UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE MECATRÓNICA

DESARROLLO DE SISTEMA DE BANDA TRANSPORTADORA PARA LA DETECCIÓN DE DEFECTOS EN CONSERVAS ENLATADAS POR MEDIO DE VISIÓN ARTIFICIAL

Trabajo de titulación previo a la obtención del
Título de Ingeniero en Mecatrónica

AUTOR: Juan Carlos Espinoza Allauca
TUTOR: Jonathan Salvador Paillacho Corredores

Guayaquil - Ecuador

2024 - 2025

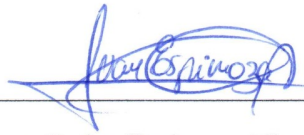
CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Yo, **Juan Carlos Espinoza Allauca** con documento de identificación N° **0950428284**; manifiesto que:

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

Guayaquil, 12 de febrero del año 2025

Atentamente,



Juan Carlos Espinoza Allauca
0950428284

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

Yo, **Juan Carlos Espinoza Allauca** con documento de identificación Nº **0950428284**, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del **Dispositivo Tecnológico: DESARROLLO DE SISTEMA DE BANDA TRANSPORTADORA PARA LA DETECCIÓN DE DEFECTOS EN CONSERVAS ENLATADAS POR MEDIO DE VISIÓN ARTIFICIAL**, el cual ha sido desarrollado para optar por el título de: Ingeniero en Mecatrónica, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

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

Guayaquil, 12 de febrero del año 2025

Atentamente,

Juan Carlos Espinoza Allauca
0950428284

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, **Jonathan Salvador Paillacho Corredores**, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: **DESARROLLO DE SISTEMA DE BANDA TRANSPORTADORA PARA LA DETECCIÓN DE DEFECTOS EN CONSERVAS ENLATADAS POR MEDIO DE VISIÓN ARTIFICIAL**, realizado por **Juan Carlos Espinoza Allauca** con documento de identificación Nº **0950428284**, obteniendo como resultado final el trabajo de titulación bajo la opción **Dispositivo Tecnológico** que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 12 de febrero del año 2025

Atentamente,

Ing. Jonathan Salvador Paillacho Corredores, M.Sc.
1718907874

DEDICATORIA

Dedico este trabajo primero a Dios, a mi madre Rosario Allauca Amaguaya y hermana Karina Enriquez Allauca, que me han impulsado en cada momento a seguir adelante, me han enseñado el esfuerzo diario, la perseverancia, la paciencia y el trabajo diario; a mis tíos que fueron y son un pilar fundamental para que alcance el éxito en esta etapa universitaria, que nada es fácil en la vida y que todo se gana con el sudor de la frente.

Juan Carlos Espinoza Allauca

AGRADECIMIENTO

Agradezco a cada docente que formaron parte de mi vida universitaria, personas con un alto valor ético y profesional, con sus conocimientos me formaron permitiéndome adquirir habilidades valiosas para mi carrera profesional. También a cada amigo que me a puesto la universidad, quienes me mostraron su verdadera amistad y sinceridad absoluta, sin ellos esto tampoco seria posible para alcanzar la meta pero se logró, después de esto les deseo lo mejor de los éxitos.

Juan Carlos Espinoza Allauca

RESUMEN

El presente trabajo tiene como objetivo el diseño e implementación de un sistema de clasificación de conservas de latas, mediante el uso de redes neuronales convolucionales de visión artificial en python 3. El sistema se centra en el uso de una banda transportadora, la cual lleva las latas hacia la cámara, para su respectivo análisis y clasificación en base a las condiciones físicas de la lata.

Para la implementación de la presente propuesta, se emplea un sistema de reconocimiento por visión artificial mediante el uso de tensorflow, para el respectivo análisis y funcionamiento correcto del sistema. La IA es entrenada mediante el uso de un dataset, el cual proporciona todas las clases de desperfectos y condiciones viables de las latas, una vez el sistema evalúe el elemento, este tomara una acción de empujar la lata a través del uso de un pistón, si esta no esta en buenas condiciones, caso contrario la lata seguirá su trayecto correspondiente por la banda transportadora.

Por último, para el diseño y entrenamiento de la IA se utilizó el entorno de python 3, debido a la estabilidad y rendimiento que ofrece para la creación de distintos tipos de IA, el cual es ejecutado en una raspberry pi 4b con el fin de reducir gastos económicos y optimizar recursos. Para la creación de la banda se realizaron cálculos y simulaciones matemáticas mediante el uso de Matlab que permiten determinar el correcto funcionamiento y viabilidad.

Palabras claves: redes neuronales, entrenamiento, visión artificial, IA, python 3, raspberry pi 4b, clasificación de latas.

ABSTRACT

The objective of this work is the design and implementation of a classification system for canned goods, through the use of artificial vision convolutional neural networks in Python 3. The system focuses on the use of a conveyor belt, which carries the cans to the chamber, for their respective analysis and classification based on the physical conditions of the can.

For the implementation of this proposal, an artificial vision recognition system is used through the use of TensorFlow, for the respective analysis and correct operation of the system. The AI is trained using a dataset, which provides all kinds of defects and viable conditions of the cans. Once the system evaluates the element, it will take an action to push the can through the use of a piston, if it is not in good condition, otherwise the can will continue its corresponding path along the conveyor belt.

Finally, for the design and training of the AI, the Python 3 environment was used, due to the stability and performance it offers for the creation of different types of AI, which is executed on a Raspberry Pi 4b in order to reduce economic costs and optimize resources. For the creation of the belt, mathematical calculations and simulations were carried out using Matlab that allow determining the correct operation and viability.

Keywords: neural networks, training, computer vision, AI, python 3, raspberry pi 4b, can classification.

Índice

Índice de figuras

Índice de tablas

I. Introducción

El mundo vive cambios constante de avances tecnológicos y las cuales se convierten en herramientas fundamentales dentro del sector alimenticio por tener la capacidad de analizar y etiquetar imágenes de manera precisa, esto ha dado paso a una gran variedad de usos en distintos ámbitos. Por esta razón, se diseñó y desarrolló un prototipo de banda transportadora para la detección de defectos en conservas enlatadas y por medio de la visión artificial, con el propósito de implementar un sistema capaz de identificar automáticamente latas buenas y latas malas. Para ello, se utilizan imágenes capturadas por una cámara conectada a un Raspberry Pi 4 modelo b, lo que facilita la clasificación y acelera el proceso de selección de estos productos.

Se decidió por conservas enlatadas debido a su gran importancia y demanda en los mercados, al ser un alimento fácil y rápido de consumir por tal motivo las industrias que no realice una inspección rigurosa en sus conservas enlatadas puede que una de ellas salga en mal estado comprometiendo la salud de quienes lo consumen y futuras demandas, dañando así el prestigio de la empresa y futuras perdidas económicas. En la detección de defectos en conservas enlatadas y el proceso de reconocimiento de imágenes tiene la capacidad de mejorar la precisión de detección y separación de conservas enlatadas de mal estado, lo que conllevara a beneficios económicos y a una mayor credibilidad.

II. Problema

La competencia en un mercado que crece rápidamente y está desafiando a todas las industrias a producir productos con alta calidad, flexibilidad y valor agregado, pero a costos más altos y en tiempos más prolongados, debido a que muchos de los procesos actuales dependen de un método tradicional como la inspección humana y de la capacidad visual para detectar defectos de fabricación en los productos enlatados [1]. En las industrias manufactureras existen varios problemas, uno de los más relevantes es la falta de automatización en el proceso de cierre del producto enlatado, esto ocasiona un colapso durante la transportación de latas cerradas, debido a que no existe una separación entre el producto defectuosas y no defectuoso, esto ocasiona que el personal del departamento de calidad tenga que realizar un control riguroso para que de forma visual y manual identifique los diferentes daños en el producto semisellado.

Las principales causas de las latas envasadas con defectos son: máquinas cerradoras sin el mantenimiento adecuado, calibración y programación inadecuada, insumos (latas y tapas) con defecto de fábrica, insuficiente recubrimiento de barniz en la parte externa de la lata y el proceso de esterilización con sobrecarga de agua y vapor que contiene metales pesados.

La generación del producto terminado con los defectos antes señalados pueden ocasionar retrasos significativos en la producción y distribución, esto impacta de manera negativa en los tiempos de entrega y eficiencia operativa causando pérdida de clientes en el mercado, pérdida de la confiabilidad del cliente y consumidor, pérdida económica y problemas de salud cuando una lata sufre una abolladura que genera una grieta, permitiendo la entrada de organismos externos. Esto contamina internamente el alimento que se consumirá, producido por microbios más peligrosos que generan un agente tóxico en los alimentos enlatados son las bacterias del género clostridium, causante de la intoxicación alimentaria más grave, el clostridium botulinum, un peligro reconocido mundialmente en los alimentos herméticamente sellados [2].

Además, se generará un impacto económico negativo al retirar del mercado el producto procesado ya listo para la distribución y comercialización, a esto añadirá un problema grave debido a las demandas legales, que generarán costos significativos para la empresa, dañando su imagen y posición en el mercado a largo plazo.

III. JUSTIFICACIÓN

El propósito de este proyecto es promover un prototipo mecatrónico que utilizará la visión artificial para optimizar el proceso de verificación de excelencia. Con esta tecnología, se podrá asegurar que solo los productos en buen estado lleguen al mercado, optimizando productos de la más alta calidad [3].

La presente investigación explora el papel de sistemas inteligentes y el aprendizaje automático mediante la automatización de los procesos de cierre en productos enlatados con el fin de desarrollar modelos de visión artificial capaces de identificar y adaptarse a diferentes condiciones y necesidades de las empresas y optimizar la exactitud y la flexibilidad de los sistemas de producción, reduciendo los costos operativos.

La implementación de sistemas automatizados permite una mayor flexibilidad y capacidad de respuesta ante los problemas que causan al momento del cierre de una lata, las deficiencias en el sellado de una lata de producto terminado da paso a consecuencias graves como los costos elevados para la empresa y el riesgo en la salud de los consumidores. Por medio de la ejecución de un sistema de visión artificial que funcione a tiempo real [4] y sea eficaz de reconocer desperfectos en el codificado de las latas se propone a primera instancia mitigar las dificultades de salud que afectan a los técnicos operativos, como el agobio, agotamiento visual y desgaste, que se provoca por la ejecución de labores monótonas.

La banda transportadora integrará visión artificial mediante una red neuronal desarrollada en python, que permitirá crear una base de datos para identificar conservas enlatadas en buen y mal estado, luego se analizará las imágenes en tiempo real y posterior a eso detecte las conservas enlatadas en mal estado, que ayudará a las tomas de decisiones automáticos en cuestión de productos defectuosos. Esta tecnología de vanguardia no solo eleva significativamente la calidad de los productos, sino que también agiliza las operaciones, disminuyendo los costos asociados a la inspección tradicional por una banda transportadora [5].

IV. OBJETIVOS

IV-A. Objetivo general

Implementar un sistema de banda transportadora con visión artificial para la detección automática de defectos en conservas enlatadas en una línea de producción.

IV-B. Objetivos específicos

- Construir una banda transportadora para la clasificación de conservas enlatadas.
- Implementar un sistema de visión artificial para la detección de deformaciones en conservas enlatadas.
- Validar el funcionamiento del sistema de visión artificial en una línea de producción de conservas enlatadas para la evaluación de su rendimiento en condiciones reales.

V. Marco teórico referencial

V-A. La inteligencia artificial

La Inteligencia Artificial (IA) es un campo de la informática dedicada a generar sistemas que pueden llevar a cabo actividades que, en general, solicitan inteligencia humana. Este concepto se manifiesta cuando una máquina reproduce entornos cognitivos similares a los de los seres humanos, interactuando con otras mentes humanas en aspectos como el pensamiento, el aprendizaje y la percepción [6].

El mundo atraviesa la cuarta revolución industrial, también conocida Industria 4.0. La integración de la IA en la vida habitual de las personas se ha vuelto algo habitual [7], obteniendo como mejoras imprevistas en la economía y bienestar de las empresas que implementan IA en sus líneas de producción dando como resultado altos estándares de seguridad, eficiencia y sostenibilidad [8].

V-A1. La inteligencia artificial en la industria:

La IA es la tecnología que está revolucionando rápidamente diversas industrias y es un campo de la informática dedicado a crear técnicas y algoritmos que permiten a las máquinas realizar funciones que, hasta hace poco, solo podían ser ejecutadas por humanos. Permitiendo una optimización en los procesos industriales de la ingeniería con una mayor exactitud en el análisis predictivo y en la toma de decisiones dentro de la industria de la ingeniería [9]. A modo de ejemplo, en el proceso de los enlatados puede detectar las conservas con defectos, es capaz de identificar problemas en tiempo real durante la producción, promoviendo una mejora continua y aumentando la productividad como se observa en la Figura 1.

Figura 1: Línea de producción de conservas enlatadas [10].

V-B. Conservas enlatadas de aluminio

En los últimos años se han evolucionado varios tipos de envases con diferentes formas, reconociendo la maleabilidad y facilidad de aluminio, hoy en día se dispone de latas en diversas formas, como cuadradas, rectangulares y ovaladas, entre otras [11]. Las latas internamente tienen un recubrimiento tipo barniz sanitario que son totalmente aptos para los alimentos, estos sufren un proceso térmico de esterilización para aislar la inactividad de microorganismos que pueden ingresar y contaminar el alimento. Una conserva en optimas condiciones permite conserva el alimento por extensos periodos de tiempo.

La demanda de envases de aluminio para alimentos ha experimentado un notable crecimiento, a continuación se puede observar en la Figura 2, el procedimiento de creación de una lata desde aluminio.

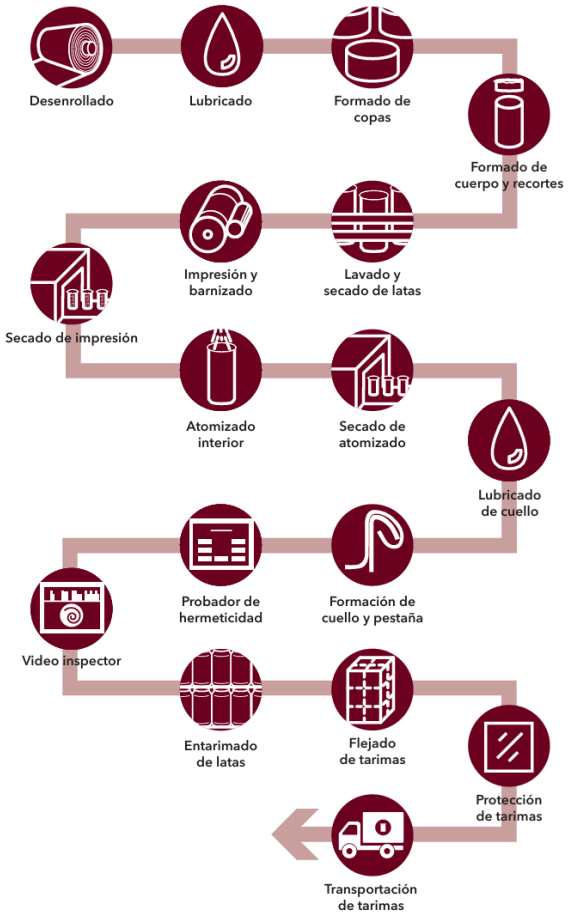


Figura 2: Proceso de elaboración de latas de aluminio [11].

