# Control de calidad en la producción de materiales publicitarios para empresas de diseño y marketing usando imágenes e inteligencia artificial

Henry Nicolás Cortés Bolaños y Jhonatan Felipe Valest Flores

Trabajo de Grado para Optar al Título de Ingeniero Electrónico

Director

Jaime Guillermo Barrero

Magíster en Potencia Eléctrica

Universidad Industrial de Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Ingeniería Electrónica

Bucaramanga

2025

## Dedicatoria

Dedico este logro con profunda gratitud y amor a mi padre, Omar Valest, por ser mi ejemplo de fortaleza, disciplina y sabiduría; a mi abuela Benilda Vesga, por su ternura infinita, sus oraciones constantes y su fe inquebrantable en mí; y a mi madre, Danila Flores, por su apoyo incondicional, sus sacrificios silenciosos y su amor que ha sido mi impulso constante.

También extiendo esta dedicatoria a todas aquellas personas cercanas que, con palabras de aliento, gestos de cariño o compañía sincera, fueron luz en los momentos de incertidumbre y fuerza en los días más exigentes. Cada uno de ustedes forma parte de este logro, y a todos les llevo en el corazón con profunda gratitud.

***Jhonatan Felipe Valest Flores***

A mi amada madre, Ana Cristina, quien con su amor incondicional, su inquebrantable fe y su incansable apoyo, ha sido la inspiración y la fuerza motora en cada paso de mi camino. Tus sacrificios, tus enseñanzas y tu constante aliento hicieron posible la realización de este sueño. Este logro es tuyo también.

A mi querido hermano, Joan, por ser el pilar que me brindó la oportunidad de iniciar este camino en la universidad brindándome el dinero para pagar el pin y por el regalo invaluable de mi primera y única calculadora, fiel compañera a lo largo de toda mi carrera. Tu apoyo y generosidad fueron esenciales para alcanzar este logro.

***Henry Nicolás Cortés Bolaños***

**Agradecimientos**

Expreso mi más sincero agradecimiento a la Universidad Industrial de Santander, institución que me brindó las herramientas académicas, tecnológicas y humanas necesarias para llevar a cabo este proyecto de grado. Su compromiso con la formación integral y la excelencia académica ha sido fundamental en mi proceso como estudiante y futuro profesional.

Agradezco de manera especial a los profesores, quienes, con su dedicación, conocimiento y guía constante, aportaron significativamente a mi desarrollo académico y personal. Cada clase, cada consejo y cada corrección fueron piezas clave en la construcción de este trabajo.

Este proyecto también es el reflejo del esfuerzo colectivo, de un entorno educativo que inspira, motiva y reta. A todos quienes hicieron parte directa o indirectamente de este camino, gracias por ser parte de esta etapa tan valiosa de mi vida.

***Jhonatan Felipe Valest Flores***

A mí novia espectacular novia Geraldine, agradezco sus constantes palabras de motivación, apoyo y amor incondicional ya que me ha acompañado en mi proceso estudiantil y ayudado a ser una mejor persona. También por su comprensión y paciencia durante este arduo proceso. Gracias por estar a mi lado en cada paso del camino y por ser mi pilar en los momentos difíciles.

A mi compañero, Jhonatan Felipe Valest Flores, por la colaboración incansable, el trabajo en equipo, las horas de discusión y el apoyo mutuo que hicieron esta jornada más llevadera y enriquecedora.

A la Universidad Industrial de Santander, por ser el espacio donde mi conocimiento y mi persona pudieron crecer y desarrollarse plenamente. Gracias por las oportunidades y las enseñanzas que moldearon mi trayectoria profesional y personal.

Finalmente, a todos aquellos amigos y colegas que, de una u otra forma, aportaron su granito de arena, sea con una palabra de aliento, una idea o un simple momento de distensión, gracias por ser parte de este camino.

***Henry Nicolás Cortés Bolaños***

## Tabla de contenido

## Índice de Tabla

## Índice de Figura

## Resumen

**Título:** Control de calidad en la producción de materiales publicitarios para empresas de diseño y marketing usando imágenes e inteligencia artificial*

**Autor:** Henry Nicolás Cortés Bolaños y Jhonatan Felipe Valest Flores**

**Palabras Clave:** Control de calidad, inteligencia artificial, procesamiento de imágenes, materiales publicitarios, redes neuronales convolucionales, marketing, diseño gráfico, impresión digital

**Descripción:** Para este proyecto se desarrolló un sistema embebido de bajo costo para el control de calidad automatizado en la producción de materiales publicitarios impresos, empleando procesamiento de imágenes e inteligencia artificial. Se seleccionó la Raspberry Pi 4 por su balance entre capacidad de procesamiento y costo, integrando una cámara para la captura en tiempo real. Se construyó un conjunto de datos de 3.278 imágenes, junto con filtros ofrecidos por Roboflow lo que generó un total de 14.254. El modelo de detección elegido fue YOLOv11 Large, por su alta precisión, entrenado con 100 épocas y evaluado con métricas como mAP@0.5 (83.4%), precisión (82.1%) y recall (74.6%), mostrando un buen equilibrio entre detecciones correctas y mínimas omisiones. La implementación del sistema permitió realizar inferencia en tiempo real, identificando y clasificando defectos incluso en condiciones de iluminación variable o baja calidad de captura. Los resultados evidencian la viabilidad del sistema como alternativa económica y reproducible para empresas con recursos limitados, optimizando procesos, reduciendo desperdicio y mejorando la calidad de impresión. Las pruebas demostraron su capacidad de generalización en diferentes escenarios y dispositivos, lo que abre la posibilidad de su adopción en entornos de producción reales, así como futuras mejoras mediante hardware con aceleración de IA y ampliación del dataset.

---

* Trabajo de Grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director Jaime Guillermo Barrero Perez.

**Abstract**

**Title:** Quality control in the production of advertising materials for design and marketing companies using images and artificial intelligence*

**Author(s):** Henry Nicolás Cortés Bolaños y Jhonatan Felipe Valest Flores**

**Key Words:** Quality control, artificial intelligence, image processing, advertising materials, convolutional neural networks, marketing, graphic design, digital printing

**Description:** For this project, a low-cost embedded system was developed for automated quality control in the production of printed advertising materials, using image processing and artificial intelligence. The Raspberry Pi 4 was selected for its balance between processing power and cost, integrating a camera for real-time capture. A dataset of 3,278 images was constructed, along with filters offered by Roboflow, generating a total of 14,254. The detection model chosen was YOLOv11 Large, due to its high accuracy. It was trained with 100 epochs and evaluated with metrics such as mAP@0.5 (83.4%), precision (82.1%), and recall (74.6%), showing a good balance between correct detections and minimal omissions. The implementation of the system enabled real-time inference, identifying and classifying defects even in conditions of variable lighting or low capture quality. The results demonstrate the system's viability as a cost-effective and reproducible alternative for companies with limited resources, optimizing processes, reducing waste, and improving print quality. The tests demonstrated its generalization capabilities across different scenarios and devices, opening up the possibility of its adoption in real-world production environments, as well as future improvements through hardware with AI acceleration and dataset expansion.

---

* Thesis

**Faculty of Physical and Mechanical Engineering. School of Electrical, Electronic, and Telecommunications Engineering. Director: Jaime Guillermo Barrero Perez.

## Introducción

En el competitivo panorama actual del diseño y marketing, la calidad de los materiales publicitarios impresos es un pilar fundamental para la imagen de marca y la efectividad de las campañas. Sin embargo, la industria enfrenta un desafío persistente en el control de calidad durante la producción de estas impresiones. Los métodos tradicionales, predominantemente manuales, son lentos, costosos y propensos a pasar por alto errores como inconsistencias de color, manchas o corrimientos (Nunsys, s.f.). Estos defectos no solo afectan la percepción de profesionalismo y credibilidad de la marca, sino que también pueden generar costos adicionales por desperdicio y reimpresión (Imprenta Minerva, 2024; FasterCapital, s.f.).

Para superar estas limitaciones, este proyecto propone una solución innovadora: la implementación de un sistema embebido de control de calidad basado en procesamiento de imágenes e inteligencia artificial. Esta decisión se fundamenta en el potencial de la IA y la visión artificial para automatizar la detección de defectos con alta precisión y eficiencia (E2M COUTH, 2023; GFT Technologies, s.f.). A diferencia de la inspección humana, que es inviable a las velocidades de impresión actuales (Nunsys, s.f.), los sistemas automatizados pueden analizar el diseño y la composición visual en tiempo real, identificando anomalías de manera consistente (E2M COUTH, 2023). La aplicación de inteligencia artificial en la impresión ya ha demostrado su capacidad para reducir tiempos de producción, minimizar errores humanos y optimizar recursos (Omán Impresores, 2024; Konica Minolta, 2024).

Los impactos de este proyecto son significativos. A nivel empresarial, permitirá a las compañías de diseño y marketing (mercadeo) ofrecer productos de calidad superior, fortaleciendo

su competitividad y la satisfacción del cliente. Económicamente, la detección temprana de defectos reducirá considerablemente los costos de producción y desperdicio. Además, al optimizar los procesos, se contribuirá a prácticas más sostenibles, minimizando el consumo de papel y tinta. En síntesis, este trabajo busca no solo mejorar la calidad en la producción de materiales publicitarios, sino también optimizar la eficiencia operativa y promover un enfoque más consciente con el medio ambiente en la industria.

# 1. Objetivos

## 1.1. Objetivo General

Implementar un sistema embebido de control de calidad durante el proceso de producción de materiales impresos, para empresas de diseño y marketing, mediante una emulación a nivel de laboratorio basada en procesamiento de imágenes e inteligencia artificial.

## 1.2. Objetivos Específicos

Identificar y estudiar los errores comunes como las fallas, manchas y corrimientos del diseño en las impresiones de materiales publicitarios a ser detectados en el proyecto.

Crear una base de datos de imágenes dedicadas al reconocimiento de los errores más importantes, con el propósito de validar cada diseño, para el entrenamiento de modelos de inteligencia artificial.

Desarrollar algoritmos, que por medio de técnicas de procesamiento digital de imágenes y modelos de aprendizaje profundo como redes neuronales convolucionales (CNN), permitan detectar errores en las impresiones publicitarias.

Implementar un sistema embebido de bajo costo con capacidad de adquisición de imágenes mediante una cámara integrada y algoritmos que permitan identificar defectos en las imágenes utilizadas en los procesos de impresión publicitaria.

Validar el sistema embebido de control de calidad implementado en la producción de materiales publicitarios mediante la evaluación de su desempeño utilizando las métricas PSNR (Proporción Máxima de Señal a Ruido) y SSIM (Índice de Similitud Estructural), asegurando que las anomalías detectadas sean precisas.

# 2. Marco Teórico

Esta sección aborda los conceptos fundamentales y las bases teóricas que sustentan el desarrollo de este proyecto. Posteriormente, se presenta la investigación detallada que justificó la selección del sistema embebido utilizado en la implementación del proyecto.

## 2.1. Aprendizaje Automático

El aprendizaje automático, o *Machine Learning* (ML), es un subcampo de la inteligencia artificial que dota a los sistemas informáticos de la capacidad de aprender y mejorar a partir de la experiencia, sin necesidad de ser programados explícitamente para cada tarea. Funciona mediante algoritmos que analizan datos, identifican patrones y, basándose en ellos, realizan predicciones o toman decisiones, optimizando su rendimiento a medida que procesan una mayor cantidad y diversidad de información (Khan Academy, s.f.; Microsoft Azure, s.f.). Este proceso imita cómo un sistema ajusta y mejora continuamente sus capacidades al acumular

El aprendizaje automático se clasifica principalmente en cuatro categorías fundamentales, cada una con un enfoque distinto en la forma en que los algoritmos adquieren conocimiento, tal como se muestra en la Figura 1.

- **Aprendizaje Supervisado:** Consiste en entrenar un modelo utilizando un conjunto de datos previamente etiquetado, es decir, donde cada entrada está asociada a la salida o respuesta correcta. El objetivo del algoritmo es aprender la relación entre

las variables de entrada y salida, de modo que pueda predecir respuestas para datos nuevos con un nivel de precisión aceptable (HPE Lamerica, s.f.; Prometeo FP, s.f.).

- **Aprendizaje No Supervisado:** Se aplica cuando los datos no cuentan con etiquetas o categorías predefinidas. En este enfoque, el algoritmo busca estructuras ocultas, patrones de similitud o agrupaciones dentro de la información, permitiendo la segmentación y reducción de dimensionalidad de los datos sin intervención directa del usuario (INESDI, s.f.).

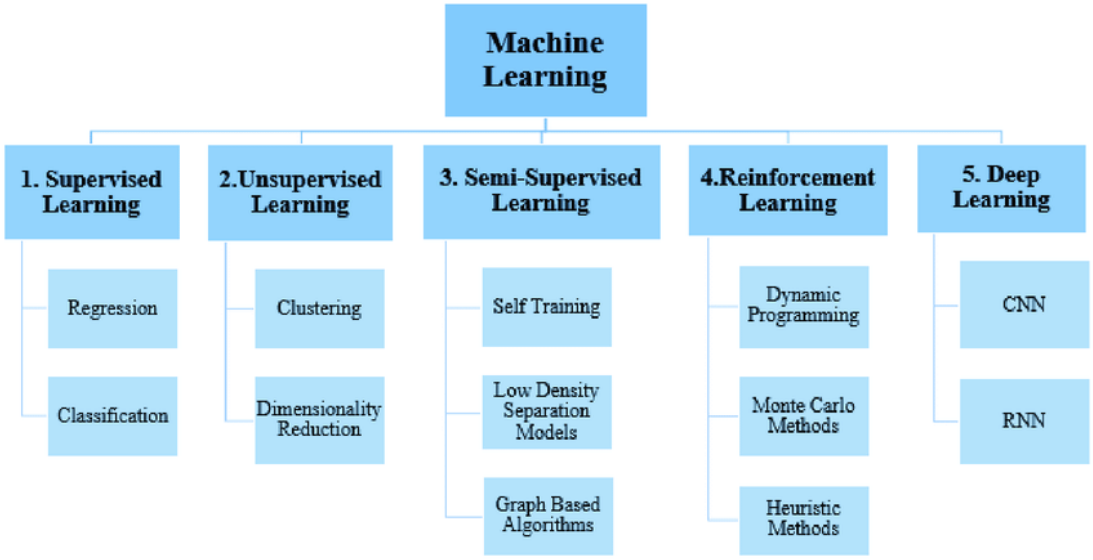
- **Aprendizaje por Refuerzo:** Este enfoque se basa en la interacción de un agente con un entorno. El agente ejecuta acciones y recibe retroalimentación en forma de recompensas o penalizaciones. A través de un proceso iterativo de ensayo y error, el modelo optimiza su política de decisión con el fin de maximizar la recompensa acumulada a largo plazo (AWS, s.f.; CEUPE, s.f.; IBM, s.f.b).

