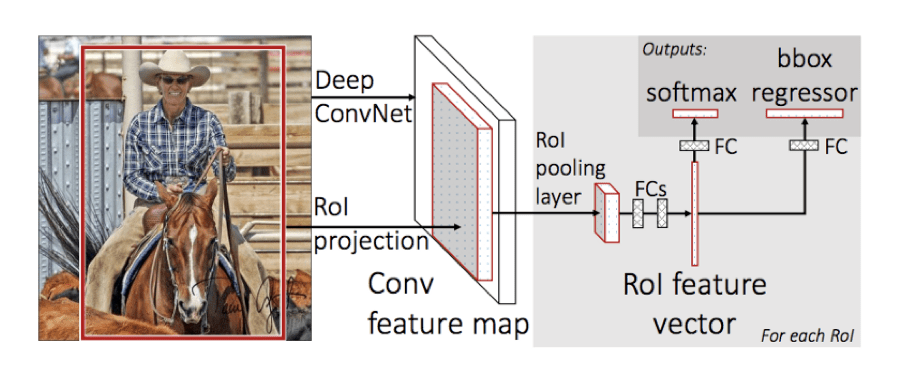
Los **bounding boxes** son rectángulos que enmarcan objetos de diferentes clases, y especifican su posición, clase y confianza. Cada clase representa un grupo semántico representativo de un objeto, animal, humano o cosas ya establecidos, como por ejemplo perro, gato, silla, casa o árbol.

El modelo **R-CNN** (Regions with CNN Features) se usa para la detección de objetos en imágenes y funciona de manera similar al Faster R-CNN. La principal diferencia es que en lugar de emplear una red neuronal convolucional completa, emplea una CNN preentrenada para extraer características de las regiones propuestas de la imagen. Este modelo fue superado por el Faster R-CNN en términos de precisión y velocidad, sigue siendo una opción viable para la detección de objetos en imágenes.

Por otro lado, está el modelo Faster R-CNN para la Detección de Objetos en un Dataset Personalizado. Esta arquitectura utiliza la técnica de búsqueda selectiva para unir las propuestas de región, y cuenta con un módulo de agrupación de regiones de interés, denominado Rool Pooling Layer. Este modelo extrae una ventana de tamaño fijo del mapa de características y usa dichas características para obtener la etiqueta de la clase final, permitiendo el entrenamiento completo de la red neuronal, con información de entrada en forma de bounding boxes. El Rool Pooling genera un vector de características de tamaño fijo, que se dirige a una capa densa para su procesamiento.



*Ilustración 2 R-CNN Algoritmo de detección de objetos*



*Ilustración 3 Faster R-CNN Algoritmo de detección de objetos*

La Arquitectura del modelo **Faster R-CNN** incluye el **Regional Proposal Network** (RPN), que se encarga de generar propuestas de regiones y evitar el uso de algoritmos de búsqueda selectiva. Con la implementación del RPN, la red puede operar a una velocidad de 7 a 10 fps, lo que permite realizar la Detección de Objetos en tiempo real mediante el empleo del Deep Learning. Microsoft es un ejemplo de cómo emplear esta técnica para detección de objetos, y se destaca la rapidez de la red [11].

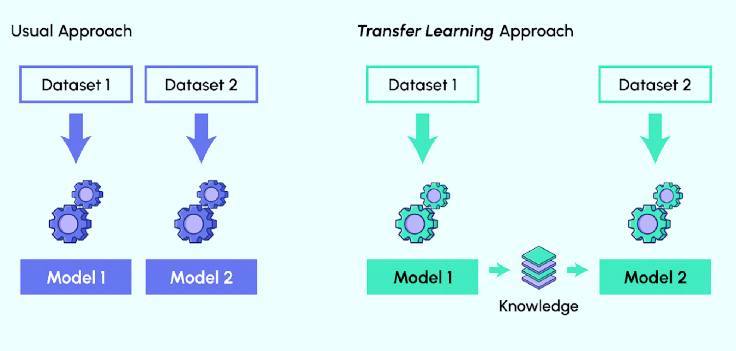
Además, ResNet es una red neuronal convolucional que se ha destacado por su eficacia en la resolución de problemas de aprendizaje profundo. En una red convolucional, las capas se apilan y se entrenan para aprender características de nivel bajo, medio y alto. En el aprendizaje residual, en lugar de aprender características, se aprenden residuos, que se pueden entender como la sustracción de la característica aprendida de la entrada de esa capa. ResNet utiliza conexiones de acceso directo para esto, conectando directamente la entrada de la enésima capa a alguna capa (n + x). Se ha demostrado que entrenar estas redes es más fácil que entrenar redes convolucionales profundas simples, y se resuelve el problema de la precisión degradante [12]. ResNet50 es una variante de esta red que tiene 48 capas de convolución, una capa MaxPool y una capa de Average Pool. Cuenta con 3.8 x 10^9 operaciones de puntos flotantes [13]. ResNet-101 es una red neuronal convolucional de 101 capas de profundidad, que puede cargar una versión previamente entrenada de la red en más de un millón de imágenes de la base de datos ImageNet. La red previamente entrenada puede clasificar imágenes en 1000 categorías de objetos y ha aprendido ricas representaciones de características para una amplia gama de imágenes. El tamaño de entrada de la imagen es de 224 por 224 [14]. Estos modelos son útiles para la inferencia inmediata en categorías ya presentes en los conjuntos de datos, así como para el Transfer Learning en nuevos conjuntos de datos. Los códigos de ambos modelos se pueden encontrar en https://keras.io/api/applications/resnet/, la página web de Keras.

El concepto de **Transfer Learning** es útil para aprovechar modelos preentrenados como ResNet y aplicarlos a nuevos datasets, lo que permite lograr una inferencia inmediata para categorías que ya están en los conjuntos de datos, así como para modelos de dataset nuevos para la red [15].

