DESARROLLO DE UN SISTEMA DE DETECCIÓN DE ABEJAS PORTADORAS DE POLEN MEDIANTE EL PROCESAMIENTO DE IMÁGENES Y TÉCNICAS DE APRENDIZAJE PROFUNDO.

ANDRES FELIPE URBANO ESPINOSA

DIRECTOR

MAG. CARLOS IVÁN VÁSQUEZ VALENCIA

PHD (C). ANDRÉS FELIPE SOLIS PINO

CORPORACIÓN UNIVERSITARIA DE COMFACAUCA - UNICOMFACAUCA

FACULTAD DE INGENIERÍAS

INGENIERÍA MECATRÓNICA

POPAYÁN, CAUCA

2024

DESARROLLO DE UN SISTEMA DE DETECCIÓN DE ABEJAS PORTADORAS DE POLEN MEDIANTE EL PROCESAMIENTO DE IMÁGENES Y TÉCNICAS DE APRENDIZAJE PROFUNDO.

ANDRES FELIPE URBANO ESPINOSA

PROYECTO DE GRADO PRESENTADO A LA FACULTAD DE INGENIERÍAS PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO MECATRÓNICO

DIRECTOR

MAG. CARLOS IVÁN VÁSQUES VALENCIA

PHD (C). ANDRÉS FELIPE SOLIS PINO

CORPORACIÓN UNIVERSITARIA DE COMFACAUCA - UNICOMFACAUCA

FACULTAD DE INGENIERÍAS

INGENIERÍA MECATRÓNICA

POPAYÁN, CAUCA

2024

Nota de Aceptación

_____

_____

_____

_____

Presidente del Jurado

_____

_____

_____

Jurado

_____

_____

_____

_____

CONTENIDO

Lista de Figuras

Lista de tablas

Lista de ecuaciones

Resumen

La polinización es un proceso fundamental para la supervivencia de los ecosistemas. Se considera que el 90% de las plantas con flor depende de la polinización para reproducirse (formar semillas o frutas) siendo así indispensable para la biodiversidad. Uno de los principales polinizadores de la naturaleza son las abejas melíferas, las cuales debido a su cuerpo cubierto de pelos recogen fácilmente los gránulos de polen cuando se mueven al interior de las flores, lo cual provoca que estos granos se esparzan a medida que visitan más flores. El monitoreo de las abejas en la forma tradicional se ha realizado por personal experto teniendo como objetivo conocer las condiciones de salud y el transporte de alimentos (polen) hacia el enjambre. Esta tarea puede convertirse en un proceso arduo y difícil. El método tradicional de observación puede tomar largos periodos de tiempo presentando detecciones tardías ante anomalías en las abejas. Los constantes avances en inteligencia artificial (IA) como lo son la automatización de aprendizaje, el descubrimiento repetitivo a través de datos o los algoritmos de aprendizaje profundo, brindan un panorama favorable para el seguimiento y clasificación de abejas portadoras y no portadoras de polen. El objetivo de este trabajo es desarrollar un sistema que permita detectar y hacer seguimiento a abejas portadoras de polen usando técnicas de IA. El sistema inteligente se desarrolló en un entorno embebido, mediante la tarjeta de desarrollo Jetson Nano de Nvidia ®, a través del framework de Deep learning Pytorch y finalmente se validó el sistema en un entorno real.

Palabras clave: Aprendizaje profundo, batch-size, épocas, entrenamiento, jetson nano, red neuronal.

Abstract

Pollination is a fundamental process for the survival of ecosystems. It is considered that 90% of flowering plants depend on pollination to reproduce (form seeds or fruits), making it essential for biodiversity. One of the main pollinators in nature are honey bees, which, due to their body covered with hair, easily collect pollen granules when they move inside flowers, causing these grains to spread as they visit more flowers. Monitoring of bees in the traditional way has been carried out by expert personnel with the aim of knowing the health conditions and the transport of food (pollen) to the swarm. This task can become an arduous and difficult process. The traditional method of observation can take long periods of time, presenting late detections of anomalies in bees. Constant advances in artificial intelligence (AI) such as learning automation, repetitive discovery through data or deep learning algorithms, provide a favorable outlook for the monitoring and classification of pollen-carrying and non-pollen-carrying bees. The objective of this work is to develop a system that allows the detection and monitoring of pollen-carrying bees using AI techniques. The intelligent system was developed in an embedded environment, using the Nvidia ® Jetson Nano development card, through the Pytorch Deep Learning framework and finally the system was validated in a real environment.

Keywords: Deep learning, batch-size, epochs, training, jetson nano, neural network.

CAPÍTULO 1

INTRODUCCION

En este capítulo se abarcará el planteamiento del problema, justificación, objetivos del proyecto y metodología. Dando a conocer el porqué del proyecto, los alcances que tendrá y la forma en que se trabajará a lo largo del mismo.

1.1. Planteamiento del problema

Colombia cuenta con una biodiversidad y plantas tropicales (lavanda, menta de gato, girasoles, entre otros) las cuales podrían contribuir a un aumento de producción nacional de miel y sus derivados. De acuerdo con la Organización de las Naciones Unidas para la Alimentación y Agricultura (FAO) Colombia ocupa el puesto 70 en la producción mundial de miel [1]. Sin embargo, la producción colombiana de miel no satisface las demandas internas, por consiguiente, provoca una necesidad latente de importación. Para satisfacer esta necesidad, es necesario realizar un estudio de las colmenas de los apicultores para determinar el estado de estas, así como la actividad de las abejas [2].

La implementación de tecnologías es imprescindible en la supervisión de las abejas. En la actualidad, la observación de las colmenas requiere de mucho tiempo, generando una detección tardía ante posibles anomalías en los enjambres como virosis, bacteriosis, nosemosis, entre otras[3]. Por consiguiente, los constantes avances tecnológicos en detecciones de objetos apoyados en aprendizaje profundo (Deep learning) han permitido dar un nuevo panorama al progreso de la apicultura. Un ejemplo de ello es el método de visión por computadora y aprendizaje profundo [2], el cual es más rápido y eficiente que las observaciones y anotaciones manuales realizadas por apicultores. Recientemente, Arcega Dan, et al., [4] propusieron el uso

de sensores infrarrojos, sensores acústicos, radares armónicos, radiofrecuencia y métodos de procesamiento de imágenes para el monitoreo de abejas, concluyendo que la mayor eficiencia se obtenía con sistemas de procesamiento de imágenes.

Este procesamiento de imágenes se ha apoyado en el uso de redes neuronales convolucionales (CNN), con clasificadores como el algoritmo de k vecinos más cercanos (KNN), máquina de vectores de soporte (SVM), Naive Bayes y arquitecturas como VGG16, VGG19 (Visual Geometry Group de 16 o 19 capas de profundidad), ResNet, entre otros [5]. No obstante, se presenta una dificultad al entrenar los modelos para diferentes especies de abejas debido a la variación del tipo de polen, y variaciones en la iluminación en escenarios reales, lo cual sugiere que existe una necesidad de desarrollar modelos robustos que permitan mejorar la detección de abejas portadoras de polen ante las limitaciones mencionadas.

Por consiguiente, para un manejo integrado de las colmenas, es necesario llevar a cabo estudios de campo que permitan obtener resultados eficientes en diferentes condiciones. Para ello, es recomendable utilizar métodos de supervisión por computadora que permitan a las granjas apícolas hacer la transición de métodos artesanales a métodos automatizados más eficientes

Con base en el aporte que pueden generar estos métodos modernos de supervisión de abejas y pensando en un incremento de la producción de miel, surge la siguiente pregunta de investigación:

¿Cómo implementar en un sistema embebido una aplicación de inteligencia artificial para la detección de abejas portadoras de polen?

1.2. Justificación

Las abejas melíferas son los principales polinizadores de la naturaleza. Su importancia radica en que se alimentan y al mismo tiempo transportan el polen de una flor a otra, lo que permite que muchas plantas se reproduzcan para generar semillas o frutos. Para finales del 2018, Colombia tuvo una producción de 3600 toneladas de miel, cantidad en la cual el Cauca aportó un total de 136 toneladas. Estas 3600 toneladas no lograron satisfacer el 70% y 80% la demanda interna nacional lo que ha generado la entrada de la misma desde otros países ocupando un 12% de la producción nacional [6]. El objetivo de este proyecto es generar un sistema de detección de abejas melíferas portadoras de polen mediante una tarjeta de desarrollado Jetson Nano que permita a los profesionales apícolas estudiar sus comportamientos en las colmenas para un posterior análisis, orientado a generar un incremento en la productividad del sector.

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar un sistema de detección de abejas portadoras de polen mediante procesamiento de imágenes y técnicas de aprendizaje profundo en sistemas embebidos.

1.3.2. Objetivo Específico

- Definir un dataset de imágenes de abejas portadoras de polen para el entrenamiento del modelo basado en aprendizaje profundo.
- Diseñar un sistema de detección embebido basado en aprendizaje profundo que permita detectar abejas portadoras de polen.
- Evaluar el desempeño del sistema de clasificación de abejas basado en aprendizaje profundo.

1.4. Metodología

Para el desarrollo de este proyecto se implementó la metodología hardware – software. Mediante esta metodología se busca adquirir información del exterior para luego ser procesada y validada en el software (Fig. 1).

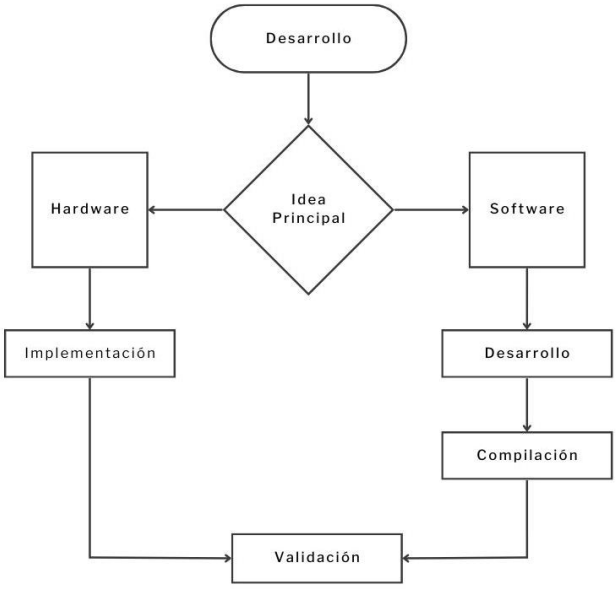

Fig. 1 Metodología Hardware – Software. Fuente: Autor propio

En lo concerniente al hardware se utilizó una cámara Logitech para la adquisición de imágenes y validación del sistema, una tarjeta Jetson nano para el desarrollo software y finalmente un computador para la preparación del dataset. Con base en lo anterior se describen las actividades a realizar de la siguiente manera:

1.4.1. Adquisición de información

En esta primera actividad se buscó y estableció información relevante para el desarrollo de la red, se estableció un dataset para su procesamiento y se determinaron los softwares necesarios para llevar a cabo el proyecto.

1.4.2. Clasificación de imágenes

En esta segunda fase, se establecieron las imágenes de abejas portadoras y no portadoras de polen. Una vez clasificadas, se creó el dataset que se utilizó en el entrenamiento de la red. Para la creación del dataset, se procedió a etiquetar las imágenes y posteriormente se procesaron para dividir el dataset en dos grupos: prueba y validación.

1.4.3. Entrenamiento de red

En esta tercera fase se implementó el dataset de la fase 2 a la red neuronal, se utilizaron diferentes épocas de entrenamiento y batch-size para obtener diferentes archivos onnx y labels para la posterior validación del sistema.

1.4.4. Validación del sistema

En esta cuarta fase, se desarrolló un código en Phyton en formato .py para el procesamiento de videos e imágenes captadas mediante una cámara web. Una vez configurado el archivo .py con el código necesario, se llevaron a cabo el procesamiento junto con los archivos onnx y las etiquetas obtenidas en la fase 3, lo que permitió realizar la detección demarcada en videos o imágenes procesados.

CAPÍTULO 2

MARCO TEORICO Y CONCEPTUAL

En este se abarcará todo lo relacionado a marco teórico y antecedentes del proyecto, explicando conceptos fundamentales, formulas e investigaciones previas.

2. Marco de referencia

2.1. Marco teórico

2.1.1. Polen

También conocido como polvo fino (Fig. 2). Son partículas fecundantes con potencialidad masculinas en la reproducción de las flores, dando lugar a la formación de semillas. Su transporte se da bien sea por el aire denominada polinización anemófila o por insectos denominada polinización anemófila. Su color varía en base a la especie vegetal siendo el color amarillo o marrón claro el más significativo, sin embargo, se pueden encontrar también en colores blanco o negro [7].

Fig. 2 Polen. Fuente: Thomas Bresson

2.1.2. Abejas

Son insectos polinizadores por excelencia (Fig. 3). Cumplen una función vital en el ecosistema ya que contribuyen de una manera activa a la supervivencia de muchas especies de plantas [8]. Las abejas recogen el polen de los estambres de las plantas, los humedecen con miel o néctar para formar acúmulos, los cuales son transportados

a la colmena en especie de cesta llamada corbículas ubicadas en las patas posteriores [9].

Fig. 3 Abeja Polinizadora. Fuente: [8]

2.1.3. Inteligencia Artificial

La inteligencia artificial (IA) es un campo de la ciencia informática centrada en la creación de programas y mecanismos con comportamientos inteligentes, tales programas son capaces de analizar grandes cantidades de datos identificando las tendencias y patrones, analizándolos y posterior a ellos generando una predicción-acción de manera automática. La IA utiliza tres técnicas, la primera de ellas es el aprendizaje automático que se encarga de que las computadoras realicen acciones sin necesidad de una programación explícita, la segunda es el aprendizaje profundo el cual usa redes neuronales para la extracción de características y por último el análisis predictivo, es el análisis numérico que se hace con referencia a una base de datos para generar una probabilidad estadística de un evento [10].

2.1.4. Machine learning (aprendizaje automático)

El machine learning es una disciplina del área de la IA que, mediante algoritmos, dota a los ordenadores con habilidades capaces de identificar patrones. Los algoritmos de machine learning cuentan con tres categorías: el aprendizaje supervisado, basado en un sistema de etiquetas asociadas a unos datos que le permitan tomar decisiones o hacer predicciones según es requerido; el aprendizaje no supervisado, en el cual no se cuenta con un conocimiento previo por ende se requiere identificar patrones que

permitan organizar la información; y por último, el aprendizaje por refuerzo el cual aprende de la propia experiencia propia, haciendo que sea capaz de tomar la decisión más óptima con base en un proceso de prueba y error [11].