- **Aprendizaje Semisupervisado:** Este enfoque combina un conjunto reducido de datos etiquetados con un volumen considerable de datos no etiquetados, con el propósito de mejorar la capacidad predictiva del modelo. Lo fundamental es que, aunque los datos sin etiquetas no contienen una respuesta explícita, sí aportan información valiosa sobre la distribución y estructura del conjunto, lo que permite al algoritmo generalizar de manera más efectiva. Se trata de una estrategia especialmente útil en escenarios donde la obtención de datos etiquetados resulta

costosa o limitada, pero existe disponibilidad de grandes volúmenes de información sin etiquetar (IBM, s.f.; Mailchimp, s.f.).

**Figura 1**

*Tipos y algoritmos de aprendizaje automático*




*Nota***:** Esta imagen hace referencia a los tipos de algoritmos implementados en el Machine Learning. **Tomado de:** (Shaalan, n.d.).

## 2.2. Redes neuronales y Deep Learning

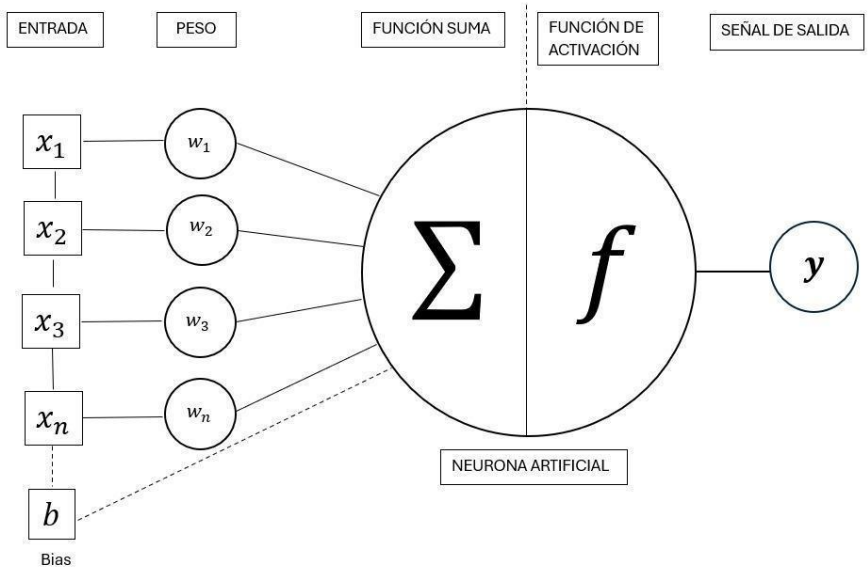
Las Redes Neuronales son un tipo de algoritmo de aprendizaje automático inspirado en la estructura y funcionamiento del cerebro humano, compuestas por capas de nodos interconectados (como neuronas) que procesan información de manera progresiva. Estos sistemas aprenden a reconocer patrones complejos ajustando las conexiones entre sus nodos a medida que se exponen a más datos, lo que les permite, por ejemplo, identificar objetos en imágenes o comprender el

lenguaje hablado (Google Cloud, s.f.a; IBM, s.f.a). Cuando estas redes tienen muchas capas —a menudo decenas o cientos— se les conoce como Deep Learning o Aprendizaje Profundo. Esta "profundidad" les permite aprender patrones extremadamente complejos y abstractos directamente de grandes volúmenes de datos brutos, sin necesidad de que un humano les diga qué características buscar (Google Cloud, s.f.b; Oracle, s.f.). Es el motor detrás de avances significativos en áreas como el reconocimiento facial, la traducción automática y los vehículos autónomos.

La Figura 2. muestra la representación matemática de una neurona artificial, mostrando sus componentes clave: la entrada (Xn), el peso (Wn), el sesgo (bias), la función de suma (∑), la función de activación y la señal de salida. Estas neuronas artificiales son los elementos que constituyen las redes neuronales.

**Figura 2**

*Representación esquemática del modelo matemático de una neurona artificial*




*Nota*: **Tomado de**: *Elaboración propia*

### *2.2.1. Redes neuronales convoluciones*

Las Redes Neuronales Convolucionales (CNN) son un tipo especializado de redes neuronales, particularmente eficaces en el procesamiento de datos con una estructura de rejilla, como imágenes, videos o señales de audio (HPE Lamerica, s.f.; IBM, s.f.a). Su diseño está inspirado en la corteza visual humana, lo que les permite identificar patrones y características de forma jerárquica: las primeras capas detectan elementos simples como bordes y texturas, mientras que las capas más profundas combinan estos para reconocer características cada vez más complejas, como formas o rostros completos (UNIR, s.f.; Juan Barrios, s.f.).

Las CNN logran esto mediante operaciones matemáticas llamadas "convoluciones" y capas de agrupación (pooling) que extraen información relevante y reducen la complejidad de los datos (IBM, s.f.a; MathWorks, s.f.). Son fundamentales en aplicaciones de visión artificial, como el reconocimiento facial, la detección de objetos, el diagnóstico médico por imágenes y los vehículos autónomos (DataCamp, s.f.; Lean Componentes, s.f.; Universidad Europea, s.f.).

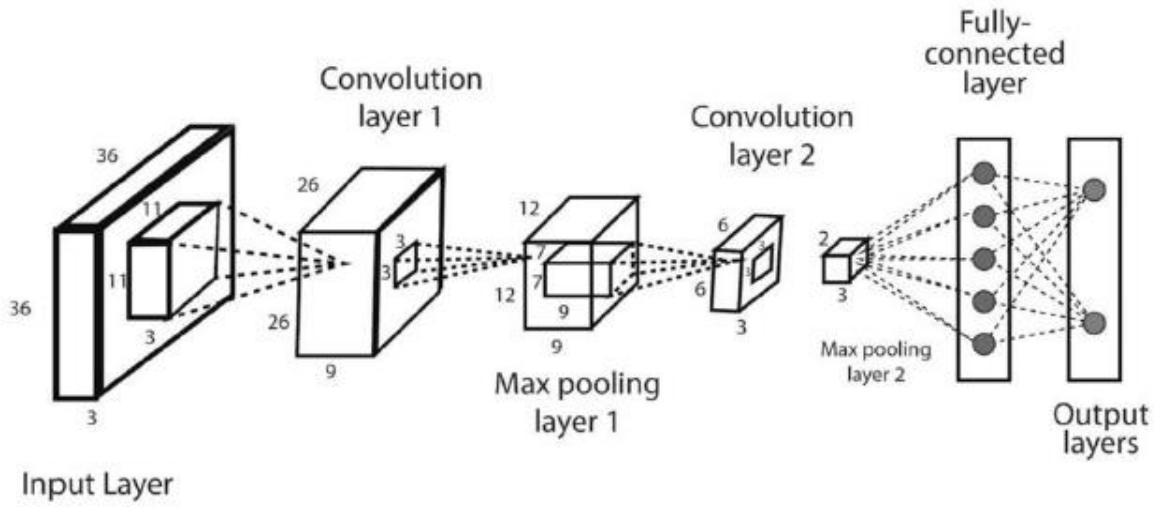
- **Capa convolucional:** Es el corazón de la CNN. Aquí, la red aplica filtros (también llamados *kernels*) sobre la imagen de entrada para detectar características específicas, como bordes, texturas o formas. Cada filtro crea un "mapa de características" que resalta la presencia de ese patrón en la imagen. Las primeras capas detectan características simples, y a medida que se avanza en la red, los filtros se combinan para reconocer patrones más complejos.
- **Capa de agrupación:** La agrupación de capas, también conocida como submuestreo, realiza una reducción de dimensionalidad; esto reduce la cantidad de parámetros en la entrada. De manera similar a una capa convolucional, la operación

de agrupación barre un filtro sobre toda la imagen, pero la diferencia es que este filtro no tiene pesos.

- **Capa totalmente conectada:** Después de que las capas de convolución y agrupación han "digerido" la información, extrayendo las características más relevantes de, por ejemplo, una imagen, esos hallazgos se compactan, como si se resumieran en una lista. Luego, esa lista de información "resumida" pasa a unas capas finales que funcionan de forma muy parecida a cómo nuestro cerebro integra diferentes piezas de un rompecabezas. Es en este punto donde la red toma todas esas características de alto nivel y las usa para aprender las conexiones sutiles y no tan obvias entre ellas, permitiéndo finalmente tomar una decisión, como identificar de qué se trata la imagen o predecir un resultado.

**Figura 3**

*Red neuronal convolucional (CNN) con múltiples capas de convolución y agrupación*



*Nota:* **Tomado de:** (Sarker, 2021)

En la Figura 3, la Red Neuronal Convolucional (CNN) representa una evolución del diseño de las Redes Neuronales Artificiales (ANN) convencionales, incorporando capas convolucionales, capas de agrupación y capas completamente conectadas (LeCun et al., 1998).

### *2.2.2. Redes neuronales profundas preentrenadas*

En el ámbito de la inteligencia artificial, es posible utilizar una red neuronal preentrenada, lo que significa que ya ha sido configurada y entrenada para una función específica en un entorno diferente. Estos modelos preentrenados son versátiles y pueden aplicarse para tareas como la clasificación, la extracción o transferencia de características, entre otras. Al seleccionar una de estas redes para resolver un problema, es fundamental considerar sus características distintivas, siendo la precisión, la velocidad y el tamaño de la red los factores más importantes a evaluar. Una red neuronal considerada eficiente se caracteriza por ofrecer una alta precisión y una rápida ejecución. (*Redes neuronales profundas preentrenadas* - MATLAB & Simulink - MathWorks España, s. f.).

Algunos ejemplos de redes neuronales preentrenadas son:

- Darknet-19
- EfficientNetV2
- RegNet

- Resnet
- alexnet
- MobileNet

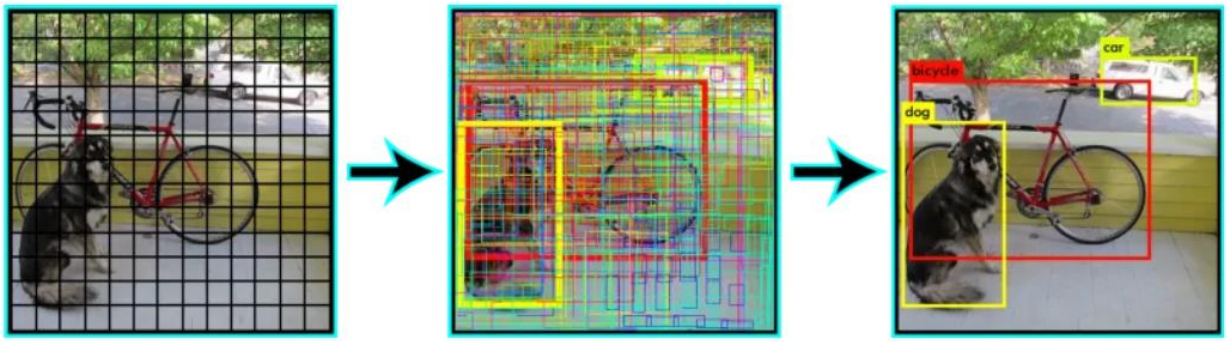
Los modelos de aprendizaje profundo basados en YOLO (You Only Look Once) son una evolución en la detección de objetos, destacando por su habilidad para identificar elementos en tiempo real con gran exactitud (Supeshala, 2021). Introducido en 2016, YOLO se convirtió en un hito en la investigación gracias a que es uno de los modelos de reconocimiento de objetos más prominentes. Emplea una red de detección de objetos de una sola etapa, lo que le permite una alta eficiencia. Este modelo utiliza una Red Neuronal Convolucional (CNN) de aprendizaje profundo para generar proyecciones, las cuales son interpretadas por el detector de objetos para crear los cuadros delimitadores.

Para ejecutar la detección, el proceso inicial implica segmentar la imagen en una cuadrícula de SxS como se muestra en la Figura 4.a. Dentro de cada una de estas celdas, se anticipan N posibles "bounding boxes" o cuadros de anclaje, observado en la Figura 4.b. Estos cuadros se generan empleando el algoritmo K-Means, el cual agrupa los valores de ancho y alto de todos los cuadros previamente etiquetados en el conjunto de datos de entrenamiento. Adicionalmente, se determina el nivel de confianza asociado a cada uno de estos cuadros. Esto resulta en un cálculo de SxSxN cajas distintas, de las cuales la mayoría presentan un nivel de confianza muy bajo. Posteriormente, se eliminan las cajas cuya certidumbre se encuentra por debajo de un umbral predefinido. Finalmente, a las cajas restantes se les aplica un procedimiento de "supresión no máxima" (*non-max suppression*), cuyo propósito es erradicar detecciones duplicadas del mismo

objeto, conservando únicamente la predicción más precisa como se observa en la Figura 4.c. (Getting Started with YOLO v2 - MATLAB & Simulink, s. f.).