## Transfer Learning

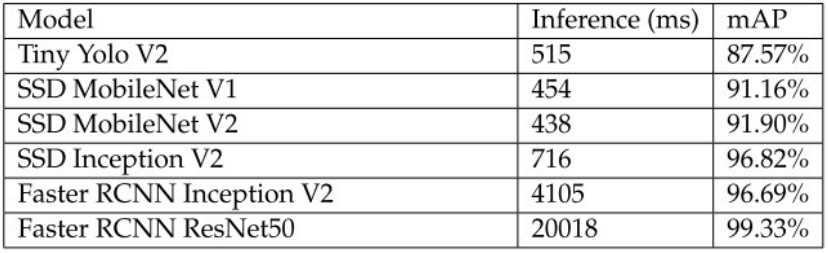
El concepto de Transfer Learning se refiere a la capacidad de transferir el conocimiento aprendido por un modelo previamente entrenado a un nuevo modelo. Cuando se entrena un modelo desde cero, ya sea una red neuronal convolucional, un modelo de regresión o un MLP, los pesos y valores de los parámetros se inicializan aleatoriamente y se entrena el modelo desde el principio durante un número de épocas hasta obtener una solución. Con Transfer Learning, se aprovecha el conocimiento ya adquirido por otros modelos y se utiliza como punto de partida en lugar de iniciar el entrenamiento desde cero. Esto es útil tanto en modelos propios como en modelos desarrollados por otras compañías como Nvidia. La transferencia de conocimiento de modelos ya creados puede emplearse para clasificar imágenes personalizadas y solucionar problemas de manera más efectiva.

Según Datascientest, el Transfer Learning, o aprendizaje transferido en español, se refiere al conjunto de métodos que permiten transferir conocimientos adquiridos de la resolución de problemas para resolver otros problemas. Frecuentemente, los modelos empleados en este campo necesitan grandes tiempos de cálculo y muchos recursos. Sin embargo, al usar modelos preentrenados como punto de partida, el Transfer Learning permite desarrollar rápidamente modelos eficaces y resolver problemas complejos de visión artificial o procesamiento del lenguaje natural.



*Ilustración 4 Enfoque tradicional vs enfoque de Tranfer Learning*

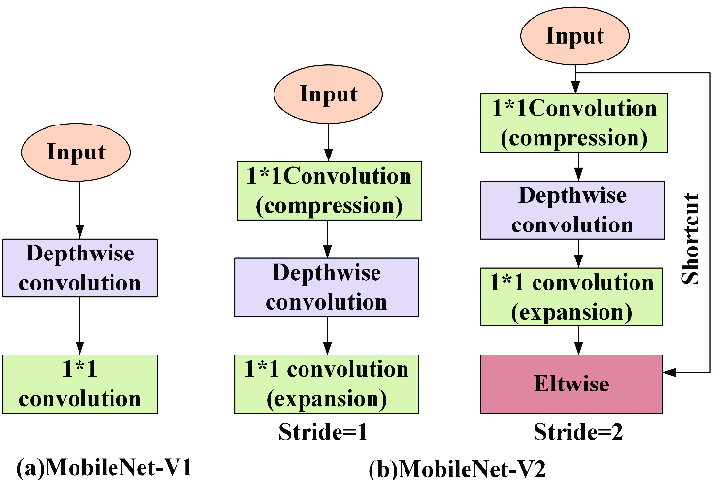
## Selección del modelo



*Tabla 1 Tiempos de inferencia y mAP*

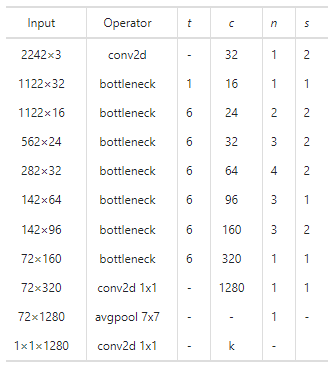
Para realizar la selección del modelo preentrenado se deben considerar tres parámetros importantes. En primer lugar, la velocidad de predicción, que hace referencia a la rapidez con la que el modelo puede procesar las imágenes y dar una respuesta. En segundo lugar, el COCO mAP o Mean Average Precision, que es una métrica utilizada para evaluar los modelos de detección de objetos como Fast R-CNN, YOLO, Mask R-CNN, entre otros. Esta métrica se calcula como la media de los valores de precisión media (AP) para los valores de recall desde 0 hasta 1 [16]. Finalmente, el output o salida del modelo es la información que el modelo proporciona, en el caso de la detección de objetos, se busca obtener los cuadros delimitadores (boxes) que indican la ubicación de los objetos detectados en la imagen.

Se decidió usar los modelos **SSD MobileNet v2 320x320** y **SSD MobileNet V1 FPN 640x640** debido a que los modelos Faster R-CNN presentaron errores durante la implementación y no se logró corregirlos. A pesar de que estos modelos no son los más precisos y veloces, tienen un rendimiento relativamente bueno. Por ejemplo, el modelo SSD MobileNet V1 FPN 640x640 tiene una precisión de 29.1, que se asemeja a algunos modelos basados en R-CNN. Estos modelos usan el modelo base MobileNet como columna vertebral para la extracción de funciones, el cual cuenta con un módulo mejorado con estructura residual invertida que permite eliminar las no linealidades en las capas estrechas, logrando prestaciones de última generación para la detección de objetos y la segmentación semántica [17].



*Ilustración 5 Estructura de MobileNet-V1 y MobileNet-V2*

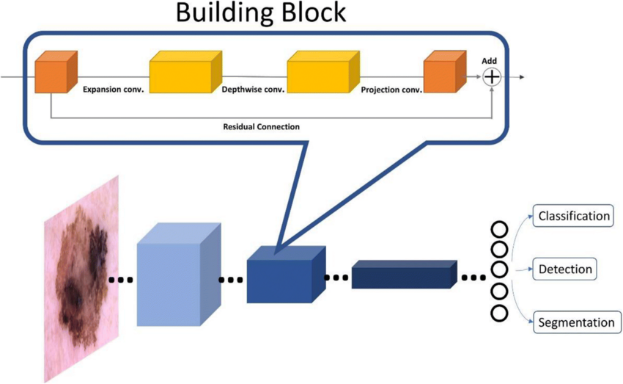
El funcionamiento de la arquitectura de MobileNetV2 se da de la siguiente manera desde la caracterización de sus parámetros:



*Tabla 2 Arquitectura de MobileNetV2*

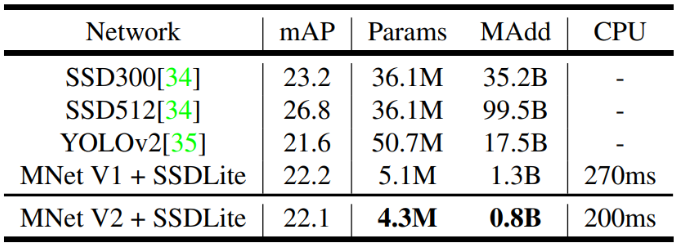
La tabla anterior ilustra los parámetros que se utilizan en la convolución, donde “t” corresponde al factor de expansión, “c” al número de canales de salida, “n” al número de repeticiones y “s” a la zancada. Además, se especifica que se usan núcleos de 3x3 para la convolución espacial.