V-B1. Evaluación del estado de conservación de latas:

Las conservas logran sufrir daños físicos a causa de diversos factores externos [12]. A continuación las diferencias entre una lata con abolladuras o deformaciones, rayaduras y oxido:

- Lata con Abolladura:
  Este daño es causado por movimientos bruscos durante el traslado al área de etiquetado, como se puede ver en la Figura 3.

Figura 3: Abolladura lateral de lata cilíndrica [13].

- Lata con Rayadura:
  Estos defectos se generan durante el proceso de fabricación, ya se por el uso de herramientas desgastadas o mal alineadas, como se ve en la Figura 4..

Figura 4: Raya parte superior de lata cilíndrica [14].

- Lata con Oxido:
  Cada lata cuenta con un recubrimiento protector diseñado para evitar la oxidación. Si este recubrimiento es demasiado delgado o se daña, el metal base queda expuesto, como se puede ver en la Figura 5..

Figura 5: Oxido parte lateral de lata cilíndrica [15].

V-C. Visión artificial

La tecnología de visión facilita mejorar el control de calidad de una manera rápida y precisa, además de disminuir los costos en comparación con la inspección visual. Esta tecnología se utiliza como una herramienta muy ventajosa en las etapas de manufactura para clasificar los productos en buen o mal estado, asegurando el cumplimiento de normas [16]. En lo que concierne el uso la visión artificial es empleado en todos los campos de producción dando mejoras continua, a continuación se ilustrara en la Figura 6 un sistema de verificación de etiquetas con códigos uniformes y alta calidad de impresión.

Figura 6: Verificación de códigos con inspección artificial [17].

V-D. Arquitecturas de visión artificial

Con el paso del tiempo, se han implementado múltiples tecnologías, entre las más comunes se encuentran:

- Visión por computadora tradicional
- El aprendizaje profundo
- Las arquitecturas mixtas

Permiten la mejora de secuencias de verificación empleando la visión artificial. Los sistemas de inspección óptica programados, genera dicha capacidad en las arquitecturas enfocadas en CNN implementadas en el proceso de verificación detectando los fallos en la fabricación de múltiples artículos [18].

V-D1. Redes neuronales convolucionales:

Las CNN son la piedra angular de la visión artificial moderna. Estas redes están planificadas para analizar datos visuales y han demostrado ser extremadamente efectivas en diversas tareas:

- Clasificación de imágenes
- Detección de objetos
- Segmentación
- Generación de imágenes

Las CNN aplicadas a las imágenes diagnósticas procurar replicar el funcionamiento de la corteza visual primaria del cerebro humano, además operan tanto en imágenes 2D como en 3D y generalmente se componen de tres capas: la capa de convolución, la capa de reducción (max pooling) y las capas densas (fully connected) [19], como se puede ver en la Figura 7.

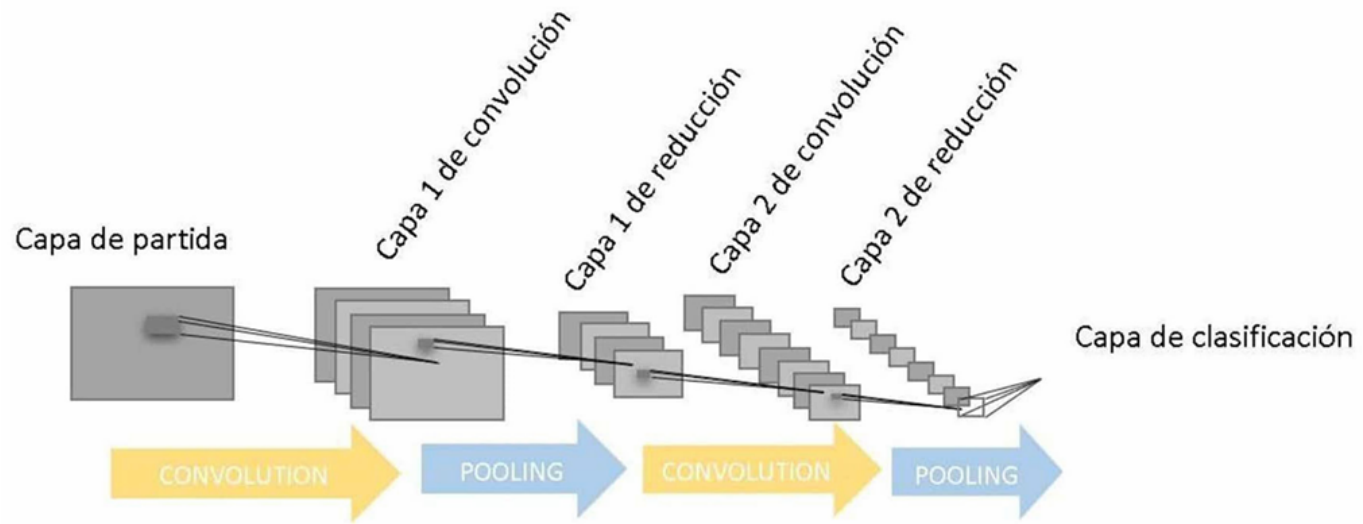


Figura 7: Arquitectura de una red neuronal convolucional[19].

V-E. Tensorflow

Es una plataforma de código abierto diseñada para el desarrollo de sistemas de aprendizaje automático. Su versatilidad permite aplicarlo en diversas tareas, aunque su enfoque principal es el entrenamiento y la ejecución de redes neuronales. [20]. El protocolo keras enfocado en redes neuronales, cuenta con una biblioteca abierta en aprendizaje profundo. [21].

En la Figura 8, se ilustra un flujo de trabajo estándar:

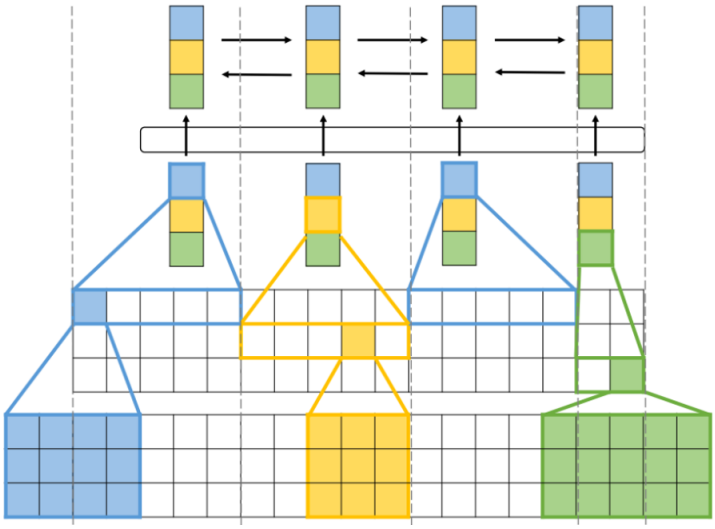


Figura 8: Aplicacion de keras [22].

V-F. Banda o sistema de cinta

Han emergido innovaciones significativas con el propósito de mejorar el mundo, sin embargo, en el ámbito industrial, se pueden destacar importantes contribuciones que han transformado considerablemente la modalidad en que se implican a cabo los procedimientos, facilitando la aceleración de los procesos y mejorando el rendimiento de las fabricas, las bandas transportadoras siguen siendo utilizadas hoy en día en diversos sectores [23].

V-F1. Clases de cintas transportadoras:

- Cintas Transportadoras de rodillos: Están formadas por rodillos cilíndricos que pueden ser accionados por un motor o por gravedad. Son perfectas para el manejo de objetos de forma regular y estable, como cajas y paquetes, como se puede ver en la Figura 9.

Figura 9: Cinta transportadora de rodillos [24].

- Cintas Transportadoras de banda: Emplean una banda continua y flexible que se desplaza sobre rodillos o poleas motorizadas para transportar objetos. Son ampliamente utilizadas en sectores como la industria alimentaria, agrícola y logística, como se puede ver en la Figura 10.

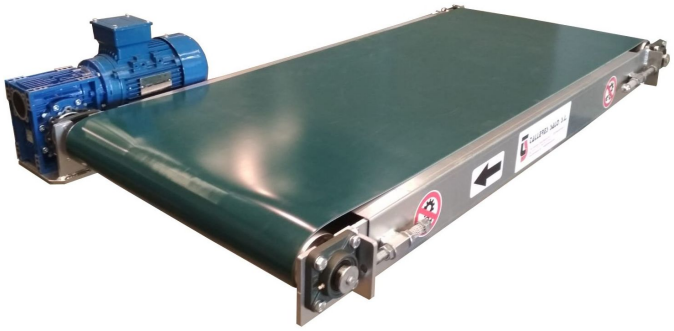


Figura 10: Cintas transportadoras bandas PVC [25].

- Cintas Transportadoras de cadena: Emplean una cadena metálica equipada con dispositivos de arrastre para sostener y mover objetos. Este tipo de cinta es especialmente adecuado para el manejo de objetos pesados, voluminosos o de formas irregulares, como se puede ver en la Figura 11.

Figura 11: Cinta transportadora de cadena [26].

- Cintas Transportadoras modulares: Consisten en módulos individuales que se conectan entre sí. Estos módulos pueden ser de plástico, metal u otros materiales, y se ensamblan para crear una banda continua. Este tipo de cinta es muy flexible y adaptable, permitiendo configuraciones variadas en forma y tamaño según los requisitos de transporte, como se puede ver en la Figura 12.

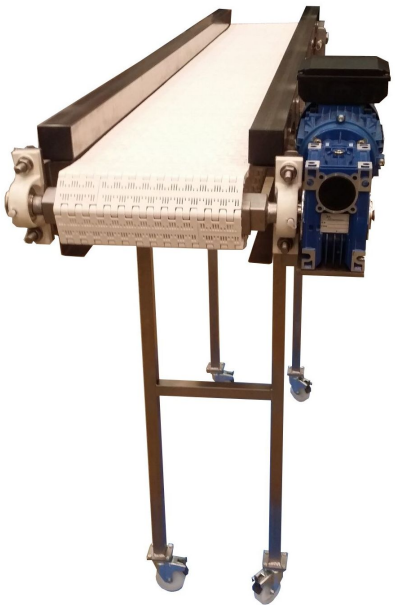


Figura 12: Cinta transportadora banda modular [27].

VI. METODOLOGÍA

VI-A. Propuesta de Diseño

En este estudio se propone implementar un sistema eficiente para la clasificación de conservas enlatadas según su estado, utilizando técnicas de visión por computadoras. Este prototipo consta de una banda transportadora de las latas que pasan a través de una cámara ubicada en la parte superior para una detección más eficiente de cada lata; se procesa cada imagen mediante una red neuronal convolucional y determina diferentes tipos de latas.

El prototipo de banda transportadora tendrá incluido un actuador cilíndrico de simple efecto, que ayudará a desplazar la lata en mal estado, continuando el proceso de producción para que la lata defectuosa pase a una revisión. En la siguiente Figura 13, se ilustra el diagrama de funcionamiento del prototipo mencionado:

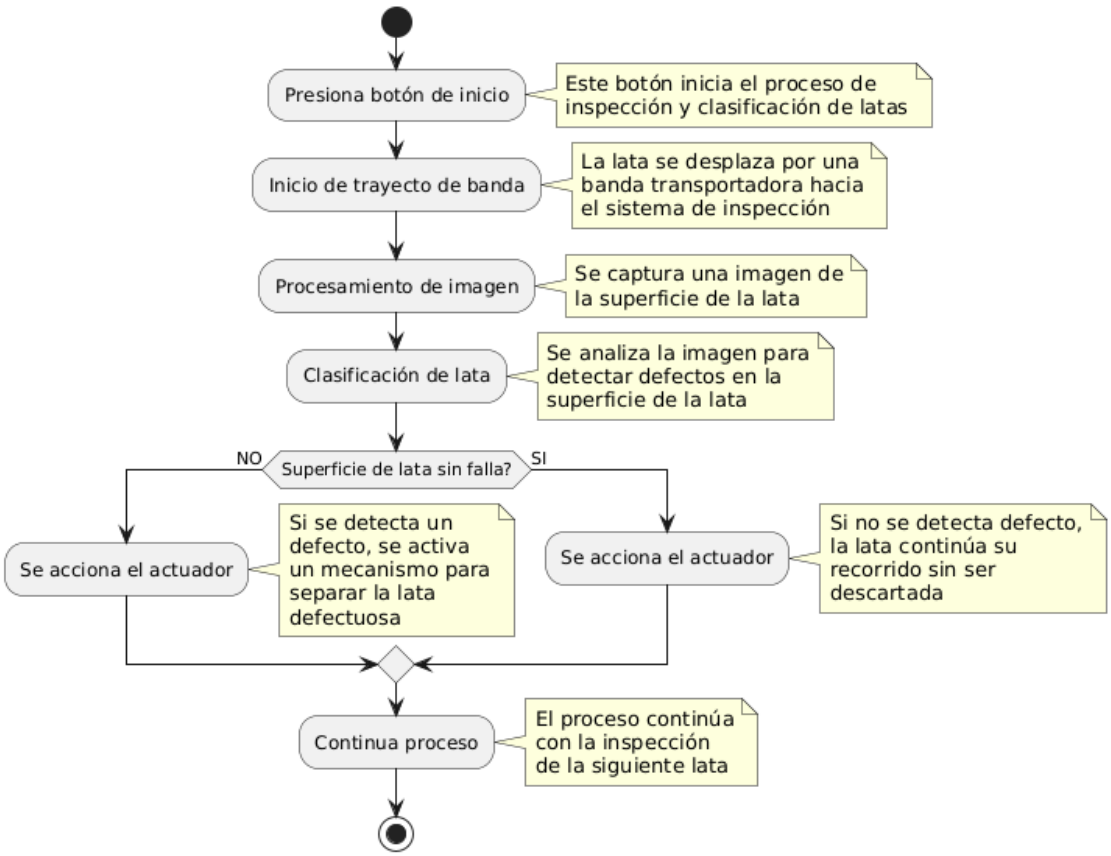


Figura 13: Diagrama de flujo. Fuente: Autor.

VI-B. Diseño Mecánico

La estructura base será fabricada en acero inoxidable grado alimenticio 304, debido que se transportará conservas enlatadas, por tal motivo, en la industria alimenticia es usado este tipo de material. Es muy resistente a la corrosión, fácil limpieza y con una amplia capacidad para soportar altas temperaturas.