**Figura 4**

*Detección de objetos basados en YOLO*




**(a)** **(b)** **(c)**

*Nota:* **Tomado de:** (*Detección De Objetos Con YOLO*, 2018)

Algunos principios fundamentales que facilitarán la comprensión de elementos relevantes al entrenar un modelo son:

**Conjunto de datos (Dataset):** es una colección de información histórica o existente. Esta colección sirve como la base a partir de la cual un algoritmo de aprendizaje automático es entrenado para que pueda identificar patrones y aprender a tomar decisiones o hacer predicciones. Es la materia prima con la que trabaja el modelo.

**Etiquetas (Labels):** En el contexto del aprendizaje automático, las etiquetas (o labels) son atributos o valores que se asignan a los datos sin procesar, como imágenes, textos o videos. Este proceso de etiquetado implica añadir información significativa y contextual a los datos, lo cual es

fundamental para que los modelos de aprendizaje automático puedan aprender de ellos, especialmente en el aprendizaje supervisado. Por ejemplo, en una imagen, la etiqueta podría ser "gato" o "perro".

**Conjunto de Entrenamiento (Training Set):** Cuando se tiene un *dataset* completo, es fundamental dividirlo en subconjuntos. El conjunto de entrenamiento es la porción principal de datos que se utiliza directamente para enseñarle al modelo, permitiéndole aprender los patrones, relaciones y características inherentes a los datos. Es la información con la que la red neuronal práctica y ajusta sus parámetros.

**Conjunto de Validación (Validation Set):** Subconjunto de datos no vistos durante el entrenamiento, usado para evaluar el rendimiento del modelo y detectar sobreajuste.

**Tasa de Aprendizaje (Learning Rate):** Hiperparámetro que define el tamaño de los pasos que el algoritmo de optimización da al ajustar los pesos de la red.

**Función de Pérdida (Loss Function):** Métrica que indica qué tan bien rinde el modelo; un valor alto significa mal desempeño y uno bajo, buen trabajo.

**Época (Epoch):** Una iteración completa donde todo el conjunto de datos de entrenamiento pasa una vez por la red neuronal para que aprenda.

**Sobreajuste (Overfitting):** El modelo aprende demasiado los datos de entrenamiento, memorizando el ruido, lo que lo hace ineficaz con datos nuevos.

**Subajuste (Underfitting):** El modelo no aprende lo básico de los datos de entrenamiento, resultando en un rendimiento pobre en todas las predicciones.

### 2.3. Framework

Es un conjunto de herramientas, librerías y componentes ya listos que te dan una base para construir programas o sistemas de manera rápida y eficiente. Los frameworks generalmente son extensibles, lo que implica que se puede añadir o alterar su funcionalidad de acuerdo a tus requerimientos particulares. Algunos ejemplos de framework son:

- **PyTorch:** Originalmente desarrollado por Facebook AI Research y actualmente administrado por la Fundación PyTorch, este es un marco de aprendizaje profundo de código abierto que emplea la biblioteca back-end de Torch y una API avanzada en Python. Facilita la formación de diversas redes neuronales, que van desde las más sencillas hasta las más sofisticadas, empleadas en labores como la visión artificial y el procesamiento de lenguaje natural. Se emplea extensamente en la investigación y evolución de la inteligencia artificial. (*What Is PyTorch?*, 2023)

- **Tensorflow:** Se trata de una plataforma de aprendizaje automático de alta intensidad, de código abierto, creada inicialmente por Google Brain con el objetivo de investigar y desarrollar tecnologías de redes neuronales. Ofrece un ecosistema integral de herramientas y bibliotecas para investigadores y programadores, facilitando la elaboración, puesta en marcha y conservación de aplicaciones fundamentadas en el aprendizaje automático de forma eficaz. Proporciona APIs estables en Python y C++, además de soporte para otros

lenguajes, lo que lo convierte en flexible y ajustable a diversas necesidades de desarrollo. (*Tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone*, n.d.)

- **Darknet:** Se trata de un marco de red neuronal de código abierto, creado en C y CUDA, que proporciona rapidez y sencillez en la instalación. Es compatible con el cálculo en CPU y GPU, lo que lo convierte en flexible para una diversidad de aplicaciones de aprendizaje automático y visión computacional. Es posible acceder a su código fuente en GitHub y examinar sus habilidades extra para entender de manera más profunda su potencial y usos. *Darknet: Open Source Neural Networks in C. (2013-2016).* (Redmon, n.d.)

- **Keras:** Keras es una API de machine learning que proporciona una experiencia de desarrollo ágil, sofisticada y sencilla de conservar. Se centra en la rapidez de depuración, la capacidad de lectura del código y la sencillez de implementación, facilitando a los programadores la creación de modelos de aprendizaje profundo más eficaces y escalables. Keras está concebido para ser accesible para las personas, con una API sencilla y consistente, documentación exhaustiva y avisos de fallos claros, lo que disminuye la carga cognitiva y simplifica el proceso de desarrollo. (*Keras*, n.d.)

### 2.4. Integración de IA en sistemas embebidos

La integración de IA en sistemas embebidos se refiere a la capacidad de ejecutar modelos de inteligencia artificial y aprendizaje automático directamente en dispositivos de hardware con recursos limitados, como microcontroladores, sensores o pequeñas computadoras de placa única.

Este enfoque, a menudo denominado "IA en el borde" (Edge AI) o TinyML, busca trasladar la capacidad de procesamiento inteligente lo más cerca posible de la fuente de los datos. Para lograrlo, los modelos de IA son optimizados y adaptados para funcionar eficientemente bajo las restricciones de memoria, capacidad de procesamiento y consumo de energía inherentes a estos sistemas embebidos.

La Inteligencia Artificial en los sistemas embebidos puede ser utilizada en varias áreas, tales como la visión computacional para la identificación de objetos, el tratamiento del lenguaje natural para la interacción con el usuario, la toma de decisiones independientes en tiempo real, entre otras. Estas habilidades inteligentes posibilitan que los sistemas embebidos brindan funciones sofisticadas, incrementen la automatización y mejoren su desempeño.

### 2.5. Detección de objetos

La detección de objetos es un método de visión computacional que busca identificar un objeto concreto en una serie de imágenes o cuadros de video a través del tiempo, una vez que este objeto ha sido detectado inicialmente. En contraste con la detección de objetos, que determina la presencia y localización de un objeto en un instante específico, el seguimiento se enfoca en preservar la identidad de dicho objeto mientras se desplaza o altera su apariencia en fotogramas consecutivos.

Este proceso se lleva a cabo típicamente en dos fases:

1. **Detección inicial:** En el primer cuadro o cuando un nuevo objeto aparece, se utiliza un algoritmo de detección de objetos (como YOLOv11) para identificar el objeto de interés.

2. **Asociación y predicción:** En los cuadros siguientes, el sistema predice la posible ubicación futura del objeto y busca su coincidencia con nuevas detecciones. Si el objeto se pierde brevemente, el algoritmo puede intentar re-identificarlo cuando reaparezca. Esto implica asociar la misma identidad al objeto a través de diferentes momentos en el video.

### 2.6. PSNR (Relación Señal a Ruido-Máxima) y SSIM (Índice de Similitud Estructural)

En el análisis de la calidad de imágenes, dos de las métricas más utilizadas son la Relación de Señal a Ruido-Máixima (PSNR, por sus siglas en inglés) y el Índice de Similitud Estructural (SSIM, por sus siglas en inglés). Ambas permiten comparar la similitud entre una imagen de referencia y otra que ha sufrido algún tipo de degradación o defecto, pero desde enfoques distintos: el primero basado en la magnitud del error numérico y el segundo en la percepción visual humana.

El cálculo de PSNR parte del Error Cuadrático Medio (MSE), que mide la diferencia promedio de intensidad entre los píxeles de la imagen original $I(x, y)$ y los de la imagen evaluada $I'(x, y)$:

$$MSE = \frac{1}{MN} \sum_{Y=1}^{M} \sum_{X=1}^{N} [I(x, y) - I'(x, y)]^2 \quad (1)$$

Con este valor se determina el PSNR como:

$$PSNR = 10 \cdot \ log_{10}(\frac{(max_I)^2}{MSE}) \tag{2}$$

Donde $max_I$ es la intensidad máxima de los píxeles (255 para imágenes de 8 bits). Valores altos de PSNR indican menor distorsión y, por tanto, mayor similitud con la imagen de referencia.

Por otro lado, la métrica SSIM incorpora factores perceptuales, evaluando luminancia, contraste y estructura. Se define como:

$$SSIM_{(x,y)} = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x{}^2 + \mu_y{}^2 + c_1) \ \ (\sigma_x{}^2 + \sigma_y{}^2 + c_2)} \tag{3}$$

donde:

- $\mu_x\mu_y$: medias de las intensidades de las imágenes.

$$\mu_x = \frac{1}{T}\sum_{i=1}^{T} \quad x_i \tag{4}$$

$$\mu_y = \frac{1}{T}\sum_{i=1}^{T} \quad y_i \tag{5}$$

- $\sigma_x{}^2, \sigma_y{}^2$: varianzas de cada imagen.

$$\sigma_{\boldsymbol{x}}{}^{\mathbf{2}} = \frac{\mathbf{1}}{\boldsymbol{T}-\mathbf{1}} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{T}} (\boldsymbol{x_i} - \underline{\boldsymbol{x}})^{\mathbf{2}} \tag{6}$$

$$\sigma_y{}^2 = \frac{1}{T-1} \sum_{i=1}^{T} (y_i - \underline{y})^2 \tag{7}$$

- $\sigma_{xy}$: covarianza entre ambas imágenes.

$$\sigma_{xy}{}^2 = \frac{1}{T-1} \sum_{i=1}^{T} (x_i - \underline{x})(y_i - \underline{y}) \tag{8}$$

- $c_1, c_2$ : Constantes de estabilidad.

El valor de SSIM varía entre 0 y 1, siendo 1 la similitud estructural perfecta. Mientras PSNR mide diferencias absolutas en el dominio de píxeles, SSIM captura la percepción visual de las diferencias, por lo que ambos indicadores se complementan en la validación de sistemas de visión artificial. (León-Batallas et al., 2020).

# 3. Metodología

Tras llevar a cabo el estudio correspondiente acerca de los conceptos y herramientas para la identificación de objetos mediante redes neuronales, se realizaron una serie de etapas para su desarrollo y puesta en marcha. A continuación, se detallan los pasos de este procedimiento:

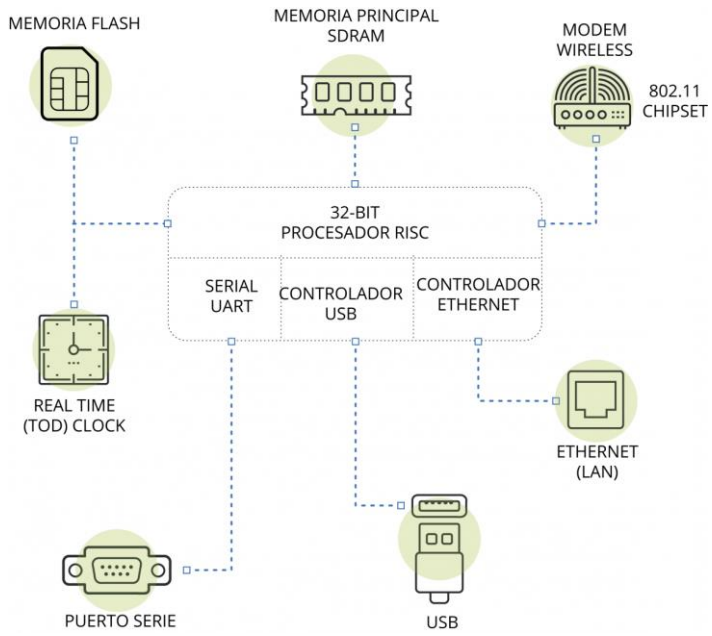
## 3.1. Selección del sistema embebido

A diferencia de los sistemas de propósito general, los sistemas embebidos están concebidos para llevar a cabo funciones concretas con recursos de hardware restringidos. Esta particularidad les facilita ser eficaces, fiables y asequibles. Su diseño perfeccionado para una función concreta necesita un entendimiento detallado de la aplicación y de métodos de programación especiales. Los sistemas embebidos están presentes en una extensa variedad de aparatos y constituyen un componente crucial de la vida contemporánea. (Jaramillo & Potosí, 2017).

Con el paso del tiempo, estos sistemas han avanzado y han jugado un rol crucial en la revolución del Internet de las Cosas (IoT). Los sistemas embebidos suelen incorporar una CPU, memoria RAM y ROM, y se vinculan a través de buses de corriente para compartir información con el ambiente mediante sensores y actuadores. *(Innovación Digital 360, 2023)*

**Figura 5**

*Descripción de un sistema embebido*




*Nota:* **Tomado de:** (*INCIBE*, 2018)

Tomando en cuenta las particularidades de un sistema embebido, se examinaron cuatro opciones de implementación sugeridas y se eligió el modelo apropiado para su implementación.

Las opciones consideradas son:

- Jetson Nano, una computadora de placa única de alto rendimiento para la inteligencia artificial (IA) (*Jetson Nano*, n.d.)
- Maix-II-Dock, una plataforma de desarrollo asequible y versátil para el aprendizaje automático e IoT *Maix-II-Dock(M2dock) introduction. (s.f).*
- AMB82 MINI, una placa de desarrollo de bajo costo para el desarrollo de aplicaciones de Internet de las Cosas (IoT) e Inteligencia Artificial (IA) *Ameba ARDUINO: Getting Started with AMB82 MINI (RTL8735B) (s.f).*

- Raspberry Pi 4 Model B, una excelente opción para una amplia gama de proyectos, desde el aprendizaje de programación hasta la creación de dispositivos IoT (*Raspberry Pi 4*, 2019)

Las propiedades fundamentales de los sistemas se encuentran estructuradas en la Tabla 1, un cuadro comparativo que facilita una mejor representación de los datos.