Usualmente, la red principal de MobileNetV2 tiene un costo computacional de 300 millones de multiplicaciones y emplea 3,4 millones de parámetros, con una resolución de entrada de 224 x 224 y un multiplicador de ancho de 1. Sin embargo, se exploran las compensaciones de rendimiento para diferentes resoluciones de entrada (de 96 a 224) y multiplicadores de ancho (de 0,35 a 1,4), lo que afecta el costo computacional y el tamaño del modelo. Para la red con las características mencionadas, el costo computacional es de 585 millones de Madds, mientras que el tamaño del modelo varía entre 1,7 millones y 6,9 millones de parámetros. En cuanto al entrenamiento de la red, se emplean 16 GPU y un tamaño de lote de 96. MobileNetV2 presenta una representación visual de sus procesos de la siguiente manera:



*Ilustración 6 Building Block de MobileNetV2*

La MobileNetV2 es una arquitectura efectiva para la detección de objetos, lo cual se demuestra al agregarle el SSDlite. Al evaluar esta combinación en el conjunto de datos de MS COCO en comparación con otras arquitecturas, se obtiene una precisión comparable con menos parámetros y menor complejidad computacional.



*Tabla 3 Comparación de MobileNetV2 + SSDLite con otros detectores de objetos en tiempo real en el dataset COCO*

El gráfico anterior ilustra que el tiempo de procesamiento de inferencia es más rápido en comparación con MobileNetV1. Específicamente, la combinación de MobileNetV2 con SSDLite es 20 veces más eficiente y 10 veces más pequeña; y, sigue superando a YOLOv2 en el conjunto de datos COCO [18].

# DESCRIPCIÓN DEL PROBLEMA

Resumidamente, el equipo ha decidido enfocarse en implementar una red neuronal convolucional para la detección de fauna silvestre en Colombia. Se define la fauna silvestre como aquellos organismos vivos de especies animales terrestres y acuáticas que no han sido criados, mejorados genéticamente o domesticados, o que han vuelto a su estado salvaje; esto según la Ley 611 de 2000 del Ministerio de Medio Ambiente [19]. Teniendo en cuenta lo anterior, se identifican varias características únicas del ecosistema en Colombia.

* No han sufrido procesos de manipulación genética o reproductiva por parte del ser humano.
* Su desarrollo evolutivo y adaptaciones al medio ambiente no han tenido intervención por parte del ser humano.
* Su sistema inmunológico se desarrolla de forma natural sin que sea estimulado por el ser humano.
* Su reproducción se da de forma natural en su hábitat sin ninguna intervención humana.
* Su comportamiento está condicionado por genética y aprendizaje según las características de su entorno, sin ninguna intervención del ser humano.
* Su supervivencia no depende del ser humano, así se hayan acostumbrado a estar en contacto continuo con él.

Con el objetivo de obtener información significativa sobre la situación de la fauna silvestre en Colombia, se busca una noticia publicada en el Ministerio de Ambiente de Colombia que enumera los tipos de delitos ambientales más frecuentes en el país y las sanciones correspondientes, incluyendo aquellos relacionados con la caza y el tráfico ilegal de animales [20].

**Aprovechamiento ilícito**. Quien se apropie, acceda, capture, mantenga, introduzca, extraiga, explote, aproveche, exporte, transporte, comercie, explore, trafique o de cualquier otro modo se beneficie de los especímenes, productos o partes de los recursos fáunicos, forestales, florísticos, hidrobiológicos, corales, biológicos o genéticos de la biodiversidad colombiana, incurrirá en prisión de sesenta (60) a ciento treinta y cinco (135) meses, y multa de ciento treinta y cuatro (134) a cuarenta y tres mil setecientos cincuenta (43.750) salarios mínimos legales mensuales vigentes.

**Tráfico de fauna.** El que trafique, adquiera, exporte o comercialice, sin permiso de la autoridad competente o con incumplimiento de la normatividad existente, los especímenes, productos o partes de la fauna acuática, silvestre o especies silvestres exóticas, incurrirá en prisión de sesenta (60) a ciento treinta y cinco (135) meses, y multa de trescientos (300) hasta cuarenta mil (40.000) salarios mínimos legales mensuales vigentes.

**Deforestación.** El que, sin permiso de autoridad competente o con incumplimiento de la normatividad existente, tale, queme, corte, arranque o destruya áreas iguales o superiores a una hectárea continua o discontinua de bosque natural, incurrirá en prisión de sesenta (60) a ciento cuarenta y cuatro (144) meses, y multa de ciento treinta y cuatro (134) a cincuenta mil (50.000) salarios mínimos legales mensuales vigentes.

**Daños en los recursos naturales y ecocidio.** Quien destruya, inutilice, haga desaparecer o cause un impacto ambiental grave o de cualquier otro modo dañe los recursos naturales, incurrirá en prisión de sesenta (60) a ciento treinta y cinco (135) meses, y multa de ciento sesenta y siete (167) a dieciocho mil setecientos cincuenta (18.750) salarios mínimos legales mensuales vigentes.

**Contaminación ambiental.** El que contamine, provoque o realice directa o indirectamente emisiones, vertimientos, radiaciones, ruidos, depósitos o disposiciones al aire, la atmósfera o demás componentes del espacio aéreo, el suelo, el subsuelo, las aguas superficiales, marítimas o subterráneas o demás recursos naturales en tal forma que contamine o genere un efecto nocivo en el ambiente, que ponga en peligro la salud humana y los recursos naturales, incurrirá en prisión de sesenta y nueve (69) a ciento cuarenta (140) meses, y multa de ciento cuarenta (140) a cincuenta mil (50.000) salarios mínimos legales mensuales vigentes.

El enfoque del mini proyecto estará en la caza y el tráfico ilegal de fauna silvestre debido a la grave situación que se presenta en Colombia, donde 234 especies de aves, 76 de mamíferos, 27 de reptiles y 9 de anfibios han sido afectadas. Además, Colombia ocupa el segundo lugar en el tráfico de especies a nivel global y es responsable de la disminución de muchas poblaciones animales [21]. Esto se debe en gran parte a que se comercializan más de 6.000 millones de animales silvestres anualmente para satisfacer la demanda de animales vivos para mascotas en hogares, zoológicos y laboratorios en todo el mundo [22].

