Sistema autónomo para clasificación de material reciclable implementado en un sistema embebido usando IA.

Cristian Alejandro Sastre Lozada.

Juan Diego Ramírez Ardila.

Nicodemo Galeano Díaz.

Trabajo de Grado para Optar el Título de Ingeniero Electrónico.

Director

Jaime Guillermo Barrero Pérez.

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2023

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Resumen

Título: Sistema autónomo para clasificación de material reciclable implementado en un sistema embebido usando IA. *

Autores: Juan Diego Ramírez Ardila, Cristian Alejandro Sastre Lozada, Nicodemo Galeano Díaz. **

Palabras Claves: Redes Neuronales Convolucionales, TensorFlow, Keras, Material Reciclable, Clasificación Autónoma, Basura.

Descripción: En este trabajo se presenta un algoritmo de inteligencia artificial, implementado en dos sistemas embebidos buscando desarrollar y comparar dos productos posibles. las tarjetas escogidas fueron: la tarjeta Jetson nano 2GB de NVIDIA y la Raspberry Pi 3 Model B +. Este algoritmo reconoce automáticamente distintos materiales reciclables. Para esto se creo una base de 3500 imágenes de diferentes fuentes, el cual pasó por un proceso de tratamiento y filtrado. Posteriormente se entrenaron tres algoritmos: EfficientNet, MobileNetV2 y MobileNetV2-FPN. El algoritmo escogido fue el basado en la MobileNet-FPN mostrando el mejor desempeño teniendo en cuenta características como la función de perdida y la precisión de detección del modelo. Luego se realizaron pruebas de funcionamiento con materiales reciclables, usando la implementación propuesta en las dos tarjetas. Finalmente se escogió la Jetson nano para la implementación, ya qué mostró mejor relación de costo-beneficio, desempeño y velocidad en la detección.

---

* Trabajo de Grado.

** Facultad de Ingeniería Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: Jaime Guillermo Barrero Peréz

Abstract

Title: Autonomous system for sorting recyclable material implemented in an embedded system using AI. *

Authors: Juan Diego Ramírez Ardila, Cristian Alejandro Sastre Lozada, Nicodemo Galeano Díaz. **

Keywords: Convolutional Neural Networks, TensorFlow, Keras, Recyclable Material, Autonomous Sorting, Garbage.

Description: This paper presents an artificial intelligence algorithm implemented in two embedded systems seeking to develop and compare two possible products. the chosen cards were: the NVIDIA Jetson nano 2GB card and the Raspberry Pi 3 Model B+. This algorithm automatically recognizes different recyclable materials. For this, a database of 3500 images from different sources was created, which went through a process of treatment and filtering. Subsequently, three algorithms were trained: EfficientNet, MobileNetV2 and MobileNetV2-FPN. The chosen algorithm was the one based on MobileNet-FPN that presented the best performance taking into account characteristics such as the loss function and the detection accuracy of the model. Then, performance tests were carried out with recyclable materials, using the implementation proposed in the two cards. Finally, the Jetson nano was chosen for the implementation, since it presented the best cost-benefit ratio, performance and speed of detection.

---

* Degree Work

** Facultad de Ingeniería Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: Jaime Guillermo Barrero Peréz

Introducción

En la actualidad, el reciclaje y clasificación de los residuos juegan un papel importante en la conservación y protección del medio ambiente, por lo tanto, es fundamental la ejecución de proyectos y acciones concretas en pro de éste. Según el documento No. 3874 del Consejo Nacional de Política Económica y Social (CONPES) el país genera más de 12 millones de toneladas de residuos sólidos al año, de las cuales, tan solo el 17 % es reciclado, desaprovechando más del 80 % de los residuos, que se podrían transformar en oportunidad de negocio y descongestión en los rellenos sanitarios. Asimismo, afirma que si Colombia continúa en la misma dinámica de generación de residuos, sin hallar soluciones para mejorar el aprovechamiento de estos, en el año 2030 el país tendrá emergencias sanitarias en la mayoría de las ciudades y una alta generación de emisiones de gases efecto invernadero, lo que afectará la calidad del aire (Departamento Nacional, 2016).

De acuerdo con Daniel Bender, Jefe de equipo de *Deep Learning* en TOMRA Sorting Recycling, en los meses y años venideros, el *Deep Learning*, uno de los componentes más importantes de la IA (inteligencia artificial), tomará fuerza en la industria del reciclaje. “El deep learning es una solución prometedora. Las empresas de reciclaje más vanguardistas que integran la inteligencia artificial en la clasificación de material se encuentran en clara ventaja sobre las que no lo hacen” (Technology, 2020). Por otro lado, Sadako Technologies, especialistas en el desarrollo de tecnología de Inteligencia Artificial y Robótica, creó Max-AI Autonomous QC, un robot capaz de identificar y clasificar objetos muy complejos en una corriente de residuos, la primera unidad

ya está en funcionamiento desde abril de 2017 en una planta de tratamiento de residuos en Sun Valley (California) y su lanzamiento ha tenido un gran impacto en el sector ya que contribuye a una operación de la planta más automatizada, segura y rentable.(Technologies, 2022)

Tras consultar con las empresas encargadas del reciclaje local, se reconoció que la principal problemática en el proceso de reciclaje de los materiales es la mala clasificación de los residuos en la fuente. Es por esto, que se plantea el desarrollo de un algoritmo de reconocimiento autónomo de residuos haciendo uso de redes neuronales covolucionales, que ayude a disminuir la proporción de materiales que están mal clasificados en las entidades comerciales y educativas. Con esto se busca mejorar el proceso de selección y de esta manera incrementar la cantidad de residuos que pueden ser utilizados como materia prima, minimizando con ello el impacto negativo que generan en el ecosistema.

Igualmente, se realizaron investigaciones sobre la tarjeta que presenta las mejores utilidades enfocado a nuestro proyecto, teniendo en cuenta factores como el tamaño, costo y velocidad. Finalmente se decidió usar la tarjeta Jetson Nano 2GB de Nvidia, y la cámara Logitech c270 HD, acoplada al sistema embebido. También se propone un modelo comparativo teniendo la tarjeta Raspberry Pi 3 Model B +. Estos dispositivos fueron adquiridos por los autores del presente trabajo.

Es así como, con el desarrollo de este proyecto, se desea exponer a la comunidad sistemas autónomos que propicien cambios culturales y modernicen el proceso del reciclaje a través del dispositivo presentado, que contribuya en el equilibrio del medio ambiente.

1. Objetivos

1.1. Objetivo General

Reconocer automáticamente materiales reciclables mediante un algoritmo de inteligencia artificial implementado en un sistema embebido.

1.2. Objetivos Específicos

Obtener una base de imágenes, recolectando imágenes de fuentes propias y externas, para posteriormente realizar el tratamiento, filtrado y etiquetado de las imágenes.

Seleccionar y entrenar un algoritmo de inteligencia artificial capaz de reconocer materiales reciclables con valores óptimos de precisión y pérdidas para la aplicación específica.

Implementar el algoritmo de inteligencia artificial en el sistema embebido seleccionado para probar su funcionamiento con datos reales.

2. Marco Conceptual

2.1. Material Reciclable

Los materiales reciclables son aquellos que pueden ser reutilizados de nuevo tras su uso principal, gracias a un tratamiento de reciclaje. Ya sea en su forma elaborada (como el plástico hecho botella) como en su forma más pura (como el anticongelante o el aceite), los materiales reciclables son aquellos de los que aún puede extraerse un valor. Un material reciclable no necesariamente se convierte siempre en el mismo material, listo de nuevo para usar. Algunos se reciclan aprovechando el valor que aún conservan para, por ejemplo, generar energía.

Figura 1.
Compactadora Paperlab

Figura 2.
Materiales compactados

Tras visitar la empresa de reciclaje Paperlab, la cual se encuentra ubicada en la ciudad de Bucaramanga, Colombia. Se evidenció el proceso de aprovechamiento que se lleva a cabo actualmente, el cual mediante una compactadora (Ver Figura 1) se reduce el volumen de los desechos sólidos con el fin de poder manejar más fácilmente estos residuos, como se muestra en la Figura 2.

Según la empresa los materiales más comunes que llegan al centro de aprovechamiento son aluminio, papel, cartón, pasta gruesa, periódico, PET, plega, PVC y vidrio, de los cuales los que se obtiene mayor beneficio económico son los metales como el cobre. En la Tabla 1 se muestran los precios de estos materiales a fecha de enero de 2023.

Empresas locales que funcionan como comercializadoras, compran los residuos aprovechables. Tras realizar la separación y procesamiento, los venden como materia prima a empresas como Bavaria.

Tabla 1
Precios materiales aprovechables (enero/2023)

Material	Precio por kg ($ COP)
Aluminio	4.000
Cartón	400
Papel	1.000
Plátisco	800
Vidrio	100

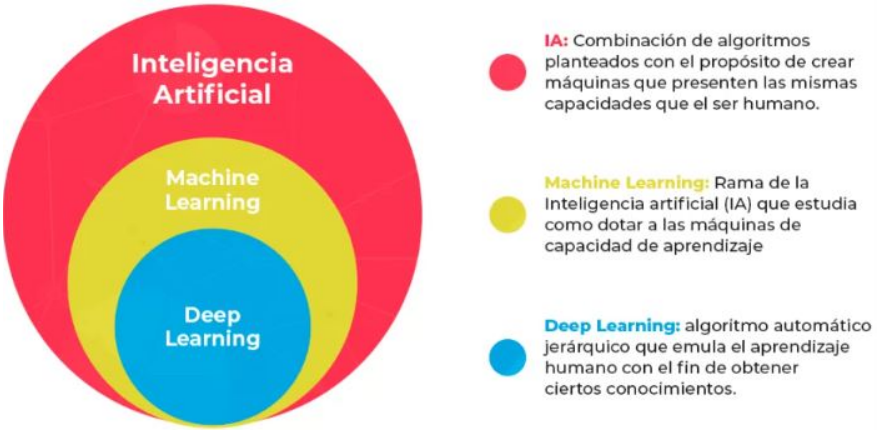
Se determino que para crear el algoritmo es necesario tomar diversas fuentes de imágenes tanto propias como externas, y así emular el trabajo o proceso que llevaría a cabo un operario de una empresa de aprovechamiento de residuos.

2.2. Inteligencia Artificial

La teoría de la inteligencia artificial llamó la atención por primera vez en 1950, cuando un famoso informático británico, Alan Turing, propuso un "juego de imitación" para comprobar si un ordenador podía engañar a los humanos haciéndoles creer que estaban interactuando con otro ser humano (Vatsa, 2021)

Después que se propusiera por primera vez el concepto de inteligencia artificial, se dieron una serie de progresos significativos, tales como la demostración de teoremas en máquina, el programa checker, LISP, etc. Desde finales de la década de 1960 hasta la de 1970, se inició un nuevo auge de la investigación en IA con la aparición de los sistemas expertos, tales como DENDAL, MTCIN, PROSPECTIOR, Hearsayy entre otros, que llevaron la IA a la práctica. En la década de 1980, con el desarrollo de los ordenadores de quinta generación, la IA se desarrolló enormemente

Figura 3.
Clasificación inteligencia artificial



Nota. Imagen tomada de *https://www.masterdatascienceucm.com/que-es-machine-learning/.* Accedido el dia 12 de diciembre del 2022

(Ling-fang, 2010).

El objetivo de esta tecnología es emular una inteligencia similar a la humana, incluyendo actividades como “razonamiento”, juicio, identificación, percepción, pensamiento, diseño y solución de problemas. (Harika *et al.*, 2022)

La IA siempre ha estado en el extremo pionero de la informática. Los lenguajes informáticos avanzados, el reconocimiento facial, los procesadores de texto entre muchos otros deben su existencia a la investigación en inteligencia artificial. Los conocimientos derivados de la investigación en IA marcarán la tendencia en el futuro de la informática, una imagen que puede dar algo de nocion sobre los campos de IA se puede ver en la Figura 3.

2.2.1. Redes Neuronales

Una red neuronal artificial es un sistema de procesamiento de información que tiene ciertas características de funcionamiento en común con las redes neuronales biológicas. Está formada por un gran número de elementos de procesamiento simples denominados neuronas, unidades, células o nodos. cada neurona está conectada a otras neuronas mediante enlaces de comunicación dirigidos, cada uno de los cuales tiene un peso asociado. los pesos representan la información que utiliza la red para resolver un problema.