**Tabla 1**

*Propiedades fundamentales de Jetson Nano, Maix-II-Dock, AMB82 Mini y Raspberry Pi 4*

| | Sistemas embebidos | | | |
|---|---|---|---|---|
| Características | Jetson Nano | Maix-II-Dock | AMB82 Mini | Raspberry Pi 4 |
| Procesador | Quad-core ARM® Cortex®-A57 MPCore processor | ARM Single-core Cortex-A7 800-1000MHz | ARMv8M at 500MHz | Quad core 64-bit ARM-Cortex A72 running at 1.5GHz |
| CPU | NVIDIA Maxwell™ architecture with 128 NVIDIA CUDA® cores 0.5 TFLOPs (FP16) | No | Broadcom BCM2711 VideoCore VI | No |
| NPU | No | 0.2 TOPS | Intelligent Engine | No |

| | | | | |
|---|---|---|---|---|
| | | | @ 0.4 TOPS | |
| Almacenamiento | 16 GB eMMC 5.1 Flash | Choosable 16M flash(Blank default) | Selectable micro SD card | Selectable micro SD card |
| Memoria | 2GB LPDDR4 | SIP 64MB DDR2 | 512KB RAM, support DDR2 128MB | 4GB LPDDR4-3200 SDRAM. |
| Wi-Fi | 1<br>Wi-Fi requires external chip | Wi-Fi module*1 | 802.11 a/b/g/n 1×1, dual Wi-Fi 2.4GHz/5GHz Simple Wi-Fi Setup | 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless |
| Codificador de Video | 250 MP/sec<br>1x 4K @ 30 (HEVC)<br>2x 1080p @ 60 (HEVC)<br>4x 1080p @ 30 (HEVC) | H.264, up to 1080p@30fps<br>H265, up to 1080p@30fps<br>JPEG, up to 1080p@30fps | Max 5-megapixel resolution for H.264/H.265 encoding up to 1080p@30fps, 720p@30fps | H.265 (4kp60 decode), H.264 (1080p60 decode, 1080p30 encode) |
| Pantalla | HDMI 2.0 or DP1.2 \| eDP 1.4 \| DSI (1 x2) 2 simultaneous | 8bit MCU LCD, can use other screen by convert board | LCD interface \| Web graphical user interface | 2 micro-HDMI® ports (up to 4kp60 supported) |
| Cámara | 12 lanes (3x4 or 4x2) MIPI CSI-2 DPHY 1.1 (18 | 2lane MIPI, Up to 1080P@60fps | JXF37<br>1920×1080<br>CMOS Full HD | 5 Mpx<br>1080P@60fps |

| | | | | |
|---|---|---|---|---|
| | Gbps) | | image sensor with wide view angel FOV 128º optical lens | |
| ADC | No | 1-ch 6bit LRADC for key | Sí | No |
| Audio | No | LINEOUTP + MICIN1P/N | ADC/DAC/I2S | Sí |
| Ethernet | 10/100/1000 BASE-T Ethernet | 10/100 Mbit/s Ethernet port with RMII interface | No | Gigabit Ethernet |
| SPI | SÍ | SPI x2 (SPI0, SPI1) | SÍ | SÍ |
| Consumo de energía | 5W, 10W<br>5V \| 1A, 2A | 5W<br>5V \| 1A | 5W<br>5V \| 1A | ~6W<br>5V \| 3A |
| Precio | $1’041.802 | $216.960 | $123.780 | $321.300 |

*Nota:* **Tomado de:** *Maix-II-Dock(M2dock) introduction. (s.f), Jetson Nano (s.f)., Raspberry Pi 4 (s.f) & Ameba ARDUINO: Getting Starde with AMB82 MINI (RTL8735B) (s.f.)*

Previo a la construcción del conjunto de datos y a la selección definitiva del modelo de aprendizaje automático, se evaluó la posibilidad de implementar el sistema en la tarjeta AMB82 Mini, utilizando los modelos de detección soportados en su entorno: YOLOv3-Tiny, YOLOv4-Tiny y YOLOv7-Tiny. Sin embargo, las métricas obtenidas en pruebas iniciales —particularmente

mAP, precisión y recall— resultaron considerablemente bajas como se aprecia en la Figura 6, lo que evidenció limitaciones importantes para el objetivo del proyecto. Dado que la meta principal era lograr la detección confiable de defectos de impresión con porcentajes de precisión elevados, se descartó el uso de la AMB82 Mini y sus variantes "Tiny", ya que estas priorizan la reducción de complejidad computacional a costa de la capacidad de detección. En consecuencia, se optó por explorar arquitecturas y plataformas con mejor desempeño, lo que posteriormente condujo a la selección de YOLOv11 en Raspberry Pi 4 como la alternativa más adecuada.

**Tabla 2**

*Desempeño del modelo con YOLOv4-tiny*

| Class_id | Name | Average Precision |
|---|---|---|
| 0 | corrimientos | 25.03% |
| 1 | fallas | 5.06% |
| 2 | manchas | 19.01% |
| Recall | | 11% |
| mAP | | 16.36% |

*Nota: Entrenamiento del modelo YOLOv4-tiny con un conjunto de datos de 560 imágenes.* **Tomado de:** (Google Colab, 2025).

La Raspberry Pi 4 emergió como la solución ideal, ofreciendo un balance superior de características a un precio competitivo. Su procesador Quad-core 64-bit ARM-Cortex A72 a

1.5GHz y sus 4GB de memoria LPDDR4 proporcionan una capacidad de procesamiento sustancialmente mayor que la AMB82 Mini y la Maix-II-Dock, permitiendo la ejecución de la lógica compleja del sistema y la gestión eficiente de los fotogramas de video. Aunque la Raspberry Pi 4 no incluye una NPU dedicada como la Jetson Nano, su CPU más potente es capaz de manejar la inferencia de modelos de IA. Además, cuenta con conectividad Wi-Fi 2.4 GHz y 5.0 GHz IEEE 802.11ac, Gigabit Ethernet, y soporte para cámaras de 5 Mpx a 60 FPS, lo que garantiza una adquisición de datos y comunicación. Esta combinación de potencia, flexibilidad y costo-efectividad hizo de la Raspberry Pi 4 la plataforma embebida más adecuada.

### 3.2. Construcción del conjunto de datos

La construcción del conjunto de datos, partió de la recopilación de imágenes reales y algunas creadas, de impresiones publicitarias que presentaban defectos visuales comunes en entornos de impresión industrial. Estas imágenes fueron capturadas directamente desde productos físicos y diseños digitales mediante una cámara de celular y programas de edición como Photoshop, asegurando un nivel de detalle suficiente para identificar imperfecciones de distintos tipos. El proceso contempló una amplia variedad de defectos visuales, entre ellos:

- Manchas de tinta.
- Corrimientos de color por mala alineación de cabezales o registro.
- Fallas de impresión como líneas interrumpidas o parches sin impresión.

**Tabla 3**

*Recolección del conjunto de datos para el entrenamiento del modelo*

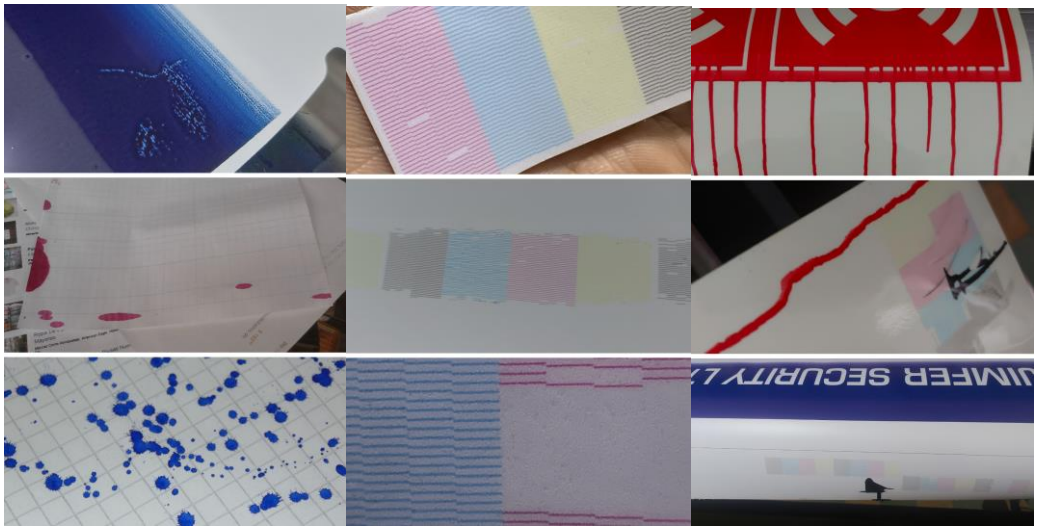
| Conjunto de datos | | |
|---|---|---|
| | | Cantidad |
| Imágenes reales | Fallas | 1.204 |
| | Manchas | 258 |
| | Corrimientos | 180 |
| Imágenes creadas | Manchas | 798 |
| | Corrimientos | 663 |
| | Total | 3.278 |

*Nota: Las imágenes reales se capturaron mediante una cámara de celular en un ambiente de laboratorio y las imágenes creadas se realizaron con programa de edición como Photoshop.* **Tomado de:** (Valest, 2025).

**Figura 6**

*Defectos visuales comunes en impresiones publicitarias*

**MANCHAS** **FALLAS** **CORRIMIENTOS**



**(a)** **(b)** **(c)**

Se eligieron manchas, corrimientos y fallas de impresión porque son los defectos más frecuentes durante la impresión en materiales publicitarios (banners, vinilos, lienzos, papel). Cada falla se presenta de una manera distinta: las manchas se observan como depósitos localizados de tinta fuera de lugar (salpicaduras, sangrados o contaminación) que alteran el tono y la uniformidad en el material; los corrimientos aparecen como franjas de color desplazadas o efectos de doble imagen; y las fallas de impresión comprenden ausencias o interrupciones de tinta —bandas, líneas o parches no impresos— producidas por boquillas obstruidas o falta de tinta.

La diversidad en el tipo de defectos, fondo, color, y condiciones de iluminación permitió construir un conjunto de datos representativo de los escenarios reales del entorno industrial. Cada imagen fue guardada en Google Drive. (Valest, 2025).[1]

### *3.2.1. Procesamiento y etiquetado de datos*

Una vez recopiladas las imágenes, se utilizó la plataforma Roboflow para la gestión completa del dataset. Este proceso incluyó las siguientes etapas:

**Etiquetado manual de defectos:** Cada imagen fue inspeccionada visualmente y anotada mediante cuadros delimitadores (bounding boxes) que señalaban la ubicación exacta del defecto presente. Las etiquetas asignadas incluyeron clases como “manchas”, “corrimientos” y “fallas”, dependiendo del tipo de imperfección observada.

**Aumento de datos (Data Augmentation):** Para incrementar la robustez del modelo frente a variaciones visuales, se aplicaron filtros de procesamiento a las imágenes a través de Roboflow como se muestra en la Tabla 4. Entre ellos se destacan: Rotaciones aleatorias, volteo horizontal/vertical, ajuste de brillo, contraste, desenfoque, ruido digital y recortes parciales (crop).

[1] El conjunto de datos al que se hace referencia es el DATASET COPUDI.

**Tabla 4**

*Filtros de procesamiento para las imágenes*

| Preprocessing | |
|---|---|
| Auto-Orient | Applied |
| Resize | 640x640 |
| **Augmentations** | |
| Flip | Horizontal, Vertical |
| 90° Rotate | Clockwise, Counter-Clockwise, Upside Down |
| Crop | 0% Minimum Zoom, 5% Maximum Zoom |
| Grayscale | Apply to 10% of images |
| Hue | Between -10° and +10° |
| Saturation | Between -25% and +25% |
| Brightness | Between -15% and +15% |
| Exposure | Between -10% and +10% |
| Blur | Up to 0.4px |
| Bounding Box: Brightness | -2% and +2% |
| Total de imágenes generadas | 14.254 |

*Nota:* Se muestra el conjunto de datos final, luego del procesamiento y aumento, estuvo compuesto por un total de 14.254 imágenes, asegurando una cobertura adecuada para la tarea de detección **Tomado de:** *(Roboflow, 2025).*

### 3.3. Selección del modelo de aprendizaje automático

La elección se basó en el requisito fundamental de proporcionar información espacial precisa, lo que descartó los modelos de clasificación tradicional —dado que éstos sólo ofrecen una etiqueta general a nivel de imagen como defectuosa o no defectuosa— a favor de un modelo de detección de objetos como lo es YOLO. Este enfoque es importante para la automatización, ya que realiza dos tareas esenciales de forma simultánea: la localización de la posición exacta del defecto mediante coordenadas (el recuadro delimitador) y la clasificación del tipo específico de error (mancha, corrimiento o falla).

En la Tabla 5 se muestran opciones disponibles para la detección de objetos, el cual se determinó que YOLOv11 Large era el candidato más prometedor para la implementación.