Se llevará a cabo un Modelo de Detección de Animales con el propósito de controlar la red trófica y monitorear los ecosistemas para proteger a los animales en peligro de extinción. Este modelo permitirá ubicar el bounding box de los animales y calcular la distancia entre ellos, lo que ayudará a tomar decisiones en dos aspectos: el Control de Animales en vía de extinción y el Control de la Cadena Alimenticia. Es importante que los seres humanos prevengan y reduzcan la alteración de las cadenas alimenticias, por ejemplo, evitando la tala de árboles, reduciendo la contaminación de fuentes hídricas, no causando incendios en áreas verdes y respetando los tiempos de animales acuáticos. Con lo anterior, se espera tener un algoritmo capaz de detectar los diferentes animales silvestres, que se han subdividido en seis clases:

| **Animal** | **Rol** | **Descripción** |
| --- | --- | --- |
| Caimán Negro | Depredador/Presa | Este animal habita en cuerpos de agua dulce como lagos y ríos, especialmente en la cuenca del Amazonas en Colombia, Ecuador y Perú. Es considerado el depredador más grande en estos humedales, y su capacidad de caza es apoyada por su aguda vista y audición, lo que le permite cazar durante la noche. Sin embargo, su existencia está en peligro debido a la caza ilegal y la pérdida y fragmentación de su hábitat natural. |
| Guacamaya Roja | Presa | Las guacamayas se encuentran en diferentes regiones de Colombia, incluyendo la planicie del Caribe y la región Orinoco-Amazónica, así como los valles interandinos del Cauca y el Magdalena, la región pacífica del Darién colombiano y las estribaciones de las cordilleras. Su alimentación se basa principalmente en semillas, y para reproducirse necesitan árboles o palmas huecas en los bosques. |
| Iguana | Presa | En Colombia, la iguana verde vive en los bosques cercanos a cuerpos de agua y también puede habitar en bosques secos, de galería, sabanas con poca vegetación arbórea e incluso en islas o costas con vegetación arbustiva. Sin embargo, esta especie es una de las más traficadas en el país, tanto para su tenencia como para su comercialización. Muchas iguanas son vendidas para su consumo humano, no solo por su carne, sino también por sus huevos, que pueden llegar a ser hasta 50 por hembra. Además, sus partes son utilizadas como carnada para la pesca o para la fabricación de productos de piel. |
| Jaguar | Depredador | La población mundial de jaguares ha disminuido en un 45%, y en algunos países, como Estados Unidos, El Salvador y Uruguay, se ha declarado extinto. En Colombia, las poblaciones más grandes de jaguares se encuentran en la región del Amazonas, la Orinoquia, Chocó y el Caribe. Debido a que esta especie necesita grandes extensiones de tierra para sobrevivir, la pérdida de hábitat, el desarrollo de infraestructura, la expansión de las actividades agrícolas y ganaderas, y la pérdida de vegetación son las principales amenazas que enfrenta. |
| Ocelote | Depredador | El ocelote vive en diferentes ecosistemas de baja altura en Colombia, siempre por debajo de los 2,000 metros sobre el nivel del mar. A menudo comparte su hábitat con jaguares, pumas y margays. A diferencia del puma y el jaguar, el ocelote prefiere usar caminos y senderos en lugar de marcar territorio con rascas. En el pasado, esta especie estuvo en peligro por la caza furtiva para obtener su piel manchada. En las décadas de los 60 y 70, miles de individuos fueron cazados por indígenas y colonos, especialmente en las regiones de la Orinoquia y la Amazonia. |
| Puma | Depredador | Se trata de un hermoso felino de color leonado que puede ser encontrado en bosques secos y húmedos, así como en el páramo, y puede habitar prácticamente cualquier hábitat, excepto aquellos que son inundables. Generalmente, habitan bosques maduros y bosques intervenidos. Sin embargo, la intervención humana ha llevado a una disminución en sus principales presas que son los roedores, lo que ha permitido que los cazadores ilegales o traficantes se acerquen a ellos. |

*Tabla 4 Dataset*

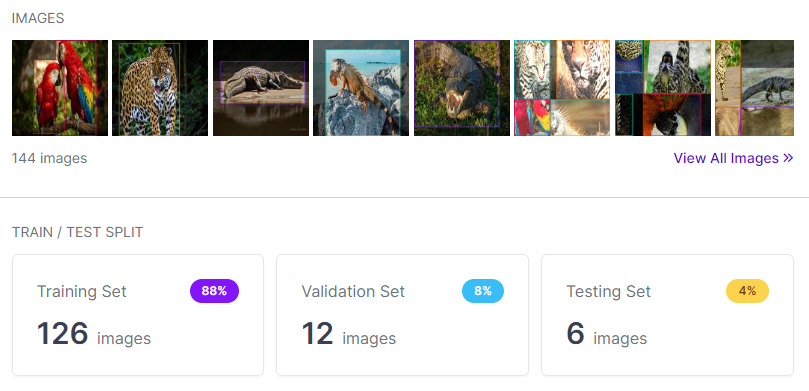
## Desarrollo del Etiquetado con Roboflow

“Con Roboflow, los clientes pueden anotar imágenes mientras evalúan la calidad de los conjuntos de datos para prepararlos para la capacitación. (La mayoría de los algoritmos de visión artificial requieren etiquetas que esencialmente «enseñan» al algoritmo a clasificar objetos, lugares y personas). La plataforma permite a los desarrolladores experimentar para generar nuevos datos de entrenamiento y ver qué configuraciones conducen a un mejor rendimiento del modelo. Una vez que se completa el entrenamiento, Roboflow puede implementar el modelo en la nube, el borde o el navegador y monitorear el modelo en busca de casos extremos y degradación con el tiempo”



*Ilustración 7 Etiquetado*

En Roboflow, se obtuvo un archivo TFRecord. El etiquetado y entrenamiento se realizaron con un total de 126 imágenes. De estas imágenes, se dividió un porcentaje para el entrenamiento, validación y prueba del modelo.



*Ilustración 8 Validación*

Una característica relevante de esta herramienta es que permite realizar una preparación previa de las imágenes mediante el redimensionamiento (resize with edge) de las mismas, dejándolas en un tamaño uniforme requerido por cada modelo de MobileNet.

## Planteamiento del enunciado

La Guía del Miniproyecto del Curso, dirigida por el profesor Andrés Felipe Escobar, sugiere entrenar una Red Neuronal Convolucional para resolver un problema de detección de objetos, segmentación (semántica o de instancia) o estimación de postura utilizando imágenes como datos de entrada. Para este fin, se sugiere utilizar las herramientas de Deep Learning Keras y/o Tensorflow.