Las redes neuronales se caracterizan por su patrón de conexiones entre las neuronas (llamado arquitectura), su método para determinar los pesos de las conexiones (llamado algoritmo de entrenamiento o aprendizaje) y su función de activación.

Al principio, todas las ponderaciones son aleatorias y las respuestas que resultan de la red son, posiblemente, disparatadas. La red aprende a través del entrenamiento. Continuamente se presentan a la red ejemplos para los que se conoce el resultado, y las respuestas que proporciona se comparan con los resultados conocidos. La información procedente de esta comparación se pasa hacia atrás a través de la red, cambiando las ponderaciones gradualmente. A medida que progresa el entrenamiento, la red se va haciendo cada vez más precisa en la replicación de resultados conocidos. Una vez entrenada, la red se puede aplicar a casos futuros en los que se desconoce el resultado.

El estudio de las redes neuronales es un campo extremadamente interdisciplinar, tanto en

su desarrollo como en su aplicación. un muestreo de algunas de las áreas en las que se aplican actualmente las redes neuronales sugiere la amplitud de su aplicabilidad (Fausett, 1993).

2.2.1.1. Redes Neuronales Convolucionales. Una red neuronal convolucional (CNN o ConvNet) es una arquitectura de red para *deep learning* que aprende directamente de los datos, sin necesidad de extraer características manualmente.

Estas redes son particularmente útiles para encontrar patrones en imágenes, para reconocer objetos, caras y escenas (T. Guo y Gao, 2017). También resultan eficaces para clasificar datos sin imágenes, tales como datos de audio, series temporales y señales (S. Jadhav y Rege, 2019)

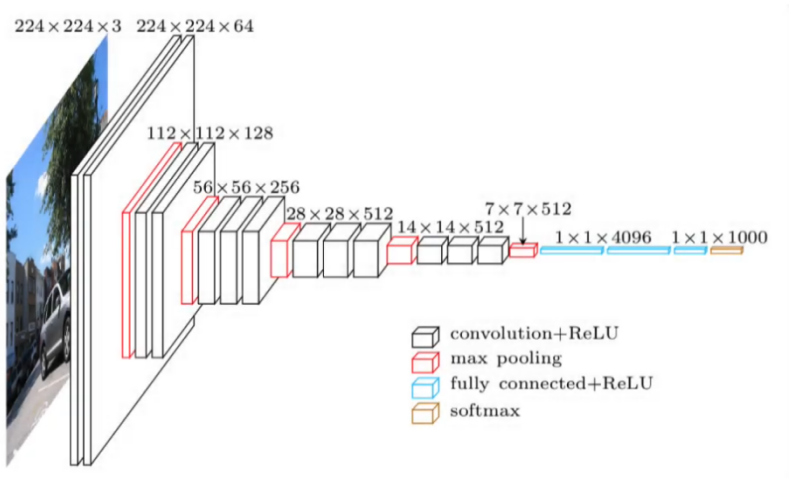
2.2.1.2. Estructura de las Redes Reuronales Convolucionales. En general, la estructura de las redes neuronales convolucionales esta construida por 3 tipos diferentes de grupos de capas (La Figura 4 muestra un ejemplo de red neuronal tipo convolucional), estos grupos son:

1. Capa convolucional: Se encarga de extraer características de las entrada mediante filtros que realizan diferentes operaciones sobre la imagen.

2. Capa de *pooling*: Permiten reducir la cantidad de datos e información, y extraer los datos mas representativos de la imagen de entrada.

3. Capa clasificadora: esta es la capa final, que es una red neuronal convencional totalmente conectada, la cual toma las características que se obtuvieron de la capa convolucional, las representa en un vector de datos y realiza la clasificación final de la imagen.

Figura 4.
Estructura de una red neuronal convolucional



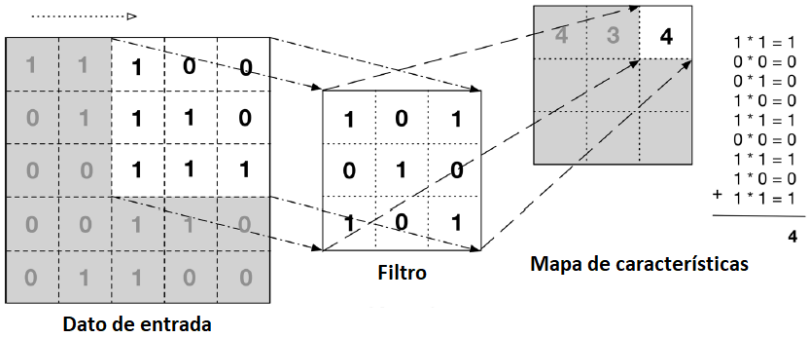
Nota. Imagen tomada de https://www.codificandobits.com/blog/redes-convolucionales-introduccion/. Accedido el día 23 de noviembre del 2022

2.2.1.3. Convolución. Las redes neuronales convolucionales tienen como su operación distintiva la convolución, a diferencia de otras redes neuronales que usan la multiplicación matricial. La convolución se aplica en alguna de sus capas y consiste en tomar grupos de píxeles vecinos de la imagen de entrada e ir operando matemáticamente (producto escalar) contra una pequeña matriz que se llama kernel, el cual puede tener por ejemplo un tamaño de 3×3 (Ver Figura 5) pixeles, luego, con ese tamaño se logra visualizar todas las neuronas de la capa de entrada para que se genere una nueva matriz de salida, que será la nueva capa de neuronas ocultas.

2.2.2. Overfitting

Los modelos de aprendizaje automático tienen como fin entrenar y obtener patrones generales de cierto grupos de datos de entrenamiento y extrapolarlos a nuevos datos de testeo, el

Figura 5.
Proceso de convolución de matrices



Nota. Imagen tomada de https://rpubs.com/dsotilccama/CNN_Markdown/. Accedido el 14 de diciembre de 2022

overfitting o sobre ajuste, en su traducción al español, se produce cuando el modelo "memoriza" los datos de entrenamiento y no "generaliza" los patrones y características de estos, lo que genera bajos niveles de desempeño cuando se ingresan datos nuevos a la red. Dada la variedad de características en la basura, el problema de sobre ajuste resulta ser recurrente en la tarea de clasificación (Kulkarni *et al.*, 2019). Para lograr minimizar las consecuencias del sobre ajuste en el desempeño del modelo, se presenta una solución mediante el data augmentation. Para esto, se puede aplicar diferentes técnicas, tales como giro y rotación de las imágenes con recorte aleatorio y ajuste de brillo (Meng y Chu, 2020; Cao y Xiang, 2020).

2.2.3. Aprendizaje por Transferencia

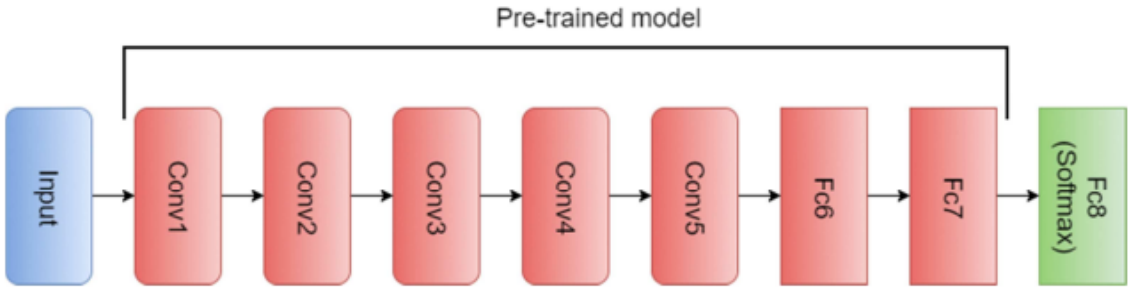
Consiste en usar una red que ya ha sido entrenada, el objetivo principal del aprendizaje por transferencia es aprovechar las características que ha aprendido el modelo de la red neuronal convolucional en un entrenamiento previo y haciendo un re-entrenamiento de las ultimas capas del

modelo este se ajuste para reconocer un nuevo set de imágenes y ejecute tareas similares. Esto se hace con la finalidad de reducir tiempos de entrenamiento, costo computacional y la obtención mas rápida y eficiente de los resultados(Li *et al.*, 2021).

El aprendizaje por transferencia puede ser usado en tres formas diferentes, utilizar el modelo pre-entrenado, usar solo la arquitectura o usar solo algunas capas de la arquitectura.

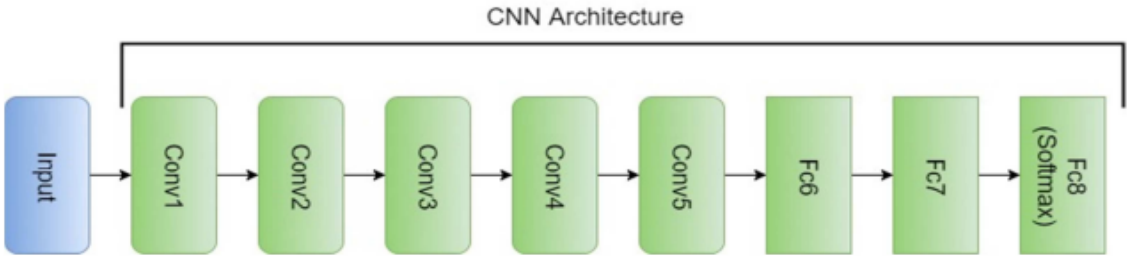
2.2.3.1. Utilizar el Modelo Pre-entrenado para Extracción de Características. En esta técnica se remueve la capa de salida de la red neuronal convolucional que ha sido entrenada previamente, esta es la capa que genera las probabilidades de cada clase, para luego ser reemplazada por una capa clasificadora con las clases de la nueva tarea a ejecutar. Se utiliza frecuentemente cuando los datos de la tarea son similares a los datos de entrenamiento del modelo base, por ejemplo como se muestra en la Figura 6 (N. Diaz, 2018).

Figura 6.
Extracción de características del modelo pre-entrenado



Nota. Imagen tomada de https://red.uao.edu.co/bitstream/10614/10153/6/T07815.pdf. Accedido el 02 de enero de 2023

Figura 7.
Modelo usando la arquitectura del modelo preentrenado



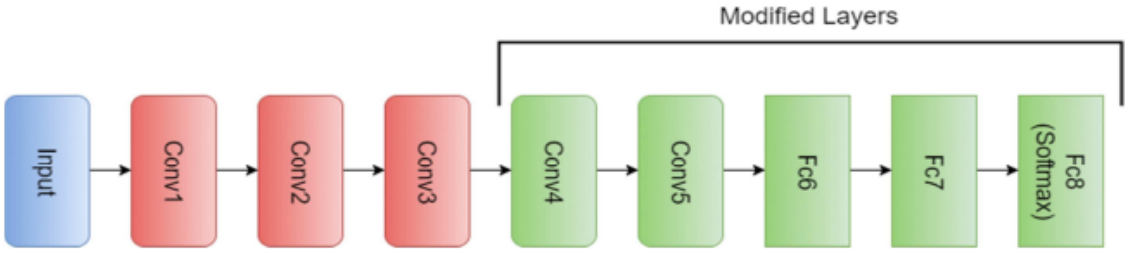
Nota. Imagen tomada de https://red.uao.edu.co/bitstream/10614/10153/6/T07815.pdf. Accedido el 02 de enero de 2023

2.2.3.2. Utilizar la Arquitectura del Modelo pre-entrenado. En esta técnica se inicializan los pesos del modelo pre-entrenado para posteriormente ser entrenados con los datos de la tarea requerida. Este método se usa cuando el tamaño de los datos es grande y hay baja similitud entre los datos del modelo base y los datos de la nueva tarea, como se muestra en la Figura 7 (N. Diaz, 2018).

2.2.3.3. Utilizar Algunas Capas del Modelo Pre-entrenado para Posteriormente Entrenarlas. En esta técnica se entrena parcialmente la red, usualmente se conservan los pesos de las capas inferiores ya que estas capas contienen características generales y se realiza el entrenamiento de las capas superiores ya que estas contienen características especificas y de gran importancia para la eficiencia del modelo, esto se evidencia en la Figura8 (N. Diaz, 2018).