**Tabla 5**

*Modelos de entrenamiento*

| Modelo | Tamaño (Pixel) | mAP 50-95 (COCO) | Velocidad CPU (ms) | Velocidad T4 (ms) | Parámetros (M) | FLOPs (B) |
|---|---|---|---|---|---|---|
| YOLO11n | 640 | 39.5 | 56.1 | 1.5 | 2.6 | 6.5 |
| YOLO11s | 640 | 47.0 | 90.0 | 2.5 | 9.4 | 21.5 |
| YOLO11m | 640 | 51.5 | 183.2 | 4.7 | 20.1 | 68.0 |
| YOLO11l | 640 | 53.4 | 238.6 | 6.2 | 25.3 | 86.9 |
| YOLO11x | 640 | 54.7 | 462.8 | 11.3 | 56.9 | 194.9 |

*Nota:* Las medidas de mAP reportadas en esta tabla muestran los valores de un conjunto de datos de referencia obtenidos por los desarrolladores del modelo sobre la tarea de detección de objetos

dentro del conjunto de datos COCO (Common Objects in Context). **Tomado de:** *(Ultralytics, 2025).*

YOLOv11 Large fue elegido por su capacidad demostrada para ofrecer la máxima precisión dentro de la familia YOLOv11, lo cual es fundamental para el objetivo de un control de calidad riguroso. Su rendimiento justifica la inversión en recursos computacionales, asegurando que el sistema pueda identificar los defectos. Las especificaciones que se usan para entrenar un modelo en YOLOv11l son:

**Tabla 6**

*Parámetros de entrenamiento del modelo*

| Parámetros | Valor |
|---|---|
| Dimensión de imagen | 640 |
| Época | 100 |

*Nota:* **Tomado de:** *(Ultralytics, 2025)*

Durante la fase de selección del framework se evaluaron PyTorch, TensorFlow, Keras y Darknet, considerando tanto la compatibilidad con YOLOv11 como las necesidades del proyecto en términos de rendimiento y despliegue. Inicialmente se contempló el uso de Darknet, por ser el framework original de YOLO; sin embargo, su falta de actualizaciones, soporte limitado y escasa optimización para hardware actual lo descartaron como opción viable. Keras fue analizado por su

sencillez en el desarrollo de prototipos, pero se evidenció que, al depender del backend de TensorFlow**,** es decir, el motor interno que ejecuta realmente las operaciones matemáticas y de cómputo de bajo nivel no ofrecía la flexibilidad necesaria para ajustar el modelo. TensorFlow, ampliamente adoptado en la industria, representaba una alternativa sólida; no obstante, su integración con el modelo era menos directa, requiriendo conversiones adicionales y aumentando la complejidad del flujo de trabajo. Finalmente, se seleccionó PyTorch debido a su arquitectura dinámica, su comunidad activa de investigación y, principalmente, porque cuenta con soporte oficial para YOLOv11. Esto facilitó el entrenamiento en GPU y permitió una transición más sencilla hacia formatos optimizados para la Raspberry Pi 4. En consecuencia, PyTorch ofreció el mejor equilibrio entre facilidad de implementación y eficiencia, superando en la práctica a las demás alternativas consideradas.

### 3.4. Entrenamiento del Modelo en Roboflow

La fase de entrenamiento del modelo YOLOv11 Large, esencial para su alto rendimiento, se llevó a cabo sobre la plataforma de Roboflow. Esta elección fue estratégica, dada la significativa complejidad computacional del modelo y las limitaciones de recursos (GPU y tiempo de ejecución) encontradas en entornos locales de desarrollo. El proceso de entrenamiento se ejecutó después de que el conjunto total de 14.254 imágenes fuera rigurosamente dividido en los subconjuntos de Entrenamiento, Validación y Prueba, según el detalle proporcionado en la Tabla 7**.** Gracias a la división ya establecida, Roboflow pudo proporcionar un entorno optimizado, utilizando hardware en la nube (como las GPU NVIDIA A100), lo cual fue importante para garantizar la eficiencia y validez del entrenamiento.

**Tabla 7**

*Distribución de las imágenes en Roboflow*

| Conjunto de datos | |
|---|---|
| | Cantidad |
| Entrenamiento | 13.720 imágenes |
| Validación | 520 imágenes |
| Pruebas | 14 imágenes |
| Total | 14.254 imágenes |

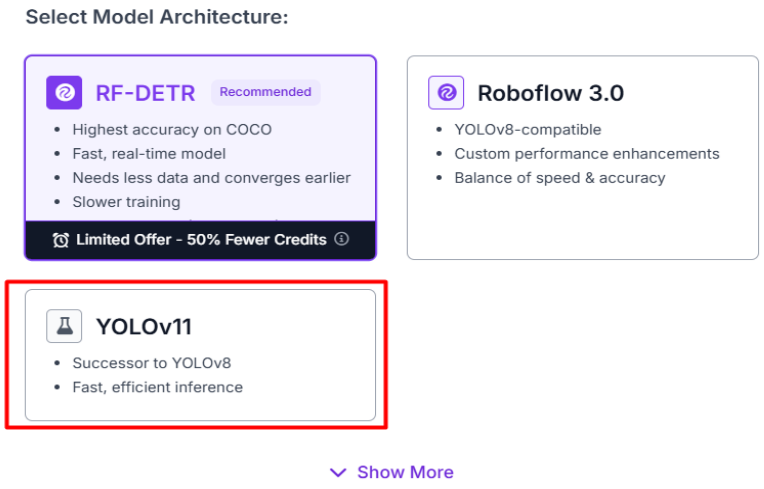
*Nota:* La división fue realizada automáticamente en la plataforma Roboflow, después de aplicar las técnicas de aumento y preprocesamiento. **Tomado de:** *(Roboflow, 2025).*

El proceso de entrenamiento incluyó los siguientes pasos:

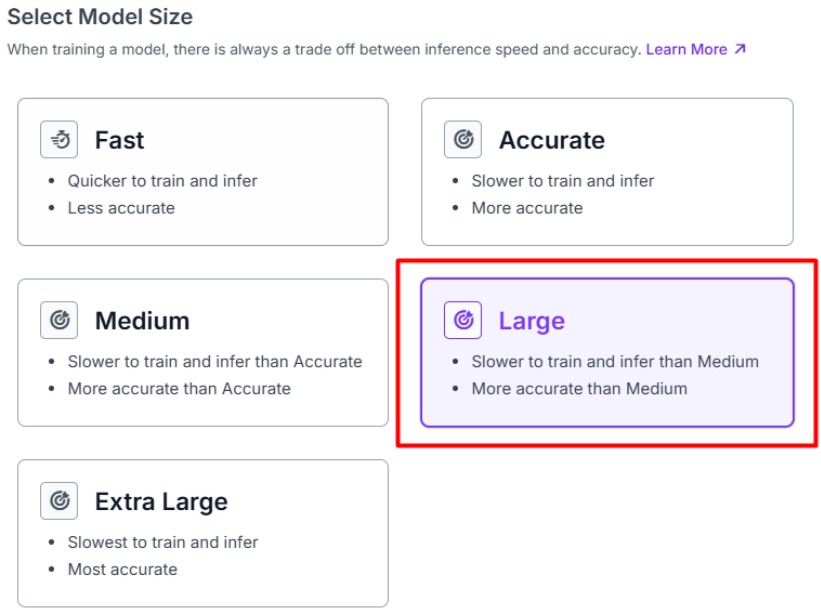
- **Carga y Versión del Dataset:** El dataset personalizado, ya pre-procesado y aumentado, fue subido y versionado en la plataforma Roboflow, asegurando la consistencia de los datos de entrada.
- **Configuración del Modelo:** Como se muestra en la Figura 7.a., seleccionó el modelo base YOLOv11 y en la Figura 7.b. se evidencia la selección del tamaño del modelo que es la versión Large y por defecto los hiperparámetros de entrenamiento (ej., número de épocas) según las recomendaciones de Roboflow.

**Figura 7**

*Selección del modelo de aprendizaje en Roboflow*




**(a)**



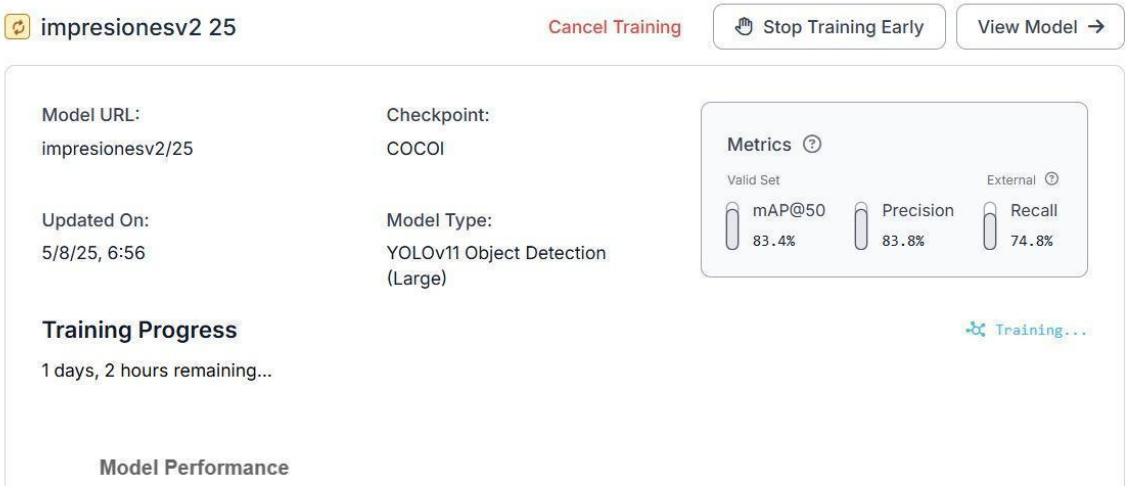

**(b)**

*Nota:* **Tomado de:** *(Roboflow, 2025).*

En la Figura 8, el entrenamiento se inició en los servidores de Roboflow. Se realizó un seguimiento continuo de su progreso a través de las herramientas de visualización integradas en la plataforma, las cuales mostraban en tiempo real las métricas de precisión media (mAP) en el conjunto de validación.

**Figura 8**

*Entrenamiento del modelo de aprendizaje en la plataforma Roboflow*




*Nota:* **Tomado de:** *(Roboflow, 2025).*

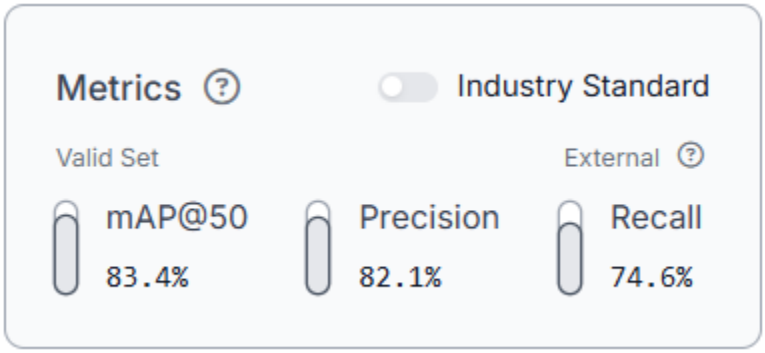
Al finalizar este proceso, Roboflow generó los archivos de pesos del modelo (.pt), encapsulando el conocimiento adquirido por la IA para la detección precisa de manchas, fallas y corrimientos. Adicionalmente, se obtuvieron las métricas finales al completar el entrenamiento las cuales fueron las siguientes:

- mAP@50: 83.4%

- Precisión: 82.1%
- Recall: 74.6%

**Figura 9**

*Métricas obtenidas del entrenamiento del YOLOv11 Large*



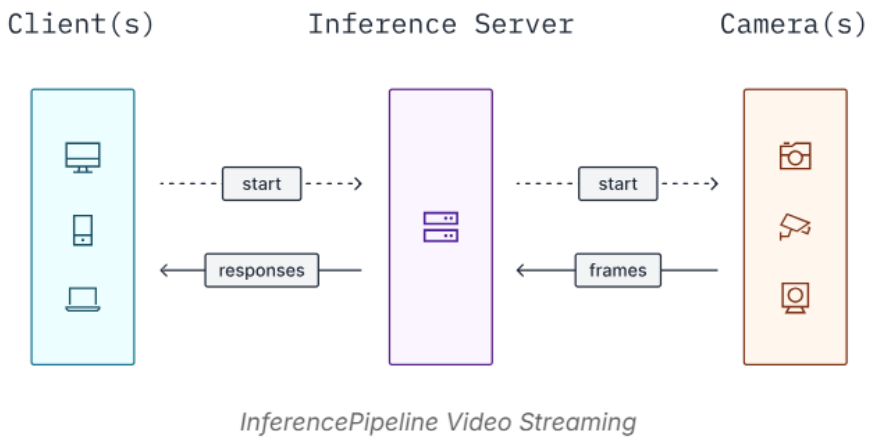

*Nota:* **Tomado de:** *(Roboflow, 2025).*

### 3.5. Roboflow Inference

Mediante Roboflow Inference, permitió ejecutar modelos de visión por computadora entrenados (como los de detección de objetos, clasificación o segmentación) de manera sencilla. En esencia, actúa como una API que te permite enviar imágenes o videos a los servidores de Roboflow, donde los modelos son ejecutados en hardware optimizado (generalmente GPUs en la nube), y luego los resultados de la inferencia son devueltos a tu aplicación. *(Roboflow Inference, s.f.).*

**Figura 10**

*Transmisión de video por Inference Pipeline*




*Nota:* **Tomado de:** *Architecture. Roboflow Inference (s.f)*

En la Figura 10, la inferencia se realiza mediante la Raspberry Pi 4 que interactúa con la nube de Roboflow, validando la alta eficacia del algoritmo de IA como componente central del sistema.

### *3.5.1. Inferencia en video*

Una vez entrenado y validado, el modelo YOLOv11 (Large) se utiliza para realizar inferencia de detección de objetos en tiempo real a través de video, aprovechando la API de inferencia de Roboflow. Este enfoque es crucial ya que permite que el procesamiento computacional de la inteligencia artificial se ejecute en la infraestructura de la nube de Roboflow, mientras que el cliente local gestiona únicamente la adquisición de video y la visualización de los resultados.

A continuación, se explica el funcionamiento conceptual:

- **Captura de Fotogramas**: Un script Python ejecutado en la Raspberry Pi 4 que utiliza la librería OpenCV para capturar continuamente fotogramas desde una cámara web.
- **Envío a la Nube:** Cada fotograma es enviado, vía conexión a Internet, a la API de inferencia de Roboflow. Para este envío, se utilizan credenciales específicas del proyecto y la versión del modelo entrenado.
- **Procesamiento de IA:** En los servidores de Roboflow, el modelo YOLOv11 (Large) procesa el fotograma recibido, aplicando sus capacidades de detección para generar predicciones (coordenadas de las cajas delimitadoras y etiquetas de clase.
- **Recepción y Visualización**: Las predicciones son devueltas al cliente local, donde el script Python las interpreta y las superpone gráficamente sobre el fotograma original, mostrando las detecciones en vivo.