## Metodología

Se llevó a cabo una búsqueda exhaustiva de modelos en la API de TensorFlow para el proyecto. Aunque se intentó implementar tres modelos, ninguno de ellos dio resultados satisfactorios debido a conflictos entre Colab y la versión de TensorFlow. Por esta razón, se optó por utilizar el modelo SSD MobileNet V1 FPN 640x640 y entrenarlo con imágenes redimensionadas de 640x640 obtenidas a través de Roboflow. La metodología del proyecto se desarrolló en base a los pasos necesarios para entrenar el modelo, incluyendo el redimensionamiento de las imágenes. Los pasos se presentan a continuación:

1. Definición de las imágenes y clases que se va a usar en el proyecto.
2. Etiquetado de las imágenes para el proceso de entrenamiento de la red.
3. Entrenamiento del modelo de red neuronal a utilizar, utilizando alguno de los múltiples tutoriales en internet.
4. Validación del modelo de red neuronal utilizado
5. Explicación de la arquitectura de red neuronal usada.

# DESARROLLO DE LA PROPUESTA

En el proceso de llevar a cabo el proyecto, se emplearon dos cuadernos (Notebooks) disponibles en Google Colab, los cuales fueron proporcionados por el profesor a cargo del curso.

## Entrenamiento del modelo

El modelo puede ser encontrado en el siguiente enlace:

[Enlace del entrenamiento del modelo](https://drive.google.com/file/d/16Gf9jVpe_RMO5CXxBHCdFYYs37nZVuk4/view?usp=share_link)

Al principio del proceso, se importaron e instalaron las bibliotecas necesarias para el reentrenamiento de los modelos y para manipular los componentes, como los directorios y los archivos del sistema operativo.

import os

import pandas as pd

import json

import shutil

!pip install tf\_slim

Después de eso, se procede a obtener el repositorio "models" de Tensorflow y se accede a la carpeta "research".

%cd /content

!git clone --quiet https://github.com/tensorflow/models.git

%cd /content/models/

%cd /content/models/research

!protoc object\_detection/protos/\*.proto --python\_out=.

os.environ['PYTHONPATH'] += ':/content/models/research/:/content/models/research/slim/'

Luego, se procede a obtener el modelo necesario para el proyecto. Para ello, se accede al enlace del repositorio llamado "Tensorflow 2 Detection Model Zoo" en GitHub y se guarda en la carpeta "content".

!wget --no-check-certificate http://download.tensorflow.org/models/object\_detection/tf2/20200711/ssd\_mobilenet\_v1\_fpn\_640x640\_coco17\_tpu-8.tar.gz \

-O /content/ssd\_mobilenet\_v1\_fpn\_640x640\_coco17\_tpu-8.tar.gz

Una vez que se ha descargado el modelo, el siguiente paso es descomprimir los archivos y guardarlos en una carpeta con el nombre del modelo dentro de la carpeta de investigación.

!tar -zxvf /content/ssd\_mobilenet\_v1\_fpn\_640x640\_coco17\_tpu-8.tar.gz

output\_path = 'ssd\_mobilenet\_v1\_fpn\_640x640\_coco17\_tpu-8'

output\_path = os.path.join(os.getcwd(), output\_path)

print("La carpeta se almacenó en {}".format(output\_path))

Se genera un directorio donde se guardarán los archivos de configuración del pipeline

path\_training = '/content/ssd\_mobilenet'

os.mkdir(path\_training)

Se mueven los archivos de configuración a la carpeta ssd\_mobilenet

source\_config = "{}/pipeline.config".format(output\_path)

target\_config = "{}/pipeline.config".format(path\_training)

shutil.copyfile(source\_config, target\_config)

Ahora, se realiza la importación de ciertas librerías necesarias para la ejecución del pipeline y el entrenamiento.

Se guarda el archivo de configuración en una variable llamada "config" y se muestra en pantalla. En la configuración se puede ver que el modelo tenía 90 clases y había sido entrenado con 25000 pasos.

config = config\_util.get\_configs\_from\_pipeline\_file(target\_config)

Se lleva a cabo la configuración del archivo de configuración para permitir su edición posterior con el número actualizado de clases, datos de entrenamiento y otros parámetros.

pipeline\_config = pipeline\_pb2.TrainEvalPipelineConfig()

with tf.io.gfile.GFile(target\_config, "r") as f:

  proto\_str = f.read()

  text\_format.Merge(proto\_str, pipeline\_config)

Se almacena la ruta de los archivos de entrenamiento y validación, correspondiente a los TFRecord y Label Map.

label\_map\_pbtxt\_fname = "/content/drive/MyDrive/Parcial2/TFRecord/animals\_label\_map.pbtxt"

train\_record\_fname = "/content/drive/MyDrive/Parcial2/TFRecord/train/animals.tfrecord"

test\_record\_fname = "/content/drive/MyDrive/Parcial2/TFRecord/valid/animals.tfrecord"

Luego se editan los datos del archivo de configuración del pipeline para indicar el uso de 6 clases, que son Jaguar, Ocelote, Puma, Caimán, Guacamayo e Iguana. Además, se especifica el tamaño del lote de datos, los checkpoints del modelo y se proporciona la ruta de los archivos de validación y entrenamiento, que son los TFRecord y label Map.

pipeline\_config.model.ssd.num\_classes = 6

pipeline\_config.train\_config.batch\_size = 4

pipeline\_config.train\_config.fine\_tune\_checkpoint = "{}/checkpoint/ckpt-0".format(output\_path)

pipeline\_config.train\_config.fine\_tune\_checkpoint\_type = "detection"

pipeline\_config.train\_input\_reader.label\_map\_path = label\_map\_pbtxt\_fname

pipeline\_config.train\_input\_reader.tf\_record\_input\_reader.input\_path[0] = train\_record\_fname

pipeline\_config.eval\_input\_reader[0].label\_map\_path = label\_map\_pbtxt\_fname

pipeline\_config.eval\_input\_reader[0].tf\_record\_input\_reader.input\_path[0] = test\_record\_fname

Se realiza otros ajustes a los archivos de configuración.

config\_text = text\_format.MessageToString(pipeline\_config)

with tf.io.gfile.GFile(target\_config, "wb") as f:

  f.write(config\_text)

Se instalan otras librerías necesarias para el reentrenamiento del modelo.