Se realizó el diseño con la siguientes medidas: el alto de la base es de 250 mm por el largo de 900 mm, el ancho de tope a tope es de 255 mm; estas medidas se realizó en base a las medidas de la conserva enlatada de forma ovalada (lata de sardina) la cual tiene un ancho de 110 mm y de alto 165 mm.

Se diseñaron los rodillos de diámetro de 55 mm, para mejor manipulación al momento de instalar tanto la mesa, como la banda, ya que el material de la banda es rígido y resistente a la tensión. La implementación de la mesa

es importante para que la banda no se hunda con el producto enlatado, ya que tiene un peso de 290 g cada lata respectivamente; si no se realiza dicha implementación, se perdería estabilidad al momento de transportar de un punto al otro.

El diseño de la mesa por donde se deslizará la banda para transportar el producto enlatado, tiene un ancho de tope a tope de 255 mm y la banda de PVC de grado alimenticio color blanco es de 245 mm dejando un espacio de lado y lado de 5 mm respectivamente, de tal manera, la banda no sufrirá un atasco con los extremos, causando un retraso de la transportación del producto.

El peso del prototipo es medianamente ligero, teniendo en consideración que todo es realizado en acero inoxidable y sus medidas no superan a grandes medidas tanto el alto por largo y ancho.

Para un mejor entendimiento del diseño se puede observar en anexos los planos de trabajo, cada diseño de pieza acotado en milímetros.

VI-C. Fórmulas de análisis de banda transportadora

1. Cálculo de masa: El calculo de la densidad de material de la banda por el volumen que se determina por las dimensiones de la banda. Donde:
   ρ : Densidad
   L : Longitud
   W : Ancho
   t : Espesor

$$m_{banda} = \rho \cdot L \cdot W \cdot t \tag{1}$$

$$m_{total} = m_{banda} + m_{producto} \tag{2}$$

   Solución:

$$m_{banda} = 1240 \times 1{,}8056 \times 0{,}245 \times 0{,}002 = 1{,}0971 \text{ kg} \tag{3}$$

$$m_{total} = m_{banda} + m_{producto} = 1{,}3871 \text{ kg} \tag{4}$$

2. Velocidad angular: Considerar que una vuelta completa equivale a 2 pi radianes. Donde:
   ω : Velocidad angular
   $v_{nominal}$: Velocidad nominal

$$\omega = \frac{v_{nominal} \cdot 2\pi}{60} \tag{5}$$

   Solución:

$$\omega = \frac{1725 \times 2\pi}{60} = 180{,}64 \text{ rad/s} \tag{6}$$

3. Velocidad tangencial: La velocidad angular por el radio de la polea, esta velocidad representa la rapidez con la que se mueve la banda en la superficie de la polea. Donde:
   ω : Velocidad angular
   $v_{tangencial}$: Velocidad tangencial

$$v_{tangencial} = \omega \cdot r_{polea} \tag{7}$$

   Un radio de polea de 0.03 m:

$$V_t = 180{,}64 \times 0{,}03 = 5{,}419 \text{ m/s} \tag{8}$$

VI-D. Fuerzas actuantes

4. Fuerza gravitacional: El peso de la banda. Donde:
   F_g : Fuerza gravitacional
   g : gravedad

$$F_g = m_{total} \cdot g \tag{9}$$

Solución:

$$F_g = 1{,}3871 \times 9{,}81 = 13{,}60 \text{ N} \tag{10}$$

5. Fuerza de tensión: La potencia transmitida es el producto de la fuerza y la velocidad, considerando la eficiencia del motor. Donde:
   F_T : Fuerza de tensión

$$F_T = \frac{P \cdot \eta}{V_t} \tag{11}$$

$$F_T = \frac{(0{,}25 \times 745{,}7) \times 0{,}75}{5{,}419} = 25{,}80 \text{ N} \tag{12}$$

6. Fuerza de Fricción: Se calcula multiplicando la fuerza de gravedad por el coeficiente de rozamiento del material. Donde:
   F_f : Fuerza de fricción
   μ : coeficiente de ficción

$$F_f = F_g \cdot \mu \tag{13}$$

$$F_f = 13{,}60 \times 0{,}35 = 4{,}76 \text{ N} \tag{14}$$

7. Calculo de Momento: La fuerza de tracción tiene la capacidad de rotar la polea en este punto.

$$M = F_T \cdot R \tag{15}$$

$$M = 25{,}80 \times 0{,}1 = 2{,}58 \text{ N} \cdot \text{m} \tag{16}$$

VI-E. Simulación de diseño de la banda transportadora

VI-E1. Simulación del movimiento de rotación sobre soporte de la banda:

El siguiente análisis de elementos finitos se emplea para determinar el desplazamiento de la banda sobre el soporte rodillo.

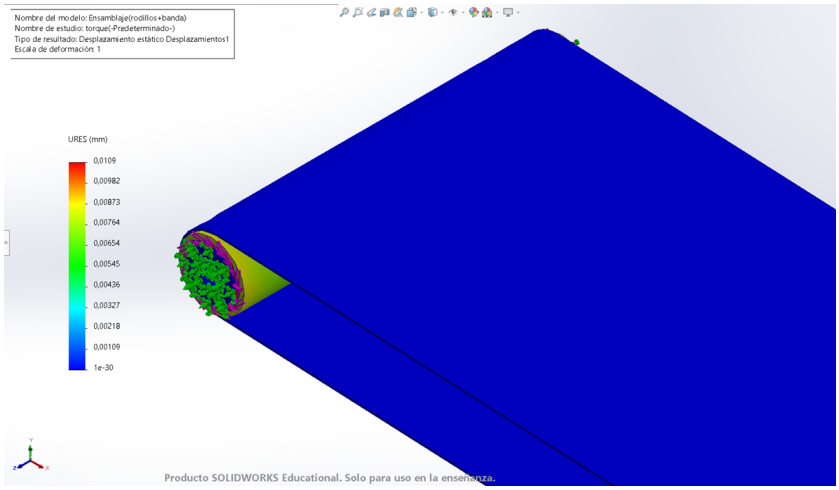


Figura 14: Desplazamiento de banda. Fuente: Autor

El análisis del prototipo en la que se usa rodillo de acero inoxidable y banda de grado alimenticio entendiendo que el acero empleado en este prototipo tiene una gran durabilidad frente a la corrosión y una resistencia al estiramiento.

El desplazamiento empleado sobre la velocidad de la banda y el peso del producto, se observa en la escala de colores el rango de 0.001 a 0.01 mm obteniendo valores muy bajos lo cual indica que el soporte rodillo esta soportando la carga sin deformaciones significativas.

VI-E2. Simulación de peso y fuerzas actuantes:

El sistema incluye la banda (m = 0.884 kg) y una lata (m = 0.29 kg), ejerce una fuerza de 11.51 N sobre el rodillo. Esta fuerza se distribuye de manera uniforme, asegurando un funcionamiento estable. Además, el coeficiente de fricción entre el PVC y el acero inoxidable ($\mu = 0{,}35$) garantiza un agarre óptimo entre la banda y el rodillo como se puede visualizar en la Figura 15, minimizando el riesgo de deslizamiento y manteniendo la eficiencia del sistema.

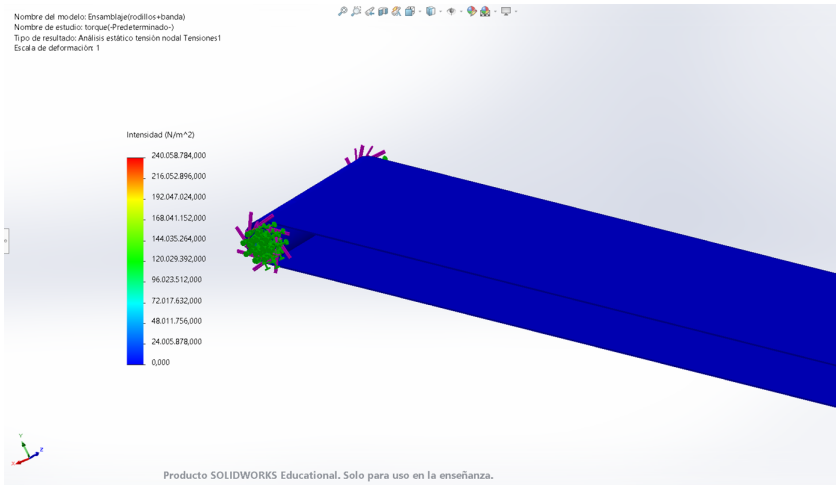


Figura 15: Fuerzas actuantes de rodillo. Fuente: Autor.

El torque es la fuerza de rotación necesaria para que la banda se mueva. Para calcularlo, consideramos el peso de la carga y el tamaño del rodillo (48 cm de radio). El resultado es 5.5248 Nm. Este valor es bajo, lo que significa que el sistema no tendrá problemas para mover la banda y su carga, lo que garantiza un funcionamiento eficiente y sin interrupciones.

VI-E3. Simulación de banda sin mesa base:

La banda es rígida pero el contenido de cada lata varia en peso teniendo en cuenta que el peso inicial (m = 0.29 kg) puede que aumente al presenciar una lata hinchada, dando deformidad en la banda si no cuenta con una mesa base como se puede apreciar en la Figura 16, dando así un desequilibrio.

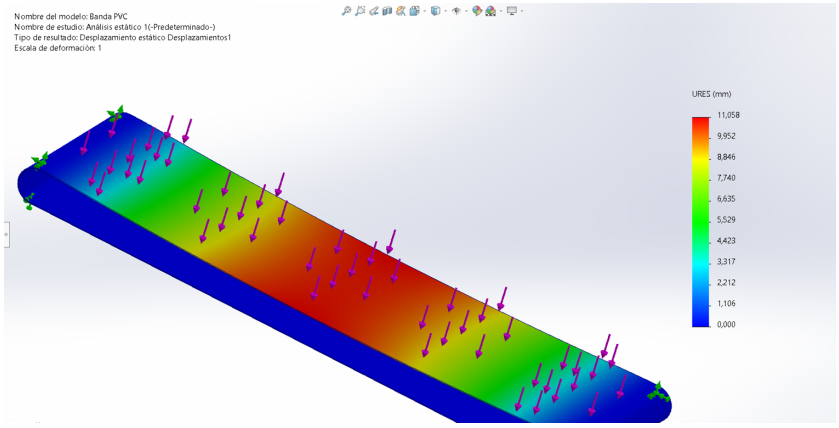


Figura 16: Deformación de banda. Fuente: Autor.

VI-E4. Simulación de banda con mesa base:

El análisis realizado demuestra que la banda transportadora, con soporte inferior, mantiene estabilidad estructural al transportar tres latas de 290 g cada una, aplicando una carga total de 8.52 N. La emulación, tal como se puede ver en la Figura 17 muestra zonas azules con bajos esfuerzos, indicando que la estructura es resistente y estable. Considerando el material de acero inoxidable y su límite elástico (200-250 MPa) frente a un esfuerzo máximo de 90 MPa, se estima un coeficiente de seguridad de 2.22. Para garantizar un funcionamiento óptimo y prevenir deformaciones, se recomienda un FS entre 2.5 y 3, asegurando así la durabilidad y eficiencia del sistema en operación continua.

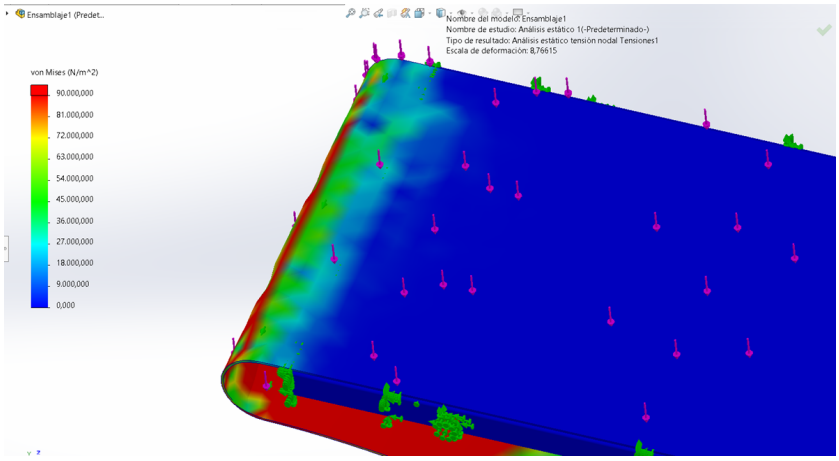


Figura 17: Soporte debajo de la banda. Fuente: Autor.

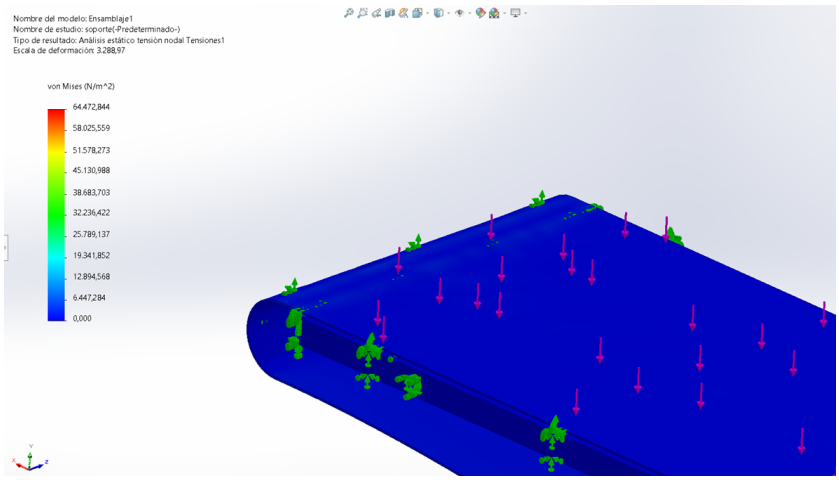


Figura 18: Soporte debajo de la banda. Fuente: Autor.

VI-F. Modelo de Red Convolucional

VI-F1. Análisis Matemático del Modelo TFLite en Dimensiones de Entrada: La ecuación [1 × 224 × 224 × 3], la dimensión de entrada al modelo es crítico para entender cómo se pretenden procesar las imágenes al principio. En primer lugar, se procesa siempre solo una imagen a la vez, que se conoce como tamaño de batch. En segundo lugar, la imagen tiene que tener un tamaño de 224×224 ya que el modelo ha sido entrenado para ello, con la excepción que no permite una dimensión distinta. La imágenes poseen tres canales, RGB, es decir, representan a valores en Rojo, Verde y Azul. Entonces, la dimensión de 224×224×3 supone que se está procesando una imagen en color, ya que existen modelos diferentes a la imagen en escala gris. Esto, a su vez, supone que cada píxel tendrá tres valores que están dentro del rango de 0–255 y podrían servir para describir la intensidad de cada color.

$$\text{Input Shape} = [1 \times 224 \times 224 \times 3] \tag{17}$$

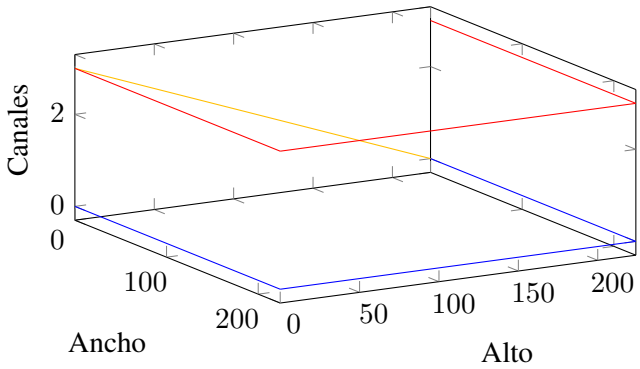


Figura 19: Representación 3D de las dimensiones de entrada del modelo TFLite. Fuente: Autor.