**Figura 11**

*Script de Python para realizar la inferencia en Roboflow*

```python
# Import the InferencePipeline object
from inference import InferencePipeline
# Import the built in render_boxes sink for visualizing results
from inference.core.interfaces.stream.sinks import render_boxes

# initialize a pipeline object
pipeline = InferencePipeline.init(
    model_id="impresionesv2/25", # Roboflow model to use
     api_key = "APIKEY", #agrega tu api key
    video_reference=0, # Path to video, device id (int, usually 0 for built in webcams), or RTSP stream
url
    on_prediction=render_boxes, # Function to run after each prediction
)
pipeline.start()
pipeline.join()
```

*Nota: la API Key que se utiliza en el código es una API privada generada por Roboflow*. **Tomado de:** *(Roboflow, 2025).*

En el Script de la Figura 11, es necesario instalar las siguientes librerías para el correcto funcionamiento del código en Python:

- opencv-python (la versión del opencv debe coincidir con la versión de Python)
- numpy
- roboflow
- inference

Para la implementación en la Raspberry Pi 4, se cargó un script en Python, que utiliza la librería InferencePipeline para ejecutar el modelo de detección entrenado en Roboflow. El código inicializa el pipeline, carga el modelo YOLOv11l previamente ajustado y establece la captura de video desde la cámara para realizar inferencias en tiempo real, renderizando los resultados en pantalla.

### 3.6. Métricas de desempeño PSNR y SSIM

Con el fin de validar el desempeño del sistema embebido implementado en la detección de fallas de impresión, se utilizaron dos métricas reconocidas en el análisis de calidad de imágenes: PSNR (Peak Signal-to-Noise Ratio) y SSIM (Structural Similarity Index Measure).

Para este proyecto, ambas métricas se calcularon comparando imágenes de impresiones correctas frente a impresiones con defectos. De esta manera, los valores obtenidos permiten

determinar si el sistema embebido logra capturar y diferenciar cuantitativamente los defectos presentes en los materiales impresos.

**Tabla 8**

*Rangos de interpretación de PSNR Y SSIM*

| PSNR | | SSIM | |
|---|---|---|---|
| >40 dB | Imágenes casi idénticas | 0.95 - 1 | Imágenes casi idénticas |
| 30 - 40 dB | Alta similitud, diferencias pequeñas | 0.80 - 0.95 | Alta similitud |
| 20 - 30 dB | Diferencias visibles moderadas | 0.60 - 0.80 | Diferencias visibles moderadas |
| 10 - 20 dB | Diferencias claras y significativas | 0.40 - 0.60 | Diferencias claras y significativas |
| <10 dB | Imágenes distintas | < 0.40 | Imágenes distintas |

*Nota: Las métricas PSNR y SSIM son indicadores de referencia completa. Estos rangos confirman que el subsistema de captura garantiza la fidelidad estructural y el requerido en la imagen de entrada (PSNR en dB, SSIM de 0 a 1).* **Tomado de:** *(Wang, 2004).*

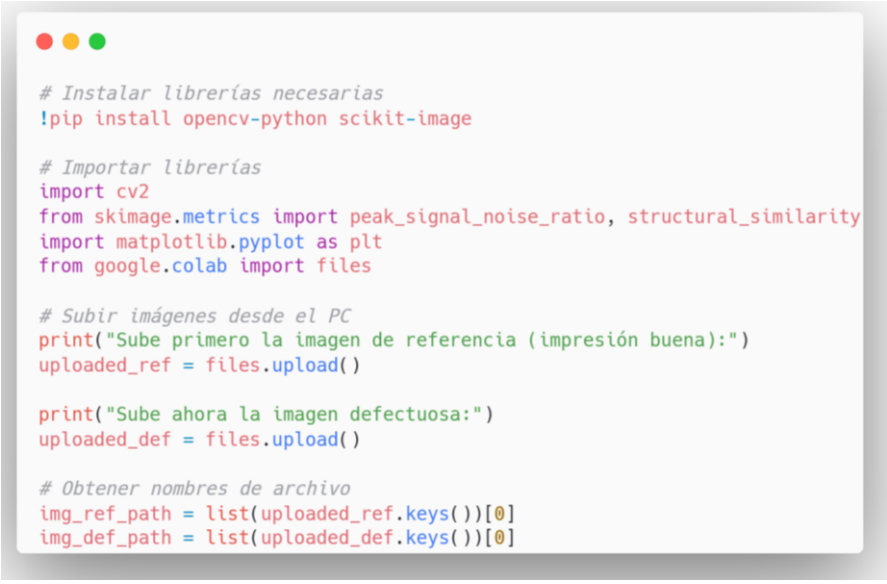
## 4. Resultados

### 4.1 Resultados de Validación del Sistema con PSNR y SSIM

El cálculo de las métricas PSNR y SSIM se realizó en Google Colab mediante un script en Python. El programa permitía cargar una imagen de referencia (impresión buena) y una imagen defectuosa (impresión con fallas), obteniendo de manera automática los valores de ambas métricas y generando como salida un gráfico comparativo con los resultados sobreimpresos.

**Figura 12**

*Código en Python implementado para la validación del sistema embebido*



```python
# Instalar librerías necesarias
!pip install opencv-python scikit-image

# Importar librerías
import cv2
from skimage.metrics import peak_signal_noise_ratio, structural_similarity
import matplotlib.pyplot as plt
from google.colab import files

# Subir imágenes desde el PC
print("Sube primero la imagen de referencia (impresión buena):")
uploaded_ref = files.upload()

print("Sube ahora la imagen defectuosa:")
uploaded_def = files.upload()

# Obtener nombres de archivo
img_ref_path = list(uploaded_ref.keys())[0]
img_def_path = list(uploaded_def.keys())[0]
```

**(a)**

```python
# Cargar imágenes
img_ref = cv2.imread(img_ref_path)
img_def = cv2.imread(img_def_path)

# Redimensionar defectuosa para que coincida con la referencia ===
h, w = img_ref.shape[:2]
img_def_resized = cv2.resize(img_def, (w, h))

# Convertir a escala de grises
ref_gray = cv2.cvtColor(img_ref, cv2.COLOR_BGR2GRAY)
def_gray = cv2.cvtColor(img_def_resized, cv2.COLOR_BGR2GRAY)

# Calcular métricas
psnr_val = peak_signal_noise_ratio(ref_gray, def_gray)
ssim_val, _ = structural_similarity(ref_gray, def_gray, full=True)

# Mostrar imágenes en HD con métricas
plt.rcParams['figure.dpi'] = 200  # aumentar nitidez
fig, axes = plt.subplots(1, 2, figsize=(14, 7))

axes[0].imshow(cv2.cvtColor(img_ref, cv2.COLOR_BGR2RGB))
axes[0].set_title("Impresión Buena", fontsize=12)
axes[0].axis("off")

axes[1].imshow(cv2.cvtColor(img_def_resized, cv2.COLOR_BGR2RGB))
axes[1].set_title("Impresión Defectuosa", fontsize=12)
axes[1].axis("off")

# Agregar los resultados como texto en la figura
plt.figtext(0.5, 0.02, f"PSNR: {psnr_val:.2f} dB   |   SSIM: {ssim_val:.3f}",
            ha="center", fontsize=12, bbox={"facecolor":"lightgray", "alpha":0.7, "pad":5})

plt.show()

# Guardar imagen en HD
fig.savefig("comparacion_metricas_hd.png", dpi=300, bbox_inches='tight')
print("Imagen guardada como comparacion_metricas_hd.png en alta resolución ✅")
```

**(b)**

**Tabla 9**

*Valores obtenidos del PSNR Y SSIM*

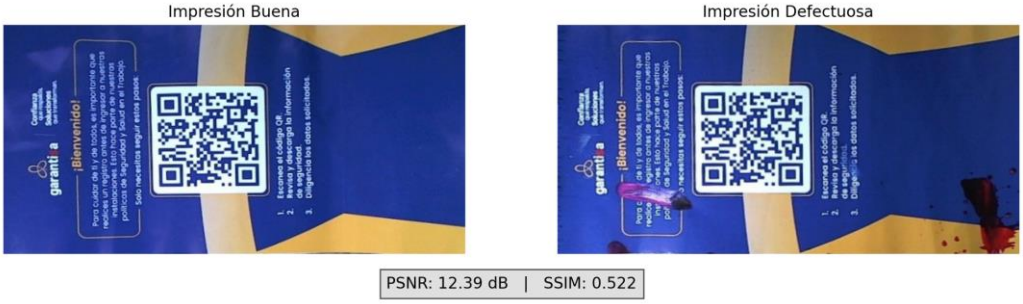
| Imagen a | | Imagen b | |
|---|---|---|---|
| PSNR | inf dB | PSNR | 12.39 dB |
| SSIM | 1.00 | SSIM | 0.258 |

*Nota: En la Figura 12.a. la comparación de dos imágenes idénticas arroja PSNR infinito y SSIM = 1.0, indicando ausencia de diferencias. En la Figura 12.b., la comparación entre una impresión buena y una defectuosa muestra PSNR de 12.39 dB y SSIM de 0.522, reflejando pérdida de calidad por las fallas.* **Tomado de:** *(Wang, 2004).*

**Figura 13**

*Comparación entre impresión buena y defectuosa con los valores de PSNR y SSIM*


**(a)**




**(b)**

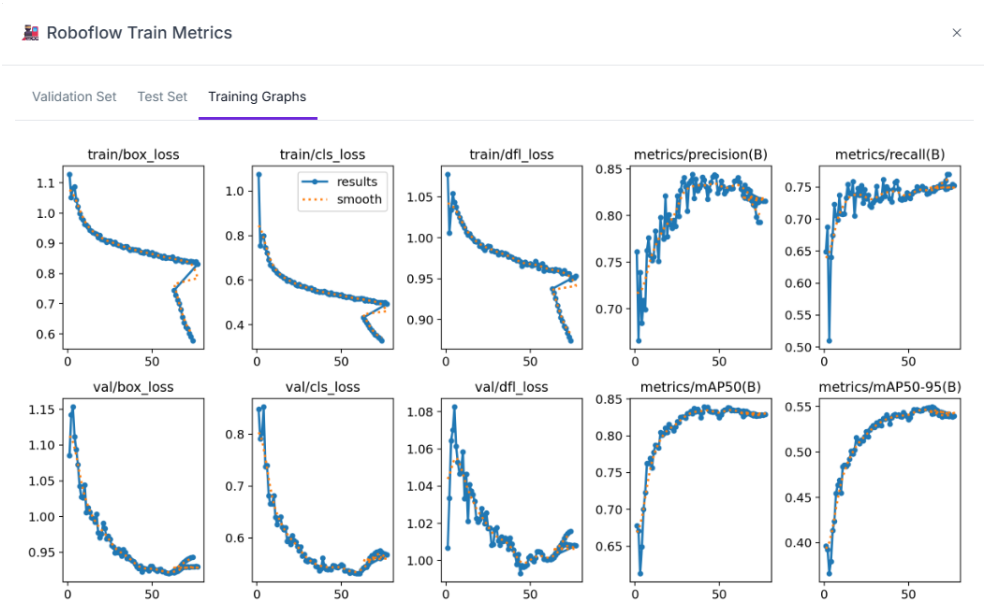
De este modo, se demuestra la viabilidad del enfoque propuesto para evaluar la calidad de las impresiones y diferenciar aquellas que presentan anomalías.

### 4.2. Evaluación del modelo (YOLOv11) y métricas de desempeño

En su versión final, el modelo fue evaluado utilizando un conjunto de validación independiente y las métricas estándar en tareas de detección de objetos. En la Figura 14, se presenta el comportamiento de las métricas más relevantes, como las pérdidas de clasificación, localización y regresión (box loss, cls loss, dfl loss), tanto en el conjunto de entrenamiento como en el de validación.

**Figura 14**

*Gráficas avanzadas de entrenamiento generadas por Roboflow para el modelo YOLOv11l*




*Nota:* **Tomado de:** *Roboflow (2025).*

Las métricas obtenidas fueron las siguientes:

- **mAP@0.5 (Mean Average Precision a IoU ≥ 0.5):** Obtuvo un valor de 83.4%, lo cual indica un excelente nivel de precisión media en la detección de los defectos, considerando una superposición mínima del 50% entre las predicciones y la anotación real. Esta métrica refleja la capacidad del modelo para localizar correctamente los defectos dentro de las imágenes.

- **Precisión (Precision):** Registró un 82.1%, lo que significa que, de todas las predicciones positivas realizadas por el modelo (es decir, regiones marcadas como defectuosas), más del 82% fueron correctas. Este resultado es fundamental para garantizar que el sistema no rechace impresiones sin fallas.

- **Recall (Sensibilidad):** Alcanzó un 74.6%, indicando que el modelo fue capaz de detectar correctamente más del 74% de todos los defectos reales presentes en las imágenes. Aunque menor que la precisión, este valor sigue siendo competitivo y sugiere un equilibrio razonable entre detecciones acertadas y errores por omisión.

### 4.3. Análisis del comportamiento del entrenamiento

Durante el proceso de entrenamiento, se generaron diversas gráficas que permiten evaluar el desempeño del modelo en cada etapa del aprendizaje como se evidencia en la Figura 17. También se observa la evolución progresiva de los indicadores de desempeño como la precisión, el recall y la precisión media (mAP@0.5 y mAP@0.5–0.95), los cuales muestran una tendencia

ascendente y una estabilización adecuada, lo que indica un aprendizaje efectivo y controlado del modelo

### *4.3.1. Pérdidas de entrenamiento y validación*

- **train/box_loss y val/box_loss:** Reflejan la pérdida asociada a la predicción de las cajas delimitadoras (bounding boxes). Se observa una disminución progresiva y sostenida de este valor tanto en entrenamiento como en validación, indicando que el modelo aprendió a ajustar correctamente la ubicación de los defectos en las imágenes.