!pip install lvis

!pip install tf-models-official

!pip install tensorflow-io

Con la personalización de los archivos de configuración, se procede a reentrenar el modelo con 5000 épocas, en el cual se guardan los checkpoints del entrenamiento en una carpeta dentro del drive.

num\_steps = 5000

model\_dir = "/content/drive/MyDrive/Parcial2/Models\_saved"

!python /content/models/research/object\_detection/model\_main\_tf2.py \

--pipeline\_config\_path={target\_config} \

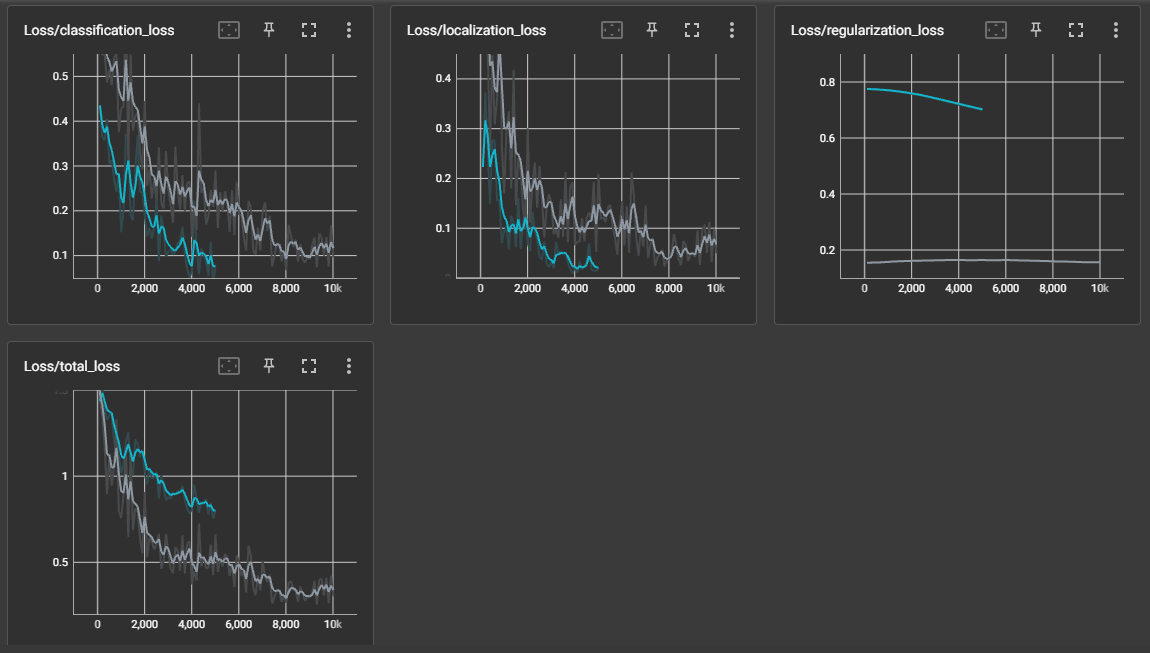
--model\_dir={model\_dir} \

--num\_train\_steps={num\_steps}

Luego de haber reentrenado el modelo, se carga la información del entrenamiento en Tensorboard. En la figura 9, se muestra que, en tan solo 5000 iteraciones, el modelo logra tener valores de pérdida en la clasificación y localización bastante pequeños, alrededor de 0.05 y 0.02 respectivamente.

%reload\_ext tensorboard

%tensorboard --logdir "/content/drive/MyDrive/Parcial2/Models\_saved"



*Ilustración 9 Pérdidas generadas por el tensorboard*

Con el modelo ya entrenado, se procede a salvarlo o guardarlo con el fin de poder utilizarlo en las etapas de inferencia y posteriores aplicaciones de ser necesario.

model\_dir = "/content/drive/MyDrive/Parcial2/Models\_saved/v3\_640X640/checkpoint1"

output\_directory = "/content/drive/MyDrive/Parcial2/Models\_saved/v3\_640X640/Model\_3"

!python /content/models/research/object\_detection/exporter\_main\_v2.py \

 --input\_type image\_tensor \

 --pipeline\_config\_path {target\_config} \

 --trained\_checkpoint\_dir {model\_dir} \

 --output\_directory {output\_directory}

Finalmente, se comprime el modelo sintonizado o afinado (fine tuned) en formato zip.

!zip -r /content/drive/MyDrive/Parcial2/Models\_saved/v3\_640X640/Model\_3/fine\_tuned\_model3.zip /content/drive/MyDrive/Parcial2/Models\_saved/v3\_640X640/Model\_3

## Inferencia

Se puede encontrar el desarrollo en el siguiente enlace:

[Enlace de la inferencia del modelo](https://drive.google.com/file/d/1oVCpDEt9S6ypc65dMZA9S_hl0lx5EtzZ/view?usp=share_link)

Inicialmente, se instalan las librerías necesarias para el proceso de inferencia del modelo reentrenado.

!pip install tf\_slim

!pip install lvis

!pip install tf-models-official

import os

De igual forma, se instalan ciertas librerías, repositorios y se realiza el desplazamiento a la carpeta de research.

%cd /content

!git clone --quiet https://github.com/tensorflow/models.git

%cd /content/models/

!apt-get update && apt-get install -y -qq protobuf-compiler python-pil python-lxml python-tk

!pip install -q Cython contextlib2 pillow lxml matplotlib

!pip install -q pycocotools

%cd /content/models/research

!protoc object\_detection/protos/\*.proto --python\_out=.

os.environ['PYTHONPATH'] += ':/content/models/research/:/content/models/research/slim/'

!python object\_detection/builders/model\_builder\_test.py

En esta sección, se importan las librerías necesarias para la inferencia del modelo y la visualización de imágenes.

import cv2

import zipfile

from object\_detection.utils import label\_map\_util

from object\_detection.utils import visualization\_utils as viz\_utils

import tensorflow as tf

import numpy as np

from PIL import Image

import matplotlib.pyplot as plt

from google.colab.patches import cv2\_imshow

Posteriormente, se descomprime el modelo sintonizado o afinado y se ubica en el directorio de content.

path = '/content/drive/MyDrive/Parcial2/Models\_saved/v3\_640X640/Model\_3/fine\_tuned\_model3.zip'

zip\_ref = zipfile.ZipFile(path, "r")

zip\_ref.extractall("/content/fine\_tuned\_model")

zip\_ref.close()

Se carga el modelo reentrenado.