2.1.5. Deep learning (aprendizaje profundo)

El deep learning es un algoritmo automático estructurado o jerárquico asemejando el aprendizaje humano para obtener nuevos conocimientos por sí mismo, basado en un conocimiento previamente enseñado. La información de entrada, al igual que los nuevos conocimientos, se entrelazan mediante redes neuronales artificiales con el propósito automatizar los análisis predictivos, es así como el aprendizaje profundo se divide en tres principales capas (Fig. 4), la capa de entrada (Intup layer), compuestos por neuronas que asimilan los datos de entrada, capa oculta (hidden layer), encargada de procesar la información y realizar los cálculos intermedios; y finalmente, la capa de salida (Output layer), encargado de tomar la decisión a realiza [12].

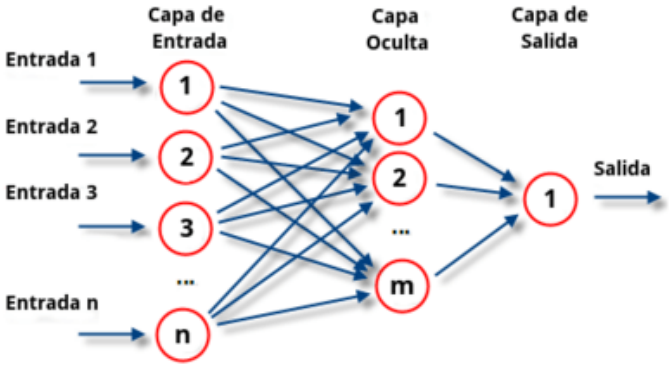

Fig. 4 Capas Deep Learning. Fuente: [12]

2.1.6. Red neuronal convolucional (CNN)

Una red neuronal es un tipo de red neuronal artificial con aprendizaje supervisado, esta red tiene múltiples capas de filtros convolucionales de una o más dimensiones, añadiendo una función entre capa para realizar un mapeo no lineal. La primera capa

es la encargada de extraer las características, la segunda capa se encarga de una reducción por muestreo, y la última capa utiliza neuronas de perceptrón para la clasificación final. El entrenamiento de la CNN se realiza por medio de pixeles y neuronas que toman como entrada los píxeles de la imagen (ancho x alto), lo que equivaldría al número de neuronas a utilizar (imágenes de 1 solo color, para imágenes de 3 seria ancho x alto x 3), luego se procede con un Pre-procesamiento: se convertir los valores a 0 y 1, dividiendo los datos entre 255 (los colores de los pixeles toman valores de 0 a 255). Posteriormente, se avanza a las Convoluciones, donde se toma un grupo de píxeles cercanos de la imagen y se realiza un producto escalar matemático con una matriz de kernel (generalmente de 3x3 píxeles), seguido por la función de activación, siendo la más común ReLU (Rectifier Linear Unit) (ecuación 1).

$$f(x) = max(0, x)$$

Ecuación 1 CNN. Fuente: [13].

Una vez completada la función de activación, se realiza un Muestreo (subsampling) donde se toma una muestra de las neuronas más representativas antes de hacer una nueva convolución, con el fin de reducir el tamaño de la próxima capa. El muestreo más utilizado es el Max-Pooling (recorre los pixeles de derecha – izquierda, arriba – abajo, 2x2) dando como resultado una primera convolución (Fig. 5). [13].

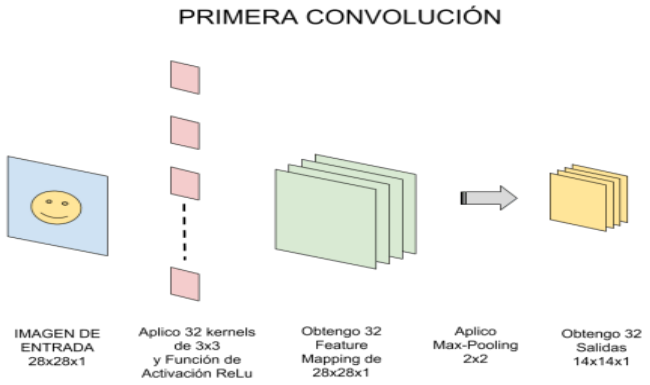

Fig. 5 primera capa de convolución. Fuente: [14]

Para finalizar, se llega a las convoluciones subsecuentes en las que se aplican los pasos anteriores, teniendo en cuenta que el resultado de la convolución anterior será multiplicando por la nueva capa (Fig. 6).

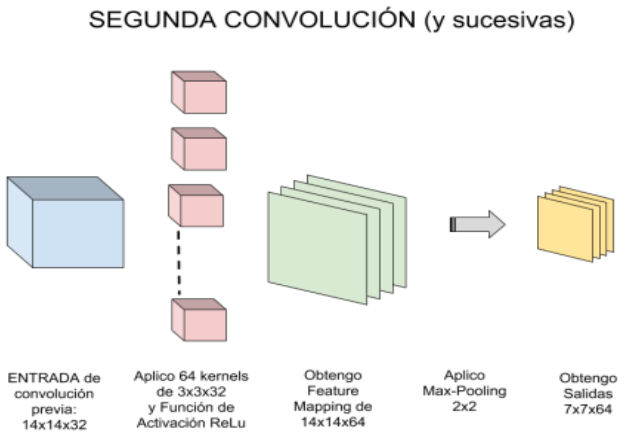

Fig. 6 Segunda capa de convolución. Fuente: [14]

2.1.7. Detección de objetos

La detección de objetos es una técnica en procesamiento de imágenes que permite encontrar instancias de una o varias clases presentes en imágenes como: autos, humanos, casas, entre otros (Fig. 7). Para la detección de objetos en imágenes se utilizan algoritmos de machine learning en donde se asigna una clase por cada objeto a detectar. Estos algoritmos tienen la capacidad de detectar múltiples objetos, dar la posición del objeto y dibujar un rectángulo a su alrededor. Para la detección de objetos las arquitecturas más conocidas son R-CNN, Yolo y Single Shot Detector (SSD) [14].

Fig. 7 Detección de Objetos. Fuente: [14]

2.1.8. Técnicas de detección de objetos

2.1.8.1. YOLO

You Only Look Once (YOLO) fue creada en 2016 y es una red utilizada para la detección de objetos en tiempo real. El funcionamiento de YOLO consiste en la división de una imagen en celdas (Fig. 8). Sobre esas celdas intentará detectar el objeto valiéndose de archivos fijos (Fig. 9), los cuales corren a 45 FPS sin procesamiento de lote en una GPU Titan X. Hace uso de la métrica IoU y de la supresión no máxima (Non-max suppression). Para su preentrenamiento, utilizan 20 capas convolucionales seguidas de una capa promediadora y una capa de conexión completa para finalmente convertirse en una red de detección de objetos completa. Una de las limitaciones de YOLO es el número de objetos que se puede detectar, dado que cada celda predice únicamente dos cajas y una clase [14].

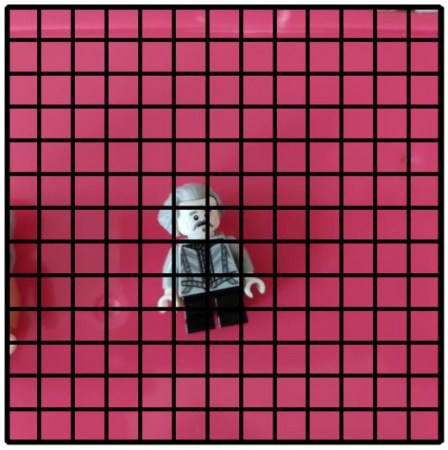

Fig. 8 Celdas Yolo. Fuente: [14]

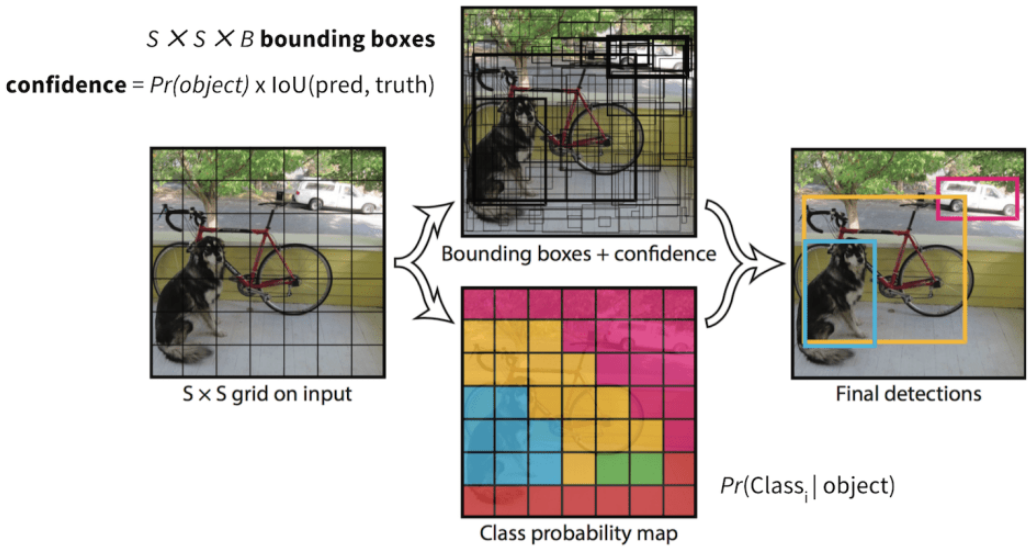

Fig. 9 Detección Yolo. Fuente: [14]

2.1.8.2. RCNN

La red neuronal convolucional basada en regiones (R-CNN) es un modelo diseñado en el año 2013, el cual toma una imagen de entrada y produce un conjunto de cuadros delimitadores como salida; cada cuadro delimitador contiene una región de interés (ROI). Este detector R-CNN está dividido en 4 etapas: la primera etapa se conforma por la imagen de entrada; la segunda etapa genera las propuestas a regiones de

interés; la tercera etapa consiste en una red neuronal convolucional la cual extrae un vector de características de longitud fija para cada región; y finalmente, la etapa 4 consiste en un conjunto de clases lineales Support Vector Machine (SVM) (Fig. 10). Este modelo calcula de forma independiente las características de la red neuronal en cada región de interés, generando más tiempo en el entrenamiento de la red. Posterior a esto en 2015 se lanzó un nuevo modelo llamado fast R-CNN el cual permite ejecutar la red neuronal una vez en toda la imagen y se implementó al final de la red, un método llamado ROIPooling encargado de cortar cada ROI del tensor de salida de la red para reformarlo y clasificarlo (Fig. 11). En 2017, se actualizó nuevamente el modelo a Mask R-CNN en donde se permitió agregar segmentación instantánea reemplazando el método ROIPooling por ROIAling, el cual podría representar fracciones de un píxel. Finalmente, la última actualización se dio en 2019 en donde aparece Mesh R-CNN, la cual tiene la capacidad de generar una malla 3D a partir de una imagen 2D [15].

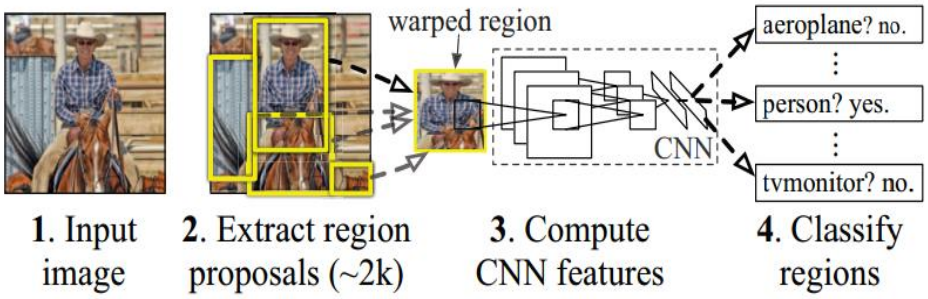

Fig. 10 Modelo R-CNN por etapas. Fuente: [15]

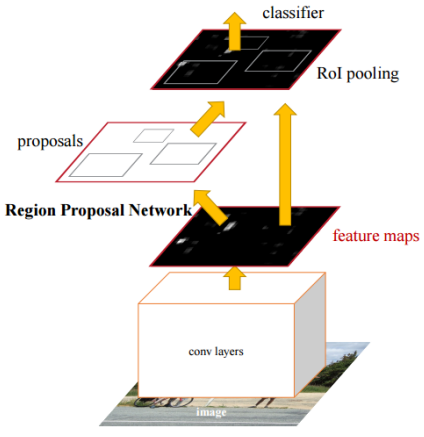

Fig. 11 Modelo Fast R-CNN. Fuente: [15]

2.1.8.3. SSD

Cuenta con un modelo de columna vertebral compuesto por una red troncal basada en una red profunda de clasificación de imágenes pre-entrenadas para extraer las características (Fig. 12). La red más utilizada es la Resnet, la cual es entrenada en ImageNet la cual permite extraer el significado semántico de la imagen, en donde se conserva su estructura original, pero con una resolución baja. El segundo modelo es el cabezal SSD, compuesto por una o más capas convolucionales agregadas hasta la columna vertebral; sus salidas se presentan en cuadros delimitadores y clase de objetos en la ubicación. La SSD no utiliza una grilla predefinida; sin embargo, utiliza anclas de distintas proporciones que se van escalando a medida que se avanza en la red. ([14, 16].