- **train/cls_loss y val/cls_loss:** Representan la pérdida de clasificación, es decir, cuán bien el modelo asigna la clase correcta a cada objeto detectado. Estas curvas muestran una tendencia descendente con convergencia estable, señal de una buena capacidad del modelo para diferenciar entre los distintos tipos de defectos (manchas, corrimientos, fallas).

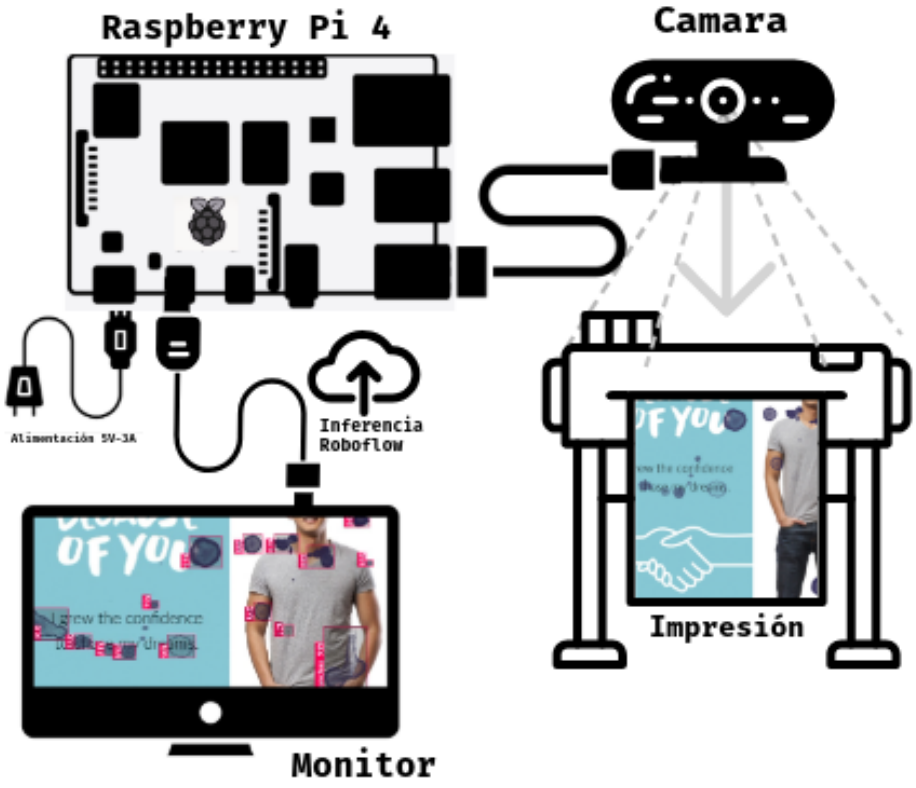
- **train/dfl_loss y val/dfl_loss:** La pérdida DFL (Distribution Focal Loss) se asocia a la precisión de la predicción de las coordenadas dentro de las bounding boxes. Esta pérdida también muestra un descenso sostenido, especialmente en las primeras 40 épocas, lo cual refleja un aprendizaje efectivo en la regresión de las posiciones exactas.

### 4.4. Implementación del sistema embebido

La puesta en marcha del sistema de detección de errores en impresiones publicitarias se realizó en un entorno controlado, utilizando una cámara convencional conectada a la Raspberry Pi 4, lo cual permitió simular con fidelidad una estación de inspección visual. Esta decisión respondió a un factor principal: la necesidad de realizar pruebas rápidas y observables.

**Figura 15**

*Esquema de hardware y flujo de inferencia del sistema propuesto*



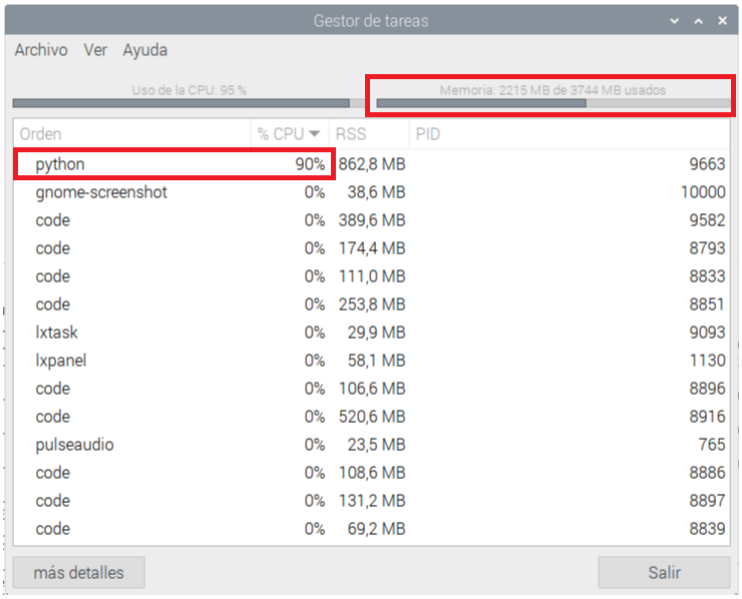

Nota: *La Raspberry Pi 4 actúa como unidad de procesamiento principal, conectada a una cámara encargada de capturar las imágenes en tiempo real. Estas imágenes son enviadas al motor de*

*inferencia de Roboflow, donde el modelo entrenado detecta posibles defectos de impresión. Finalmente, los resultados se visualizan en un monitor para el análisis del operario.* ***Tomado de:*** *Elaboración propia.*

Durante la ejecución del sistema se observó un uso de CPU cercano al 90% y un consumo de memoria de aproximadamente 2.2 GB de los 3.7 GB disponibles observados en la Figura 16. A pesar de esta alta demanda de recursos, el sistema logró mantener la operación estable en la Raspberry Pi 4.

**Figura 16**

*Consumo de memoria RAM en la Raspberry Pi 4 al iniciar el Script de Python*

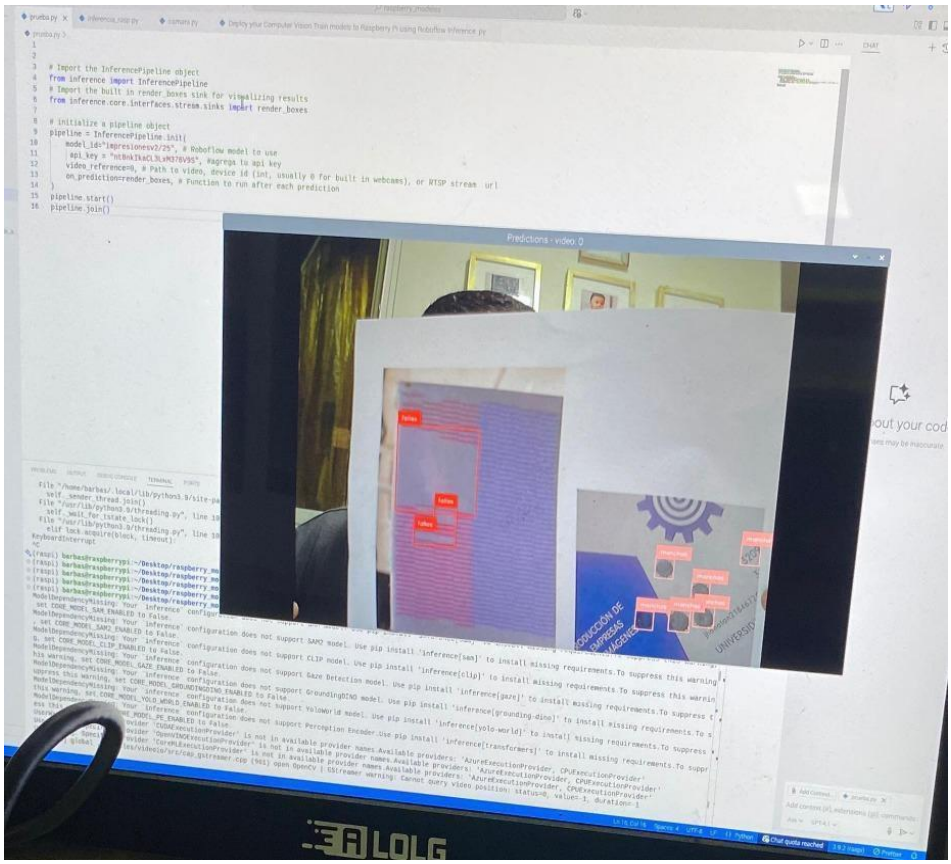



Cada detección se representó mediante cuadros delimitadores que marcaban la ubicación de los defectos identificados, junto con una etiqueta de clase. Esto permitió validar visualmente

que el sistema no solo estaba reconociendo correctamente las fallas, sino también clasificándolas según el tipo de anomalía: manchas de tinta, errores de alineación o corrimientos en la impresión. Se observaron resultados satisfactorios tanto en imágenes estáticas como en pruebas con movimiento, lo cual demuestra que el modelo tiene un buen desempeño incluso en condiciones de variabilidad lumínica o enfoque imperfecto.

**Figura 17**

*Script en correcto funcionamiento del sistema embebido*

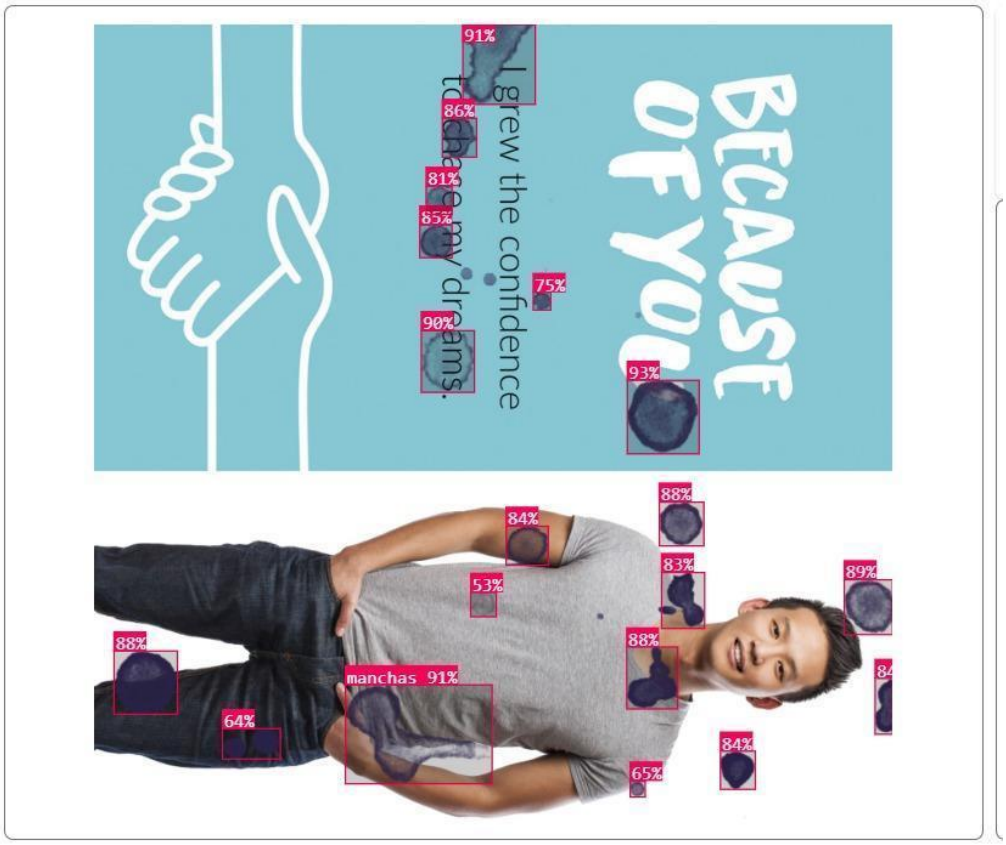



Nota: *En cuanto al desempeño temporal, el sistema requirió un promedio de 1 minuto por inferencia, lo que confirma la viabilidad de la implementación.* **Tomado de:** (Valest, 2025).

Los resultados obtenidos durante la ejecución del sistema pueden apreciarse en las siguientes figuras, que recogen distintos escenarios de prueba:

**Figura 18**

*Capturas del sistema de detección de fallas y resultados de YOLOv11l, "manchas en impresiones"*

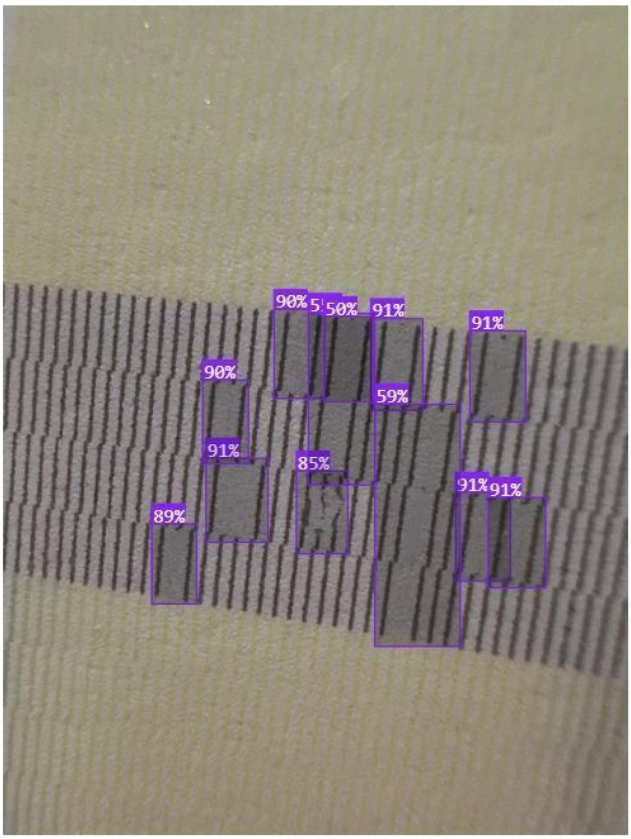



*Nota:* **Tomado de:** *Roboflow (2025).*

En la Figura 18 se observan múltiples manchas identificadas en una camiseta gris, así como sobre una superficie impresa de fondo claro, con precisión y sin falsos positivos evidentes.