VI-F2. Capa de Salida: El vector de salida [1 × 4] en el código indica la arquitectura final de la red configurada. El 1 inicial en la lista continúa es el recuento de salidas de capas y coincide con el tamaño de lotes con los que el usuario espera calcular. El modelo en su conjunto procesará una imagen a la vez. El 4 es el número de salidas en total. En este caso, el modelo clasifica las imágenes en 4 categorías diferentes que son: lata hinchada, rasgada, golpeada y perforada. El siguiente conjunto de números especifica el rango de valores en el que cada elemento de esa serie puede estar. En este caso, siempre sumarán 1. Estos son valores de probabilidad para cada clase que una imagen dada podría contener. Por lo tanto, este resultado es típico de un problema de clasificación multiclase.

$$\text{Output Shape} = [1 \times 4] \tag{18}$$

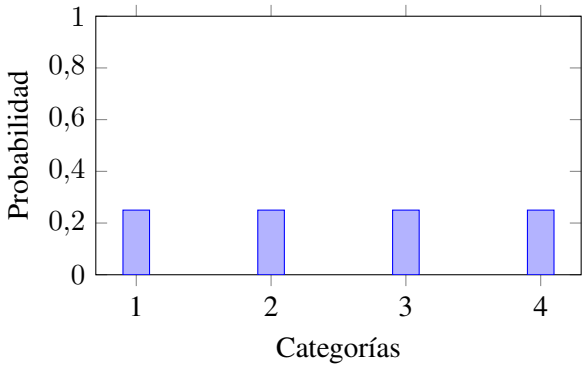


Figura 20: Distribución de probabilidades de la capa de salida.

VI-F3. Estadísticas de Tensores: La fórmula muestra cómo se calcula la media aritmética de n valores sumándolos entre sí y dividiendo por n. Esas áreas en la imagen lo muestran como el punto medio de una distribución normal, que es exactamente donde la línea punteada es efectivamente la media aritmética. La curva de campana es simétrica en torno a la media porque los datos se distribuyen naturalmente de esa manera alrededor.

Media:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{19}$$

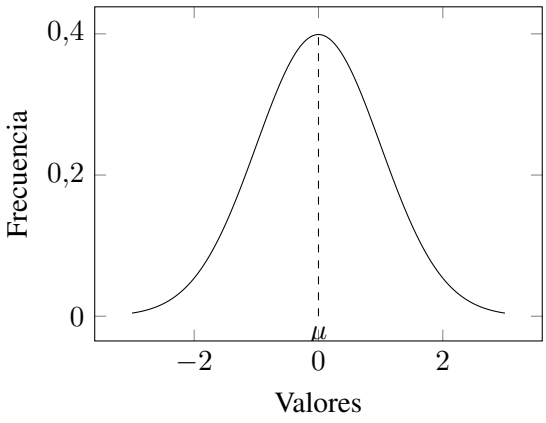


Figura 21: Distribución normal con la media indicada.

VI-F4. Estadísticas de desviación: La desviación estándar, es una medida de dispersión. Cualifica la manera en que los valores difieren unos de otros por parte de la media. Para las redes neuronales, es una métrica especial, ya que describe cómo las activaciones y pesos están distribuidos. En general, se calcula por la raíz cuadrática de la media de las desviaciones cuadráticas de la media para cada valor. Una sigma alto señala que los valores están distribuidos a aumentados, lo que puede ser un indicio del problema con los gradientes explosivos. También, si es demasiado baja, eso podría ser una señal que la red no está aprendiendo rasgos característicos de manera efectiva.

Desviación Estándar:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{20}$$

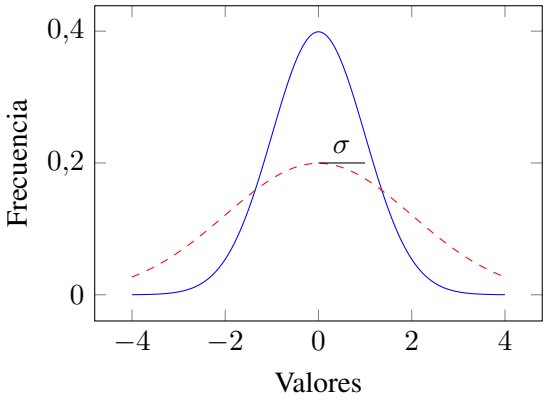


Figura 22: Distribución con desviación estándar representada gráficamente.

VI-F5. Métricas de Normalización:

$$x_{norm} = \frac{x - \mu}{\sigma} \tag{21}$$

La normalización no es más que un proceso que lleva a cabo una transformación de los datos a fin de dotarles de las propiedades estadísticas solicitadas, a saber, media 0 y desviación estándar 1. Dicho proceso consiste en restar a cada valor la media y dividir el resultado por la desviación estándar, latente en la fórmula. La normalización resulta vital por varias razones a la vez: capacita a diferentes características de aprender al mismo paso al margen de las escalas originales; acelera la marcha del entrenamiento al elaborar por defecto un paisaje de búsqueda más uniforme; y previene los problemas aritméticos que puedan emerger en el proceso de entrenamiento. Con respecto a las redes profundas, la normalización va de la mano no solo con los datos de entrada, también con las activaciones intermedias mediante normalización por lotes, normalización de capas y normalización de instancias.

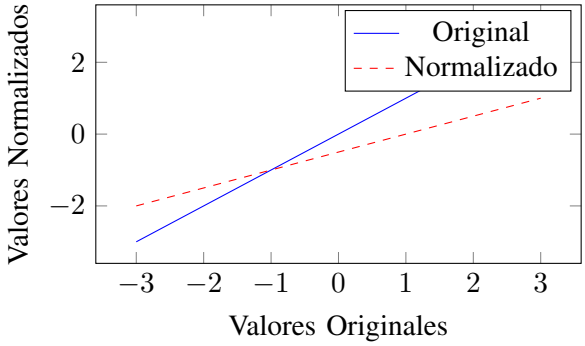


Figura 23: Normalización de valores respecto a la media y desviación estándar.

VI-F6. Rango de Activaciones:

$$-39{,}8223 \leq x \leq 33{,}0995 \tag{22}$$

El rango de activaciones [-39.8223, 33.0995] es el rango en que las señales pueden encontrarse en la red neuronal, vale la pena notar que este rango es importante por varias razones, una de estas razones es que este da una idea acerca de el rango en que se manejan las señales que fluyen a través de la red y luego el rango también pude

indicar problemas por estos tal como que la señal es muy grande y tengas problemas como desvanecimiento de gradientes, el rango también puede indicar si se necesita regularización o normalización en un rangos muy amplios puede indicar que el proceso de entrenamiento no es estable, por otro lado, un rango muy angosto pude indicar que la red no esta tomando considerando las variaciones presentes en los datos.

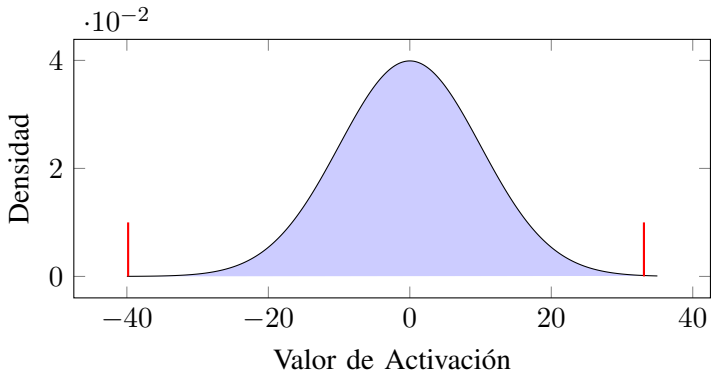


Figura 24: Distribución de valores de activación con rangos extremos destacados.

VI-F7. Estructura de Convolución: Reformula que es una cnn

$$\text{Conv2D}(x) = \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} \sum_{c=1}^{c_{in}} w_{ijc} \cdot x_{ijc} + b \tag{23}$$

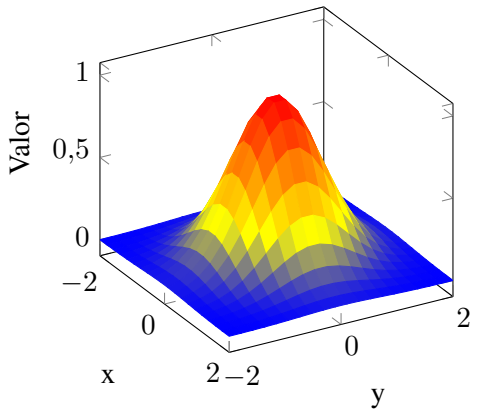


Figura 25: Superficie 3D que ejemplifica la convolución.

VI-F8. Función de Activación Final: La función softmax es una operación matemática esencial en aprendizaje profundo que convierte el resultado de una red neuronal, típicamente un vector de números llamados logits, en una distribución de probabilidad. Para cada entrada, exponencia la entrada y normaliza esas valiosas exponenciales dividiendo por la suma de todas las otras exponenciales. Por esa razón, la función softmax es importante.

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{4} e^{x_j}} \tag{24}$$

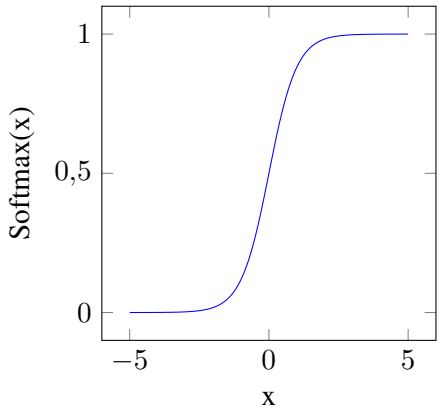


Figura 26: Representación gráfica de la función softmax.

VI-G. Simulación de rendimiento de la IA propuesta en Matlab

Este análisis trata del área de la acumulación y la distribución de parámetros, donde se observa la inspección sobre cuántos pesos se encuentran en las capas y se proporciona una idea general de la estructura del modelo. Permite identificar en qué capas se encuentran los recursos y acentuar en qué capas se puede realizar una regularización o reducción. De la regularización realizada en escala logarítmica y verifica qué capas afectan más en términos de órdenes de magnitud. A veces es útil saber qué capas afectan más el modelo en términos de complejidad para priorizar la optimización. La complejidad empírica también ayuda a entender cómo crece la carga computacional a medida que avanza por las capas, y se representan las etapas críticas. La densidad del gráfico circular nos muestra cuáles son las cinco entidades más caras en términos de parámetros, lo que permite enfocar en la optimización donde se agrupan la mayoría de los recursos.

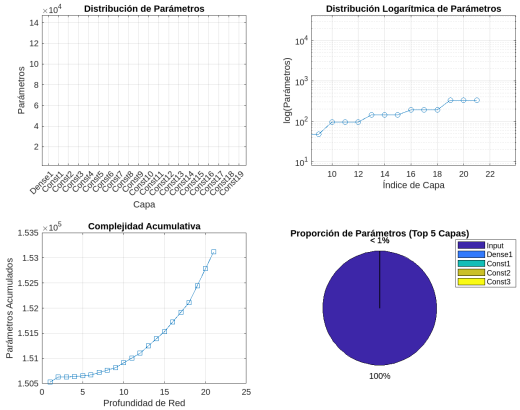


Figura 27: Análisis de Arquitectura y Complejidad

El proceso de evaluación de las características estadísticas de los datos manejados por cada tensor, son esenciales para asegurar la estabilidad numérica y la eficacia de los modelos. El boxplot proporciona una vista global de los valores más altos, más bajos y promedio de cada tensor, identificando posibles outliers o distribuciones anómalas. La dispersión estadística, que compara medias y desviaciones estándar, refleja la consistencia de los datos, y resalta la identificación de problemas de escala. El rango dinámico entre el valor máximo y mínimo, muestra la amplitud de variación dentro de cada tensor, y destaca los tensores con mayor sensibilidad a cambios relativo en los datos. El coeficiente de variación normaliza la variabilidad relativa entre los tensores, permitiendo comparar directamente tensores con escalas muy diferentes, una métrica importante para la estrategia de normalización.

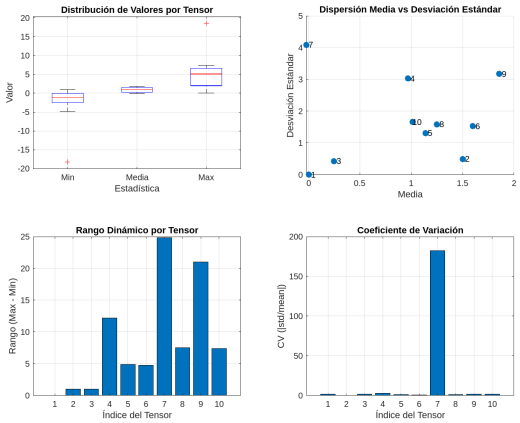


Figura 28: Análisis Estadístico de Tensores

Es importante analizar cómo puede cuantificar la red el uso de los recursos de hardware y su capacidad de procesamiento teórica. Por una parte, el gráfico de uso de memoria por capa estima el uso de la memoria. El gráfico muestra que se utiliza 32 bits para representar los datos y parámetros, proporciona la cantidad de datos sobre la que opera la red en cada capa y calcula el consumo de recursos. El análisis acumulativo de la memoria facilita ver cómo se incrementa la carga con cada capa adyacente y puede ser útil para determinar las oportunidades para la implementación de la red en dispositivos con recursos limitados. Por otra parte, la capacidad de cálculo teórica se estima por el número de operaciones de punto flotante. La eficiencia computacional puede expresarse en operaciones por cada byte de la memoria. Tasa de utilización de recursos computacionales se puede utilizar para identificar hasta qué punto los recursos se están empleando de forma eficiente en comparación con la memoria y señalar los errores de diseño.