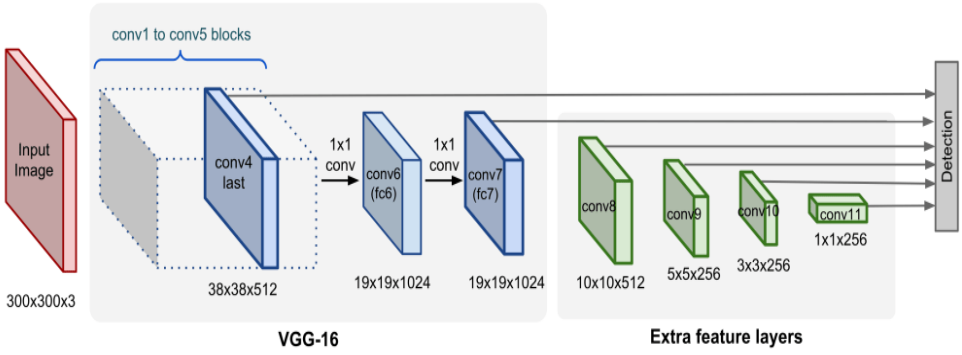

Fig. 12 Red SSD. Fuente: [14]

2.1.9. Segmentación semántica

La segmentación semántica es una técnica de Deep Learning en donde se asigna una etiqueta o clasifica cada pixel presente en una imagen; esta tiene como finalidad reconocer un conjunto de píxeles que conforman distintas categorías (Fig. 13). En la segmentación semántica, pueden existir diferentes categorías para clasificar una imagen permitiendo tener una amplia variedad de aplicaciones, las cuales van desde interpretar el contenido de escenas urbanas para tareas de conducción automática hasta aplicaciones en el campo de la medicina para el análisis de información del paciente para un diagnóstico u operaciones. [17, 18].

El proceso de entrenamiento de una red de segmentación semántica para clasificar imágenes consta de estos cuatro pasos:

- Analizar un conjunto de imágenes en pixeles.
- Crear una red de segmentación semántica.
- Entrenar la red para clasificar imágenes en categorías de pixeles.
- Evaluar la precisión de la red.

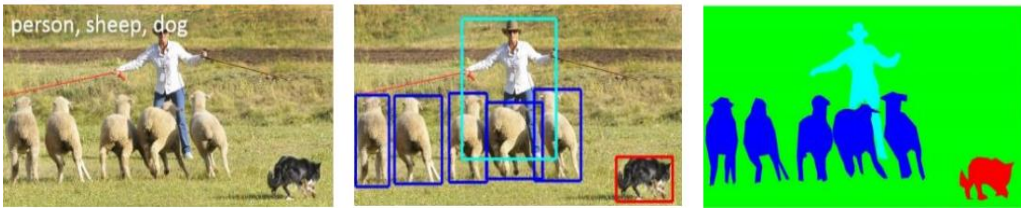

Fig. 13 Segmentación Semántica. Fuente: [19]

2.1.10. Sistemas Embebido

2.1.10.1. El kit de desarrollador Jetson Nano

Es una pequeña y potente computadora utilizada en el área de inteligencia artificial (Fig. 14). Su mayor uso se ve reflejado en aplicaciones de Deep Learning en tareas como la clasificación de imágenes en tiempo real, detección de objetos, segmentación, entre otros. [20].

Fig. 14 Tarjeta Jetson Nano. Fuente: [20]

2.1.10.2. Raspberry Pi

Es un microcontrolador de pequeñas dimensiones basado en hardware libre, así como sistemas operativos libres basados en GNU/Linux (Fig. 15). Las placas Raspberry se basan en SoC de arquitectura ARM de bajo consumo y buen rendimiento. Se componen por una RAM (la capacidad depende del modelo) así como varias salidas de video, un control jack de 4 polos (para entrada y salida de audio), un lector de tarjetas para la instalación del sistema operativo y varios puertos USB [21].

Fig. 15 Raspberry Pi. Fuente: [21]

2.1.11. Métricas de desempeño

2.1.11.1. Matriz de confusión

Es una tabla que describe el rendimiento de un modelo supervisado de Machine Learning en los datos de prueba, donde se desconocen los verdaderos datos de prueba. Esta matriz tiene 4 resultados: Verdadero Positivo (TP), cuando la clase real del punto de datos era 1 (Verdadero) y el pronóstico también 1 (Verdadero); Verdaderos Negativos (TN), cuando la clase real del punto de datos fue 0 (Falso) y el pronosticado también es 0 (Falso); Falso Positivo (FP), cuando la clase real del punto de datos era 0 (Falso) y el pronosticado es 1 (verdadero); y finalmente Falso Negativo (FN) Cuando la clase real del punto de datos era 1 (Verdadero) y el valor predicho es 0 (Falso) [22].

2.1.11.2. Accuracy

Es el porcentaje total de elementos clasificados correctamente (Medida de calidad). Se calcula con los valores obtenidos en la matriz de confusión, de la siguiente forma:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Ecuación 2 Accuracy- Fuente: [23].

El valor obtenido esta entre 0 y 1, de forma que entre más alto sea el número mejor será su precisión. [23].

2.1.11.3. Recall

Es el número de elementos identificados correctamente como positivos del total de positivos verdaderos obtenidos en la matriz de confusión [22]. Se calcula con los valores de TP y FN, relacionados de la siguiente manera:

$$Recall = \frac{TP}{TP + FN}$$

Ecuación 3 Recall. Fuente: [22].

2.1.11.4. Precision

Es el número de elementos identificados correctamente como positivo de un total de elementos identificados como positivos. [22]. Obtenidos mediante TP y FP, de la siguiente manera:

$$Precision = \frac{TP}{TP+FP}$$

Ecuación 4 Presicion. Fuente: [22].

2.1.11.5. Specificity

Es el número de ítems correctamente identificados como negativos fuera del total de negativos. [22]. Obtenidos mediante FP y TN, de la siguiente manera:

$$Specificity = \frac{TN}{TN + FP}$$

Ecuación 5 Specifity. Fuente: [22].

2.1.11.6. F1- Score

Es utilizado para combinar las medidas de precisión y Recall en un sólo valor. [22]. Se calcula por medio de la media aritmética de la siguiente manera:

$$\underline{X} = \frac{Presicion + Recall}{2}$$

Ecuación 6 F1-score. Fuente: [22].

2.1.11.7. Intersection over Union (IoU)

Es una métrica de evaluación que se utiliza para medir la precisión de un detector de objetos con un conjunto de datos en particular o, dicho de otra forma, el grado de superposición entre de dos casillas (Fig. 16). Cuanto mayor sea la región de superposición, mayor será la IOU. [24]. Se calcula:

$$IoU = \frac{Area\ de\ Superposicion}{Area\ de\ Union}$$

Ecuación 7 IoU. Fuente: [22].

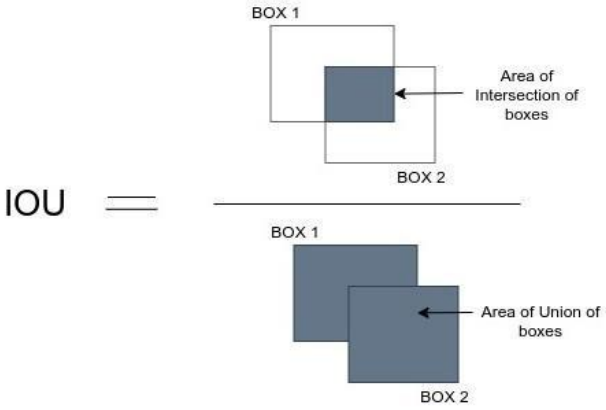

Fig. 16 IOU. Fuente: [24]

2.1.11.8. Coeficiente de similitud de datos (DSC)

Es una estadística que se utiliza para medir la similitud de la imagen de predicción y la verdad fundamental. [25]. Se calcula:

$$DSC = \frac{2 * TP}{(t + FP) + (TP + FN)}$$

Ecuación 8 DSC. Fuente: [25]

2.1.12. Frameworks

2.1.12.1. Pytorch

Es un framework de machine learning de código abierto creado para ser flexible y modular para la investigación (Fig. 17). Pytorch posee un paquete de Python para funciones de alto nivel como lo es NumPy (Tensor) con una fuerte aceleración de GPU y TorchScript, a fin de generar una transición fácil entre el modo «eager» y el modo gráfico. Esto lo ha posicionado como una pieza fundamental en el desarrollo de relevantes aplicaciones de Inteligencia Artificial, como el Autopilot de Tesla y Pyro de Uber.

PyTorch dispone de soporte para su ejecución en tarjetas gráficas (GPU), el cual utiliza internamente CUDA, así como una API que conecta la CPU con la GPU que ha sido desarrollada por NVIDIA. [12].

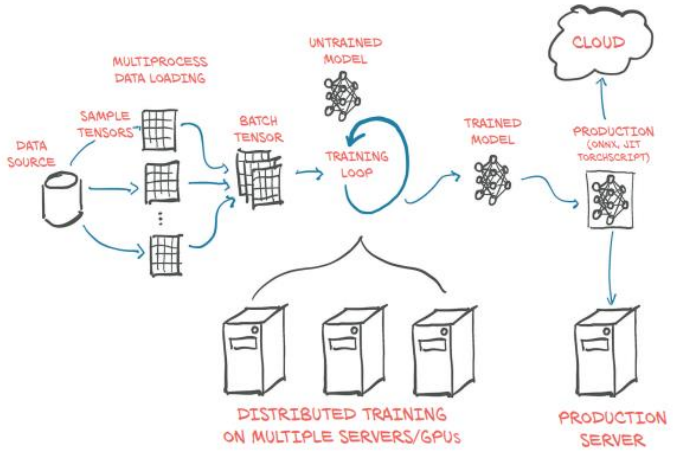

Fig. 17 Pytorch. Fuente: [12]

2.1.12.2. TensorFlow

Es una biblioteca de software de código abierto para computación numérica, la cual utiliza gráficos de flujo de datos. Cada nodo en las gráficas representa operaciones matemáticas, por otro lado, los bordes de las gráficas representan las matrices de datos multidimensionales (tensores) comunicadas entre ellos. TensorFlow cuenta con una arquitectura que permite implementar el cálculo a una o más CPU o GPU en equipos de escritorio, servidores o dispositivos móviles con una sola API (Fig. 18). Desarrollado en 2014 y lanzado como código abierto en 2015 por el equipo de Google Brain Team, con el propósito de llevar a cabo el aprendizaje automático y la investigación de redes neuronales con aplicaciones en procesamiento de imágenes (visión artificial), [26].

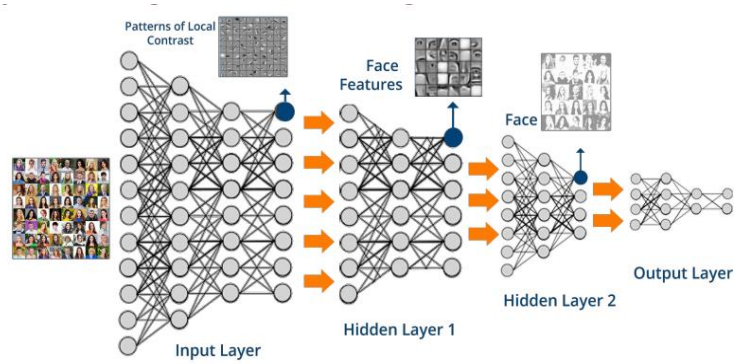

Fig. 18 Deep Learning con TensorFlow. Fuente: [27]

2.2. Antecedentes

El monitoreo de las abejas ha presentado una mejora significativa gracias a los avances en de la inteligencia artificial. Las técnicas de machine learning para la detección de objetos han sido una herramienta de gran apoyo para los apicultores reduciendo el tiempo de observación de las abejas. La detección de abejas mediante sistemas inteligentes ha permitido obtener información sobre sus actividades en la colmena, así como también ha permitido hacer un seguimiento a enfermedades como: varroa y nosema [28].

Una dificultad para la detección de abejas mediante procesamiento de imágenes radica en el entrenamiento de modelos para diferentes tipos de polen con base en la variación de sujeción del mismo. El equipo de investigación [29] implementó un filtro Kalman para la estimación de movimiento, un algoritmo para rastrear la posición de múltiples abejas y una red Global de Vecino más Cercano (GNN) para tracking; sin embargo, al realizar la detección de las abejas ante diferentes cambios de luz, este presentaba dificultades al realizar el seguimiento de los insectos. Por consiguiente, surgió la necesidad de buscar alternativas que solventaran esta problemática. Se llevaron a cabo estudios en donde se demostró que usando arquitecturas como Maquina de Soporte Vectorial (SVM) y Yolo presentaban mejoras en la detección de abejas. De tal forma que los estudios [2], [5] y [29] implementaron la arquitectura SVM para solventar tal problemática obteniendo un desempeño bajo la métrica de Accuracy de 50.66% en su detección más baja y de 77.31% en su detección más alta.

Por otro lado, algunos estudios han implementado el uso de embebidos como herramientas para los sistemas de detección y seguimiento de abejas [4], mejorando los tiempos de respuesta en la detección usando sistemas dedicados. A estos sistemas embebidos se les conectó una cámara Logitech C920, que sirve como elemento de captura de imágenes en tiempo real, con una calidad de 1080p a 30 FPS. Uno de los sistemas embebidos utilizados fue la Raspberry Pi, la cual contaba con un modelo de mezcla gaussiana en donde se obtuvo un Accuracy de 88%. El siguiente

sistema embebido fue una Jetson Nano TX2 con un modelo Yolov3-tiny en donde se obtuvo un Accuracy de 93.4%.

Ante la mejora significativa, estos resultados evidencian la ventaja de usar sistemas embebidos para la detección de objetos. La implementación de diferentes algoritmos de red como Yolo, SSD, entre otras, así como redes como AlexNet, ResNet, DarkNet, entre otras, dentro de un sistema embebido permite obtener una eficiencia mayor al detectar objetos en tiempo real. A continuación, se presentan algunos trabajos relevantes sobre la detección de abejas portadoras de polen implementando técnicas de machine learning.

CAPÍTULO 3

DESARROLLO DEL SISTEMA DE DETECCION DE ABEJAS PORTADORAS DE POLEN

En este capítulo se abarca en detalle el diseño conceptual del sistema de detección de abejas portadoras de polen, abordando tanto los aspectos teóricos como prácticos necesarios para la implementación del proyecto. Se describe de manera específica la preparación del sistema, la cual incluye el desarrollo del software y hardware necesarios para su correcto funcionamiento. Además, se analizan las condiciones y requisitos técnicos, operativos y ambientales que se deben considerar para garantizar la viabilidad y eficiencia del sistema. Además, se exponen las metodologías empleadas para integrar las distintas partes del sistema, junto con los criterios de diseño adoptados para optimizar su desempeño en la detección de abejas portadoras de polen.