Para el entrenamiento del algoritmo presentado, se usó este modelo, teniendo en cuenta que la base de imágenes es más pequeña y con una distribución diferente, comparada con la del

Figura 8.
Modelo usando algunas capas del modelo preentrenado



Nota. Imagen tomada de https://red.uao.edu.co/bitstream/10614/10153/6/T07815.pdf. Accedido el 02 de enero de 2023

modelo entrenado originalmente.

2.2.4. Modelos Pre-entrenados

Actualmente hay a disposición gran variedad de redes neuronales convolucionales pre-entrenadas que optimizan la clasificación de objetos con tan solo re-entrenar la red con los parámetros que mejor se adapten a la solución mas eficiente del problema. Algunos de estos modelos son: **VGG16, ResNet, GoogleNet, AlexNet, EfficientNet, MobileNetV2, MobileNetV2-FPN** (Yugsi, 2020).

Las últimas tres mencionadas fueron las selecionadas para realizar el entrenamiento y modelado propuesto.

2.2.4.1. EfficientNet B0. Este modelo es de ultima generación y este usa una función de activación llamada Swish en lugar de la función de activación ReLU(Unidad Lineal Rectificadora). El éxito de EfficientNet se da por el escalamiento uniforme de la profundidad, el ancho y la reso-

lución mientras se reduce la escala del modelo. El componente principal del modelo es un bloque MBConv Cuello de botella Invertido y dentro de este bloque encontramos una capa que inicialmente expande y luego comprime los canales, esto produce una conexión directa entre cuellos de botella y esto a su vez conecta menor número de canales que las capas de expansión (cumit Atila *et al.*, 2021).

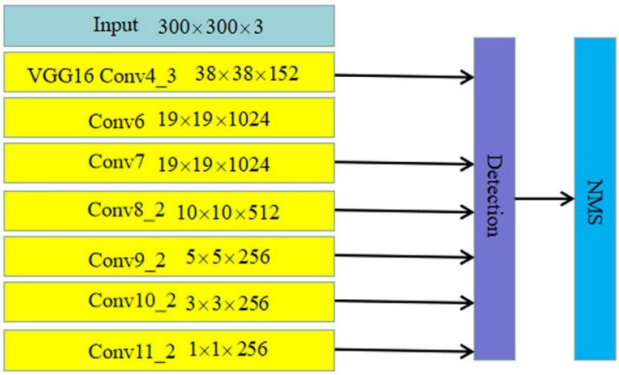
2.2.4.2. MobileNetV2. Este Modelo está conformado por una serie de bloques residuales invertidos, agrupados entre dos convoluciones, la función principal de estos es transformar la entrada en una representación intermedia y la representación intermedia en una salida. El resultado de la convolución pasa a través de una capa de agrupación de promedio global y luego atraviesa una capa de inferencia (Buiu *et al.*, 2020).

2.2.4.3. MobileNetV2FPN. En el MobileNetV2 SSD FPN-Lite, tenemos una red base (MobileNetV2), una red de detección (Single Shot Detector o SSD) y un extractor de características (FPN-Lite).

La estructura de la red SSD se muestra en la Figura 9. Los primeros cinco grupos de capas convolucionales de VGG-16 se utilizan para la extracción de características, y varias capas convolucionales adicionales para extraer las características más profundas de la imagen. La capa de predicción predice la categoría y la ubicación de la información de las características extraídas de las distintas capas, y el resultado final de la detección se obtiene después de un cribado mediante la supresión no máxima (non-maximum suppression) mediante supresión no máxima (NMS) (Meng, 2021).

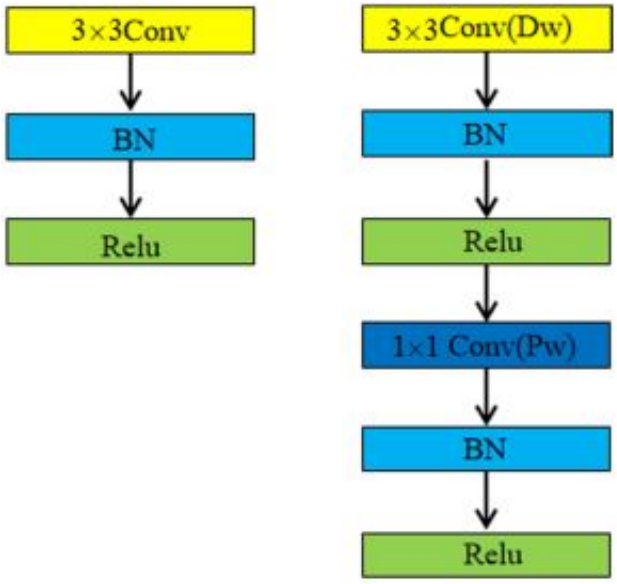
Una convolución estándar del modelo MobileNet (La arquitectura se muestra en la Figura 10) se puede dividir en una convolución 3×3 en profundidad y una convolución 1×1 denominada

Figura 9.
Topologia Red SSD



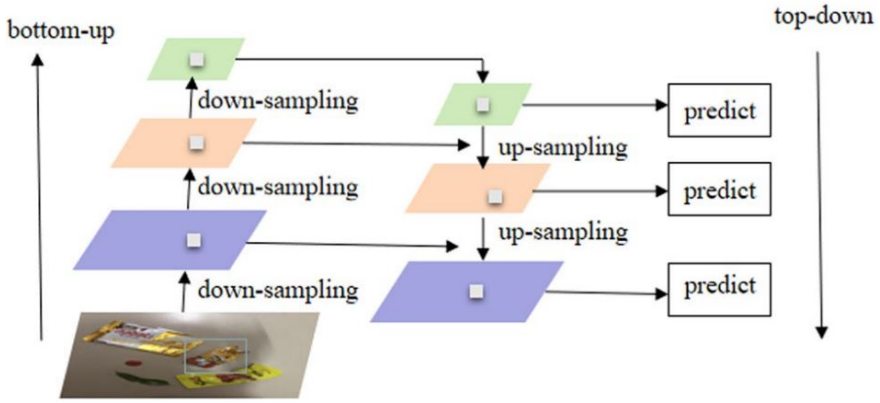
Nota. Imagen tomada de https://doi.org/10.1007/s42835-021-00960-w. Accedido el 21 de diciembre de 2022.

Figura 10.
Topologia MobilNet



Nota. Imagen tomada de https://doi.org/10.1007/s42835-021-00960-w. Accedido el 21 de diciembre de 2022.

Figura 11.
Configuración FPN



Nota. Imagen tomada de https://doi.org/10.1007/s42835-021-00960-w. Accedido el 21 de diciembre de 2022.

convolución puntual. En la convolución en profundidad se introduce un único filtro en cada canal de entrada. A continuación, se aplica una convolución 1×1 para integrar las salidas de la convolución en profundidad.

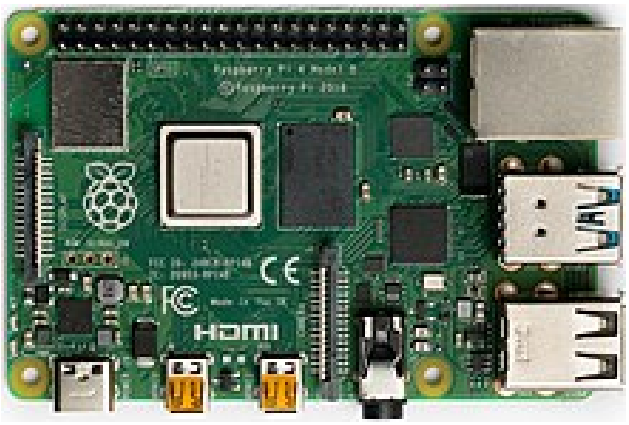
El método convencional aprovecha las *image pyramids* para construir *feature pyramids* y calcula las características en sus respectivas escalas de imagen. FPN es capaz de combinar características debiles de bajo nivel con características robustas de alto nivel para mejorar la precisión de la detección de objetos pequeños. La Figura 11 muestra la construcción de esta configuración.

Se propone un método de detección de residuos basado en MobileNet-SSD, que puede lograr un efecto preciso y eficaz. Además, se aplica FPN para mejorar mejorar el rendimiento, ya que SSD es ineficaz para detectar objetos pequeños.

2.3. Sistemas Embebidos

Un sistema embebido o sistema empotrado se define como un conjunto de componentes electrónicos agrupados entre si y combinados con software de computadora para realizar funciones especificas, usualmente en tiempo real. Está conformado por uno o varios procesadores, posee un procesador central que se encarga de recibir, analizar y procesar los datos para posteriormente enviar una señal de estimulo a los actuadores del sistema y se realicen determinadas tareas. Los sistemas embebidos son diseñados principalmente para aplicaciones donde se requiere una computación en tiempo real (Alba y Alcorta, 2020).

Figura 12.
Placa de desarrollo Raspberry PI 3 B+



Nota. Imagen tomada de https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/. Accedido el 6 de enero de 2023.

2.3.1. Raspberry PI

La Raspberry PI es una placa SBC (Single Board Computer, pc de placa única), de bajo costo, pero de alto rendimiento capaz de realizar proyectos de manera más dinámica. Por sus principales características permite la ejecución de miles de aplicaciones con grandes beneficios y bajo costo. Este dispositivo portátil fue desarrollado en Gran Bretaña en el año 2006, la Figura 12 muestra la placa (Raspberry, 2023).

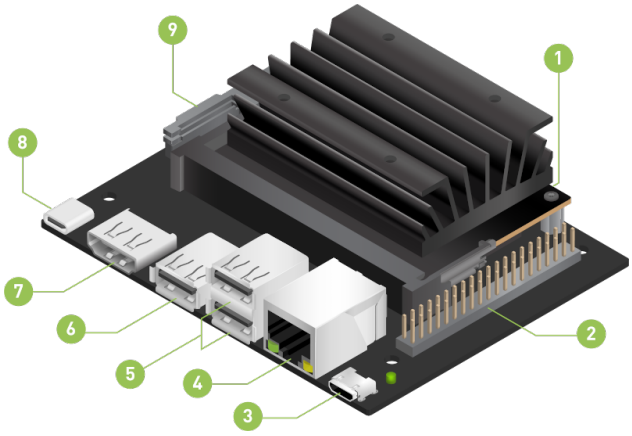
Esta placa posee un procesador Broadcom BCM2837B0, Cortex-A53 (ARMv8) de 64-bit, con cuatro núcleos que trabaja a 1.4GHz. Posee un memoria Ram de 1GB LPDDR2, conexión de 2.4 y 5GHz wireless LAN, Bluetooth de 4.2/BLE, Gigabit Ethernet y 4 puertos USB. Además la placa cuenta con conexión HDMI y puertos MIPI para cámaras y displays. Su tamaño es de 85 x 56 mm y el consumo promedio es de 5W.

2.3.2. Jetson Nano

La Jetson Nano Es una computadora de tamaño reducido, pero muy poderosa que permite ejecutar redes neuronales en aplicaciones como clasificación de imágenes, detección de objetos, segmentación y procesamiento de voz. Su tamaño es de 70 x 45 mm.

En su interior se encuentra una potente CPU ARM Cortex-A57 MP Core de 4 núcleos y genera 472 Gigaflots de potencia, también encontramos una GPU Nvidia Maxwell con 128 núcleos CUDA la cual es compatible para ejecutar la librería CUDA. El consumo de potencia oscila entre 5 y 10W. En el mercado actual se encuentra un kit en el que se adapta el modulo Jetson Nano, esta

Figura 13.
Kit de desarrollo Jetson Nano



Nota. Imagen tomada de https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano/product-development/. Accedido el 6 de Enero de 2023.

base portadora cuenta con 3 puertos USB 2.0 un puerto USB 3.0, un puerto HDMI, un DisplayPort y un puerto Ethernet, puertos necesarios y suficientes para iniciar en el mundo de la inteligencia artificial, la Figura 13 muestra la placa (Nvidia, 2023).