**Figura 19**

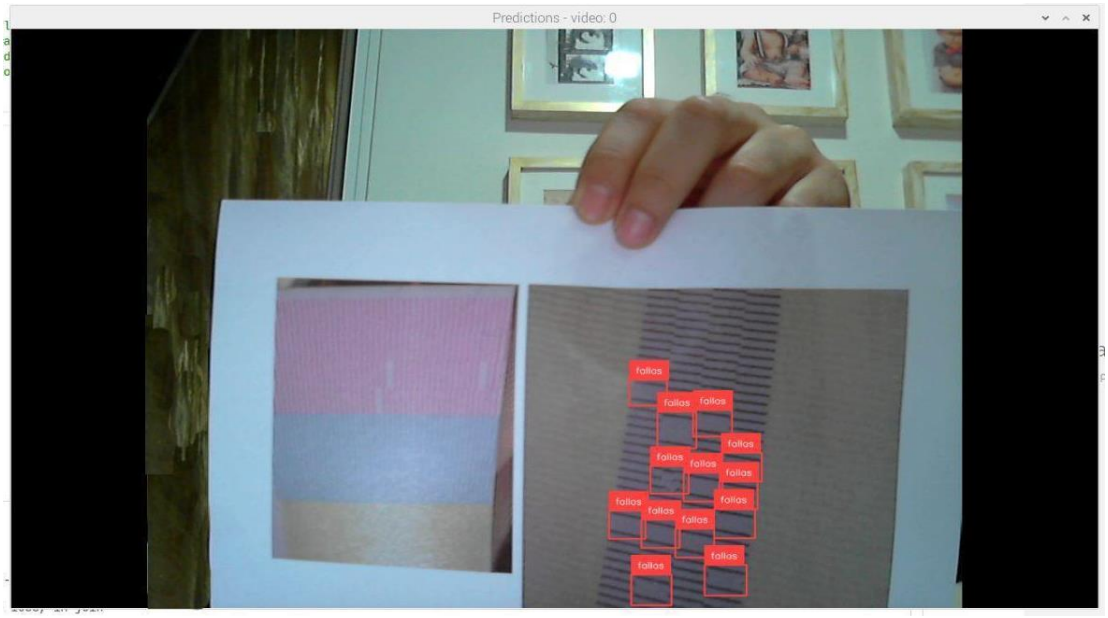
*Capturas del sistema de detección de fallas y resultados de YOLOv11l, "fallas en prueba cmyk de impresión a gran formato"*




En la Figura 19, el modelo detecta con claridad errores de impresión en líneas verticales mal alineadas, representando una falla típica por desplazamiento de cabezal. Estas detecciones se logran incluso en imágenes con bajo contraste, lo cual evidencia el buen funcionamiento del modelo frente a variaciones en iluminación o calidad de captura.

**Figura 20**

*Capturas del sistema de detección de fallas y resultados de YOLOv11*




**(a)**


**(b)**

La Figura 20.a. y Figura 20.b. muestra un par de casos de prueba en condiciones reales con la Raspberry Pi en tiempo real. Allí, el sistema es capaz de identificar manchas en una impresión publicitaria, lo que confirma que el modelo puede generalizar correctamente frente a diferentes dispositivos, resoluciones y ángulos de visualización.

**Figura 21**

*Capturas del sistema de detección de fallas, detección corrimientos YOLOv11*


Por último, en la Figura 21 se presentan detecciones exitosas de corrimientos de tinta, una falla más sutil pero igualmente crítica en la calidad de impresión, las cuales fueron correctamente localizadas y clasificadas por el sistema, incluso con niveles de confianza superiores al 70%.

## 5. Conclusiones

La implementación del sistema embebido se materializó con éxito en la Raspberry Pi 4, utilizando una cámara USB. Esta elección estratégica de hardware permitió superar las limitaciones de plataformas de menor capacidad adecuándose a la complejidad de YOLOv11 Large.

La Raspberry Pi 4 proporcionó el equilibrio ideal entre potencia de procesamiento y accesibilidad, demostrando que la inferencia de modelos de IA de alto rendimiento es factible en un dispositivo de bajo costo.

La ejecución del sistema en la Raspberry Pi, con la inferencia gestionada a través de la nube de Roboflow, validó la funcionalidad completa del sistema en un entorno de emulación a nivel de laboratorio, cumpliendo a cabalidad con el objetivo general del proyecto.

El preprocesamiento, etiquetado y aumento de datos resultaron críticos para los desafíos del modelo, permitiendo una correcta generalización frente a variaciones de iluminación, enfoque y tipo de material además la integración del sistema en un pipeline de inferencia local permitió minimizar la latencia y mantener un rendimiento fluido, incluso en escenarios con múltiples defectos simultáneos, lo que lo hace viable para entornos de producción reales.

El sistema desarrollado representa una alternativa económica y reproducible frente a soluciones industriales de alto costo, contribuyendo a la optimización de procesos, reducción de desperdicio y mejora continua en la calidad de impresión.

La validación del sistema embebido a través de las métricas PSNR y SSIM demostró ser útil, aunque no exenta de limitaciones. Durante el proceso se evidenció que pequeñas variaciones en la posición de la hoja, iluminación o encuadre de la cámara podían alterar significativamente los valores obtenidos, aun cuando las diferencias visuales fueran mínimas. Esto obligó a considerar técnicas de preprocesamiento, como el alineamiento y recorte de las imágenes, para garantizar comparaciones justas. Asimismo, se comprobó que el PSNR, al depender estrictamente del error cuadrático medio, puede sobrestimar la similitud cuando existen cambios locales poco perceptibles, mientras que el SSIM ofrece una valoración más cercana a la percepción humana, aunque sensible a artefactos externos del entorno. En conjunto, el cálculo de estas métricas permitió no solo validar el desempeño del sistema, sino también reconocer la importancia de estandarizar las condiciones de captura y aplicar correcciones de preprocesamiento para asegurar resultados confiables y reproducibles.

## 6. Recomendaciones

Se sugiere, para futuras versiones del sistema, explorar la implementación en dispositivos embebidos con aceleración por hardware, como el Google Coral, NVIDIA Jetson Nano, lo que permitiría liberar recursos del sistema central y mejorar la eficiencia energética en entornos de producción continua.

Asimismo, es recomendable ampliar el conjunto de datos incorporando más variaciones de defectos reales, impresiones en diferentes tipos de papel y condiciones ambientales adversas, con el fin de robustecer aún más la capacidad de generalización del modelo y minimizar los falsos positivos o negativos en producción.

Otra línea de mejora sería la integración de un sistema de seguimiento y registro automatizado de las detecciones, con almacenamiento en la nube y generación de reportes, lo cual facilita la trazabilidad de errores y permitiría implementar estrategias de mejora continua en los procesos gráficos.

En el proceso de validación se recomienda no limitarse únicamente al cálculo de métricas objetivas como PSNR y SSIM, sino complementarlas con pruebas de repetibilidad, evaluando el comportamiento del sistema bajo diferentes condiciones de operación (iluminación, variaciones de impresión, desgaste del hardware). Esto permitiría identificar hasta qué punto las métricas reflejan el desempeño real y garantizar que el sistema embebido mantenga un nivel de confiabilidad aceptable en escenarios de uso prolongado y en entornos de producción reales.

Finalmente, se recomienda continuar evaluando nuevas arquitecturas de detección de objetos ligeros y precisos, como el modelo RF-DETR, que puedan aportar mejoras adicionales en entornos específicos o industrias con requerimientos más exigentes.

**Lista de referencias**

*Amebapro2 AI Convert Model – Realtek IoT/Wi-Fi MCU Solutions*. (n.d.). Realtek IoT/Wi-Fi MCU Solutions. Retrieved July 25, 2025, from https://www.amebaiot.com/en/amebapro2-ai-convert-model/

*AmebaPro2 Datasheet Download*. (2023). Ameba. https://www.amebaiot.com/zh/datasheet-download-amb82-mini/

*Cómo exportar al formato PaddlePaddle desde modelos YOLO11*. (n.d.). Documentación de Ultralytics YOLO. https://docs.ultralytics.com/es/integrations/paddlepaddle/

Cortés, H. (2025, 08 05). *Versions*. Roboflow. Impresionesv2. https://app.roboflow.com/tesis-iee5f/impresionesv2/25

*DATASHEET Raspberry Pi 4 Model B*. (2019, June 21). Raspberry Pi Datasheets. Retrieved July 25, 2025, from https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

*Detección de objetos con YOLO: implementaciones y como usarlas*. (2018, May 11). Medium. Retrieved July 25, 2025, from https://medium.com/@enriqueav/detecci%C3%B3n-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246

IBM. (n.d.). *¿Qué es la Inteligencia Artificial (IA)?* IBM. Retrieved July 25, 2025, from https://www.ibm.com/es-es/think/topics/artificial-intelligence

IBM. (n.d.). *¿Qué es machine learning?* IBM. Retrieved July 25, 2025, from https://www.ibm.com/mx-es/think/topics/machine-learning

IBM. (n.d.). *¿Qué son las redes neuronales convolucionales?* IBM. Retrieved July 25, 2025, from https://www.ibm.com/es-es/think/topics/convolutional-neural-networks

*Introducción a los sistemas embebidos*. (2018). INSTITUTO NACIONAL DE CIBERSEGURIDAD. https://www.incibe.es/incibe-cert/blog/introduccion-los-sistemas-embebidos

*Jetson Nano Lleva el Poder de la IA Moderna a los Dispositivos en el Edge*. (n.d.). NVIDIA. Retrieved July 25, 2025, from https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano/product-development/

*Keras*. (n.d.). Keras: Deep Learning for humans. Retrieved July 25, 2025, from https://keras.io/

León-Batallas, A., Bermeo-Paucar, J., Paredes-Quevedo, J., & Torres-Ordoñez, H. (2020). Una revisión de las métricas aplicadas en el procesamiento de imágenes. *RECIMUNDO*, **4**(3), 267-273. https://doi.org/10.26820/recimundo/4.(3).julio.2020.267-273

*Predict on a Video, Webcam or RTSP Stream*. (n.d.). Roboflow Inference. Retrieved October 6, 2025, from https://inference.roboflow.com/quickstart/run_model_on_rtsp_webcam/

Redmon, J. (n.d.). *Darknet: Open Source Neural Networks in C*. Joseph Redmon. Retrieved July 25, 2025, from https://pjreddie.com/darknet/

*RT-DETR de Baidu: Un detector de objetos en tiempo real basado en Vision Transformer*. (n.d.). Ultralytics YOLO Docs. Retrieved July 25, 2025, from https://docs.ultralytics.com/es/models/rtdetr/#overview

*Sistemas embebidos: claves para eficiencia y automatización*. (2025, June 12). Innovación Digital 360. Retrieved July 25, 2025, from https://www.innovaciondigital360.com/iot/sistemas-embebidos-que-son-y-para-que-se-utilizan/

Solawetz, J. (2020, July 1). *Train a YOLOv4-tiny Model on a Custom Dataset*. Roboflow Blog. Retrieved July 25, 2025, from https://blog.roboflow.com/train-yolov4-tiny-on-custom-data-lighting-fast-detection/

Tensorflow. (n.d.). *tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone*. GitHub. Retrieved July 25, 2025, from https://github.com/tensorflow/tensorflow

Ultralytics. (n.d.). *Descripción general de los conjuntos de datos de detección de objetos*. Ultralytics YOLO. Retrieved July 25, 2025, from https://docs.ultralytics.com/es/datasets/detect/

Ultralytics. (2025, July 16). *Ultralytics YOLO11 - Documentación de Ultralytics YOLO*. Ultralytics YOLO Docs. Retrieved August 9, 2025, from https://docs.ultralytics.com/es/models/yolo11/#performance-metrics

Valest, J. (2025, 07 25). *Conjunto de datos*. Google Drive. https://drive.google.com/drive/folders/1wbYGIQqdGHMmi01kj1-fVAYg0xkz7vcI

Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004, 04 30). *Image quality assessment: from error visibility to structural similarity*. IEEE Xplore. https://ieeexplore-ieee-org.bibliotecavirtual.uis.edu.co/stamp/stamp.jsp?tp=&arnumber=1284395

*What is PyTorch?* (2023, October 4). IBM. Retrieved July 25, 2025, from https://www.ibm.com/think/topics/pytorch

*YOLO11 Object Detection Model: What is, How to Use*. (n.d.). Roboflow. Retrieved July 25, 2025, from https://roboflow.com/model/yolo11

*YOLOv4 Tiny*. (09, 11 2020). Roboflow. https://roboflow.com/model/yolov4-tiny

*You Only Look Once: Unified, Real-Time Object Detection*. (2015, June 8). arXiv. Retrieved July 25, 2025, from https://arxiv.org/abs/1506.02640

## Apéndice

Apéndice A. Repositorio en Github control_calidad

En el siguiente enlace público de la plataforma Github se podrá ver con más detalle los aspectos mencionados en este documento para el desarrollo del proyecto.

https://github.com/henrycortes10/control_calidad

Apéndice B. Dataset Google Drive

En el siguiente enlace público de Google Drive se podrá ver el conjunto de datos.

https://drive.google.com/drive/folders/1wbYGIQqdGHMmi01kj1-fVAYg0xkz7vcI

Apéndice C. Información del sistema embebido en Google Drive

En el siguiente enlace público de Google Drive se podrá ver evidencia respecto a la implementación del sistema embebido.

https://drive.google.com/drive/folders/1-PDu2wgup472LynxaOJ1TWqaTKxA0qiB