3. Desarrollo del sistema de detección de abejas de polen

3.1. Diseño conceptual del sistema

La inteligencia artificial se ha convertido en una herramienta de gran utilidad en el ámbito de la apicultura, proporcionando soluciones innovadoras para llevar a cabo procesos que contribuyen significativamente a la preservación y mejora de las condiciones de las colmenas. Entre las diversas técnicas de inteligencia artificial, el aprendizaje profundo o Deep Learning, ha demostrado ser especialmente eficaz en este contexto. A través de la implementación de una red neuronal, se desarrolló un sistema diseñado específicamente para la detección de abejas portadoras de polen, una tarea esencial en el monitoreo de la salud y productividad de las colmenas. Este sistema fue implementado considerando una serie de elementos interrelacionados, tanto en términos de hardware como de software, que trabajan de manera conjunta para garantizar su funcionalidad.

3.1.1. Elección de lenguaje de programación

En este caso en particular se ha optado por utilizar la librería de Pytorch, ya que esta brinda una amplia gama de herramientas y funciones para el procesamiento de datos, construcción del modelo y optimización del algoritmo de aprendizaje, todo esto gracias a su enfoque dinámico y expresivo. También permite utilizar tensores para el procesamiento de información no numérica como sonidos o imágenes, estos tensores utilizan herramientas matemáticas conocidas como escalar, vector, tupla y matriz. El escalar es un tensor de dimensión cero el cual contiene un solo número, el vector es un tensor unidimensional el cual contiene varios escalares del mismo tipo, la tupla es un sensor unidimensional el cual contiene diferentes tipos de datos, y la matriz la cual es un tensor bidimensional que contiene varios vectores del mismo tipo. Pytorch permite calcular gradientes o derivadas de sus tensores mediante una función compuesta con sus derivadas parciales, lo que permite calcular cuantos valores son verdaderos dentro del tensor, esto permite mejorar los resultados en los gráficos dinámicos en la salida del tensor. Todo ello facilita la interacción de manera eficiente con Python y todas sus librerías, lo cual permite incorporar códigos de una manera precisa y concisa para afinar la red neuronal.

3.1.2. Definición de estructura de la red neuronal

Para ello se define las capas de entrada siendo esta equivalente al número de características de los datos. Una vez definida la primera capa, se debe definir las capas ocultas, estás se encuentran situadas entre las capas de entrada y salida, estas capas se encargan de procesar la información. La capa de salida es la encargada de hacer la predicción; entre estas capas se encuentran las neuronas, cada capa tiene un conjunto de neuronas encargadas de realizar la interacción entre capas. Para la interacción de capas se utilizan hiperparámetros. El primer es el batch-size, el cual representa la cantidad de muestras utilizadas un paso hacia delante y hacia atrás a través de la red, esto permite tener un equilibrio entre precisión y velocidad. El segundo

de ellos son las épocas, utilizado para medir la cantidad de veces que el modelo observara todo el conjunto de datos. Una vez definidos los hiperparámetros se pasa a la función de activación, esta función permite transformar una señal de entrada de un nodo en una señal de salida que se trasfiere de capa en capa generando un mapeo entre ellas. Entre las funciones de activación se encuentra la función sigmoide, la cual se caracteriza por una curva en forma de S delimitada entre los valores cero y uno, la función tanh (tangente hiperbólica) caracterizada por tener la misma curva en forma de S que la función sigmoide, pero los valores están acotados entre -1 y 1. La función sofmax es utilizada generalmente como función de activación en la capa de salida. Cabe mencionar que, utilizar la función sigmoide o tanh para construir redes neuronales profundas es arriesgado, ya que es más probable que sufran el problema de desvanecimiento de gradiente. Por tal motivo se creó la función ReLu (unidad lineal rectificada), esta función esta acotada entre cero e infinito, lo que indica que para valores de entrada menores o iguales que cero, la función devuelve cero y, para valores superiores a cero, la función devuelve el valor de entrada proporcionado. Finalmente se tienen los pesos y sesgos, los cuales son parámetros que el modelo aprende durante el entrenamiento. Inicialmente, estos se establecen en valores aleatorios, estos valores se pueden ver utilizando los atributos weight y bias.

3.1.3. Datos de entrenamiento

Con respecto a los datos de entrenamiento, se seleccionaron las imágenes a utilizar para el dataset. Para ello se optó por utilizar el software de Labeligm por su fácil manejo, código abierto y diversos formatos de etiquetado. Con este software se realizó el etiquetado de las imágenes en Pollen y Not Pollen, para ello el formato de etiquetado de estas imágenes debe ser compatible con las librerías que serán utilizadas, en este caso en formato VOC (Visual Object Challenge). Una vez obtenidas las etiquetas se debe cargar la información al software de Jupyter por su interacción con Python, con él se realizaron 3 procedimientos, el primero de ellos, es la división de imágenes en entrenamiento, validación y prueba. El segundo procedimiento es la eliminación de la extensión .jpg y finalmente se deben crear los archivos .txt (train, test y trainval) con la información de las etiquetas.

3.1.4. Entrenamiento del modelo

Para ello se utilizó una Jetson nano, en ella se implementaron las librerías de Pytorch mediante el entorno de Docker, el cual es un software de código abierto que permite crear, implementar y gestionar aplicaciones en contenedores virtuales (un contenedor de Docker es un paquete de software con todo lo necesario para ejecutar una determinada aplicación). Una vez en el entorno de Docker, se procedió al procesamiento del dataset y se eligió la arquitectura para el entrenamiento (en este caso la arquitectura que se utilizó es DetecNet). También se deben ajustar los hiperparámetros hasta encontrar un resultado óptimo y obtener los archivos para su entrenamiento.

3.1.5. Evaluación del modelo

Con el fin de realizar la evaluación del modelo, se generó un archivo con extensión .py mediante el cual se cargan los archivos obtenidos en el entrenamiento, se ajustan los parámetros para crear los bounding box (Cuadro delimitadores), threshold (flexibilidad de detección) y se activa la detección por cámara web o procesamiento de video. Además, en el entorno de Docker se carga el archivo .py y se define el threshold con el cual se realizó la validación, una vez se ejecuten las líneas de código, el docker retorna una ventana con los bounding box con la clase y porcentaje de precisión correspondiente a la clase. Adicional a ello, se utilizaron las métricas de desempeño precisión y F1-socre para obtener una evaluación cuantitativa de los resultados.

A continuación, se presenta la cronología del diseño conceptual anteriormente planteado (Fig. 19).

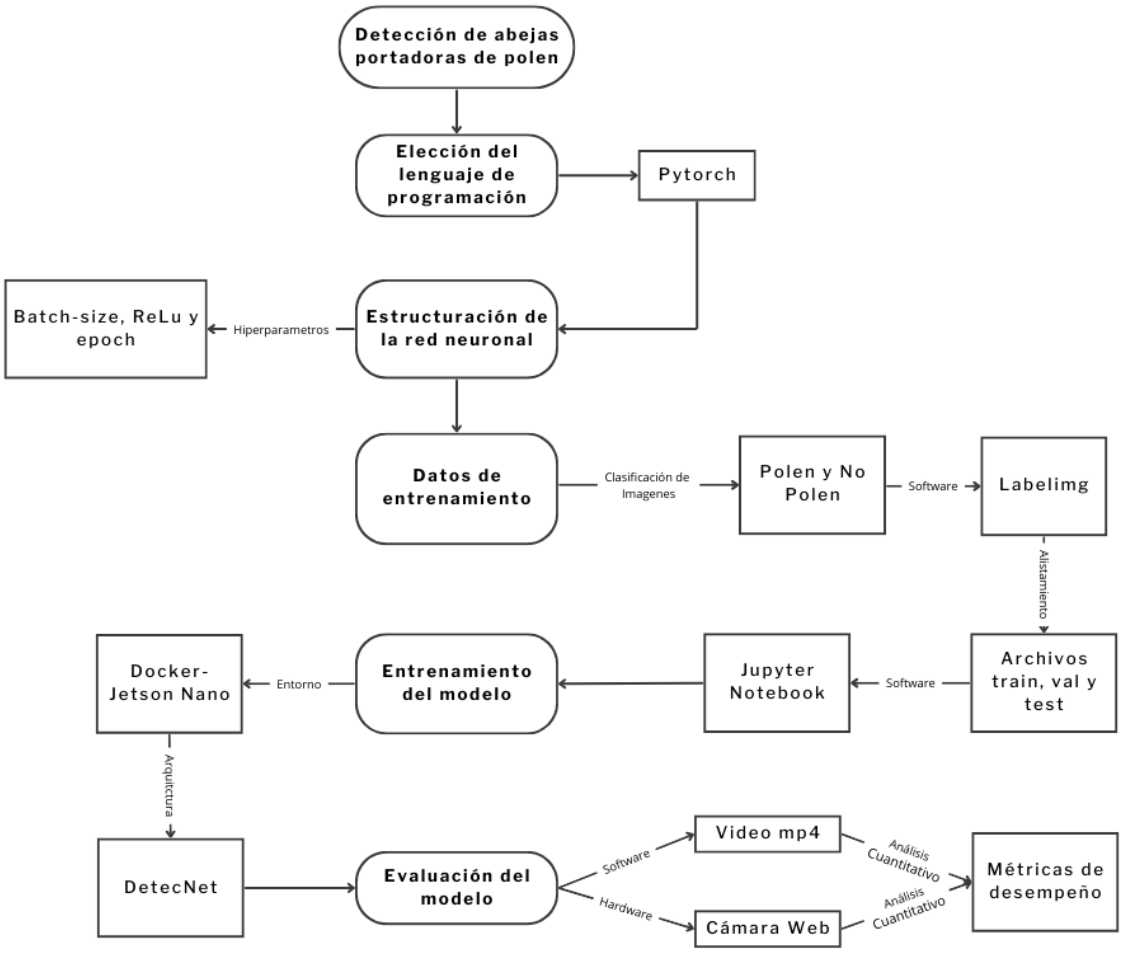

Fig. 19 Diagrama de flujo diseño conceptual. Fuente: Autor propio

3.2. Preparación de condiciones iniciales

Para el desarrollo del sistema, el proceso comenzó con la preparación del sistema embebido, utilizando como núcleo principal la tarjeta Jetson Nano. Esta herramienta, conocida por su capacidad para ejecutar aplicaciones avanzadas de inteligencia artificial, fue seleccionada por su alto rendimiento y versatilidad en proyectos de visión por computadora y aprendizaje profundo. La configuración inicial de la Jetson Nano implicó la instalación de un sistema operativo especializado. Para ello, se utilizó una tarjeta SD como medio de almacenamiento. En cuanto a la alimentación eléctrica, se optó por un adaptador de voltaje de 110V que garantizara un suministro estable y confiable de energía. Con el fin de aprovechar al máximo el rendimiento de la Jetson Nano, asegurando las demandas de procesamiento asociadas con la ejecución de redes neuronales.

3.2.1. Tarjeta SD

Para la preparación de la tarjeta SD, se debe realizar un flasheo en el software de Balena Etcher con la finalidad de eliminar cualquier dato existente y preparar la memoria microSD. Una vez completado el flasheo se procede a montar la imagen ISO (previamente descargada) la cual contiene el entorno de la tarjeta Jetson Nano [30] (Fig. 20 a 22).