2.4. Adecuación del Entorno

Para la ejecución del entrenamiento de los modelos se usaron librerías para redes neuronales tales como TensorFlow versión 2.9 que incluye la librería de Keras (Keras, 2022). La carpeta con las imágenes utilizadas se puede consultar en (Sastre *et al.*, 2022c) y los scripts de entrenamiento en el repositorio de GitHub siguiendo la ruta / Sistema-autonomo-de-clasificacion-de-material-reciclable /Proyect_TFL_train.ipynb (Sastre *et al.*, 2022d).

En cuanto a la escritura y ejecución del código, se ejecutó en la herramienta Google Colab, la cual brinda la posibilidad del trabajo colaborativo y también la posibilidad de usar GPUs (pagos

y gratuitos), que es una unidad de procesamiento gráfico que se ocupa de las tareas gráficas del sistema, lo que implica obtener un entrenamiento mucho mas rápido. Para el desarrollo del modelo se usó python en la versión 3.8 y se realizó la compra de 200 GPU usando los paquetes ofrecidos por Google Colab.

De igual manera, se recurrió a la API de TensorFlow. Una API consiste en un conjunto de protocolos que se utilizan para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software, esta herramienta fue importante para el desarrollo del modelo pre-entrenado escogido. Las API principales de TensorFlow proporcionan un conjunto de API de bajo nivel completas, componibles y extensibles para computación de alto rendimiento (distribuida y acelerada), destinadas principalmente a la creación de modelos de aprendizaje automático (ML), así como a la creación de marcos y herramientas de flujo de trabajo de ML dentro de la Plataforma TensorFlow. Estas API brindan una base para crear modelos altamente configurables con control detallado y nuevos marcos desde cero (Tensorflow, 2022a).

Para la implementación se usó TensorFlow lite, el cual es un conjunto de herramientas que ayudan a ejecutar modelos de TensorFlow en dispositivos móviles, integrados y de IoT (Tensorflow, 2022b), para este caso se usa solo el interprete de TensorFlow lite, ya que es la opción que brinda el mayor ahorro de recursos en las tarjetas, de igual forma, se utilizó la librería OpenCV, el cual brinda un marco de trabajo de alto nivel para el desarrollo de aplicaciones de visión por computador en tiempo real como estructuras de datos, procesamiento y análisis de imágenes, análisis estructural, etc (OpenCV, 2023).

Asimismo, fue importante el uso de los entornos virtuales, en el cual el intérprete Python, las bibliotecas y los scripts instalados, están dentro de un “docker”, que permite aislarlo del sistema operativo global y de otros entornos existentes. Esto con el fin de realizar instalaciones de paquetes sin poner en conflicto versionamiento de las librerías actuales y las globales del sistema.

3. Definición del Sistema

3.1. Generación de la Base de Imágenes

Los dos factores mas importantes en la clasificación de objetos usando aprendizaje por transferencia es la adecuada selección del modelo que mejor se adapte a la tarea que se desea realizar y la construcción de una base de datos que contribuya al buen desarrollo de la tarea a desarrollar.

Para lograr los mejores resultados en el entrenamiento de nuestro modelo, es necesario construir una base de datos amplia, la cual le permita a la red generalizar perfectamente las características del objeto analizado y de esta manera la precisión en la clasificación sea la mas alta posible.

Se construyó una base de datos de 3500 imágenes de elementos reciclables los cuales fueron distribuidas en 5 clases: cartón, metales, papel, plásticos y vidrios, se tomaron 3150 imágenes de fuentes externas (Santoro, 2019), el 90% del total, y 350 de fuentes propias, el 10% del total, asimismo, se buscó una distribución homogénea de los datos, se cuenta con 700 imágenes aproximadamente por cada clase(Sastre *et al.*, 2022c).

El tamaño de las imágenes tomadas de bases de datos existentes con las que se trabajó es de 512*384 pixeles al igual que el tamaño final de las imágenes propias. Para llegar al tamaño propuesto con las imágenes propias, se realizó una escala a las imágenes tomadas (4000x3000 pixeles), manteniendo la relación de cuatro a tres y quedando con un tamaño de 512*384 pixeles. Esto se hizo con el fin de tener una base de datos homogénea en tamaño de las imágenes.

A continuación, se presenta la configuración de red de cada uno de los tres modelos, primero, se muestra la Tabla 2, donde se mencionan los parámetros que se usaron en la etapa de extracción de características, la cual, toma las imágenes de entrada y extrae las características mas relevantes para después procesarlas, posteriormente, en la Tabla 3 se mencionan los parámetros usados en la etapa de predicción de clasificación, a la cual ingresa un mapeo de las características de cada imagen y con esta información predice la clase a la que pertenece, así como la ubicación de los recuadros en donde se detectan los objetos, es decir, las coordenadas de cada recuadro en el que se detecta un objeto dentro de la imagen.

La carpeta donde se encuentra la configuración de red de cada modelo, se muestra en el apartado *pipeline.config. https://drive.google.com/drive/folders/1U53f8S3HOgz-o02tq2IfBL4TX-ShH4lM*

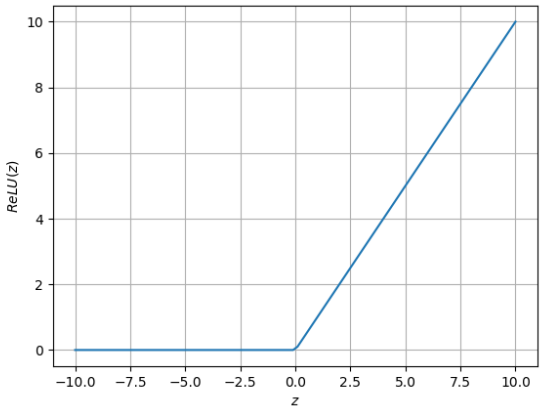
Tabla 2
Parámetros de modelos para extracción de características

Modelo	Regularizador	Inicializador	Función de activación
EfficientNet	L2_Regularizer	Truncated_Normal_Initializer	Swish
MobileNetV2	L2_Regularizer	Truncated_Normal_Initializer	Relu_6
MobileNetV2FPN	L2_Regularizer	Random_Normal_Initializer	Relu_6

Tabla 3
Parámetros de modelos para predicción de clasificación

Modelo	Regularizador	Inicializador	Funcion de activación
EfficientNet	L2_Regularizer	Random_Normal_Initializer	Swish
MobileNetV2	L2_Regularizer	Random_Normal_Initializer	Relu_6
MobileNetV2FPN	L2_Regularizer	Random_Normal_Initializer	Relu_6

Figura 14.
Función de activación Relu



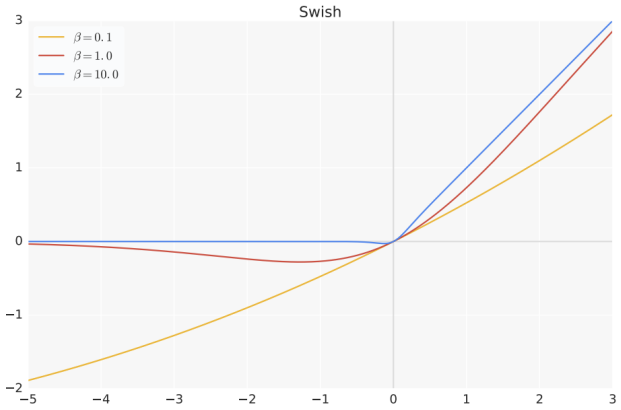
Nota. Imagen tomada de https://www.codificandobits.com/blog/funcion-de-activacion/. Accedido el 30 de Enero de 2023.

Función de activación Relu. Su nombre viene de las siglas en Inglés de Rectified Linear Unit (o unidad lineal rectificada). El comportamiento de esta función se muestra en la Figura 14.

Esta función generará una salida igual a cero cuando la entrada (z) sea negativa, y una salida igual a la entrada cuando dicha esta última sea positiva (Sotaquirá, 2018).

Función de activación Swish. La función swish se define como x · $\sigma(\beta$ x), donde σ(z) = (1 + exp(z))1 es la función sigmoidea y β es una constante o un parámetro entrenable. La Figura 15

Figura 15.
Función de activación Swish



Nota. Imagen tomada de https://arxiv.org/pdf/1710.05941v2.pdf. Accedido el 30 de Enero de 2023.

traza el gráfico de la función swish para diferentes valores de β. Si β = 1, es equivalente a la unidad lineal ponderada sigmoidea (SiL) de (Elfwing *et al.*, 2017) que se propuso para el aprendizaje por refuerzo. Si β = 0, swish se convierte en la función lineal escalada f (x) = x/2. Como $\beta \to \infty$, el componente sigmoide se aproxima a una función 0 1, por lo que swish se vuelve como la función ReLU. Esto sugiere que swish puede verse vagamente como una función suave que se interpola de forma no lineal entre la función lineal y la función ReLU. El grado de interpolación puede ser controlado por el modelo si β se establece como un parámetro entrenable (Ramachandran *et al.*, 2017).

3.1.1. Preprocesamiento de los Datos

Se realizó un proceso de etiquetado a las imágenes. Este etiquetado se desarrolló con la finalidad de aplicar entrenamiento supervisado a la red neuronal.

3.1.1.1. Etiquetado de imágenes. Se etiquetaron las imágenes usando el software LabelImg. LabelImg es una herramienta de etiquetado de datos e información, es de código abierto, muy flexible para imágenes, texto, audio, vídeo. Está desarrollado en Python y su interfaz gráfica usa QT (Tkachenko *et al.*, 2023).

El proceso que se realizó a la base de datos se centró principalmente en hacer un etiquetado, marco o caja del elemento principal en la imagen y dando el nombre a la clase a la que pertenece, con esta técnica se reduce el ruido, ya que no se tiene en cuenta la información que no hace parte del elemento y contribuye a mejorar el rendimiento del modelo.Este proceso se puede observar en la Figura 16.

Figura 16.
Etiquetado de las imágenes (Plásticos, Metales, Vidrios, Papeles y Cartón)

3.2. Diseño de Experimentos

Se eligieron 3 modelos basados en la investigación realizada. Los criterios de selección fueron:

1. Modelos usados principalmente en aplicaciones para dispositivos móviles y sistemas embebidos.

2. Disponibilidad en la API de Tensorflow y su configuración de parámetros en tamaño de las imágenes es acorde con el tamaño de las imágenes de la base de datos construida.

3. Uso frecuente en modelos de detección, por lo que presentan altos rendimientos en estas aplicaciones. (Pascasio *et al.*, 2023; Meng *et al.*, 2022)

3.2.1. Entramiento Base

El entrenamiento base se realizó con los tres modelos seleccionados, para compararlos fue necesario realizar algunas modificaciones a estos modelos. Los modelos usados están preentrenados con unos parámetros que se deben ajustar a cada aplicación. En este caso se modificó el número de clases, el paso (step), el lote (batch) y cambio de tamaño (resize).

3.2.1.1. El Paso (Step). A este parámetro se le dio un valor de 50.000. Con este valor se busca el equilibrio del modelo y minimizar las perdidas del mismo.

3.2.1.2. El Lote (Batch). El valor asignado a este parámetro es de 16. Este valor se asignó por dos factores importantes:

1. Recursos computacionales: Entre mas alto sea el lote, mayor es la cantidad de recursos computacionales que se requiere para el procesamiento del modelo.

2. Prueba y error: Teniendo en cuenta la limitación de recursos computacionales, a prueba y error se empezó a incrementar este parámetro hasta llegar a 16, ya que en este punto el procesamiento del modelo tardó 120 horas aproximadamente y se obtuvo resultados favorables.

3.2.1.3. Redimensionamiento. En la entrada de las redes se encuentra una capa de redimensionamiento que se encarga de establecer todas las imágenes con el mismo tamaño. En este caso se disminuye el trabajo de esta capa ya que todas las imágenes de la base de imágenes tienen el mismo tamaño.

3.2.1.4. Número de Clases. Los modelos usados tienen por defecto un entrenamiento con 90 clases. Este parámetro se ajustó a 5 clases que son las clases con las que se trabajó.

3.2.2. Comparativa de Desempeño

Para poder elegir el mejor modelo de los tres elegidos, calcularon dos métricas: la función de perdida total y la precisión de detección.