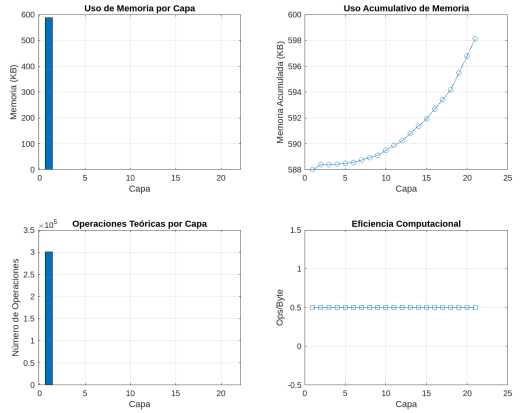


Figura 29: Análisis de Eficiencia y Rendimiento

El siguiente análisis muestra como se distribuyen los valores de los pesos y cómo deben ser preparados para el aprendizaje. El histograma muestra la cantidad de valores en diferentes rangos de valores de los pesos, de esta forma, permite captar inmediatamente los sesgos o la distribución no balanceada. En cambio, Q-Q plot compara la distribución empírica de los pesos y la esperada si la distribución fuera, por ejemplo, normal. Si existen desviaciones grandes de la linea de la distribución ideal, por ejemplo, los datos no siguen la distribución normal. La distribución acumulativa es simplemente la integral de la distribución, lo cual muestra la probabilidad acumulativa hasta el cierto valor de los pesos. La normalización trae los pesos en la misma escala lo cual mejora la convergencia durante el tiempo de entrenamiento y previene que los valores se hagan demasiado grandes causando inestabilidades numéricas.

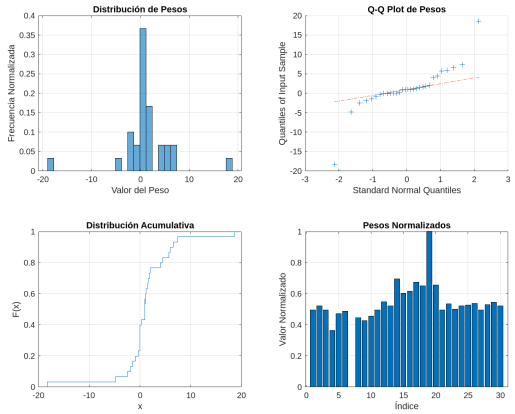


Figura 30: Análisis de Distribución y Normalización

Se presenta un análisis de varias estadísticas clave de los tensores están relacionadas entre sí. Podría ayudar a encontrar patrones globales del comportamiento del modelo. Para empezar, en la visualización del 'mapa de calor de correlaciones se muestra cómo las características tales como media, desviación estándar, mínimo, máxima están relacionadas entre sí. Se puede observar que hay dependencias fuertes que pueden influir en la estabilidad de nuestro modelo. En el análisis de dependencia media máximo se intenta utilizar la regresión lineal para encontrar una tendencia. En este caso, se podría intentar normalizar estos valores para que tengan un rango similar. El ratio de varianza es otro indicador que podría utilizarse.

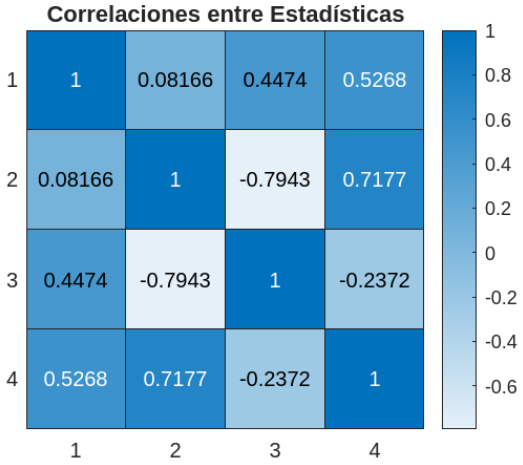


Figura 31: Análisis de Correlación y Dependencias

En esta sección se muestran las métricas clave para describir el rendimiento y la eficiencia del modelo términos globales. El número total de parámetros, su media y desviación estándar aclaran la complejidad producida por un modelo y su coherencia a la hora de distribuir recursos. La relación de compresión se ha verificado para el valor total y el de la capa más numerosa para detallar un modelo balanceado. Por último, la eficiencia global refleja qué tan efectivamente ha sido diseñado el modelo con el ratio de parámetros entonces dividido por la memoria.

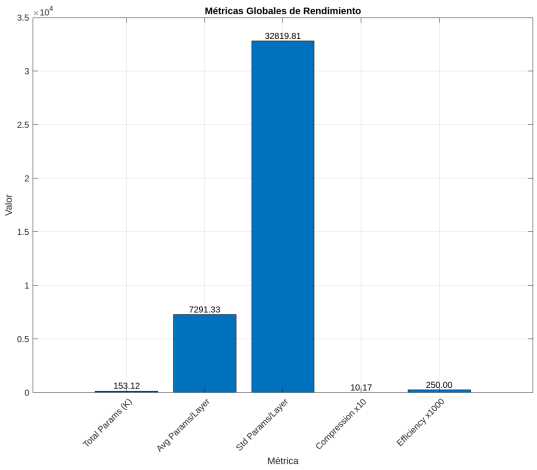


Figura 32: Métricas de Rendimiento Global

Este análisis empieza con la complejidad en capas por el tiempo de ejecución. Es de suponer que la complejidad temporal es fundamental para las procesos que manejan un volumen significativo de datos, como las convoluciones. Así, es posible ver en el gráfico de la distribución de complejidades relativas cuál es la proporción del tiempo estimado que está ocupada por cada capa. Por otro lado, la complejidad acumulativa muestra cómo las capas aumentan en su totalidad a medida que se van procesando, y es útil para planear optimizaciones temporales. En cuanto a la eficiencia temporal, se analiza cuántos parámetros útiles contribuyen al aumento de la complejidad, lo cual permite planear cómo mejorarlo para reducir los cuellos de botella computacionales.

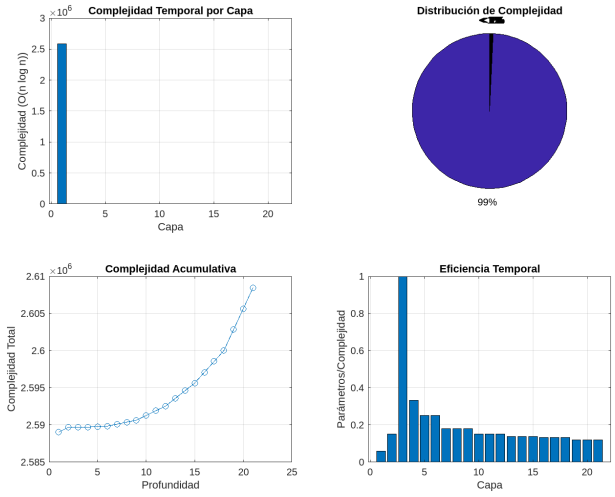


Figura 33: Análisis de Complejidad Temporal

VI-H. Diseño de estructura de programación de entrenamiento de la IA propuesta con python

Como se puede ver en la siguiente Figura 34 , la clase ClasificadorDefectosLata es el componente central del sistema, que tiene como objetivo clasificar defectos en las latas. La figura proporciona detalles sobre los atributos y métodos que constituyen la funcionalidad de la clase. En particular, muestra la definición de los datos de entrada aceptados por el clasificador, los algoritmos que implementa y las estructuras de datos utilizadas.

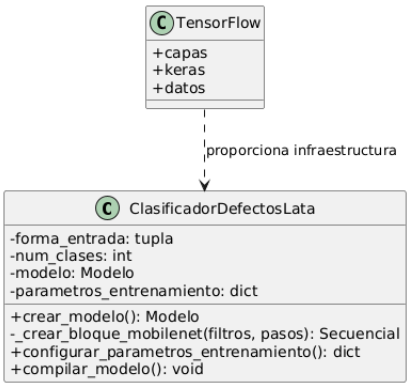


Figura 34: Estructura de la clase ClasificadorDefectosLata

En la Figura 35, se ilustra el diagrama de flujo completo de trabajo para la creación y capacitación del modelo de aprendizaje automático que beneficie al sistema de clasificación de defectos. Se presenta las etapas claves, que incluyen la recopilación y la adaptación del conjunto de datos de formación algoritmo, la toma decisión y los ajustes de formas sobre la arquitectura del modelo, la etapa iterativa de formación con validación, finalmente hasta la mejora del desempeño del modelo a través de la optimización de hiperparámetros.

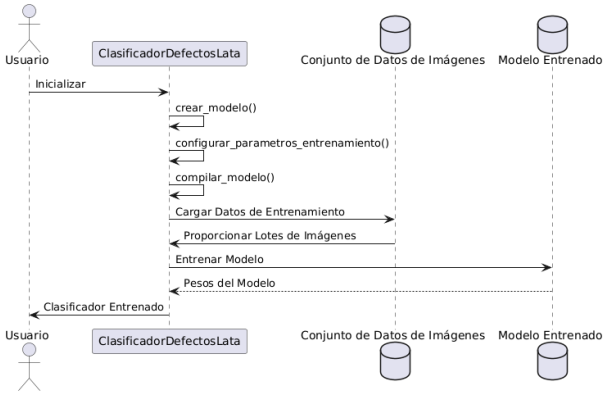


Figura 35: Proceso desde la inicialización hasta el entrenamiento del modelo

En el diagrama general de estados, se describen los estados a través de los cuales un modelo de machine learning en su ciclo de vida. Especifica las transiciones entre los estados, incluido el inicio, el preprocesamiento, el entrenamiento, la valoración del rendimiento y finalmente, la producción posterior al entrenamiento. Por lo tanto, esta esquematización ilustra la lógica del control sobre los estados y las dependencias que existen entre las etapas de desarrollo del modelo.

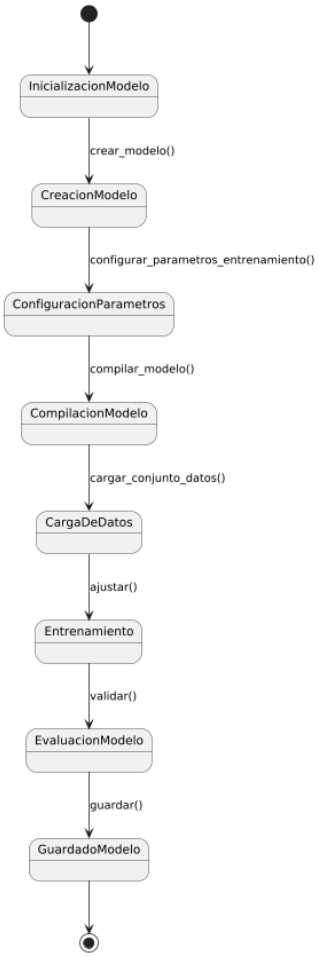


Figura 36: Transiciones de estado durante la creación y entrenamiento del modelo

La estructura de la red neuronal empleada como núcleo del modelo de clasificación de latas defectuosas se muestra en la Figura 37. Este esquema revela las diferentes capas de la red, es decir, la entrada que toma el conjunto de datos, las capas ocultas múltiples que extraen y procesan sus características y la capa de salida que hace predicción.

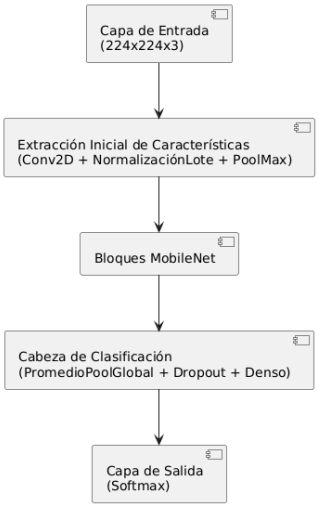


Figura 37: Estructura de capas de la red neuronal

VI-I. Arquitectura del Modelo

El esquema muestra un sistema de detección basado en una Raspberry Pi. Una cámara captura video y lo envía a la Raspberry Pi, que procesa los frames utilizando un modelo de TensorFlow Lite para realizar predicciones. Con base en estas predicciones, la Raspberry Pi puede controlar un módulo de relé para activar o desactivar la electro-válvula y esta a su vez activando el actuador neumático. Además, el sistema incluye una interfaz gráfica creada con Tkinter, que permite visualizar y actualizar la información en tiempo real. La comunicación entre los componentes es clave para la detección eficiente y la automatización de tareas.

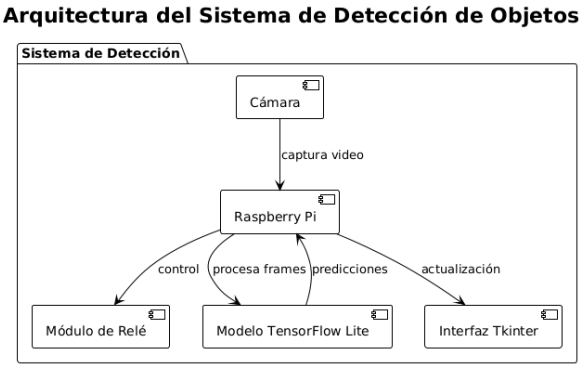


Figura 38: Arquitectura Integral del Sistema de Detección Automatizado de Objetos Defectuosos

La Figura 39 muestra la secuencia temporal de eventos en el procesamiento de imágenes. Comienza con la adquisición de datos desde la cámara, seguida del preprocesamiento para normalización e incremento del contraste. Luego, se aplican algoritmos de segmentación y extracción de características relevantes, que finalmente alimentan un clasificador. Cada etapa se diseña para minimizar la latencia y maximizar la precisión del sistema, permitiendo un procesamiento eficiente en sistemas de visión computacional embebidos.

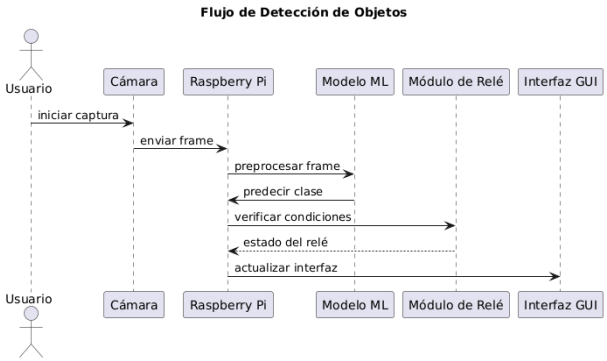


Figura 39: Secuencia Temporal de Procesamiento de Imágenes en Sistemas de Visión Computacional

La Figura 40 muestra una máquina de estado que define la lógica operativa de un sistema de detección automatizada. Como se muestra, los estados del sistema son dentro de Incialización, Recolección de Datos, Procesamiento y Decisión. Adicionalmente, se presentan las transiciones de los estados que están controlados por ciertas condiciones lógicas hechas por las entradas y las salidas obtenidas del modelo de clasificación.