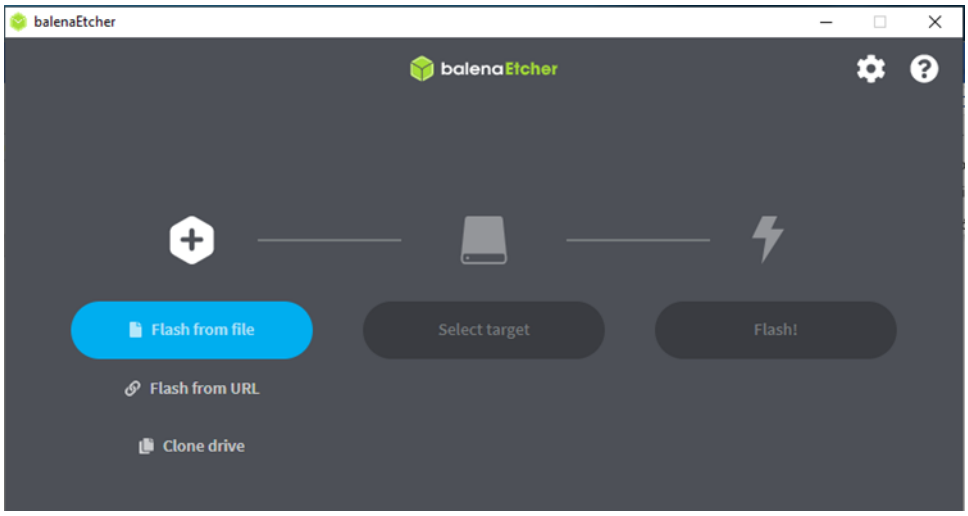

Figura 20 Software Balena Etcher. Fuente: Autor Propio

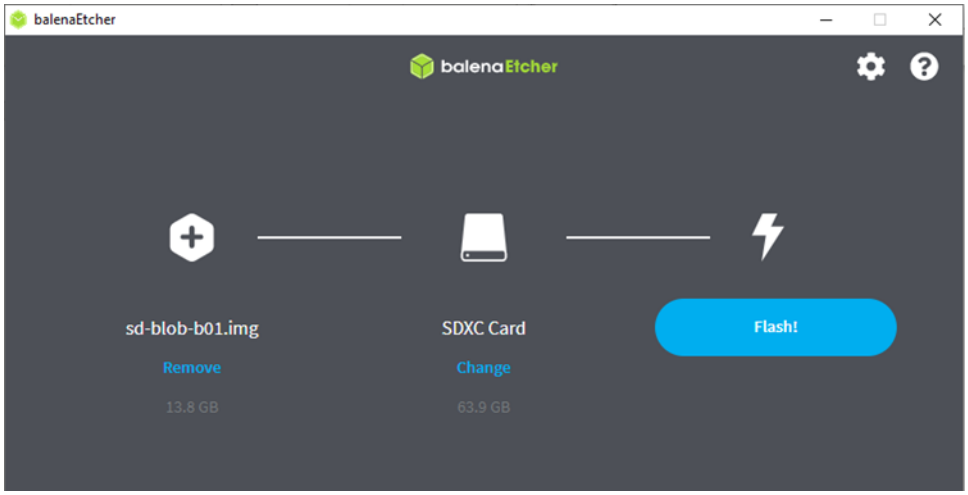

Figura 21 Montura de imagen ISO. Fuente: Autor propio

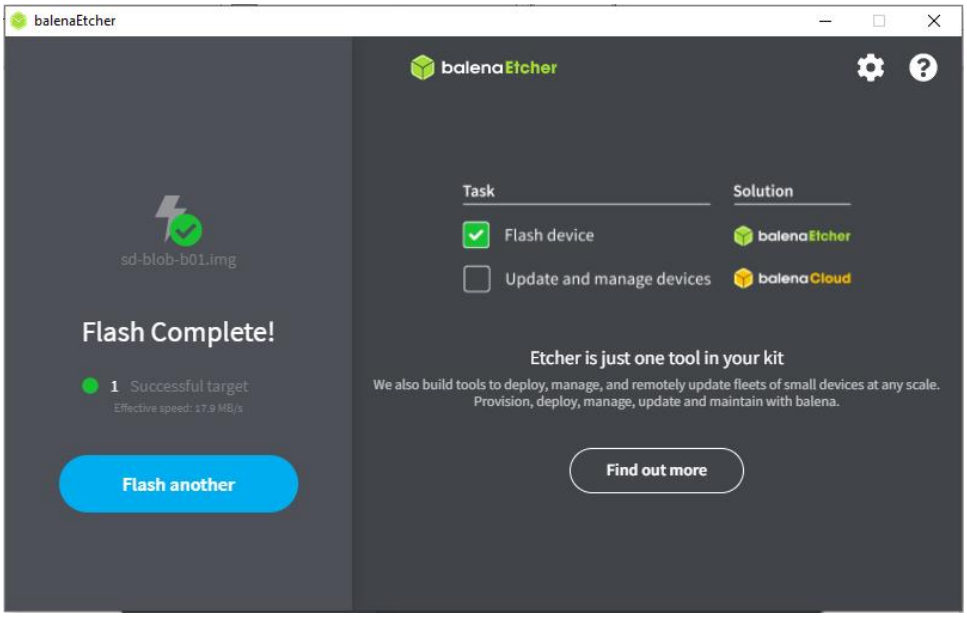

Figura 22 Flasheo microSD. Fuente: Autor propio

3.2.2. Jetson Nano

Para la inicialización y optimo funcionamiento de la tarjeta Jetson nano se instala la tarjeta SD preparada anteriormente, adicional a ello se instalan tres periféricos más a la tarjeta. Uno de ellos es un módulo wifi Tp-link AC600, que se encarga de la conectividad a internet para la descarga y actualización de las librerías de Pytorch (Fig. 23). El segundo periférico corresponde a un teclado inalámbrico empleado para la interacción con el entorno (Fig. 24). Y finalmente, una cámara web para la validación de la red neuronal (Fig. 25).

Figura 23 Modulo wifi. Fuente: Autor propio

Figura 24 Teclado inalámbrico. Fuente: Autor propio

Figura 25 Teclado inalámbrico. Fuente: Autor propio

Una vez conectado los periféricos y conectada la tarjeta Jetson nano a la fuente de alimentación se inició con la configuración del entorno Docker. En primera instancia, se configuran aspectos fundamentales del sistema, como la definición del usuario, zona horaria, red WiFi, el idioma, el espacio de memoria para el sistema operativo y la configuración de la potencia de procesamiento (máxima potencia para tarjetas Jetson Nano con alimentación 110v) (Fig. 26).

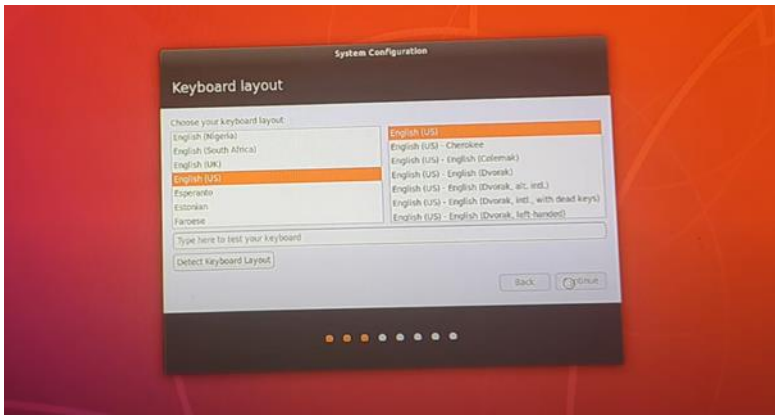

Figura 26 Inicialización Jetson nano. Fuente: Autor propio

Una vez instalada la interfaz en la tarjeta Jetson Nano, se procedió a abrir un terminal para instalar el JetPack de Jetson Nano, dentro del JetPack se encuentra el entorno de Docker (Fig. 27). Este entorno contiene las librerías necesarias para la ejecución del proyecto, para ello se descargan y actualizan las librerías necesarias para obtener un mejor rendimiento y resultado a la hora de entrenar la red. Culminada la instalación del paquete de Docker, se crea en el sistema una carpeta contenedora llamada Jetson nano la cual alberga las librerías, modelos pre-entrenados y de igual forma, es la carpeta contenedora para cargar el dataset del proyecto.

Figura 27 Interfaz Jetson nano. Fuente: Autor propio

3.3. Zona de estudio

En cuanto a la zona de estudio, cabe destacar que el entorno ideal para la preservación de las abejas debe contar con una temperatura superior a 10°C. Este rango mínimo de temperatura es crucial para garantizar el bienestar de las abejas, ya que influye directamente en su actividad metabólica, comportamiento y capacidad para realizar tareas esenciales como la recolección de néctar y polen. Para la recolección de las muestras de campo de las abejas portadoras de polen, se realizó una visita al municipio de Santa Rosa – Cauca, el cual cuenta con una extensión de 4.479 km^2 lo que corresponde al 14,7% del territorio departamental, con una altura de 1700 metros sobre el nivel del mar y una temperatura promedio de 19°C con un clima templado (Fig. 28). Al llegar al municipio, se ubicó un apiario en la zona rural en cual contaba con 7 colmenas, en las cuales se pudo recolectar muestras en cada una de ellas. (Fig. 29 y 30). Para la adquisición de muestras de campo se utilizó como hardware principal una cámara Logitech C920 HD (Fig. 25), esta cámara fue fijada en la entrada a 20cm de la entrada de la colmena para poder filmar el ingreso de las abejas con polen, en este proceso se obtuvo un total de 19 minutos de grabación para los cuales se utilizó un computador para almacenar la información, y como hardware de apoyo se utilizó un smartphone con la finalidad de enfocar un poco más las abejas, en este proceso se obtuvieron 6 minutos de grabación.

Figura 28 Ubicación geográfica. Fuente: Autor Propio

Figura 29 Granja de abejas. Fuente: Autor Propio

Figura 30 Muestras de Abejas en campo. Fuente: Autor propio

3.4. Elementos para el desarrollo del proyecto

Para el desarrollo del proyecto, se realizó una meticulosa selección de los materiales y equipos necesarios con el objetivo de garantizar la calidad de los datos recolectados y la precisión en los resultados obtenidos. Cada componente fue elegido considerando criterios técnicos, funcionales y de compatibilidad, asegurando así un desempeño óptimo del sistema en todas las etapas del proceso. para ello se utilizó:

3.4.1. Computador asus x441u

Con el fin de procesar grandes cantidades de datos, se optó por un computador Asus x441u, el cual cuenta con un sistema operativo Windows 11, 8Gb de Ram y tarjeta de video Nvidia Geforce 920MX (Fig. 31).

Figura 31 Computador Asus. Fuente: Autor propio

3.4.2. Software de etiquetado

Para el etiquetado de las imágenes se utilizó el software de Labelimg (Fig. 32). Este software permite elegir diferentes formatos para etiquetas tales como cvs, Json, Yolo y voc (xml). Para este proyecto se utilizó el formato voc el cual brinda las coordenadas en formato xlm con la ubicación del objeto (Fig.33). Adicional a ello, este permite tener diferentes clases de objetos en una misma imagen, para este proyecto se eligieron las clases pollen y not pollen (Fig. 34 y 35).

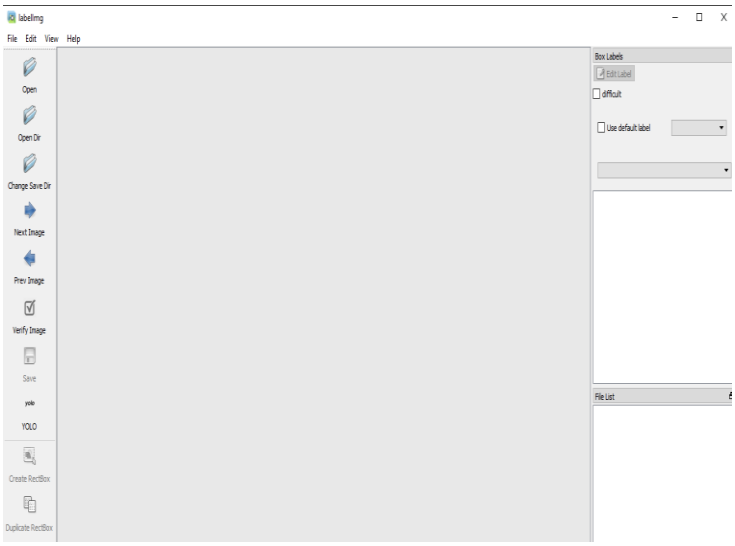

Figura 32 Software LabelImg. Fuente: Autor propio

Figura 33 Formato VOC: Fuente: Autor propio

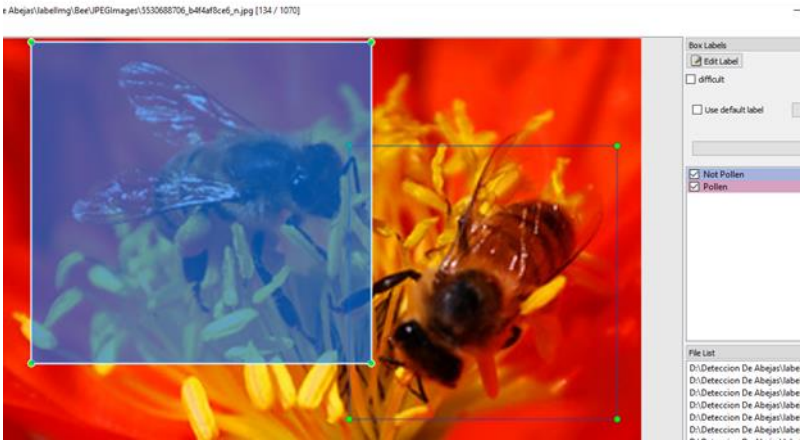

Figura 34 Clases en etiquetas. Fuente: Autor propio

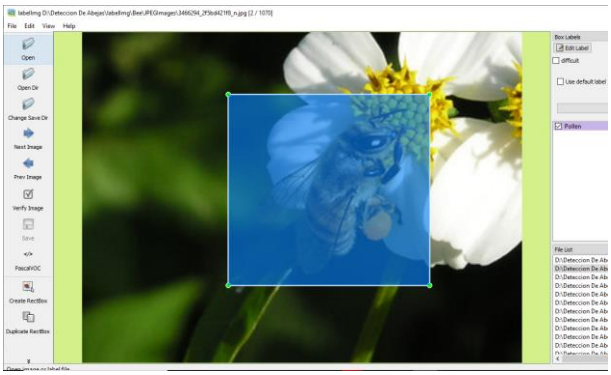

Figura 35 Etiquetado de imágenes: Fuente: Autor propio

3.4.2.1. Software para procesamiento de dataset

Para el procesamiento del dataset se utilizó el software de Jupyter Notebook Online, el cual en conjunto con Google Colab permite procesamientos en la nube. Este software permite utilizar librerías como pandas, glob y os, las cuales se encargan de eliminar las extensiones .jpg a las imágenes anteriormente etiquetadas y almacena los datos en un dataframe de pandas. Adicional a ello, permite realizar una Split para dividir los datos que se usaron para validación y entrenamiento en la red neuronal.

3.4.2.2. Software de entrenamiento y prueba

Para el entrenamiento del sistema se inicia el entorno de Docker. El cual es un sistema operativo de contenedores, es decir, cada contenedor se ejecuta de forma separada generando su propio sistema de archivos, su propia red y su propio árbol de procesos aislados. En este entorno, llamado “Bee” se creó un espacio personalizado que facilito la interacción entre los datos ahí almacenados (Dataset, códigos de compilación y archivos de prueba), permitiendo un flujo de trabajo eficiente

3.5. Preparación del dataset

Para la preparación del dataset, se inicia seleccionando las imágenes que se van a etiquetar, posterior a ello, se generaron las etiquetas que se utilizaron en un archivo txt (label.txt), acto seguido, se inicializó el software labelimg para realizar el procedimiento, al ingresar al software se navegó hasta localizar el comando “open dir” el cual permitió importar las imágenes a etiquetar, mediante el comando “Change save dir” se seleccionó la carpeta donde se almacenaron las etiquetas y se seleccionó el formato con el que se guardaron las etiquetas (Formato VOC en nuestro caso). Una vez configurado el software se procedió a realizar las etiquetas, para ello se avanzó por cada imagen cargada generando un bounding box sobre el objeto y eligiendo la clase a la que pertenece. En el proceso de etiquetado se realizó en 1.070 imágenes donde 477 imágenes son de abejas portadoras de polen y 593 imágenes son de abejas no portadores de polen.

Una vez finalizado el etiquetado, se procedió a utilizar el software de Jupyter Notebook, en donde se eliminaron las extensiones jpg. Una vez almacenado el dataframe, se realizó un Split (división de datos) mediante el cual se divide el dataset en imágenes de entrenamiento (727), imágenes de prueba (182) e imágenes de validación (161). Para culminar con la preparación del dataset, se guardaron los datos obtenidos del Split en archivos .txt. Los datos de entrenamientos se almacenan en el documento train.txt, los datos de validación en val.txt y los datos de prueba en test.txt. Es indispensable que los archivos contengan estos nombres para que el entorno de Docker pueda ejecutar los archivos (Fig. 37).