PATH\_TO\_MODEL\_DIR = "/content/fine\_tuned\_model/content/drive/MyDrive/Parcial2/Models\_saved/v3\_640X640/Model\_3"

PATH\_TO\_SAVE\_MODEL = PATH\_TO\_MODEL\_DIR + "/saved\_model"

detect\_fn = tf.saved\_model.load(PATH\_TO\_SAVE\_MODEL)

Se almacena en una variable la ruta del label map y se obtienen los índices de las clases a partir de este archivo.

PATH\_TO\_LABELS = "/content/drive/MyDrive/Parcial2/TFRecord/v3\_640x640/animals\_label\_map.pbtxt"

category\_index = label\_map\_util.create\_category\_index\_from\_labelmap(PATH\_TO\_LABELS, use\_display\_name=True)

Se define una función que tiene como objetivo leer una imagen ubicada en una ruta determinada. Una vez leída, se convierte en un arreglo y luego en un tensor para proceder a realizar la detección correspondiente.

def read\_image(path):

  image\_path = path

  image\_np = np.array(Image.open(image\_path))

  input\_tensor = tf.convert\_to\_tensor(image\_np)

  input\_tensor =  input\_tensor[tf.newaxis, ...]

  detections = detect\_fn(input\_tensor)

  num\_detections = int(detections.pop('num\_detections'))

  return detections, num\_detections, image\_np

Se crea otra función, la cual se encarga de hacer posible la visualización de la detección realizada, ubicando los bounding boxes en los respectivos objetos.

def show\_detection(detections, num\_detections, image\_np):

  detections2 = {key: value[0,:num\_detections].numpy() for key, value in detections.items()}

  detections2['num\_detections'] = num\_detections

  detections2['detection\_classes'] = detections2['detection\_classes'].astype(np.int64)

  # Copy image and draw bounding-boxes

  image\_np\_with\_detections = image\_np.copy()

  # print(np.array(image\_np\_with\_detections).shape)

  # Use object detection library to show the classification and boundingboxes

  viz\_utils.visualize\_boxes\_and\_labels\_on\_image\_array(

      image\_np\_with\_detections,

      detections2['detection\_boxes'],

      detections2['detection\_classes'],

      detections2['detection\_scores'],

      category\_index,

      max\_boxes\_to\_draw=15,

      min\_score\_thresh=0.40,

      use\_normalized\_coordinates = True

  )

  # Detection result

  image\_np\_with\_detections = cv2.cvtColor(image\_np\_with\_detections, cv2.COLOR\_BGR2RGB)

  return image\_np\_with\_detections

La última etapa del proceso consiste en realizar la detección de objetos en un conjunto de imágenes almacenadas en una carpeta específica, la cual se especifica mediante la variable "folder\_path". A partir de la concatenación del nombre de las 50 imágenes y la dirección de la carpeta, se lleva a cabo la detección de objetos y se muestra el resultado en una figura.

folder\_path = "/content/drive/MyDrive/Parcial2/Imagenes"

#path = "/content/images/frame\_1-4\_3516.jpg"

count = 0

for im in os.listdir(folder\_path):

  path = folder\_path + '/' + im

  print(im)

  (detections, num\_detections, image\_np) = read\_image(path)

  image\_np\_with\_detections = show\_detection(detections, num\_detections, image\_np)

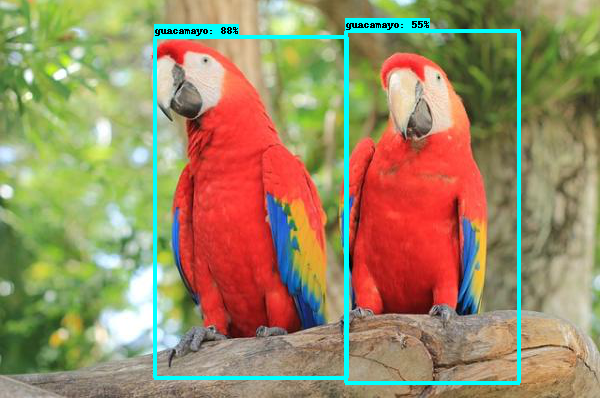
  plt.figure()

  cv2\_imshow(image\_np\_with\_detections)

  if count == 50:

    break

  count += 1



*Ilustración 10 Detección realizada por el modelo*

# RESULTADOS

Se llevará a cabo una evaluación de los resultados del modelo a través de la obtención de imágenes generadas mediante la inferencia del modelo. La evaluación se centrará en la precisión del modelo con relación a la cantidad de imágenes usadas en el entrenamiento, validación y prueba. También se analizarán los casos en los que el modelo presenta fallos, como confusión, focos nulos o muertos y escenas donde el modelo falla con mayor frecuencia.

**Clases bien detectadas**. Se estableció un criterio para determinar que la detección fue satisfactoria, el cual consistió en que la precisión de la inferencia fuera mayor al 80%. De esta forma, algunas de las clases que cumplieron con este criterio fueron:



*Ilustración 11 Clase bien detectada*



*Ilustración 12 Clase bien detectada*



*Ilustración 13 Clase bien detectada*



*Ilustración 14 Clase bien detectada*

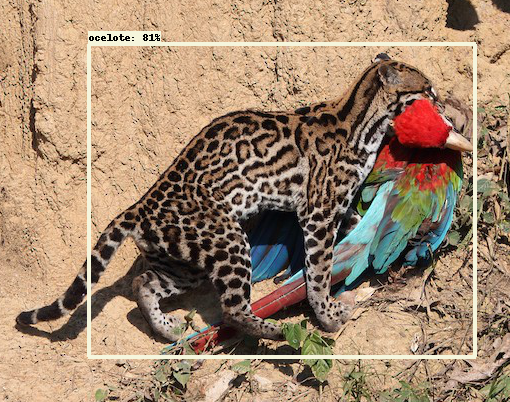
En la mayor parte de los casos, las clases bien detectadas fueron representadas por imágenes con condiciones óptimas y de buena calidad.

**Clases regularmente detectadas**. Algunas imágenes en las que se logró detectar una clase de manera considerablemente precisa, mientras que la otra clase tuvo una precisión menor al 80%. Una posible explicación de esto podría ser que se prestó mayor atención a las características físicas y corporales de los felinos en muchas imágenes, lo que no sucedió de igual manera con la Iguana ni el Caimán. A pesar de que se obtuvieron buenos resultados, se reconoce que se podría mejorar el nivel de inferencia para lograr una mayor precisión en la predicción de la localización y clasificación de las clases.