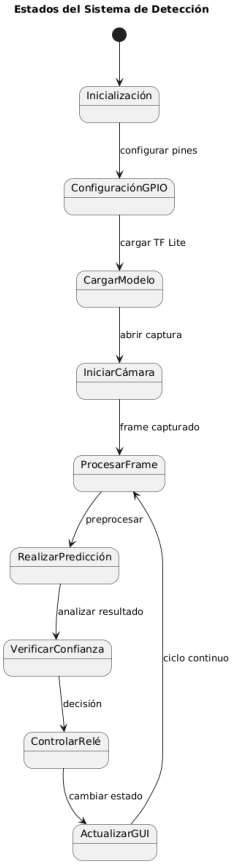


Figura 40: Transiciones de Estado y Lógica de Control en Sistemas de Detección Automatizada

La Figura 41 muestra los componentes con los que opera un sistema de inspección automatizada que emplea machine learning. Entre ellos, se encuentran módulos de hardware, como cámaras y unidades de procesamiento, y data pipelines y redes neuronales como componentes de software. El procedimiento de inspección conlleva la

recopilación de datos a través de las cámaras y su procesamiento mediante los pipelines configurados para limpiar y estructurar los mismos. Finalmente, el uso de la información recolectada para optimizar el rendimiento del proceso de inspección es considerado en este modelo.

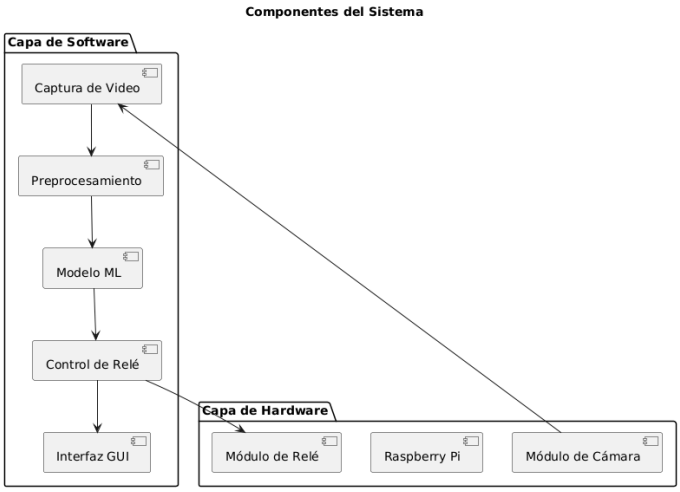


Figura 41: Modelo de Componentes para Sistemas de Inspección Automatizada con Machine Learning

La Figura 42 describe la distribución física y lógica de un sistema embebido diseñado para detección inteligente. Incluye nodos de captura de datos conectados a una unidad central de procesamiento mediante buses de comunicación. La topología considera factores como la estabilidad y el consumo de energía y latencia. Además, incluye componentes para conectividad remota, que permiten la supervisión y la actualización del sistema.

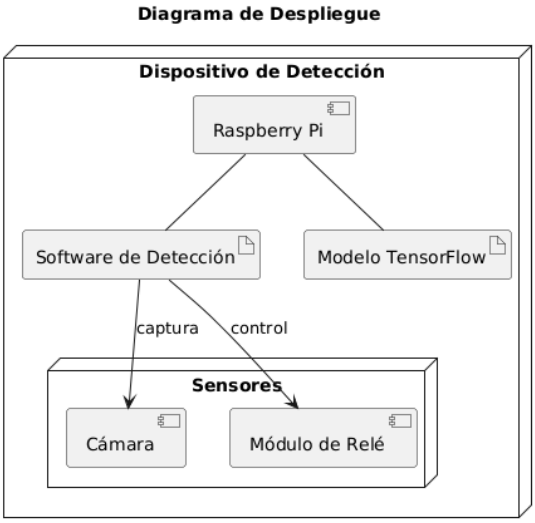


Figura 42: Topología de Despliegue de Sistemas Embebidos de Detección

La siguiente Figura 43 se puede apreciar un mapa de flujo que describe las operaciones de un sistema de categorización utilizando técnicas de aprendizaje profundo. Comenzando de la adquisición inicial de datos hasta la inferencia del modelo, todo se encuentra optimizado para minimizar los errores y ofrecer máxima precisión. Además, se pueden apreciar las operaciones de retroalimentación suplementarias, que permiten ajustes automáticos

en caso de cualquier anomalía en el rendimiento o los datos de entrada, lo que, a su vez, garantiza un desarrollo adicional del sistema.

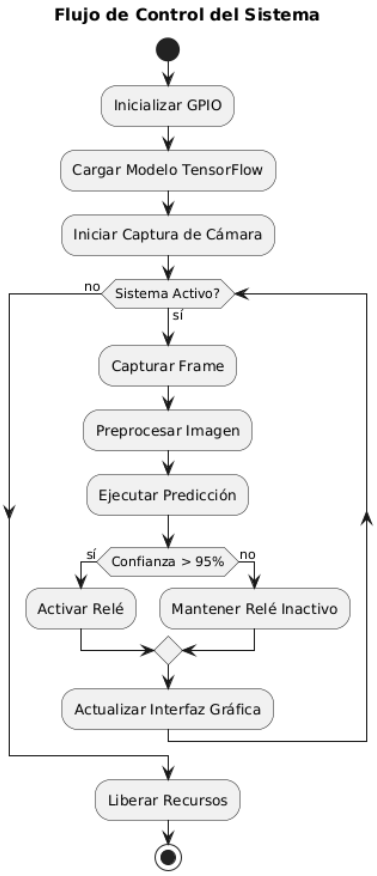


Figura 43: Mapa de flujo de Control para Sistemas de Clasificación Automática con Aprendizaje Profundo

VI-J. Estructura de procesamiento de imágenes

La Figura 44 muestra un modelo que representa la estructura de la inteligencia artificial, para procesar las imágenes. Se utiliza una imagen de entrada de 224 x 224 píxeles con 3 canales de color. Por lo tanto, se tiene un tensor de tamaño [1 x 224 x 224 x 3]. En el primer bloque, Conv2D Block, se aplican 64 filtros de convolución de 3 x 3 con activación ReLU y provoca mapa de características de 112 x 112. Luego, éstas se reducen a 56 x 56 mediante MaxPooling2D, ReLU 2 x 2. Además, en la etapa de Feature Maps se extraen múltiples representaciones de características principales, al mismo tiempo, en la capa Dense Layers se clasifica la imagen en 5 clases de salida. Así, esta capa utiliza varias técnicas como la normalización: $\left(\frac{x-\mu}{\sigma}\right)$, dropout: 0.5, tamaño de batch: 32, tasa de aprendizaje: 0.001 y Adam optimizer para actualiar el desempeño y la capacidad de universalización. Las clases de salida de C1, C2, C3, C4, C5.

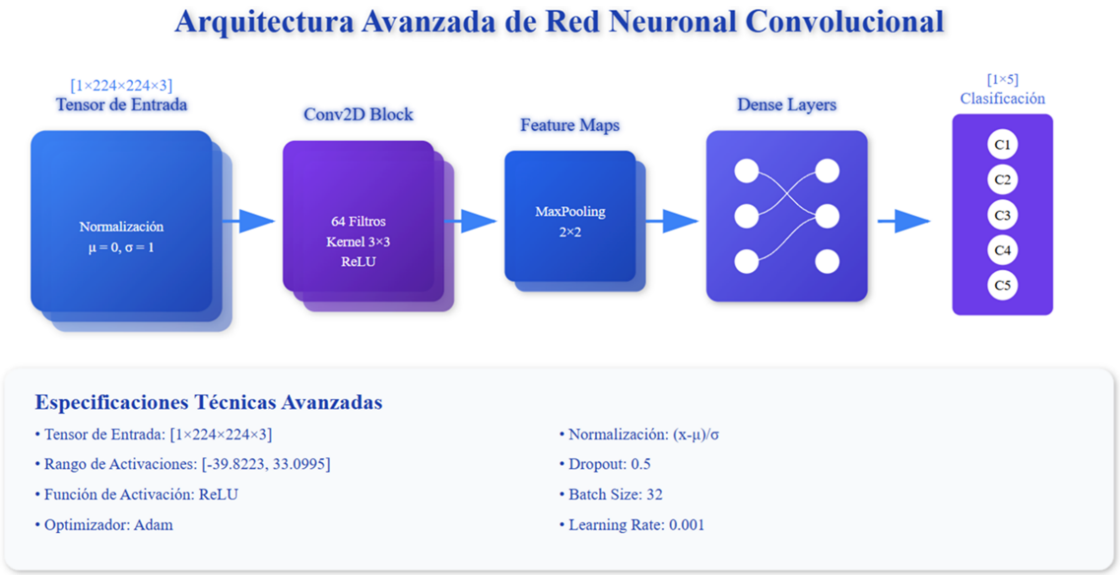


Figura 44: Representación estructural de la inteligencia artificial

VI-K. Dataset de imágenes

En la siguiente Figura 45, se muestra las carpetas con su respectivo nombre asignado para las fotos.

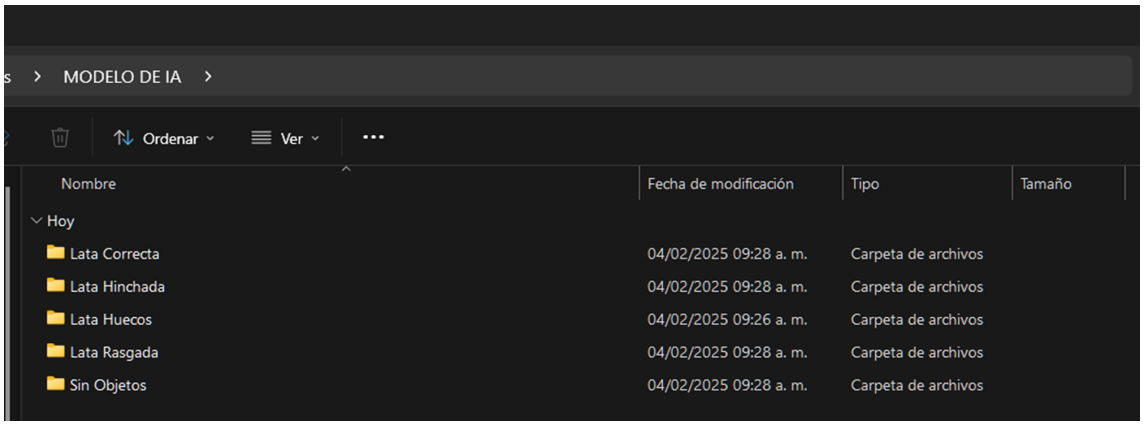


Figura 45: Carpetas contenidas de imágenes para entrenamiento. Fuente: Autor.

VI-K1. Interfaz principal de Rasbian Os:

La siguiente Figura 46 se puede ver la interfaz principal de Rasbian Os de la Raspberry pi 4 modelo B, la cual muestra la ejecución de la IA a través de la terminal del sistema y de respectiva clasificación en base a los datos que recoge la cámara a través de la conexión del puerto USB del dispositivo.

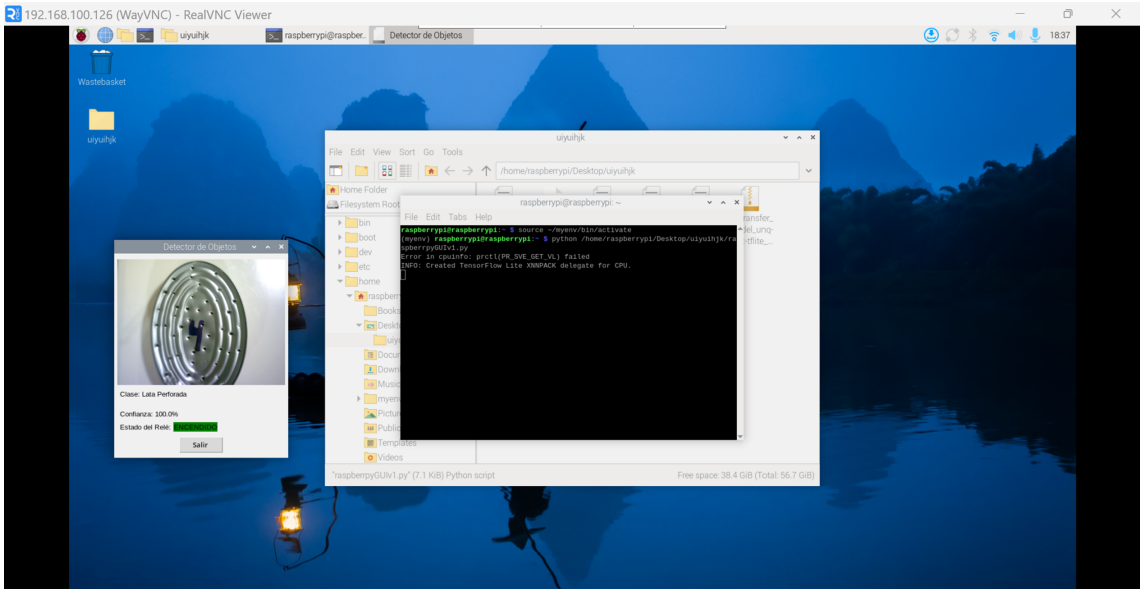


Figura 46: Interfaz de rasbian os. Fuente: Autor.

VI-K2. Detección sin objetos:

Como se observa en la Figura 47, la detección sin objetos capturada por la IA en la banda transportadora, la cual no permite que relé se active, ya que al no tener ningún objeto en presencia de la cámara para su reconocimiento, este no lo clasifica como un objeto en buen o mal estado.

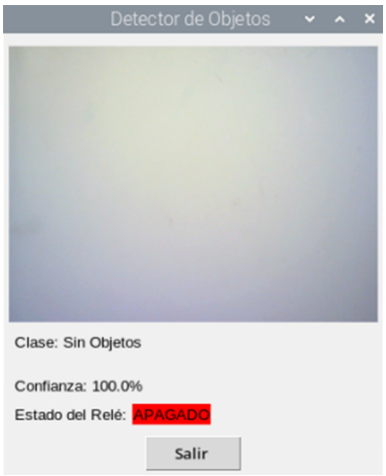


Figura 47: Detección sin presencia de objeto. Fuente: Autor.

VI-K3. Detección de lata buena:

En la Figura 48 se visualiza, la detección de una lata considerada como el caso 1, el cual es clasificado por la IA en la categoría de lata correcta, la cual hace que el relé no se active, ya que al ser un lata en buen estado esta se considera que no debe ser desechada.

Figura 48: Detección de lata en buen estado. Fuente: Autor.

VI-K4. Detección de lata hinchada:

En la Figura 49 se ve, la detección de una lata con el caso 2, el cual corresponde a la clasificación de la IA a la clase de lata hinchada, en base a las condiciones de la estructura del cuerpo de la lata, lo cual hace que el relé se active, ya que al ser una lata en mal estado, esta se considera que debe ser desechada.

Figura 49: Detección de lata mala - hinchada. Fuente: Autor.