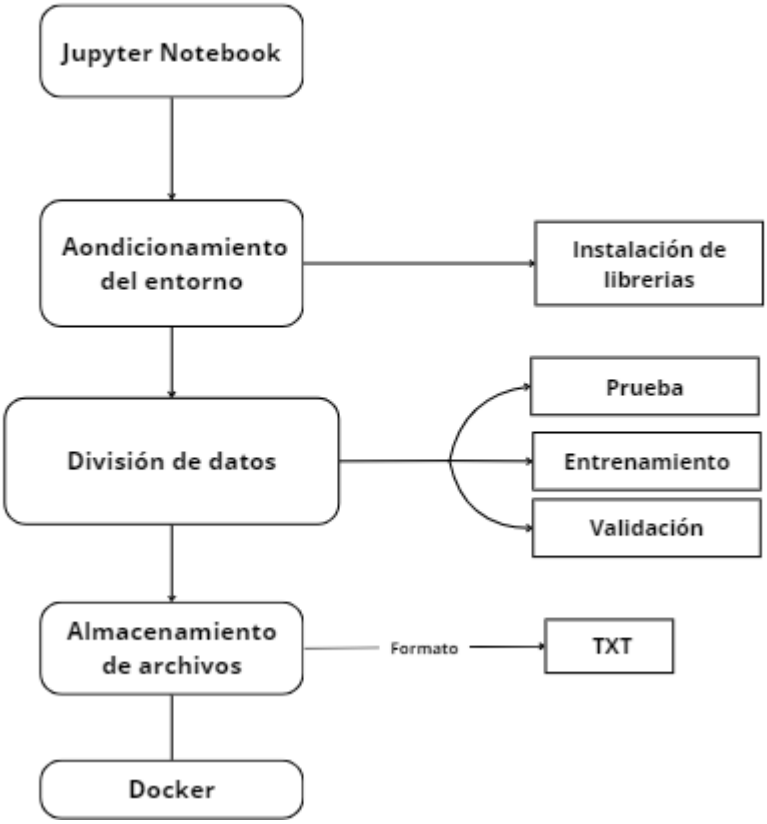

Figura 36 Diagrama de flujo de programación Jupyter Notebook. Fuente: Autor Propio

3.6. Desarrollo de la red neuronal

Una vez el dataset se encontró preparado, se ingresó al entorno de Docker, en él, se buscó la carpeta llamada Jetson nano, estando dentro de la carpeta Jetson nano se trasfirieron los archivos correspondientes al dataset en este caso las imágenes y los archivos .txt, estos archivos se cargaron dentro de la carpeta de data la cual estaba contenida en la ruta Python > Deteccion > ssd. Posterior a ello, se abrió una terminal y se ejecutó el docker (Fig. 37). Dentro de Docker se utilizó el comando “cd” para navegar hasta la carpeta que contenía el dataset y los archivos .txt, una vez dentro de la carpeta contenedora (carpeta data) se configuró el sistema para iniciar el proceso de entrenamiento. En primer lugar, se invocó la función de entrenamiento mediante el script --train_ssd.py, A continuación, se especificó el tipo de datos con el que se

etiquetó el dataset mediante el parámetro --dataset-type = voc, seguido de esto, se cargó el dataset utilizando el parámetro --data = data/Bee, Luego, se indicó la carpeta de destino para guardar el modelo entrenado con el parámetro --model-dir = models/Bee. Se definieron los hiperparámetros del proceso de entrenamiento: el tamaño del batch con el parámetro --Batch-size = 2, y el número de épocas a realizar durante el entrenamiento con el parámetro --epoch = 400, De este modo, se dio inicio al proceso de entrenamiento (Fig. 36).

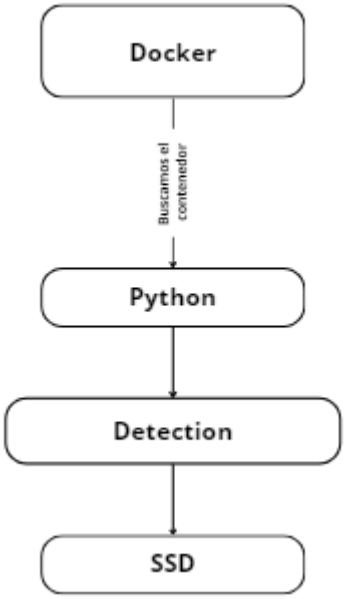

Figura 37 Navegación en Docker. Fuente: Autor propio

El tiempo de entrenamiento de la red neuronal varía según la configuración de hiperparámetros utilizados; específicamente, a mayor número de épocas y menor tamaño de lote (batch-size), mayor será el tiempo de entrenamiento. Para este caso en particular, los entrenamientos tardan entre 3 a 15 horas de entrenamiento. Una vez finalizado el entrenamiento, se genera un archivo en formato .py (onnx.py) este archivo contiene la información de la neurona que tuvo el mayor porcentaje de precisión en el entrenamiento y un archivo .txt con los labels (Labels.txt). Sin embargo, el sistema no realiza un guardado automático de estos archivos, por consiguiente, es necesario

guardarlo de forma manual mediante la función export (onnx_export.py), una vez ejecutada la función se guarda el archivo en la carpeta models.

Una vez obtenido el archivo onnx, se procedió a cargar la información necesaria para realizar pruebas de funcionamiento. Para ello, se utilizó el software Regedit ejecutado desde el Docker, en el cual se generó un código con extensión .py. Este código fue parametrizado con las condiciones necesarias para procesar el video y realizar la inferencia. En este sentido, se emplearon las librerías de pytorch, cv2, Jetson.inference, Jetson.utils, numpy (como np) y argparse.

La primera librería utilizada en el código fue argparse, la cual facilitó la activación de la función threshold. Esta función permitió definir el umbral de flexibilidad para la detección entre las clases Pollen y Not Pollen, ajustando dicho umbral de acuerdo con las condiciones del entorno de prueba. A continuación, se utilizó, fue la Jetson.inference, donde se definió la red neuronal empleada, que en este caso fue DetecNet-V2. Luego, se cargaron los archivos onnx y labels.txt obtenidos durante el entrenamiento. También se parametrizó la salida visualizada en pantalla, en este caso, los puntajes (output-cvg=scores) y los bounding boxes (output-bbox = boxes).

La siguiente librería utilizada fue la Jetson.utils, mediante la cual se dio apertura a la cámara web mediante la función Jetson.utils.videoSource y mostrar en pantalla la imagen capturada por la cámara utilizando Jetson.utils.videoOutput. Posteriormente, se utilizó cv2, que facilito la carga de archivos .mp4 mediante la función cv2.videoCapture. Para ajustar la calidad de la imagen, se definió el número de pixeles en el cuadro delimitador mediante la subfunción cv2.CAP_PROP_FRAME_HEIGHT, 480 o cv2.CAP_PROP_FRAME_WIDTHT, 480.

Finalmente, se emplearon nummpy, cv2 y Jetson.utils para generar el color de los bounding boxes, los textos y el porcentaje de precisión en las detecciones (Fig. 38). En el caso de las detecciones con cámara web, la función de cambio de color fue jetson.utils.cudaFromNumpy (cv2.cvtColor (frame,cv2.COLOR_BGR2RGBA).astype(np.float)), mientras que para los archivos .mp4, se usaron cv2.putText (para mostrar texto), (frame,item, (int

(detect.Left),int(detect.Top)) cv2.FONT_HERSHEY_SIMPLEX,1,color,3) y

cv2. Rectangle (frame, (int(detect.Left), int(detect.Top)), (int(detect.Right), int(detect.Bottom)), (255,0,0), 1) (para dibujar las cajas delimitadoras)

Para la ejecución del archivo, se utilizó Docker mediante la función --Docker/run.sh --volumen junto con la dirección de la carpeta contenedora. Además, se ejecutó el comando --device=/dev/video0 para inicializar el acceso a la camera web. Una vez que Docker cargó con normalidad, se llamó el archivo anteriormente creado y se parametrizó la flexibilidad de detección con threshold, lo que permitió iniciar las pruebas del sistema (Fig. 39).

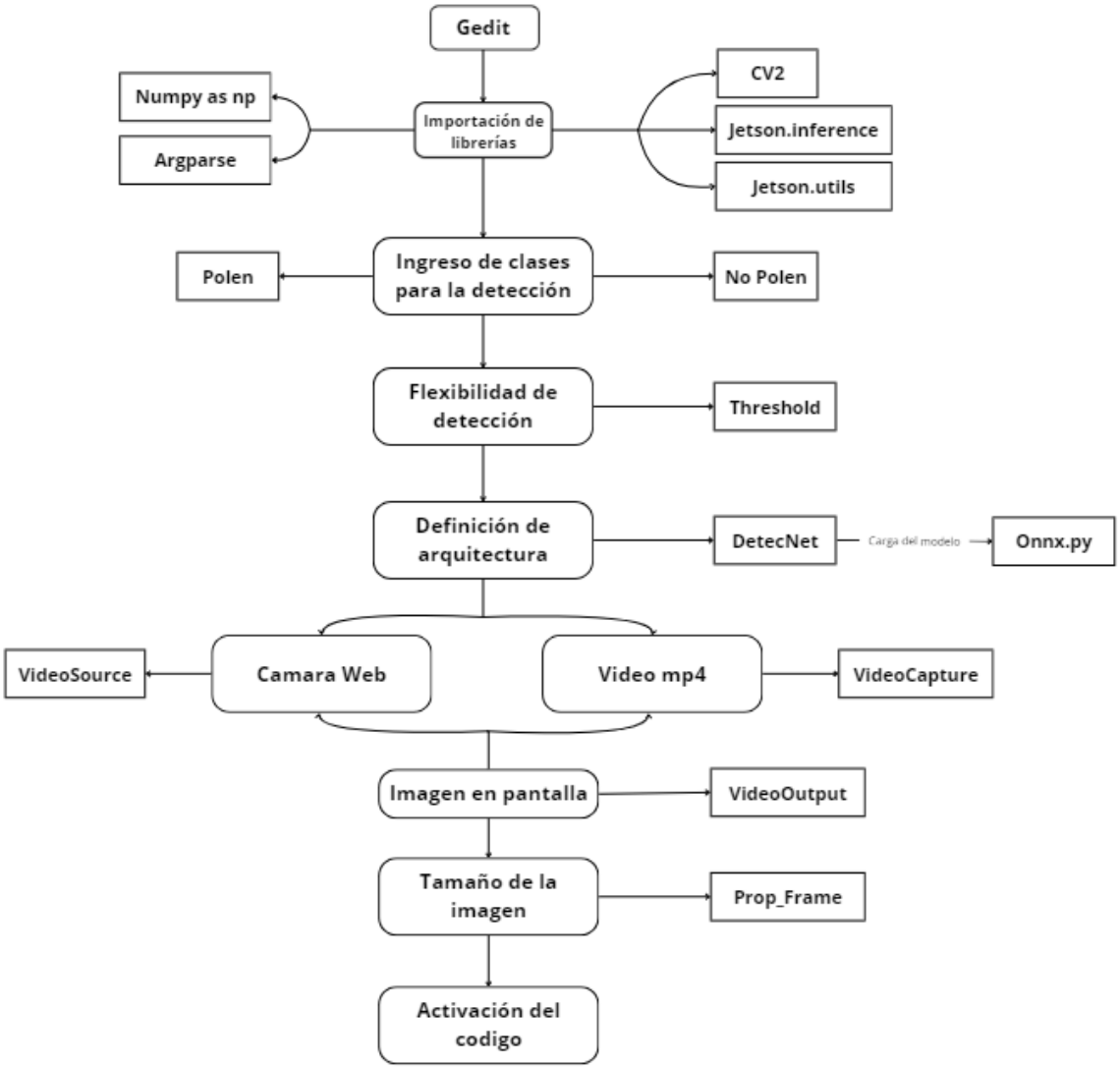

Figura 38 Diagrama de flujo en software gedit. Fuente: Autor propio

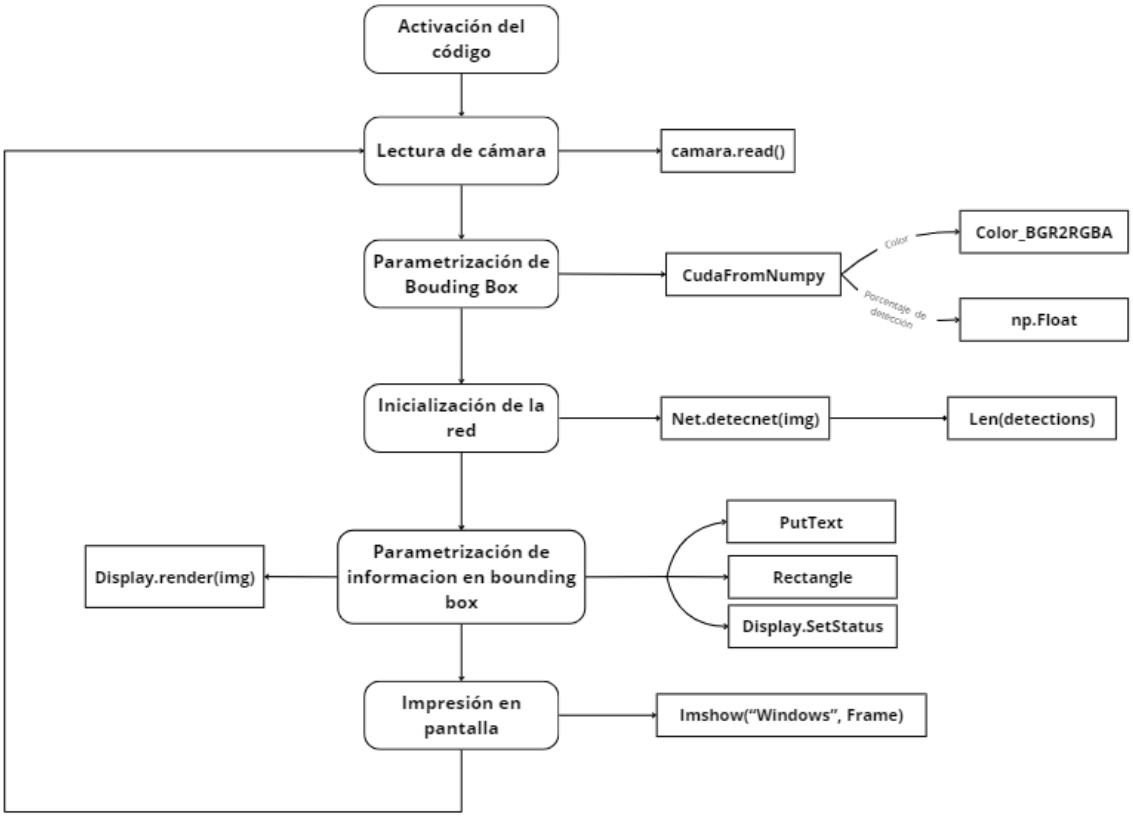

Figura 39 Diagrama de flujo de activación de código. Fuente: Autor propio

CAPÍTULO 4

PUESTA EN MARCHA DEL SISTEMA

En este capítulo se analizará el funcionamiento del sistema, apoyándonos en métricas de desempeño que permitirá analizar de forma analítica la funcionalidad del sistema.