3.2.2.1. Función de Perdida Total. Función de Perdida: Es un método para evaluar el algoritmo teniendo en cuenta el modelado de los datos.Esta función está directamente relacionada con las predicciones del modelo desarrollado. Si este valor es bajo, el modelo proporcionará resultados positivos.(Gupta, 2022)

La comparación empezó con la función de perdida total y en esta prueba el modelo que obtuvo el peor desempeño fue el modelo MobilenetV2, quedando descartado para la siguiente prueba, los otros dos modelos obtuvieron resultados similares en esta métrica.

3.2.2.2. Precisión de Detección. Es importante mencionar qué es el Algoritmo mAP, Este algoritmo fue desarrollado con el objetivo de evaluar modelos de detección de objetos. Esta po-

derosa herramienta realiza una comparación del cuadro delimitador de verdad de campo con el recuadro detectado y devuelve una puntuación. La precisión aumenta a medida que la puntuación obtenida por el mAP aumenta. Son 4 submétricas que definen el algoritmo mAP:

1. Matriz de confusión: Esta submétrica está conformada por 4 atributos:

   a) Verdaderos positivos ó *True positive* (TP)

   b) Negativos verdaderos ó *True negative* (TN)

   c) Falsos positivos ó *False positive* (FP)

   d) Falsos negativos ó *False Negative* (FN)

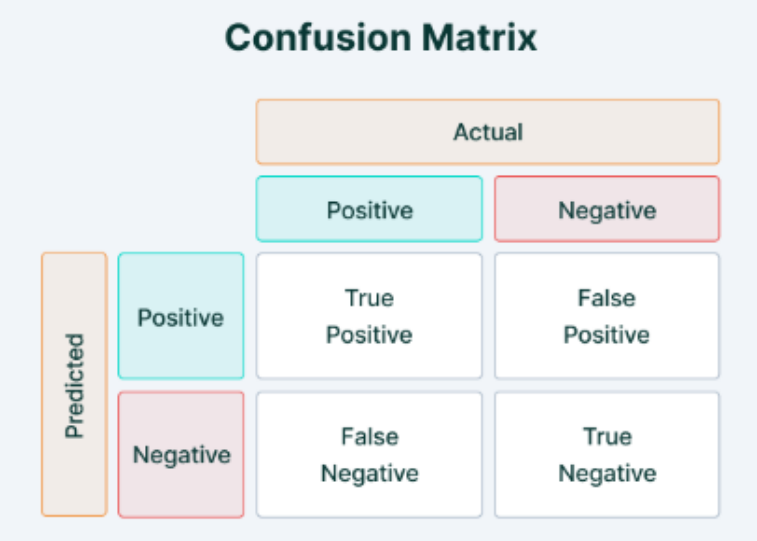
   Este proceso se ilustra en la Figura 17

2. Intersección sobre la unión: Indica la superposición del cuadro delimitador previsto con el cuadro de verdad. Cuando esta submétrica toma un valor alto indica que las coordenadas del recuadro delimitador es similar a las coordenadas del cuadro de verdad.Esta idea se ilustra en la Figura 18

3. Recall: Indica que tan bueno es el modelo para encontrar verdaderos positivos(TP) de todas las predicciones.

4. Precisión: Indica que tan bueno es el modelo para encontrar verdaderos positivos(TP) de las posibles predicciones positivas.

Figura 17.
Matriz de Confusión

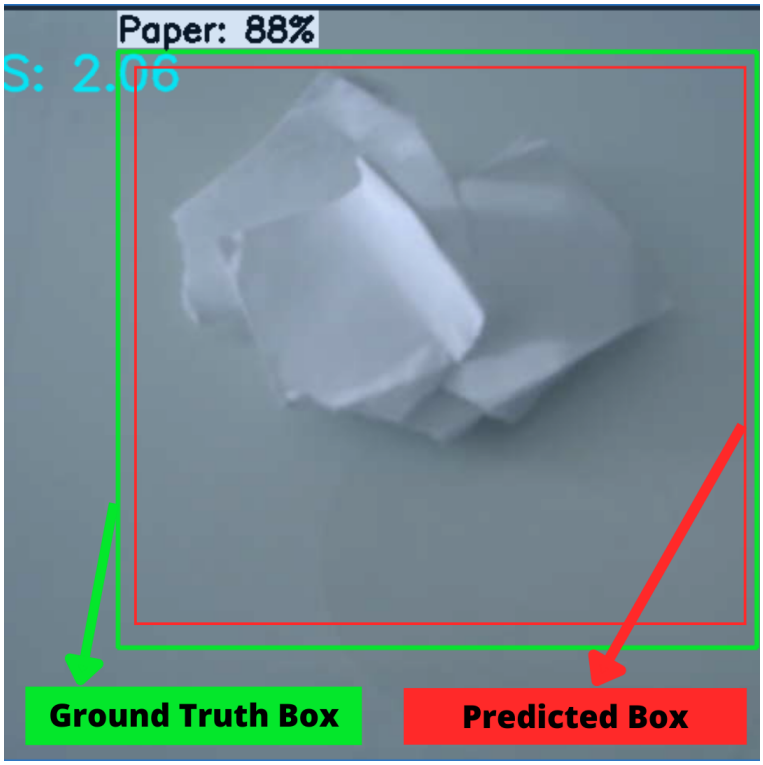


Nota. Imagen tomada de https://www.v7labs.com/blog/mean-average-precision: :text=Mean%20Average%20Precision(mAP)%20is%20a%20metric%20used%20to%20evaluate,valu
Accedido el 6 de enero de 2023.

Con las submétricas mencionadas anteriormente se obtiene la precisión promedio AP y con esta métrica obtenemos el mAP, siendo el mAP el promedio de la precisión promedio AP en cada clase. La precisión promedio AP la calculamos de la siguiente manera:

1. Generando las puntuaciones de predicción usando el modelo.
2. Convertir las puntuaciones de predicción en etiquetas de clase.
3. Calcular la matriz de confusión: TP,FP,TN,FN.
4. Calcular métricas de Precisión y Recall
5. Calcular el área bajo la curva de precisión vs Recall

Figura 18.
Intersección sobre la Unión



6. Medir la precisión promedio

Para obtener la precisión de detección se usó el algoritmo mAP Results, este algoritmo toma varios umbrales de referencia para luego hacer un barrido y promedio de los resultados obtenidos en cada umbral. En esta prueba el modelo con el mejor desempeño fue el modelo MobileNetV2 FPN.(GAD, 2020)

3.3. Presentación del Código

El código necesario para la realización del proyecto, se presenta en dos secciones. En la primera, se muestra el código utilizado para el entrenamiento del modelo, este código, se desarrolló

en la herramienta Google Colab usando Python 3.8 y apalancando el procesamiento con el uso de varias GPU. En la segunda sección, se presenta el código utilizado para la implementación, código acoplado del trabajo realizado por EdgeElectronics (Electronics., 2019).

3.3.1. Código de Entrenamiento

Como se mencionó anteriormente, en esta sección se desarrolló el código utilizado para el entrenamiento de los modelos, el cual se realizó en un notebook de Google Colab, que se encuentra en el repositorio de GitHub siguiendo la ruta / Sistema-autonomo-de-clasificacion-de-material-reciclable /Proyect_TFL_train.ipynb (Sastre *et al.*, 2022d). Este código, presenta una metodología para el re-entrenamiento de modelos de detección de objetos por medio de la API de TensorFlow.

Primeramente, se realiza la carga de las librerías y las imágenes necesarias para el desarrollo, las cuales se encuentran en la siguiente carpeta *https://drive.google.com/drive/u/1/folders/1v2LuXq5zf769nwL*
se buscan las versiones compatibles con los scripts a usar de la API para el entrenamiento y carga de los modelos. Seguidamente, se clonan los archivos de la API, al igual que los modelos a entrenar con el fin de tener un entorno de desarrollo con todas las herramientas necesarias para realizar el entrenamiento. A continuación, se ejecutan los scripts necesarios para procesar el formato de las etiquetas, con el fin de tenerlas en el formato necesario para el proceso, seguido de la configuración de los modelos. En este punto, se ejecuta el script principal, que realiza el entrenamiento de la red seleccionada, para posteriormente visualizar los resultados de entrenamiento de la misma. Finalmente, se realiza el cálculo de la precisión del modelo, usando el algoritmo mAP y se genera la imagen del modelo para convertirla al formato de Tensorflow Lite.

3.3.2. Código de Implementación

La implementación se realizó acoplando el código de detección de objetos mediante una cámara desarrollado por (Electronics., 2019). En este código se usa el intérprete de Tensorflow Lite, con el fin de simplificar y optimizar las librerías necesarias para el funcionamiento. El código presenta una implementación sencilla, en la que se leen los datos en tiempo real de la cámara conectada al sistema embebido por medio de la librería OpenCV. Seguidamente se llama el intérprete para realizar las predicciones sobre las imágenes de entrada, para finalmente retornar las coordenadas de las cajas, las cuales se dibujan en la interfaz principal con sus respectivos valores de precisión de clasificación, y la clase que el algoritmo predice para ese objeto específico.

El algoritmo se implementó en 2 sistemas embebidos, como ejemplo, en la Figura 19 se muestra el resultado obtenido al implementar el algoritmo en la tarjeta Jetson Nano.

Figura 19.
Ejemplo de la implementación en la tarjeta Jetson NANO

3.4. Implementación

Para obtener los parámetros de rendimiento en la implementación, se comenzó con la organización de las imágenes, en la cual se clasificaron en total 3.500 imágenes, divididas en 5 clases diferentes Metales, Cartón, papel, plásticos y vidrios. Para lograr un correcto proceso de validación, se dividió la base de imágenes en una proporción de 80 % para el entrenamiento y un 20 % para el test, esto con el fin de evitar el *overfitting* y obtener un modelo capaz de generalizar adecuadamente. Para el entrenamiento se obtuvieron 560 imágenes de cada clase y para el test 140 (Sastre *et al.*, 2022e).

Posteriormente, el modelo se implementó en los dos sistemas embebidos determinados, ambos con la misma cámara, la c270 hd de Logitech, esto para tener las condiciones más equitativas posibles para la implementación en vivo. Los resultados se enumeran a continuación en las secciones 3.4.1 y 3.4.2.

3.4.1. Implementación del Modelo en Tarjeta Jetson

Uno de los dos sistemas embebidos seleccionados para realizar la implementación del algoritmo es la tarjeta Jetson nano 2GB de NVIDIA,

Para poder realizar la implementación del modelo en el sistema embebido, la plataforma nos facilita una guía detallada para poner en marcha el sistema operativo de la misma. Es un sistema operativo basado en Linux que viene con una versión de Ubuntu 18.04.6 LTS con un procesador ARM de 64 bits(estos procesadores ARM son procesadores de bajo consumo de energía, pero

potentes para realizar tareas de IA).

Esta placa adicionalmente cuenta con CUDA(Compute Unified Device Architecture) que traduce Arquitectura unificada de dispositivos de computo y hace referencia a una plataforma de computación en paralelo incluyendo un compilador que permite usar una variación del lenguaje de programación CUDA para codificar algoritmos en GPU de NVIDIA.

Para iniciar la implementación se necesitan los siguientes elementos, Memoria micro SD, mouse, teclado, monitor, Adaptador de tensión a 5V y Conexión a internet.

El proceso inicia con el formateo de la memoria micro SD para luego insertar la imagen del sistema operativo de la placa que podemos descargar directamente desde la pagina de NVIDIA. Después de tener la memoria flasheada con la imagen del sistema operativo, procedemos a iniciar la placa y configurar el inicio del sistema operativo (SD, 2021).

Seguidamente se instala el entorno virtual de Python en el cual se guardaran todas las librerias necesarias, posteriormente se procede con la instalación de tensorflow y keras que son librerías de Python que facilitan el proceso de entrenamiento e implementación del modelo. Asimismo, se procede con la instalación de la librería OpenCV, la que se encarga del desarrollo de las aplicaciones de visión por computador en tiempo real.

Cuando está instalado todo el software antes mencionado se hace la verificación de dependencias para evitar que haya dependencias incumplidas (NVIDIA, 2021). Cuando se asegura que el software está en perfectas condiciones se carga el modelo que se ha desarrollado anteriormente

de forma online en Colab.