VI-K5. Detección de lata rasgada:

Como observa en la siguiente Figura 50, la detección de una lata apropiado al caso 3, el cual es clasificado por la IA a la clase de lata rasgada en base a la apertura en el centro del cuerpo de la lata, lo cual hace que el relé se active, ya que al ser un lata en mal estado se lo designa como un elemento que debe ser desechado respectivamente.

Figura 50: Detección de lata mala - rasgada. Fuente: Autor.

VI-K6. Detección de lata perforada:

En la Figura 51 se aprecia, la detección de una lata con el caso 4, el cual corresponde a la clasificación de la IA a la clase de lata perforada en base a la cantidad de huecos presentes en el cuerpo de la lata, lo cual hace que el relé se active, ya que al ser un lata en mal estado esta se considera que debe ser desechada.

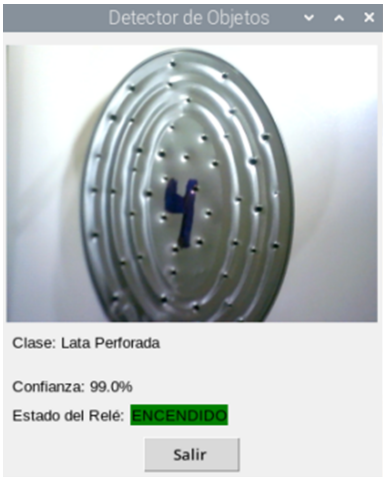


Figura 51: Detección de lata mala - perforada. Fuente: Autor.

VI-L. Implementación de la red

Para una interacción con la red neural y la Raspberry pi 4 modelo B es necesario realizar las siguientes conexiones para poder visualizar y poner a prueba los datos ingresado dentro de la misma raspberry, de tal manera que las pruebas con la lata se realice y poder corregir posiciones.

VI-L1. Conexión a computadora portátil y mouse:

Para realizar la conexión de la raspberry a la computadora portátil se necesita de un cable HDMI a micro HDMI, el mouse que también ingresa para la manipulación del mismo .

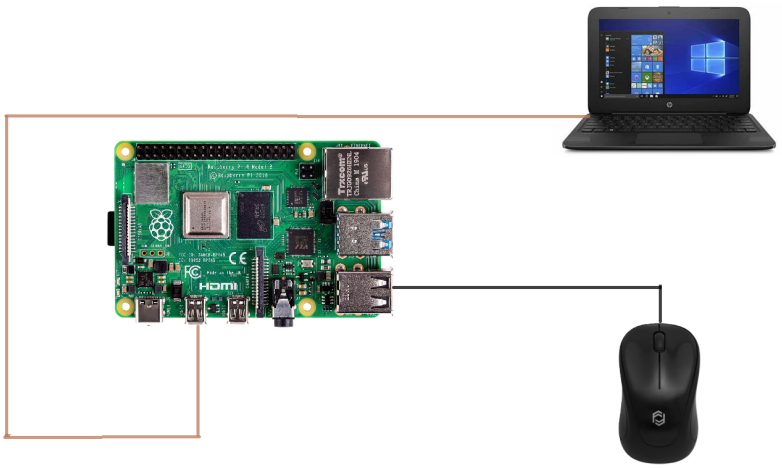


Figura 52: Conexión raspberry pi 4 a laptop. Fuente: Autor.

VI-L2. Conexión a cámara y electro-válvula:

La conexión de la cámara a la raspberry, para realizar la captura de las imágenes de las latas en la banda, cuando una lata esta en mal estado, activara un relé de 5V., que permitirá que se active la electro-válvula de 110 VAC, 5/2 vías, hacia una válvula solenoide de doble efecto, para que se separe de la banda la lata en mal estado, como se visualiza en el siguiente gráfico las conexiones Figura 53.

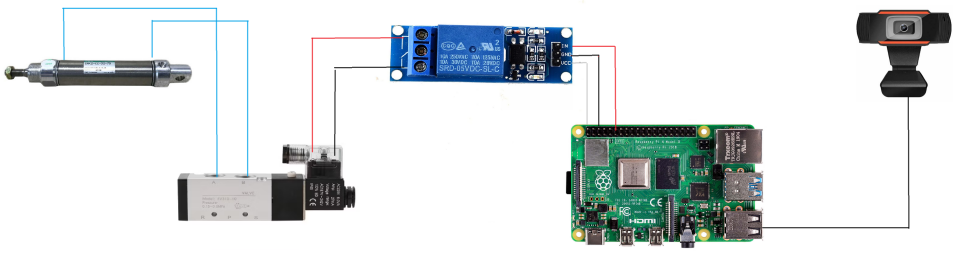


Figura 53: Conexión cámara y electro-válvula. Fuente: Autor.

VII. Resultados

VII-A. Evaluación de rendimiento de la red

VII-A1. Validación sin objeto:

Se realizan 20 pruebas, para verificar si puede detectarse si existe un objeto dentro de la banda, se elabora una gráfica como se observa en la Figura 54, se verifica el escenario en términos de desempeño en esta categoría, con un 98.8 % de precisión. Se valida, que se puede detectar si existe una lata. La poca variabilidad en los resultados demuestra que el sistema es muy confiable en esta tarea.

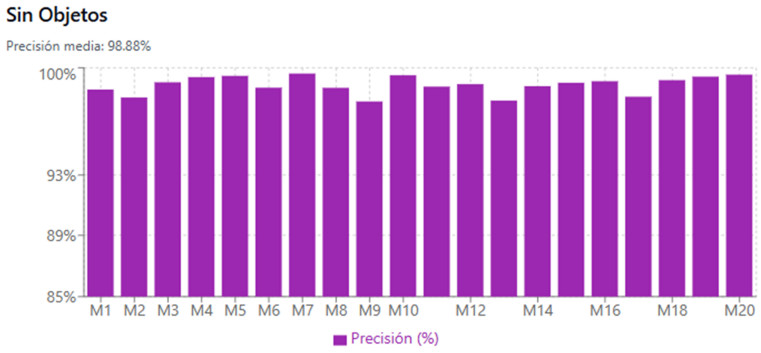


Figura 54: Gráfica de detección sin objetos. Fuente: Autor.

VII-A2. Validación de lata en buen estado:

En la Figura 55 se aprecia una clasificación de gran precisión de las latas en buen estado, logrando un 98.5 % de aciertos. Esto significa que su reconocimiento es bastante alto en relación con las latas dañadas, lo que ayuda a mejorar la producción. Su desempeño es correcto en diferentes condiciones, asegurando que las latas en buen estado no sean descartadas por error.

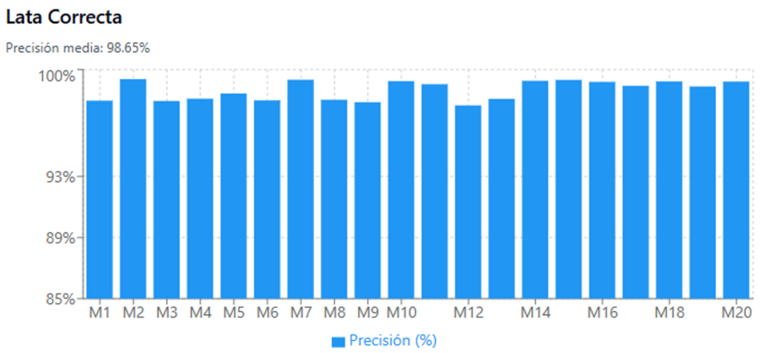


Figura 55: Gráfica de detección de lata en buen estado. Fuente: Autor.

En la Figura 56, la cámara procesa la imagen de la conserva enlatada y clasificándola como lata buena, el actuador neumático no entra en funcionamiento debido que la lata esta en buen estado y avanza a su siguiente proceso.

Figura 56: Procesamiento de la imagen de lata en buen estado. Fuente: Autor.

VII-A3. Validación de lata hinchada:

En la Figura 57 se aprecia la detección general de una lata hinchada, la cual ronda un valor promedio de precisión del 97.09 % lo que es considera un poco menos precisa en comparación con latas correctas. Esto es debido a que no se identifican perfectamente porque la hinchazón no es homogénea en todas las latas, esto podría generar ciertas complicaciones al momento de ser detectada.

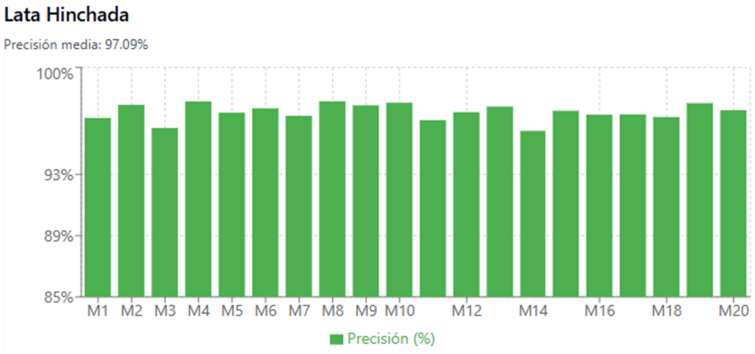


Figura 57: Gráfica de detección de lata en buen mal estado - hinchada. Fuente: Autor.

En la Figura 58, la cámara procesa la imagen de la conserva enlatada y clasificándola como lata hinchada, el actuador neumático entra en funcionamiento debido que la lata esta en mal estado y debe ser separada del proceso.

Figura 58: Procesamiento de la imagen de lata hinchada en mal estado. Fuente: Autor.

VII-A4. Validación de lata rasgada:

En la Figura 59 se observa que los resultados corresponden al caso más difícil para el modelo, ya que las rasgaduras son delgadas y poco visibles al momento de moverse el objeto por la banda transportadora. Aunque la precisión es del 94.5 % sigue siendo alta, pero es la más baja entre todas anteriores categorías analizadas. Esto significa que hay una mayor probabilidad de que el modelo confunda una lata rasgada con otro tipo de defecto.

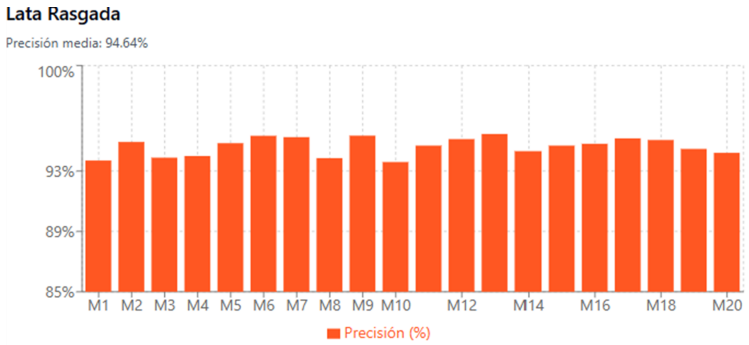


Figura 59: Gráfica de detección de lata en buen mal estado - rasgada. Fuente: Autor.

En la Figura 60, la cámara procesa la imagen de la conserva enlatada y clasificándola como lata rasgada, el actuador neumático entra en funcionamiento debido que la lata esta en mal estado y debe ser separada del proceso.

Figura 60: Procesamiento de la imagen de lata rasgada en mal estado. Fuente: Autor.

VII-A5. Validación de lata perforada:

En la Figura 61 se visualiza un buen desempeño al detectar latas con perforaciones, aunque su precisión es ligeramente menor que en comparación con las otras detecciones. Esto se debe a que los huecos pueden aparecer en diferentes tamaños y ubicaciones, lo que hace que sea más difícil de clasificar. A pesar de esto, un 95.2 % de aciertos sigue siendo un resultado bastante favorable a nivel de clasificación.

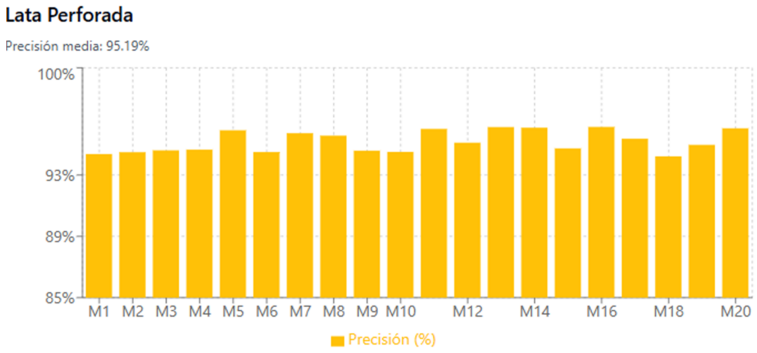


Figura 61: Gráfica de detección de lata en buen mal estado - perforada. Fuente: Autor.

En la Figura 62, la cámara procesa la imagen de la conserva enlatada y clasificándola como lata perforada, el actuador neumático entra en funcionamiento debido que la lata esta en mal estado y debe ser separada del proceso.

Figura 62: Procesamiento de la imagen de lata perforada en mal estado. Fuente: Autor.

VIII. CRONOGRAMA

Actividades a desarrollar en el periodo de la elaboración del proyecto

Tabla I: Cronograma

DESARROLLO DE SISTEMA DE BANDA TRANSPORTADORA PARA LA DETECCIÓN DE DEFECTOS EN CONSERVAS ENLATADAS POR MEDIO DE VISIÓN ARTIFICIAL.					
Titulo: Cronograma de actividades					
Actividad	Actividad especifica	Vigencia 2024 - 2025			
		Nov	Dic	Ene	Feb
Investigación.	Profundizar en investigaciones previas sobre el tema.	1	1	1	
Diseño de la estructura.	Diseñar una estructura de soporte que sea capaz de sostener y mantener en funcionamiento.	1	1	1	
Fabricación de la estructura.	Elegir los materiales adecuados para la construcción de la estructura.	1	1	1	1
Programación de algoritmo.	Crear un algoritmo eficiente y preciso para resolver el problema planteado.		1	1	1
Prueba de algoritmos para la clasificación.	Crear conjuntos de datos de prueba para evaluar el rendimiento del algoritmo.			1	1
Integración de componentes con la estructura.	Unir los diferentes componenten del sistema, configurar los parámetros del sistema.			1	1
Prueba de funcionamiento.	Verificar que los componentes interactúen correctamente entre sí.			1	1
Validación.	Verificar que el sistema cumple con los estándares y regulaciones aplicables.				1

IX. PRESUPUESTO

Nombre del elemento	Descripción	Cantidad	Valor total
Cámara	Dispositivo óptico	1	23$
Raspberry pi 4	Computadora de placa	1	50 $
Motor	Monofasico	1	50 $
Banda transportadora	Grado alimenticio	1	60 $
Tubos	Acero	2	4$
Rodamiento	Bolas	4	8 $
Plancha de acero	1.5mm	1	80 $
Sensor neumático	Actuador	1	20 $
Electrovalvula	Dispositivo electrónico	1	20.5 $
Racor recto	1/8X4MM	1	2.9 $
Silenciador	1/8 XCPC	1	1.7 $
Manguera neumática	Diam 4mm	1	1 $
Relé	interruptor	1	2.5 $
Botón de encendido	Componente electrónico	1	4 $
Perno	1/4*3/4	15	3 $
Tuerca	1/4	15	3 $
Polea de aluminio	2 pulgadas	1	6 $
Polea de aluminio	3 pulgadas	1	8 $
Banda	Tipo A	1	4 $
SUBTOTAL MAQUETA			351.6$
Mano de obra	corte y construcción	1	100$
transporte	movilización		20$
Alimentación			5$
SUBTOTAL COSTOS LOGÍSTICOS			125$
TOTAL(MAQUETA + COSTOS LOGÍSTICOS)			476.6$

X. Conclusiones

Se realizó el diseño de la banda transportadora, para la cual se realizaron análisis matemáticos y simulaciones, para determinar el correcto funcionamiento del equipo, para poder ser integrado con el sistema de visión artificial para la clasificación de conservas de latas.