4. Resultados y Análisis

Una vez finalizada la preparación de la memoria microSD, se procede a iniciar la tarjeta Jetson Nano alimentada a 110V. Posteriormente se conectan los periféricos y se configuran los parámetros de la tarjeta. Esto incluye la definición del usuario, zona horaria, red WiFi, el idioma, el espacio de memoria para el sistema operativo y la configuración de la potencia de procesamiento (máxima potencia para tarjetas Jetson Nano con alimentación 110v):

Entrenamiento de red neuronal										
Entrenamiento	1	2	3	4	5	6	7	8	9	10
Batch-size	2	1	2	4	2	2	10	2	5	4
Epochs	50	50	80	100	150	200	350	400	250	600
Threshold	4.0	5.8	2.0	2.5	3.0	3.8	5.5	3.8	1.8	4.0

Tabla 1 Entrenamiento. Fuente: Autor propio

Dentro de las validaciones realizadas se identificó que el threshold ideal para elsistema está entre 3.5 a 4.5. Con valores inferiores a 3.5 la detección generaba falsos positivos (Fig. 40 y 42) y con valores superiores a 4.5 generaba detecciones indeterminadas (detecciones sin objetos en pantallas) (Fig. 43). La variación del batch-size con relación a las épocas de entrenamiento es inversamente proporcional, a mayor batch-

size menor cantidad de épocas de entrenamiento. El batch-size más alto permitido por la tarjeta Jetson Nano fue de 10, con el cual se realizaron 350 épocas de entrenamiento. Para este sistema se concluyó que la mejor respuesta del sistema se dio en el entrenamiento número 8, en donde el tiempo de entrenamiento fue de 10 horas aproximadamente y una precisión superior al 80%.

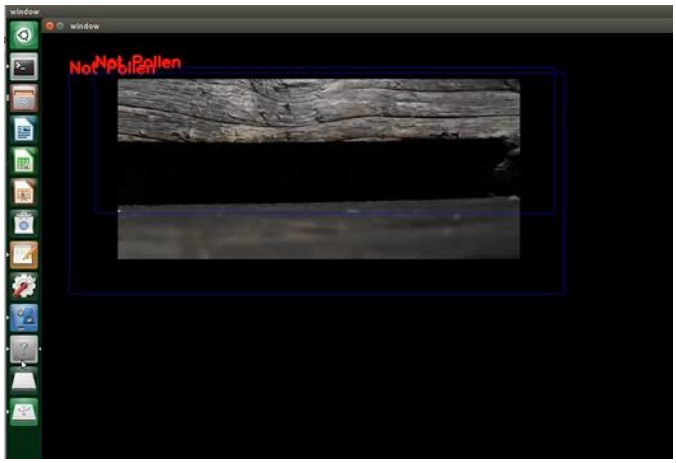

Fig. 40 Falso Positivo. Fuente: Autor propio

Fig. 41 Falso positivo. Fuente: Autor propio

Fig. 42 Detección errónea. Fuente: Autor propio

En el entrenamiento número 8 se realizaron un total de 15 validaciones, 9 de las validaciones se realizaron mediante archivos .mp4 y 6 de las validaciones con cámara web (Fig. 43 y 44).

Figura 43 Procesamiento en video. Fuente: Autor propio

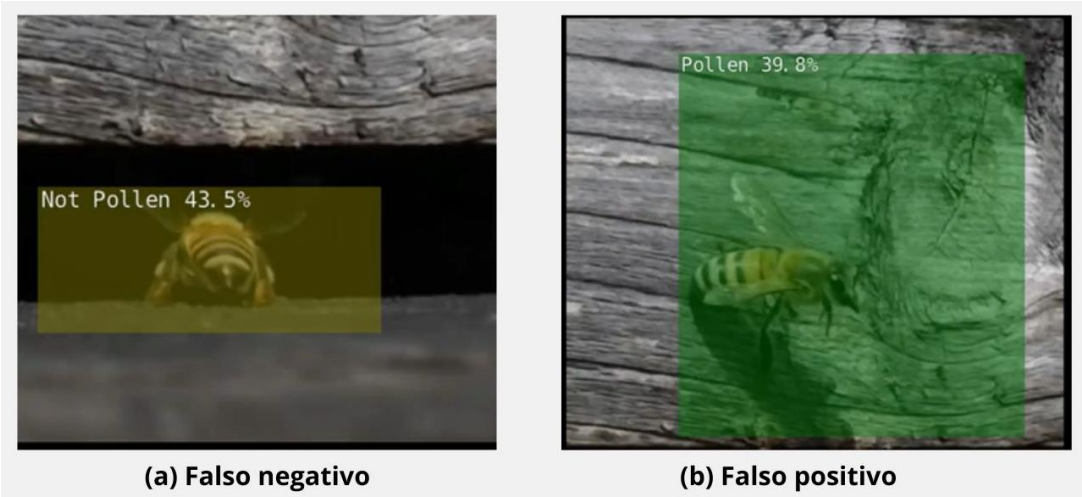

Figura 44 Procesamiento en cámara web. Fuente: Autor propio

Dentro de las validaciones y con la finalidad de generar un análisis cuantitativo se realizó un conteo de las detecciones realizadas. Para ello se consideraron 4 diagnósticos entre los cuales se tiene verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN). Realizado el conteo, los resultados se pueden observar en la Tabla 2:

Conteo de detecciones					
Detecciones / Videos	TP	TN	FP	FN	Total de imágenes
1	11	4	1	1	17
2	9	3	6	3	21
3	12	3	5	2	22
4	12	4	2	3	21
5	11	2	6	1	20
6	13	5	5	0	23
7	8	4	2	5	19
8	14	6	2	1	23
9	15	4	1	1	21
10	14	2	1	2	19
11	9	5	3	1	18
12	11	3	0	0	14
13	15	3	2	3	23
14	15	3	0	1	19
15	14	5	0	2	21
TOTAL	**183**	**56**	**36**	**26**	**301**

Tabla 2 Conteo de detecciones. Fuente: Autor propio

Para un análisis más profundo de los resulatdos obtenidos, se utilizaron dos metricas de desmpeño: Precision y F1-score. En el calculo de la precision determinó en dos momentos, el primero de ellos es la obtencion mediante las funciones de codigo anteriormente mencionadas (Fig. 45 Y 46); el segundo es el calculo analitico de las predicciones, para ello utilizamos la ecuacion 4 para el calculo de la presicion y la ecuacion 3 y 6 para calcular F1-score.

Fig. 45 Precisión del sistema. Fuente: Autor propio

Fig. 46 Precisión del sistema. Fuente: Autor propio

Como resultado de las métricas de desempeño analíticas, se calcularon los promedios de los diagnósticos obtenidos en la Tabla 2. En promedio, se detectaron 301 imágenes de abejas con y sin polen, con diagnósticos promedio de TP = 183, TN = 56, FP = 36 y FN = 26. Con estos promedios, se obtuvo una precisión del 84% (Ecuación 9) y un F1-score del 86% (Ecuación 11), lo que indica que el sistema tiene un rendimiento superior al 80% en la detección de abejas con polen. Esto lleva a concluir que el sistema es eficiente.

$$Presicion = \frac{TP}{TP + FP} = \frac{183}{183 + 36} = \frac{183}{219} = 0.84$$

Ecuación 9 Calculo de precisión. Fuente: Autor propio

$$Recall = \frac{TP}{TP + FN} = \frac{183}{183 + 26} = 0.88$$

Ecuación 10 Calculo recall. Fuente: Autor propio

$$F1 = X = \frac{Presicion + Recall}{2} = \frac{0.84 + 0.88}{2} = \frac{1.72}{2} = 0.86$$

Ecuación 11Calculo F1-score. Fuente: Autor propio

Con el propósito de validar el comportamiento de la red, en la Tabla 3 se encuentran las métricas de desempeño aplicadas a cada una de las validaciones realizadas. Se determinó que la validación número 12 obtuvo la métrica de desempeño más alta (100%), mientras que la validación número 2 alcanzó una métrica de desempeño con el índice más bajo (aproximadamente 68%).

Métricas de desempeño			
Métricas / Videos	Presicion	Recall	F1-Score
1	0,92	0,92	0,92
2	0,60	0,75	0,68
3	0,71	0,86	0,78
4	0,86	0,80	0,83
5	0,65	0,92	0,78
6	0,72	1,00	0,86
7	0,80	0,62	0,71
8	0,88	0,93	0,90
9	0,94	0,94	0,94
10	0,93	0,88	0,90
11	0,75	0,90	0,83
12	1,00	1,00	1,00
13	0,88	0,83	0,86
14	1,00	0,94	0,97
15	1,00	0,88	0,94
Total	0,84	0,88	0,86

Tabla 3 Métricas de desempeño. Fuente: Autor propio

A continuación, se exhiben las fotografías que muestran las detecciones de abejas con y sin polen realizadas por el sistema.

La figura 49 corresponde a una deteccion realizada por procesamiento de video en formato mp4, se observa un bounding box generado en la imagen corresponde a una deteccion positiva de deteccion, en este caso Pollen (Fig. 47).

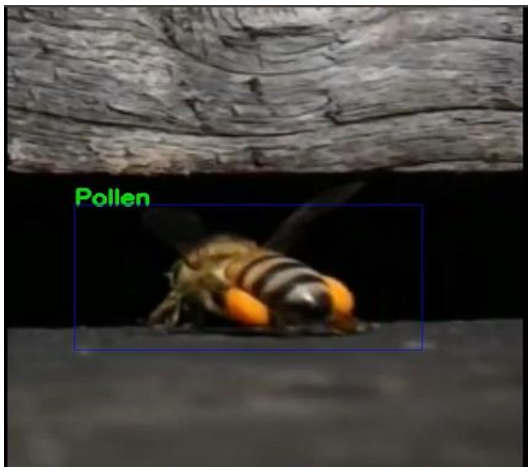

Fig. 47 Abeja portadora de polen detectada en video. Fuente: Autor propio

La figura 50 corresponde a una detección realizada mediante cámara web y procesamiento en video, en este caso se anexo un porcentaje de detección el cual arrojo una predicción positiva para la clase Pollen con un 52,9% de precisión para la imagen a, en la imagen b una predicción del 52% para la clase Not Pollen y finalmente en la imagen c y d se observa un aumento en la precisión de predicción con un 54.8% y 78.8% para clase Pollen respectivamente (Fig.48).

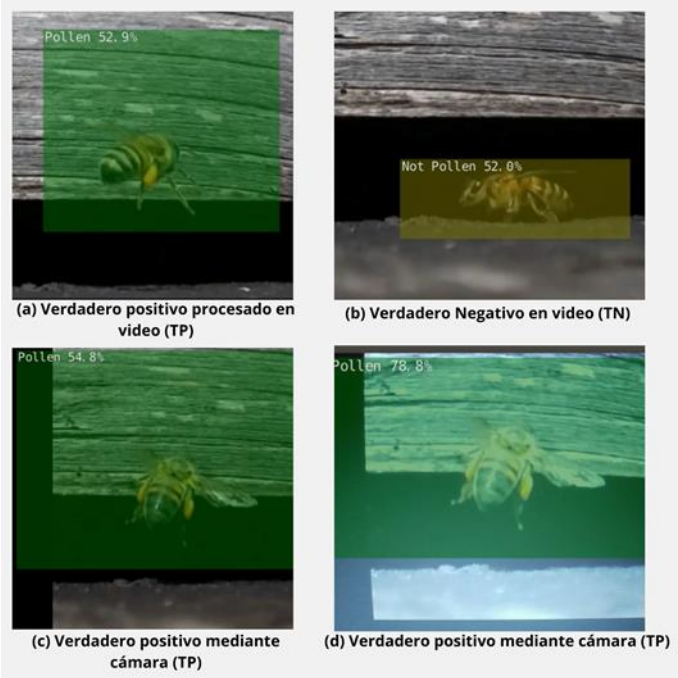

Figura 48 Detección de abejas portadoras de polen. Fuente: Autor propio

En la figura 51 se puede observar un doble bounding box lo que nos indica una doble predicción que conlleva a un falso positivo por la sobreposición de la detección (Fig. 49).

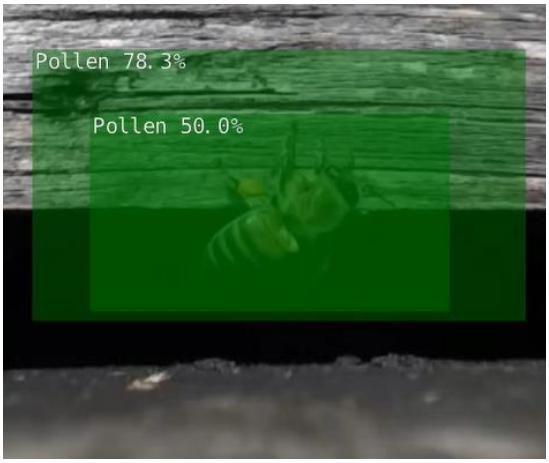

Fig. 49 Precisión de detección con falso positivo. Fuente: Autor propio

CAPÍTULO 5

CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se abarcará las conclusiones del sistema, las limitaciones que encontramos a lo largo del desarrollo del sistema y las mejoras del sistema.

5. Conclusiones

Se logró desarrollar un sistema inteligente para la detección de abejas portadores y no portadoras de polen mediante un sistema embebido. Esta combinación hardware – software permitió desarrollar un sistema de detección que mediante código alcanza una precisión del 95% en su máximo rendimiento, y un promedio analítico de precisión del 84%.