En las Figuras 20, 21, 22, 23 y 24, se presentan las fotos con los artículos que tuvieron los mejores parámetros de desempeño para cada clase:

Figura 20.
Mejor desempeño, clase metal

Figura 21.
Mejor desempeño, clase cartón

Todas las imágenes se pueden encontrar en el siguiente link: *https://drive.google.com/drive/u*

Figura 22.
Mejor desempeño, clase papel

Figura 23.
Mejor desempeño, clase plástico

/0/folders/17dcOKBDVa2CDxs40nPCLFI7lnb-aum_f

Finalmente, se toma el promedio obtenido de 5 artículos diferentes por cada clase, se registran los datos de precisión, Fps y por ultimo si acertó ó no en la clasificación del material reciclable (Ver Tabla 4).

Figura 24.
Mejor desempeño, clase vidrio

Tabla 4
Resultados placa Jetson

Placa	Precision (%)	Fps (Fps)	Número aciertos
Metal	85.75	2.30	4/5
Cartón	89.75	2.32	4/5
Papel	93	2.07	3/5
Plástico	94.4	2.43	5/5
Vidrio	89	2.36	5/5

En el caso de por ejemplo la clase papel, los datos de la precisión en la clasificación solo se tomaron de los artículos correctamente clasificados, tiene una alta precisión respecto a su nivel de asertividad de clasificación debido a que generaliza ciertos artículos mejor que otros, como por ejemplo papeles arrugados o lizos con colores, el modelo termina asociando algunos artículos de papel como plástico.

Es importante destacar el excelente comportamiento del modelo al realizar el encuadre de

las imágenes, lo que se puede evidenciar en cada una de las figuras mostradas, en las cuales, se enmarca los materiales en el espacio correcto y con las dimensiones apropiadas.

3.4.2. Implementación del Modelo en Tarjeta Raspberry

Para iniciar la implementación en la Raspberry Pi 3 model B+ se usó una memoria micro SD, mouse, teclado, monitor, Adaptador de tensión a 5V y Conexión a internet.

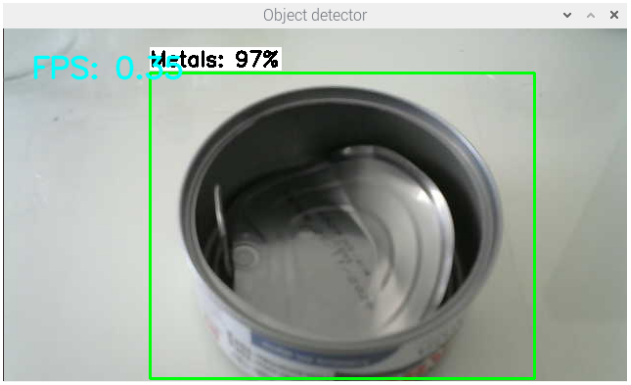
El proceso se empieza formateando la memoria micro SD para luego insertar la imagen del sistema operativo de la placa que podemos descargar directamente desde la pagina de Raspberry. Después de tener la memoria flasheada con la imagen del sistema operativo, procedemos a iniciar la placa y configurar el inicio del sistema operativo.(SD, 2021)

Seguidamente se instala el entorno virtual de python en el cual se guardaran todas las librerias necesarias, posteriormente se procede con la instalación de tensorflow y keras que son librerías de Python que facilitan el proceso de entrenamiento e implementación del modelo. Asimismo, se procede con la instalación de la librería OpenCV, la que se encarga del desarrollo de las aplicaciones de visión por computador en tiempo real.

Cuando ya tenemos instalado todo el software antes mencionado se hace la verificación de dependencias para evitar que haya dependencias incumplidas. Cuando aseguramos que el software está en perfectas condiciones se carga el modelo que se ha desarrollado anteriormente de forma online en Colab (Sastre *et al.*, 2022d), incluyendo la misma cámara con la que se hizo la implementación en la Jetson Nano.

En las Figuras 25, 26, 27, 28 y 29, se presentan las fotos con los artículos que tuvieron los mejores parámetros de desempeño para cada clase:

Figura 25.
Mejor desempeño, clase metal



Todas las imágenes se pueden encontrar en el siguiente link *https://drive.google.com/drive/u/0/folders/1SvaSCaK6b_auSSKYETckEpKPX25TYuzn*

Finalmente, se toma el promedio obtenido de 5 artículos diferentes por cada clase, se registran los datos de precisión, Fps y por ultimo si acertó ó no en la clasificación del material reciclable (Ver Tabla 5).

Tabla 5
Resultados placa Raspberry

Placa	Precision (%)	Fps (Fps)	Número aciertos
Metal	85.5	0.41	4/5
Cartón	88	0.38	3/5
Papel	91.25	0.40	4/5
Plástico	94.6	0.42	5/5
Vidrio	90.2	0.43	4/5

Figura 26.
Mejor desempeño, clase cartón

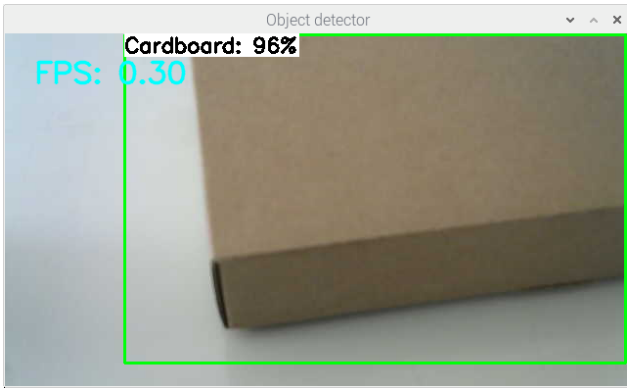


Figura 27.
Mejor desempeño, clase papel

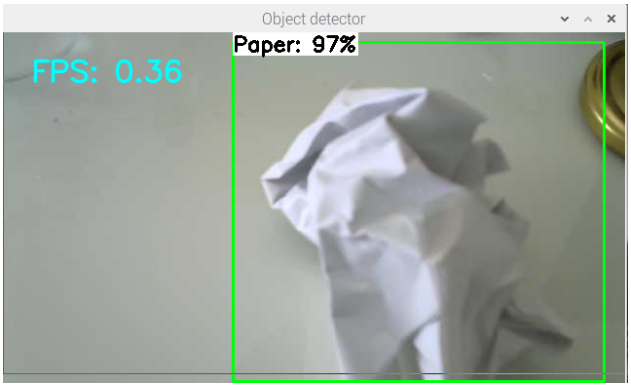
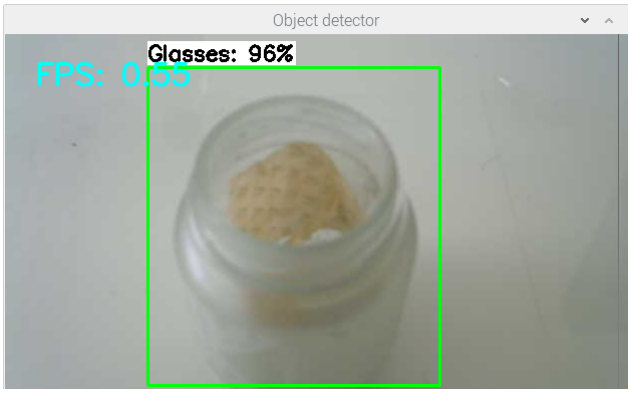


Figura 28.
Mejor desempeño, clase plástico

Figura 29.
Mejor desempeño, clase vidrio



En el caso de por ejemplo la clase Cartón, los datos de la precisión en la clasificación solo se tomaron de los artículos correctamente clasificados, tiene una alta precisión respecto a su nivel de asertividad de clasificación debido a que generaliza ciertos artículos mejor que otros, como por ejemplo cartones lizos sin ningún tipo de color o material adicional.

Es importante destacar el excelente comportamiento del modelo al realizar el encuadre de las imágenes, lo que se puede evidenciar en cada una de las figuras mostradas, en las cuales, se enmarca los materiales en el espacio correcto y con las dimensiones apropiadas.

3.5. Analisis de Resultados

A continuación, se presentan los criterios de selección del modelo *MobileNetFPN* del cual se tuvieron en cuenta parámetros como lo son la función de perdida y la precisión en la detección de los datos.

Asimismo, se realiza la comparativa entre las implementaciones de los dos sistemas embe-

bidos, la cual nos muestra el promedio del porcentaje de precisión de clasificación de 10 artículos para cada clase.

Finalmente, se hace una comparativa entre las dos placas desde un punto de vista comercial y operativo.

3.5.1. Comparativa Función de Perdida entre Modelos

Para la elección del modelo a implementar, primero se aplica la función de perdida de cada uno de estos, la cual evalúa la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas durante el aprendizaje, de acuerdo a esta evaluación, se filtran los dos primeros modelos de los tres iniciales.

Primero, se presenta las gráficas de desempeño de la función de perdida de cada modelo (Ver Figuras 30, 31, 32), la cual se divide en 4 gráficas en total, posteriormente se incluye la gráfica *total loss* (Ver Figura 33) la cual resume los 3 diferentes tipos de perdida que se muestran, *classification loss*, *localization loss* y *regularization loss*.

Figura 30.
Función de perdida *EfficientNet*

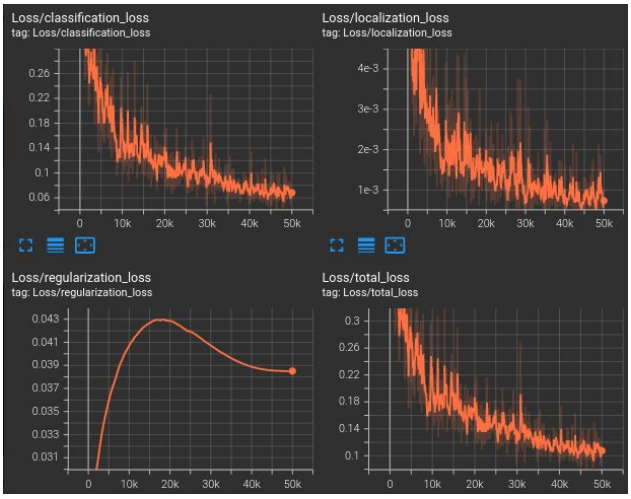


Figura 31.
Función de perdida *MobileNetV2*

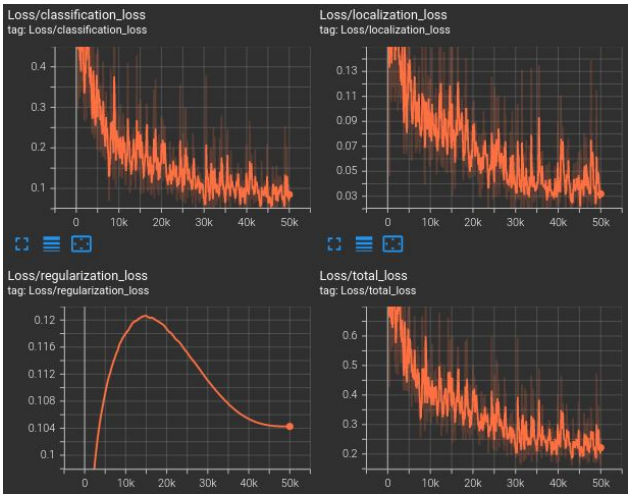
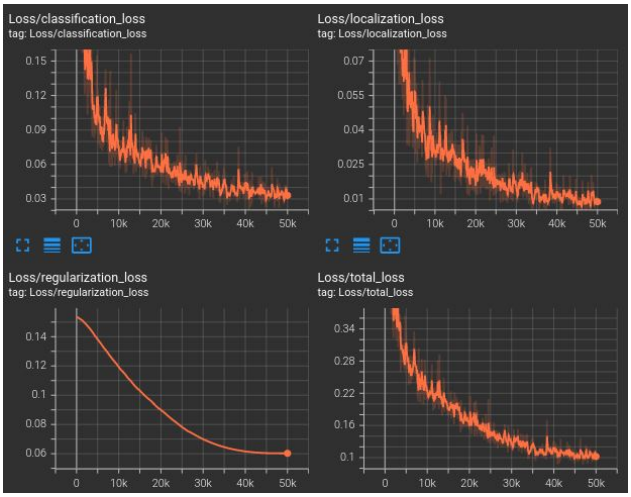
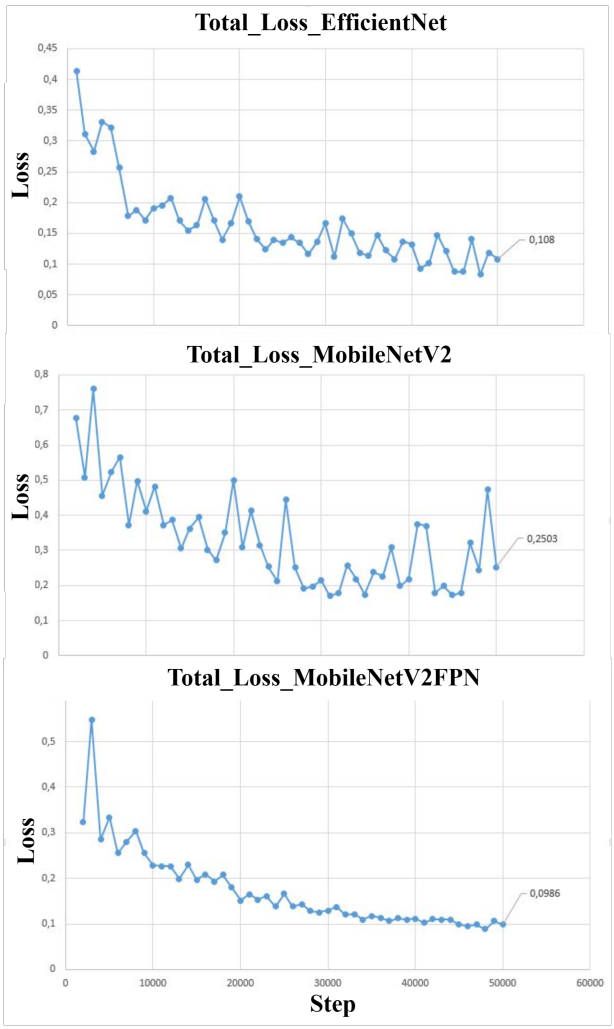