*Ilustración 15 Clase regularmente detectada*

Detección nula. Refiriéndose a las detecciones en las que el modelo no pudo identificar ningún objeto específico, es decir, cuando se detectan múltiples objetos que no están presentes en una imagen o cuando el modelo confunde completamente un objeto de una clase con otro. En este caso, no fue posible detectar el guacamayo en la imagen.



*Ilustración 16 Detección nula*

# CONCLUSIÓN

El equipo ha adquirido más experiencia práctica en el manejo, manipulación y diseño de redes neuronales artificiales para resolver problemas de detección de objetos en situaciones de la vida real, específicamente en la predicción de la ubicación y la clasificación de animales salvajes en sus biomas terrestres utilizando un modelo de red preentrenado.

Se puede decir que se han ganado más habilidades prácticas en el manejo, manipulación y diseño de redes neuronales artificiales para resolver problemas de detección de objetos en situaciones de la vida real. En particular, han utilizado un modelo de red preentrenado para predecir la ubicación y la clasificación de animales salvajes en sus biomas terrestres. Aunque no pudieron implementar el modelo recomendado en la guía, lograron entrenar con éxito una red personalizada basada en el modelo SSD MobileNet V1 FPN 640x640 para detectar objetos.

Realizar el proyecto ha permitido adquirir conocimientos en el abordaje de problemas de detección mediante el Deep Learning. Los resultados obtenidos han demostrado la eficacia de la aplicación en la detección y clasificación de diferentes animales, lo que puede tener aplicaciones en el seguimiento de especies o en la evaluación de dinámicas tróficas dentro de una red. Estos avances son importantes porque existen aplicaciones que ya ejecutan un monitoreo de las especies y estos trabajos prácticos proporcionan herramientas adicionales para mejorar la competencia del educando en el mercado.

# Referencias

1. J. Snow, «Rangers Use Artificial Intelligence to Fight Poachers,» 12 Junio 2016. [En línea]. Available: https://www.nationalgeographic.com/animals/article/paws-artificial-intelligence-fights-poaching-ranger-patrols-wildlife-conservation. [Último acceso: 15 Abril 2023].
2. PyPI, «pandas 1.5.3,» [En línea]. Available: https://pypi.org/project/pandas/. [Último acceso: 15 Abril 2023].
3. PyPI, «numpy 1.24.2,» [En línea]. Available: https://pypi.org/project/numpy/. [Último acceso: 15 Abril 2023].
4. PyPI, «matplotlib 3.7.1,» [En línea]. Available: https://pypi.org/project/matplotlib/. [Último acceso: 15 Abril 2023].
5. «json — Codificador y decodificador JSON,» [En línea]. Available: https://docs.python.org/es/3/library/json.html. [Último acceso: 15 Abril 2023].
6. «shutil — Operaciones de archivos de alto nivel,» [En línea]. Available: https://docs.python.org/es/3/library/shutil.html. [Último acceso: 15 Abril 2023].
7. T. Gamauf, «Tensorflow Records? What they are and how to use them,» 20 Marzo 2018. [En línea]. Available: https://medium.com/mostly-ai/tensorflow-records-what-they-are-and-how-to-use-them-c46bc4bbb564. [Último acceso: 16 Abril 2023].
8. Aprende Machine Learning, «¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador,» 29 Noviembre 2018. [En línea]. Available: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/. [Último acceso: 16 Abril 2023].
9. «TensorFlow 2 Detection Model Zoo,» [En línea]. Available: https://github.com/tensorflow/models/blob/master/research/object\_detection/g3doc/tf2\_detection\_zoo.md. [Último acceso: 16 Abril 2023].
10. COCO, «What is COCO?,» [En línea]. Available: https://cocodataset.org/#home. [Último acceso: 16 Abril 2023].
11. Microsoft, «Detección de objetos mediante R-CNN más rápida,» 13 Octubre 2022. [En línea]. Available: https://learn.microsoft.com/es-es/cognitive-toolkit/object-detection-using-faster-r-cnn. [Último acceso: 17 Abril 2023].
12. «Residual Network (ResNet),» [En línea]. Available: https://iq.opengenus.org/resnet/. [Último acceso: 17 Abril 2023].
13. «Understanding ResNet50 architecture,» [En línea]. Available: https://iq.opengenus.org/resnet50-architecture/.
14. «resnet101,» [En línea]. Available: https://www.mathworks.com/help/deeplearning/ref/resnet101.html. [Último acceso: 17 Abril 2023].
15. DataScientest, «¿Qué es el Transfer Learning?,» [En línea]. Available: https://datascientest.com/es/que-es-el-transfer-learning. [Último acceso: 19 Abril 2023].
16. D. Shah, «Mean Average Precision (mAP) Explained: Everything You Need to Know,» 2 Marzo 2023. [En línea]. Available: https://www.v7labs.com/blog/mean-average-precision#:~:text=Mean%20Average%20Precision(mAP)%20is%20a%20metric%20used%20to%20evaluate,values%20from%200%20to%201. [Último acceso: 19 Abril 2023].
17. S.-H. Tsang, «Review: MobileNetV2 — Light Weight Model (Image Classification),» 19 Mayo 2019. [En línea]. Available: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c. [Último acceso: 19 Abril 2023].
18. «[CNN Networks] 13. MobileNet v2,» [En línea]. Available: https://velog.io/@woojinn8/LightWeight-Deep-Learning-7.-MobileNet-v2. [Último acceso: 19 Abril 2023].
19. GOV.CO, «¿Sabes qué es fauna silvestre?,» [En línea]. Available: https://www.ambientebogota.gov.co/fauna-silvestre. [Último acceso: 19 Abril 2023].
20. GOV.CO, «Recuerde cuáles son los delitos ambientales que se penalizan en Colombia,» 5 Enero 2022. [En línea]. Available: https://www.minambiente.gov.co/comunicado-de-prensa/recuerde-cuales-son-los-delitos-ambientales-que-se-penalizan-en-colombia/. [Último acceso: 19 Abril 2023].
21. “Tráfico ilegal de especies en Colombia - FundaciónProAves - por las aves y su hábitat en Colombia”.https://proaves.org/trafico-ilegal-de-especies-en-colombia/ [Último acceso: 19 Abril 2023].
22. “Tráfico ilegal - Secretaría Distrital de Ambiente”.https://www.ambientebogota.gov.co/trafico-ilegal. [Último acceso: 19 Abril 2023].