Durante los entrenamientos, se observó que la tarjeta Jetson Nano tiene un límite de 400 épocas después del cual experimenta un bloqueo. Para obtener resultados con un mayor número de épocas de entrenamiento, se utilizó un código en el entorno de Jupyter Notebook. En Jupyter Notebook se logró realizar un total de 600 épocas de entrenamiento con un batch-size de 2. Sin embargo, Google Colab tiene limitaciones de recursos para el entrenamiento, con un límite de 10 GB de capacidad o 12 horas de funcionamiento

Se encontró un inconveniente adicional en la tarjeta Jetson Nano relacionado con el procesamiento de archivos mp4 en Docker. Se experimentó una desaceleración del video una vez que el sistema devolvía la imagen procesada, lo que requirió aumentar la velocidad en 4X para lograr una reproducción más fluida. Esto resultó en retrasos

durante las validaciones y provocó un sobrecalentamiento en la tarjeta. Los problemas de sobrecalentamiento inicialmente generaron alertas por alta temperatura, seguido de bloqueos en la tarjeta Jetson Nano y, en algunos casos extremos, fue necesario formatearla.

En el procesamiento de archivos, fue necesario aumentar el tamaño de las imágenes debido a que las imágenes con un tamaño inferior a 3 cm no permitían generar bounding boxes adecuados. Para objetos pequeños, se requiere hardware de alta definición; en este proyecto se utilizó una cámara Logitech C920, la cual no proporcionó imágenes de alta calidad. Por tanto, esto resultó en una pérdida de calidad en las muestras tomadas en campo al aumentar el tamaño de las imágenes, lo que afectó negativamente la precisión durante el procesamiento.

Aumentar el batch-size permite reducir el número de épocas necesarias para el entrenamiento, lo cual es eficiente. Sin embargo, está limitado por la Jetson Nano, con un batch-size máximo de 10. Por tanto, el análisis de los resultados sugiere que el sistema requiere una cuidadosa calibración del threshold para operar eficientemente, y que el tamaño del batch y el número de épocas deben ser ajustados de acuerdo con las capacidades del hardware

Finalmente, la tarjeta Jetson Nano presenta un consumo elevado de corriente, lo cual demanda una fuente de energía portátil de alta potencia para su operación en campo. Este requisito plantea desafíos significativos para su integración en entornos abiertos.

5.1. Trabajos futuros

A continuación, se proponen posibles mejoras para el sistema de detección de abejas portadoras de polen.

Se podría instalar un sistema de energías renovables altamente eficiente para alimentar la tarjeta Jetson Nano (esta energía dependería de la zona donde se vaya a implementar el sistema). Esto facilitaría las pruebas en campo abierto y permitiría realizar procesamientos de información en tiempo real de manera más efectiva.

También se podría adoptar el uso de equipos de alta definición como una cámara Eos rabel t7, Osmo pocket 3, Osbot tiny, entre otro, lo cual permitiría una captura en tiempo real de las abejas aumentando la precisión de las detecciones y así enriquecer la base de datos con imágenes de mayor calidad (Imágenes en 4k y videos en FHD).

Adicional a ello se podría implementar sensores que contribuyan a la detección y cuidado de las colmenas tales como sensores ultrasónicos (miniatura, con boquilla, robustos, entre otros), sensores inductivos y capacitivos, sensores de humedad (Sr300s, Mth100, entre otros) estos son fundamentales para monitorear de manera efectiva la salud y el comportamiento de las abejas.

5.2. Bibliografía

[1] c. Laverde, L. Egea, D. Ridriguez y J. Peña , «AGENDA PROSPECTIVA DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO PARA LA CADENA PRODUCTIVA DE LAS ABEJAS Y LA APICULTURA EN COLOMBIA CON ÉNFASIS EN MIEL DE ABEJAS,» Bogota, 2010.

[2] S. k. Berkaya, . E. S. Gunal y S. Gunal, «Deep learning-based classification models for beehive monitoring,» *Ecological Informatics,* 2021.

[3] Oie, «Organizacion Mundial de Sanidad Animal,» 2021. [En línea]. Available: https://www.oie.int/es/enfermedad/enfermedades-de-las-abejas/. [Último acceso: 24 03 2022].

[4] T.-t. Lin, E.-C. Yang, T. N. Ngo y D. Arcega, «Automated monitoring and analyses of honey bee pollen foraging behavior,» *Computers and Electronics in Agriculture,* 2021.

[5] I. F. Rodriguez, R. Megret, E. Acuña, J. L. Agosto-Rivera y T. Giray, «Recognition of Pollen-bearing Bees from Video using Convolutional Neural Network,» ScienceDirect, Puerto Rico, 2018.

[6] MinAgricultura, «SIOC,» 2015. [En línea]. Available: https://sioc.minagricultura.gov.co/Apicola/Documentos/2015-06-30%20Cifras%20sectoriales.pdf. [Último acceso: 30 03 2022].

[7] A. Cabo, «El Polen recogida, manejo y apliacones,» Madrid - España, 2020.

[8] National Geograpich, «National Geographic España,» 2022. [En línea]. Available: https://www.nationalgeographic.com.es/animales/abeja. [Último acceso: 07 03 2022].

[9] N. Mungsan, «Origen y diversidad del polen apicola,» Madrid, https://eprints.ucm.es/id/eprint/63006/, 2018.

[10] Y. Ocaña, L. A. Valenzuela y L. Garro, «Inteligencia artificial y sus implicaciones en la educación superior,» Lima - Perú, 2019.

[11] D. Hinestroza, «EL MACHINE LEARNING A TRAVÉS DE LOS TIEMPOS, Y LOS APORTES A LA HUMANIDAD,» Pereira , https://repository.unilibre.edu.co/handle/10901/17289, 2018, p. 17.

[12] E. Stevens, L. Antiga y T. Viehmann, Deep Learning with Pytorch, Estados Unidos De America : Manning Publications Co., 2017.

[13] O. Picazo, «Redes Neuronales Convolucionales Profundas Para El Reconocimiento De Emociones En Imagenes,» de *Politecnica Biblioteca Universitaria* , Madrid - España, https://oa.upm.es/51441/, 2018.

[14] J. I. Bagnato, Aprende Maching learnig, Spanish Edition ed., Leanpub, 2020.

[15] E. Sanchez, «DETECCIÓN DE PERSONAS MEDIANTE REDES CONVOLUCIONALES,» de *Universidad Autonoma de Madrid* , Madrid - España, https://repositorio.uam.es/bitstream/handle/10486/679353/Sanchez_Atienza_Esther_tfg.pdf?sequence=1, 2017.

[16] w. Lui, D. Anguelov , D. Erhan , C. Szegedy, S. Reed, C.-Y. Fu y A. Berg, «Cornell University,» *Computer Vision and Pattern Recognition,* vol. 9905, pp. 21-37, 2016.

[17] MathWorsk, «MathWorsk,» 2 07 2021. [En línea]. Available: https://es.mathworks.com/solutions/image-video-processing/semantic-segmentation.html. [Último acceso: 11 03 2022].

[18] P. Orellana, «SEGMENTACIÓN SEMÁNTICA Y RECONOCIMIENTO DE LUGARES USANDO CARACTERÍSTICAS CNN PRE-ENTRENADAS,» de

Repositorio Academico de la Universidad de Chile, Santiago De Chile, https://repositorio.uchile.cl/handle/2250/173733, 2019.

[19] A. Ruiz, «Universidad Zaragoza,» 2018. [En línea]. Available: https://zaguan.unizar.es/record/69800?ln=es#. [Último acceso: 30 03 2022].

[20] Nvidia, «Nvidia Corporation,» 2022. [En línea]. Available: https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano-developer-kit/. [Último acceso: 10 03 2022].

[21] Sigma Electronica, «Sigma Electronica,» 2022. [En línea]. Available: https://www.sigmaelectronica.net/producto/raspberry-pi-3/. [Último acceso: 15 03 2022].

[22] R. Borja, A. Monleon y J. Rodellar, «Estandarización de Métricas de Rendimiento para Clasificadores Machine y Deep Learning,» *Risti,* pp. 172-184, 2020.

[23] A. Noren, «Sitio Big Data,» 3 9 2019. [En línea]. Available: https://sitiobigdata.com/2019/01/19/machine-learning-metrica-clasificacion-parte-3/#. [Último acceso: 23 03 2022].

[24] E. Pacanchique, «IMPLEMENTACIÓN DE UNA TÉCNICA DE INTELIGENCIA ARTIFICIAL PARA EL ANÁLISIS DE IMÁGENES EN BÚSQUEDA DE LA IDENTIFICACIÓN DE COLILLAS DE CIGARRILLOS EN ÁREAS PÚBLICAS,» Creative Commons, Bogotá - Colombia, 2020.

[25] H. Jinchan , Y. Xiaxia y C. Chudong , «MCMC Guided CNN Training and Segmentation for Pancreas Extraction,» IEEE, China, 2021.

[26] G. Muñoz, «Deep learning con TensorFlow,» 2016.

[27] J. Tao, «Introduction to Deep Learning with Tensorflow,» Texas A&M University, 2020.

[28] A. Seshadri, «Manejo Integrado de colmenas para apicultores de Colorado,» Colorado , 2019.

[29] K.-C. Wu, E.-C. Yang, T. N. Ngo y T.-T. Lin, «A real-time imaging system for multiple honey bee tracking and activity monitoring,» *Computers and Electronics in Agriculture,* vol. 163, 2019.

[30] N. Corporation, «Nvidia,» 2024. [En línea]. Available: https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano/product-development/. [Último acceso: 02 01 2024].

[31] S. k. Berkaya, E. S. Gunal y S. Gunal, «Deep learning-based classification models for beehive monitoring,» *Ecological Informatics,* vol. 64, 2021.

[32] T.-T. Lin, E.-C. Yang, T. N. Ngo y D. Arcega , «Automated monitoring and analyses of honey bee pollen foraging behavior,» *Computers and Electronics in Agriculture,* vol. 196, 2021.

[33] I. F. Rodriguez, R. Megret, E. Acuña, J. L. Agosto-Rivera y T. Giray, «Recognition of Pollen-bearing Bees from Video using Convolutional Neural Network,» Lake Tahoe, NV, USA, 2018.

[34] J. Schaefer , M. Milling, . B. Schuller , B. Bauer, J. O. Brunner, C. Traidl-Hoffmann y A. Damialis, «Towards automatic airborne pollen monitoring: From commercial devices to operational by mitigating class-imbalance in a deep learning approach,» *Science of The Total Environment,* vol. 796, 2021.

[35] A. Cabo, «El Polen recogida, manejo y apliacones,» Ministerio De agricultura España, Madrid - España, 2020.

ANEXOS

```
    #Instalación de librerias
!pip install pandas
!pip install scikit-learn

    #Importación de librerias
import os
import glob
import pandas as pd

    #Importación de datos
data_path = "D:/Desarrollos/JetsonNano/labelImg/Bee/JPEGImages"
data_dir_list = os.listdir(data_path)
img_data_list=[]
num_images=0
for image_file in data_dir_list:
img_data_list.append(image_file.split('.jpg')[0])
dataset_df = pd.DataFrame(img_data_list)
dataset_df.head()

    #Split de archivos
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(
dataset_df,
 test_size=0.15,
random_state=2
)
train_df, valid_df = train_test_split(
 train_df,
test_size=0.2,
random_state=2
)
print(train_df.size, valid_df.size, test_df.size)

    #Generación de archivos contenedores
train_df.to_csv('Bee/ImageSets/Main/train.txt', index=False, header=None)
valid_df.to_csv('Bee/ImageSets/Main/val.txt', index=False, header=None)
test_df.to_csv('Bee/ImageSets/Main/test.txt', index=False, header=None)

  return go(f, seed, [])
}
```

Figura 50 Código de procesamiento de imágenes.

```
    # Ejecución de Docker
Cd Jetson-inference
Docker/run.sh

    # Navegación en docker
Cd Python/training/detection/ssd/

    # Activación de la red para el entrenamiento
Python3 train_ssd.py --dataset-type=voc --data=data/Bee --model-dir=models/Bee --batch-size=2 --epoch=400

    # Almacenamiento del archivo onnx
Python3 onnx_export.py --model=models/Bee

    # Inicialización de deteeción
Cd Jetson-inference
Docker/run.sh  --volume -/Abejas:/Abejas/ --device=/dev/video0
Python3 /Abejas/Abejas.py --threshold 0.4
```

Figura 51 Código de funcionamiento de Docker

```
import cv2
import jetson.inference
import jetson.utils
import numpy as np
import argparse

parser = argparse.ArgumentParser(description='Pollen y Not Pollen')
parser.add_argument('--threshold', default="0.8")
args = parser.parse_args()
threshold = "--threshold"+str(args.threshold)

net = jetson.inference.detectNet("ssd-mobilenet-v2",["--model=/Deteccion_abejas/ssd-mobilenet.onnx","--
labels=/Deteccion_abejas/labels.txt","--input-blob=input_0","--output-cvg=scores","output-
bbox=boxes",threshold])
# camera = jetson.utils.videoSource("/dev/video0)
camera = cv2.VideoCapture("/Deteccion_abejas/Deteccion.mp4)
camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
camera.set(cv2.CAP_PROP_FRAME_WIDTHT, 480)
#display = jetson.utils.video0output("Display://0")

while True:
    r,frame = camera.read()
    img = jetson.utils.cudaFromNumpy(cv2.cvtColor(frame,cv2.COLOR_BGR2RGBA).astype(np.float))

    detections = net.Detect(img)
    n_obj = len(detections)
    for detect in detections:
        Id = detect.ClassID
        item = net.GetClassDesc(Id)

        if item == "Pollen": color = (0,255,0)
        else: color = (0,0,255)

        cv2.putText(frame,item,(int(detect.Left),int(detect.Top)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
        cv2.rectangle(frame,(int(detect.Left),int(detect.Top)),(int(detect.Right),int(detect.Bottom)),
(255,0,0),1)

    cv2.imshow("window",frame)
    if cv2.waitKey(1) == ord("q"):
        break;

camera.release()
cv2.destroyAllWindows()
```

Figura 52 Código .py de procesamiento de imágenes