Figura 32.
Función de perdida *MobileNetV2FPN*



Como se puede observar en la Figura 33, los dos modelos que obtuvieron valores mínimos de la función de perdida fueron *EfficientNet* y *MobilnetV2FPN*, por lo que se descartó el modelo *MobileNetV2*.

Figura 33.
Función de perdida de los 3 modelos



3.5.2. Comparativa mAP Results entre Modelos

A continuación, se realizó la comparativa entre los dos modelos escogidos, mediante la métrica que arroja el *mAP Results*, que como ya se mencionó, es utilizada para evaluar modelos de detección de objetos.

Primero, se realiza un barrido desde 50 unidades de *threshold* hasta 95 unidades de *threshold* (Ver Figuras 34 y 36), el cual por ejemplo para el caso de *threshold* = 50, le indica a la red que solo se tomaran en cuenta los datos que alcancen un nivel superior de precisión a este valor.

Para cada nivel, se obtiene el mAP de todas las clases (Ver Figuras 35 y 37). lo que significa que en el caso de obtener un mAP de por ejemplo 90 % en un *threshold* = 80, nos indica que el modelo en el 90 % de los datos obtuvo una precisión de detección del promedio de todas las clases por encima del 80 %.

Los datos obtenidos para la gráfica de la figura 34 se encuentran en la siguiente carpeta (Sastre *et al.*, 2022a)

Los datos obtenidos para la gráfica de la figura 36 se encuentran en la siguiente carpeta (Sastre *et al.*, 2022b).

Figura 34.
mAP *EfficientNet* Threshold variable

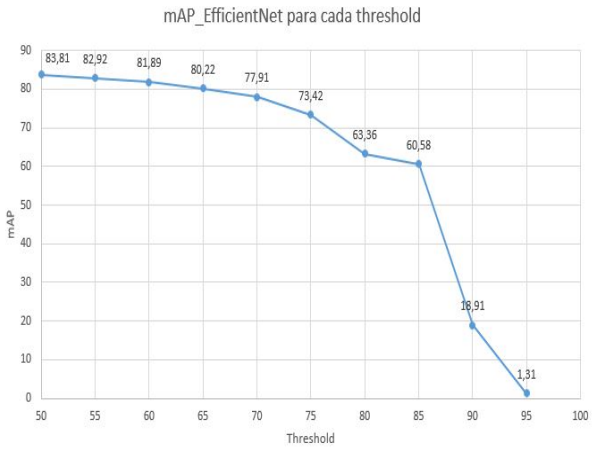
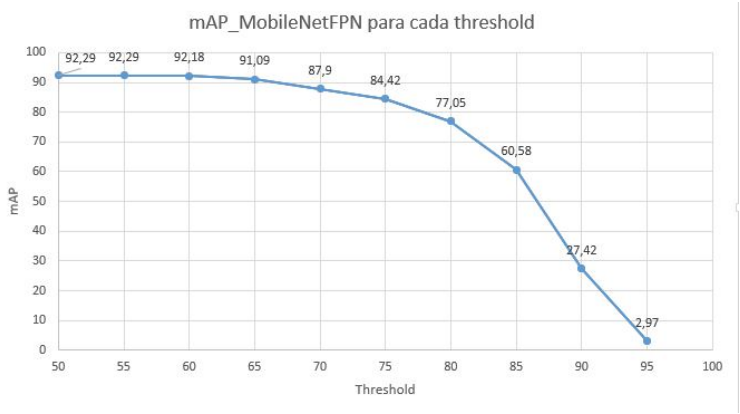


Figura 35.
mAP *results EfficientNet*

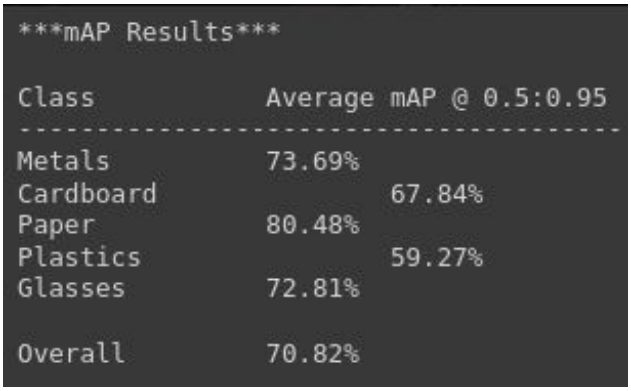
```
***mAP Results***

Class           Average mAP @ 0.5:0.95
-----------------------------------------
Metals          62.24%
Cardboard               57.51%
Paper           74.32%
Plastics                53.04%
Glasses         57.31%

Overall         60.88%
```

Figura 36.
mAP *MobileNetV2FPN* Threshold variable



Como se puede observar en las Figuras 35 y 37, el modelo que obtuvo mejor desempeño en la precisión de detección fue la *MobileNetV2FPN*, la cual obtuvo una precisión promedio de aproximadamente 10 puntos porcentuales por encima de la *EfficientNet*, resultado relevante para escoger este modelo como el idóneo para llevar acabo la implementación en las tarjetas.

Figura 37.
mAP *results MobileNetV2FPN*



3.5.3. Comparacion Implementación Jetson Vs Raspberry

Como se puede observar en la Tabla 6, el modelo tiene mejores resultados sobre la placa Jetson, tanto en la cantidad de aciertos como en el promedio de FPS.

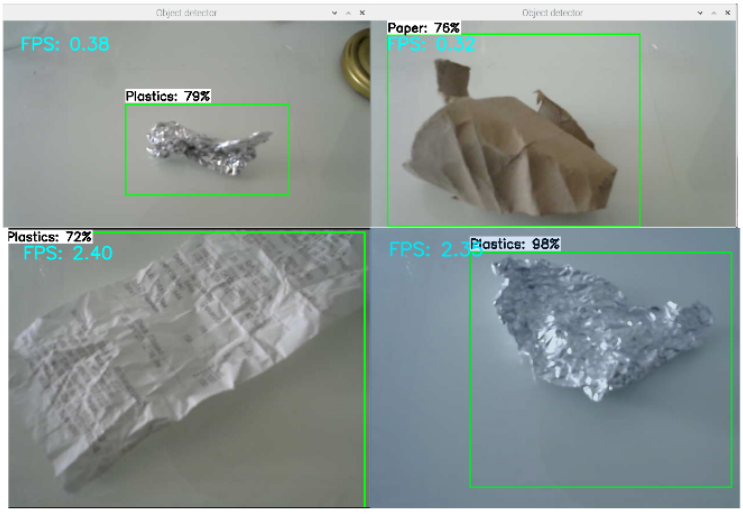
A continuación, se presentan algunos de los resultados que se obtuvo de las implementaciones en ambas tarjetas

Como se puede evidenciar en la Figura 38, el modelo tiende a clasificar los objetos que presentan texturas rugosas o deformes, en la clase de plásticos, cuando estos objetos son unicolor

Tabla 6
Resultados Totales Jetson vs Raspberry

Placa	Precisión Metal (%)	Precisión Cartón (%)	Precisión Papel (%)	Precisión Plástico (%)	Precisión Vidrio (%)	Precisión Total (%)	Aciertos Totales	Fps (Fps)
Jetson	85.75	89.75	93	94.4	89	90.38	21/25	2.296
Raspberry	85.5	88	91.25	94.6	90.2	89.91	20/25	0.408

Figura 38.
Clasificación textura rugosa



la tendencia es clasificarlo como papel.

Figura 39.
Clasificación acertada según tarjeta

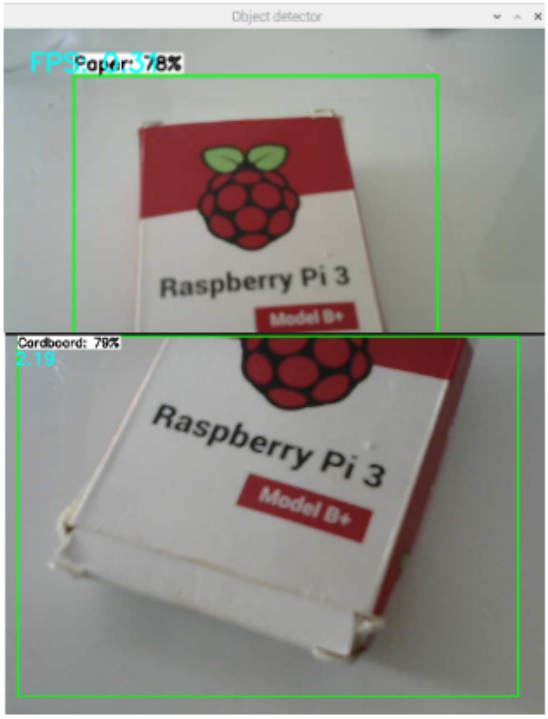


Figura 40.
Clasificación errónea

Al realizar la implementación del modelo en ambas tarjetas, se evidencia como un mismo objeto puede tener dos clasificaciones diferentes, la Figura 39 comprueba lo mencionado. Además, el comportamiento de la Figura 40, es un ejemplo de fallo de clasificación de la red.

Asimismo, es importante mencionar el precio y la disponibilidad de cada tarjeta, estos datos se muestran en la tabla 7, los precios se tomaron a día de hoy 30 de enero de 2023, en la plataforma de comercio Amazon.

Tabla 7
Comparación entre placas de precio y disponibilidad

Tarjeta	Precio (USD)	Disponibilidad (Días)
Raspberry	$149.98	10 aprox.
Jetson	$282.85	30 aprox.

3.5.4. Rendimiento sistemas embebidos

En la Tabla 8, se muestra los recursos usados por cada tarjeta a los 5, 60 y 180 minutos corriendo el modelo, para estos datos se utilizó el programa Jtop para la tarjeta Jetson y el comando Htop para la tarjeta Raspberry. Asimismo, se identificó un aumento en la temperatura en ambas tarjetas conforme pasa el tiempo, principalmente en la primera hora de ejecución, ya que posteriormente tiende a mantenerse constante. De igual forma, aumenta el espacio en memoria hasta mantenerse constate a partir de los 60 minutos. Además, es de relevancia destacar que el espacio en memoria usado en ambas tarjetas es porcentualmente similar, ya que la capacidad total de la Raspberry es de 800 MB y el de la Jetson de 2 GB. Por ultimo, se observó en ambas tarjetas que la capacidad del total de las 4 CPU, tuvo un pico en los primeros 5 minutos, posteriormente presentó una tendencia estable

Cabe mencionar que ambas tarjetas mantienen un desempeño óptimo a lo largo del tiempo, obteniendo resultados de clasificación similares respecto a los que se toman en un principio, por tanto se concluye que la implementación propuesta es viable para aplicaciones que requieran uso

Tabla 8
Recursos usados de sistemas embebidos durante 3 horas

Tarjeta/Tiempo (min)	Tempertura GPU (°C)	Espacio en memoria (MB)	CPU1 (%)	CPU2 (%)	CPU3 (%)	CPU4 (%)
Raspberry / 5	61.2	240	47	79.5	54.5	81.2
Jetson / 5	44	800	81	75	70	70
Raspberry / 60	68.8	257	50	72.3	49.4	58.4
Jetson / 60	55.5	900	73	78	68	67
Raspberry / 180	70.2	258	55.7	45.4	51.7	79.7
Jetson / 180	54.5	900	79	59	61	58

constante.

4. Conclusiones

1. Se logró entrenar un modelo de red neuronal supervisada capaz de clasificar materiales reciclables en 5 diferentes clases de residuos, estos son metales, cartón, plástico, papel y vidrio, asimismo, se implementó en dos sistemas embebidos, Raspberry pi 3 model B + y Jetson Nano 2Gb.

2. La creación de la base de imágenes para el desarrollo de este sistema fue un factor de gran relevancia en la implementación de los modelos en los sistemas embebidos. Tomó un alto porcentaje de tiempo en el proyecto desde la creación de la base de imágenes, búsqueda de imágenes externas, etiquetado y redimensionamiento, siendo este último el factor más importante en el procesamiento de la base de imágenes, se evidenció que el tener imágenes del mismo tamaño se reducen los errores de redimensionamiento.

3. Los avances tecnológicos en el campo del aprendizaje por transferencia han tenido gran auge y acogida en las últimas décadas. Por esta razón se encuentra un sinnúmero de algoritmos y modelos de inteligencia artificial que facilitan el estudio de esta rama. Se propone una metodología para el desarrollo de aplicaciones de detección en sistemas embebidos usando modelos de aprendizaje por transferencia. Se obtienen resultados óptimos teniendo en cuenta los sistemas embebidos seleccionados.

4. Tomando como base la implementación realizada en ambos sistemas embebidos, se identificó que el modelo clasifica las imágenes con texturas rugosas o deformes en la clase de plásticos (Ver Figura 39), esto posiblemente debido a que en el entrenamiento, una considerable cantidad de residuos clasificados como plásticos tienen estas características, Asimismo, se evidenció que los materiales con estas características, y además tuvieran un solo color, presentaron una tendencia a clasificarse como papel (Ver Figura 23 y 27).

5. Por medio de la implementación se determinó que los materiales identificados como plástico, vidrio y metal obtuvieron los mejores índices de desempeño y acierto en la clasificación, por el contrario, el cartón y el papel tuvieron mayores dificultades en la generalización de las etiquetas (Ver Tabla 5 y 4)., debido a la similitud entre estos dos tipos de materiales, por lo que se propone como un área de mejora el aumento en la base de imágenes y caracterización específicamente en estas dos clases.

6. Según los resultados obtenidos en la implementación, representados en la tabla 6 se identificó que la tarjeta Jetson obtuvo un rendimiento superior sobre la Raspberry, si bien la Jetson nano

tiene un costo superior (ver Tabla 7) Se sugiere esta tarjeta como la de mejor desempeño y rendimiento computacional, sustentado en mayor rapidez de identificación de elementos y acertividad de clasificación, de igual manera, cuenta con herramientas que la convierten en un sistema embebido mucho mas escalable para implementaciones mas robustas.

5. Recomendaciones

Se recomienda aumentar el numero de imágenes tomadas con diferentes ángulos, texturas y colores de cada una de las clases para evitar los errores de clasificación presentados en la implementación, de igual manera, la superficie donde se muestran los artículos a la cámara debe ser plana, blanca y sin ningún tipo de brillo o reflejo. Asimismo, se sugiere realizar la prueba del modelo en los sistemas embebidos en un sitio con buena iluminación, asi como tomar el registro de las imágenes con el mismo ángulo y sentido en ambas tarjetas, para así aumentar la confiabilidad del sistema.

Se aconseja aumentar el número de clases y ser más específico en los metales, clasificarlos por ejemplo en aluminio, cobre etc.

Por último, se recomienda incluir un parámetro que indique el estado en el que se encuentra el material, y un estimado del precio al cual se pueda vender ese articulo, en lo posible dar el resultado en precio por kilogramo.

Referencias Bibliográficas

Alba, J. L. & Alcorta, N. F. (2020). *Sistemas Embebidos, Guia metodológica para su desarrollo*. 1ª edición. https://static.upao.info/descargas/78a608f43f7702198f00faee981db7a3a84c4f8c76d77c8de61bf08dbab5b581514ea884149f697630e093afcdca9237549aca26fadd34bdd95cac8ead2db981/sistemas-embebidos–guIa-metodolOgica-para-su-desarrollo.pdf.

Buiu, C., Danaila, V.-R., & Raduta, C. N. (2020). Mobilenetv2 ensemble for cervical precancerous lesions classification. *Processes*, 8(5). doi: 10.3390/pr8050595.

Cao, L. & Xiang, W. (2020). Application of convolutional neural network based on transfer learning for garbage classification. En *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pp. 1032–1036. doi: 10.1109/ITOEC49072.2020.9141699.

cumit Atila, Uçar, M., Akyol, K., & Uçar, E. (2021). Plant leaf disease classification using efficientnet deep learning model. *Ecological Informatics*, 61:101182. doi: 10.1016/j.ecoinf.2020.101182.

Departamento Nacional, P. (21 de noviembre de 2016). Política nacional para la gestión integral de residuos solidos. url: https://colaboracion.dnp.gov.co/CDT/Conpes/Econ%C3%B3micos/3874.pdf, Accedido el 18 de Diciembre, 2021.

Electronics., E. (2019). How to train an object detection classifier for multiple objects using tensorflow (gpu) on windows 10. url: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10.

Elfwing, S., Uchibe, E., & Doya, K. (2017). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. En arXiv:1702.03118.

Fausett, L. (1993). Fundamentals of neural networks architectures, algorithms and applications. *Pearson*, pp. 3–7.

GAD, A. F. (2020). Algoritmo map. https://blog.paperspace.com/mean-average-precision/amp/.

Gupta, S. (2022). The 7 most common machine learning loss functions. https://builtin.com/machine-learning/common-loss-functions.

Harika, J., Baleeshwar, P., Navya, K., & Shanmugasundaram, H. (2022). A review on artificial intelligence with deep human reasoning. En *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, pp. 81–84. 10.1109/ICAAIC53929.2022.9793310.

Keras (2022). Keras. url: https://keras.io/.

Kulkarni, H. N., Kannamangalam, N., & Raman, S. V. (2019). Waste object detection and classification. En *CS230 Stanford*.

Li, M., Chen, D., Liu, S., & Liu, F. (2021). Detección de límites semisupervisada para granos de aluminio combinada con transferencia de aprendizaje y crecimiento de regio-

nes. *Transacciones IEEE en redes neuronales y sistemas de aprendizaje*, pp. 1–15. doi: 10.1109/TNNLS.2021.3133760.

Ling-fang, H. (2010). Artificial intelligence. En *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, volumen 4, pp. 575–578. 10.1109/ICCAE.2010.5451578.

Meng, J., Jiang, Ping, W. J., & Wang, K. (2022). A mobilenet-ssd model with fpn for waste detection. *Processes*, 17. doi: 10.1007/s42835-021-00960-w.

Meng, J., J. P. W. J. (2021). A mobilenet-ssd model with fpn for waste detection. *Springer*. https://doi.org/10.1007/s42835-021-00960-w.

Meng, S. & Chu, W.-T. (2020). A study of garbage classification with convolutional neural networks. En *2020 Indo – Taiwan 2nd International Conference on Computing, Analytics and Networks (Indo-Taiwan ICAN)*, pp. 152–157. doi: 10.1109/Indo-TaiwanICAN48429.2020.9181311.

N. Diaz, A. (2018). *Reconocimiento de objetos en imágenes usando aprendizaje profundo*. Tesis doctoral, Departamento de Automática y Electrónica, Universidad Autónoma de Occidente.

NVIDIA (2021). Deep learning frameworks documentation. url: https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html, Accedido el 22 de Diciembre, 2021.

Nvidia (2023). Kit para desarrollador jetson nano. https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano/product-development/.

OpenCV (2023). Opencv team. url: https://opencv.org/.

Pascasio, J., Mela, R., Velez, M., & Jose, R. (2023). Implementación de redes neuronales para la clasificaciÓn de desechos dentro de un cesto inteligente. http://portal.amelica.org/ameli/journal/228/2282818015/html/.

Ramachandran, P., Zoph, B., & V. Le, Q. (2017). Searching for activation functions. En https://arxiv.org/pdf/1710.05941v2.pdf.

Raspberry (2023). Raspberry pi 3 model b+. https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/.

S. Jadhav, R. P. & Rege, P. (2019). Audio splicing detection using convolutional neural network. *IEEE*, pp. 1–5. doi: 10.1109/ICCCNT45670.2019.8944345.

Santoro, A. (2019). Split garbage dataset. url: https://www.kaggle.com/andreasantoro/split-garbage-dataset, Accedido el 06 de Diciembre, 2021.

Sastre, C., Ramirez, J., & Galeano, N. (2022a). Barrido threshold efficientnet, google drive. En *https://onx.la/d20e3*.

Sastre, C., Ramirez, J., & Galeano, N. (2022b). Barrido threshold mobilenetv2fpn, google drive. En *https://onx.la/b0fd7*.

Sastre, C., Ramirez, J., & Galeano, N. (2022c). Base de datos. https://drive.google.com/drive/folders/1DZYVvedkva8gI1qGIOd5OleK0eJPqeBv?usp=sharing.

Sastre, C., Ramirez, J., & Galeano, N. (2022d). Código de entrenamiento. url: https://github.com/CristianSastre/Sistema-autonomo-de-clasificacion-de-material-reciclable.

Sastre, C., Ramirez, J., & Galeano, N. (2022e). Dataset de imagenes, google drive. En *https://onx.la/81b24*.

SD, A. (2021). Sd memory card formatter for windows download. url: https://www.sdcard.org/downloads/formatter/sd-memory-card-formatter-for-windows-download/, Accedido el 22 de Diciembre, 2021 .

Sotaquirá, M. (2018). La función de activación. url: https://www.codificandobits.com/blog/funcion-de-activacion/.

T. Guo, J. Dong, H. L. & Gao, Y. (2017). Simple convolutional neural network on image classification. *IEEE*, pp. 721–724. doi: 10.1109/ICBDA.2017.8078730.

Technologies, S. (2022). Max-ai. url: https://sadako.es/max-ai/?lang=es, Accedido el 22 de Diciembre, 2021.

Technology, T. (26 de marzo de 2020). El último e-book de tomra muestra el potencial que la inteligencia artificial supone para la industria del reciclaje. url: https://www.tomra.com/es-es/sorting/recycling/recycling-news/2020/tomra-sorting-ebook-details-potential-of-ai-holds-recycling, Accedido el 02 de Diciembre, 2021.

Tensorflow (2022a). Descripción general de las api principales de tensorflow. url: https://www.tensorflow.org/guide/core.

Tensorflow (2022b). Implemente modelos de aprendizaje automático en dispositivos móviles y perimetrales. url: https://www.tensorflow.org/lite.

Tkachenko, M., Skriabin, N., & Zhuk, S. (2023). Label images. https://github.com/heartexlabs/labelImg.

Vatsa, V. R. (2021). Analysis of global research proceedings in artificial intelligence. En *2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, pp. 19–21. 10.1109/CCICT53244.2021.00015.

Yugsi, J. (2020). *Implementación de un sistema para el reconocimiento y clasificación de la contaminación supercial de la laguna de Colta mediante algoritmos de Inteligencia Artificial*. http://dspace.unach.edu.ec/handle/51000/7210, Universidad Nacional de Chimborazo.