Se desarrolló y diseñó una red neuronal convolucional en base a las condiciones necesarias de funcionamiento y rendimiento, para adaptarlo a una raspberry pi 4 modelo b.

Se logró el correcto entrenamiento de la red en base a un dataset con las características de los elementos que se clasificó, con el fin de optimizar el rendimiento al momento de realizar la clasificación de latas en base a diversas condiciones en su estructura física.

Se consiguió el mejor desempeño de la red mediante una optimización del código y entrenamiento de 50 épocas, en relación a las condiciones de hardware de la raspberry pi 4 modelo b, con el propósito de obtener una correcta productividad al momento de clasificar las latas correctas de las defectuosas que pasan por la banda transportadora.

XI. Recomendaciones

Se sugiere mantener de manera estable la posición de la cámara, ya que perturbaciones mayores en el movimiento, pueden provocar que la cámara no gestiones de forma eficiente los datos que captura, generando un problema en el análisis de los resultados.

Al momento de cambiar los elementos del dataset, se debe volver a entrenar la red, ya que esta tiene actualizarse en base a los nuevos elementos que se vayan a utilizarse, para establecer las nuevas claves en su entrenamiento y así mantener un correcto funcionamiento.

Para futuras investigaciones se sugiere realizarla en una jetson nano, su GPU optimizada para inteligencia artificial, permite un procesamiento más eficiente de redes neuronales, logrando detecciones más rápidas y precisas. Sin embargo, su capacidad de manejar modelos más complejos sin sacrificar rendimiento la transforma en una opción ideal para aplicaciones industriales o que requieran análisis en tiempo real.

Al momento de emplear el dispositivo se sugiere calibrarlo en el lugar donde va ser utilizado, ya que cada entorno tiene distintos parámetros en tonalidades y porcentaje de intensidad de luz, lo cual puede reducir en el rendimiento de la red.

Referencias

[1] W. S. P. Santacruz, «Implementación de un sistema de visión artificial en líneas de producción de atún, para la detección de fallas de codificado en la empresa Puertomar S.A, Provincia de Manabí.,» 2020. dirección: http://dspace.espoch.edu.ec/handle/123456789/14101.

[2] T. G. Vitkova, «Medical Review and Analysis of Canned Food Production Safety,» *Journal of Biomedical and Clinical Research*, vol. 15, 1 2022. DOI: 10.2478/jbcr-2022-0001.

[3] S. M. ElShehawy y Z. S. Farag, «Safety assessment of some imported canned fish using chemical, microbiological and sensory methods,» *Egyptian Journal of Aquatic Research*, vol. 45, 4 2019, ISSN: 20903278. DOI: 10.1016/j.ejar.2019.08.005.

[4] J. Andrade-Muñoz, «Entendiendo el poder de la Inteligencia Artificial,» *TEPEXI Boletín Científico de la Escuela Superior Tepeji del Río*, vol. 10, 20 2023. DOI: 10.29057/estr.v10i20.10807.

[5] I. H. Y. Yim y J. Su, «Artificial intelligence (AI) learning tools in K-12 education: A scoping review,» *Journal of Computers in Education*, 2024, ISSN: 21979995. DOI: 10.1007/s40692-023-00304-9.

[6] J. G. Ferreira, «Análisis de la inteligencia artificial en las relaciones laborales,» *CES Derecho*, vol. 13, 1 2022. DOI: 10.21615/cesder.6395.

[7] P. O. Skobelev y S. Y. Borovik, «On the way from Industry 4.0 to Industry 5.0: From Digital Manufacturing to Digital Society,» *International Scientific Journal*, vol. II, 6 2017.

[8] B. Menezes, M. Yaqot, S. Hassaan, R. Franzoi, N. Alqashouti y A. Al-Banna, «Digital Transformation in the Era of Industry 4.0 and Society 5.0: A perspective,» en *2022 2nd International Conference on Emerging Smart Technologies and Applications, eSmarTA 2022*, 2022. DOI: 10.1109/eSmarTA56775.2022.9935399.

[9] C. C. Cheng, E. Chung y N. Correa, «inteligencia Artificial y su Impacto en la Industria de la Ingeniería,» *REICIT*, vol. 3, 1 2023. DOI: 10.48204/reict.v3n1.3948.

[10] eurostat, «Línea de Producción,» vol. 24, 2023. dirección: https://ec.europa.eu/eurostat/web/products-eurostat-news/w/ddn-20230724-1.

[11] C. D. Puga, M. D. Martínez, J. Angel y L. Solano, *EL PAPEL DE LOS ALIMENTOS ENLATADOS EN LA SALUD*, ago. de 2019.

[12] R. S. R. Cautivo, *Inspección de conservas de pescado para su custodia para la empresa Alma Perú S.A*, ago. de 2024.

[13] emergencyprepguy, «Identificación de alimentos enlatados en mal estado,» vol. 25, 2024. dirección: https://emergencyprepguy.com/best-canned-food-for-survival/.

[14] M. con salud, «Identificación de alimentos enlatados en mal estado,» vol. 26, 2022. dirección: https://www.nexofin.com/notas/1004817-cinco-formas-de-abrir-una-lata-sin-un-abrelatas-n-/.

[15] preventive vet, «¿Son seguras las latas de comida para mascotas abolladas?,» vol. 27, 2024. dirección: https://www.preventivevet.com/pets/are-dented-pet-food-cans-safe.

[16] L. Villarreal Ger, «Prototipo para la detección y clasificación de productos alimenticios mediante visión artificial en base al color,» *CONECTIVIDAD*, vol. 5, n.º 2, págs. 46-62, feb. de 2024. DOI: 10.37431/conectividad.v5i2.129. dirección: https://revista.ister.edu.ec/ojs/index.php/ISTER/article/view/129.

[17] Alicia, «¡La visión artificial puede hacer más de lo que podrías pensar!,» vol. 28, 2019. dirección: https://www.calvek.com/vision-artificial/.

[18] J. Sanchez-Romero y J. Llerena-Izquierdo, «Revisión de la literatura sobre el uso del aprendizaje profundo enfocado en sistemas de inspección ópticos automatizados para la detección de defectos superficiales en el sector de la manufactura,» *Revista InGenio*, vol. 6, 2 2023. DOI: 10.18779/ingenio.v6i2.680.

[19] F. L. Badillo, C. A. R. Hernández, B. M. Narváez e Y. E. A. Trillos, «Redes neuronales convolucionales: un modelo de Deep Learning en imágenes diagnósticas. Revisión de tema,» *Revista colombiana de radiología*, vol. 32, 3 2021, ISSN: 0121-2095. DOI: 10.53903/01212095.161.

[20] B. Pang, E. Nijkamp e Y. N. Wu, *Deep Learning With TensorFlow: A Review*, 2020. DOI: 10.3102/1076998619872761.

[21] P. Reiser, A. Eberhard y P. Friederich, «Graph neural networks in TensorFlow-Keras with RaggedTensor representation (kgcnn)[Formula presented],» *Software Impacts*, vol. 9, 2021, ISSN: 26659638. DOI: 10.1016/j.simpa.2021.100095.
[22] «Keras Applications Alternatives,» 2022. dirección: https://awesomeopensource.com/project/gyunggyung/Keras-Applications?mode=.
[23] «¿Cómo funciona una banda transportadora?,» 2020. dirección: https://sdindustrial.com.mx/blog/bandas-transportadoras/.
[24] «Cinta transportadora de rodillos: cómo funciona y sus ventajas,» dirección: https://www.vulcacas.com/blog/cinta-transportadora-de-rodillos-como-funciona-y-sus-ventajas/.
[25] «Cintas transportadoras bandas PVC,» dirección: https://www.talleressalo.es/es/fotos/img/7672701/.
[26] «Cintas transportadoras,» dirección: http://www.mbi.com.ar/maquinas/cintas-transportadoras/.
[27] «Cinta Transportadora Bandas Modulares,» dirección: https://www.talleressalo.es/es/productos/cintas-transportadoras/cinta-transportadora-con-bandas-modulares/.

ANEXO A

CÓDIGO DE PYTHON 3 V1 DE RED NEURONAL PARA RECONOCIMIENTO POR VISIÓN ARTIFICIAL

A continuación se observa el código empleado:

```
1
2  import tensorflow as tf
3  from tensorflow.keras import layers, models
4  import numpy as np
5  import cv2
6  import os
7  from sklearn.model_selection import train_test_split
8
9  class ClasificadorLatas:
10     def __init__(self, img_height=224, img_width=224):
11         self.img_height = img_height
12         self.img_width = img_width
13         self.model = self._build_model()
14         self.clases = ['Lata Correcta', 'Lata Hinchada', 'Sin Objetos', 'Lata Rasgada']
15
16     def _build_model(self):
17         model = models.Sequential([
18             layers.Input(shape=(self.img_height, self.img_width, 3)),
19             layers.Conv2D(32, 3, padding='same', activation='relu'),
20             layers.MaxPooling2D(),
21             layers.Conv2D(64, 3, padding='same', activation='relu'),
22             layers.MaxPooling2D(),
23             layers.Conv2D(128, 3, padding='same', activation='relu'),
24             layers.MaxPooling2D(),
25             layers.Flatten(),
26             layers.Dense(128, activation='relu'),
27             layers.Dropout(0.5),
28             layers.Dense(4, activation='softmax')
29         ])
30         return model
31
32     def preparar_datos(self, ruta_base):
33         """
34         Prepara los datos desde la ruta espec fica:
35         xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
36         """
37         imagenes = []
38         etiquetas = []
39         carpetas = ['Lata Correcta', 'Lata Hinchada', 'Sin Objetos', 'Lata Rasgada']
40
41         for idx, carpeta in enumerate(carpetas):
42             ruta_carpeta = os.path.join(ruta_base, carpeta)
43             if not os.path.exists(ruta_carpeta):
44                 print(f"Advertencia: La carpeta {ruta_carpeta} no existe")
45                 continue
46
47             for nombre_imagen in os.listdir(ruta_carpeta):
48                 ruta_imagen = os.path.join(ruta_carpeta, nombre_imagen)
49                 try:
50                     imagen = cv2.imread(ruta_imagen)
51                     if imagen is None:
52                         print(f"Error al cargar la imagen: {ruta_imagen}")
53                         continue
54                     imagen = cv2.resize(imagen, (self.img_width, self.img_height))
55                     imagen = imagen / 255.0  # Normalizaci n
56                     imagenes.append(imagen)
57                     etiquetas.append(idx)
58                 except Exception as e:
59                     print(f"Error procesando {ruta_imagen}: {str(e)}")
```

```
60
61          return np.array(imagenes), np.array(etiquetas)
62
63      def entrenar(self, X, y, epochs=50, batch_size=16):
64          """Entrena el modelo con los datos proporcionados"""
65          X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
        random_state=42)
66
67          self.model.compile(
68              optimizer='adam',
69              loss='sparse_categorical_crossentropy',
70              metrics=['accuracy']
71          )
72
73          # Callbacks para mejorar el entrenamiento
74          early_stopping = tf.keras.callbacks.EarlyStopping(
75              monitor='val_loss',
76              patience=10,
77              restore_best_weights=True
78          )
79
80          history = self.model.fit(
81              X_train, y_train,
82              epochs=epochs,
83              batch_size=batch_size,
84              validation_data=(X_val, y_val),
85              callbacks=[early_stopping]
86          )
87
88          return history
89
90      def predecir(self, ruta_imagen):
91          """Realiza una predicci n sobre una imagen"""
92          imagen = cv2.imread(ruta_imagen)
93          if imagen is None:
94              raise ValueError(f"No se pudo cargar la imagen: {ruta_imagen}")
95
96          imagen = cv2.resize(imagen, (self.img_width, self.img_height))
97          imagen = imagen / 255.0
98          imagen = np.expand_dims(imagen, axis=0)
99
100         prediccion = self.model.predict(imagen)
101         clase_predicha = self.clases[np.argmax(prediccion[0])]
102         confianza = np.max(prediccion[0]) * 100
103
104         return clase_predicha, confianza
105
106     def guardar_modelo(self, ruta_guardado):
107         """Guarda el modelo entrenado"""
108         self.model.save(ruta_guardado)
109
110     def cargar_modelo(self, ruta_modelo):
111         """Carga un modelo previamente entrenado"""
112         self.model = tf.keras.models.load_model(ruta_modelo)
113
114 def main():
115     # Ruta espec fica de los datos
116     RUTA_BASE = r"xxxxxxxxxxxxxxxxxxx"
117
118     # Crear instancia del clasificador
119     clasificador = ClasificadorLatas()
120
121     # Preparar datos
```

```
    print("Cargando y preparando datos...")
    X, y = clasificador.preparar_datos(RUTA_BASE)

    if len(X) == 0:
        print("No se encontraron im genes para entrenar!")
        return

    print(f"Total de im genes cargadas: {len(X)}")

    # Entrenar modelo
    print("Iniciando entrenamiento...")
    history = clasificador.entrenar(X, y)

    # Guardar modelo
    ruta_guardado = os.path.join(RUTA_BASE, 'modelo_latas.h5')
    clasificador.guardar_modelo(ruta_guardado)
    print(f"Modelo guardado en: {ruta_guardado}")

    # Ejemplo de predicci n
    # Puedes cambiar esta ruta por la de cualquier imagen de prueba
    ruta_prueba = os.path.join(RUTA_BASE, 'Lata Correcta', os.listdir(os.path.join(
    RUTA_BASE, 'Lata Correcta'))[0])
    clase_predicha, confianza = clasificador.predecir(ruta_prueba)
    print(f"\nPrueba de predicci n:")
    print(f"Imagen: {ruta_prueba}")
    print(f"Predicci n: {clase_predicha}")
    print(f"Confianza: {confianza:.2f}%")

if __name__ == '__main__':
    main()
```

ANEXO B
CÓDIGO DE PYTHON 3 V2 FINAL OPTIMIZADO DE RED NEURONAL PARA RECONOCIMIENTO POR VISIÓN ARTIFICIAL

A continuación se observa el código empleado:

```python
import tensorflow as tf
from tensorflow.keras import layers, Model, Sequential
import pathlib
import os

class CanDefectClassifier:
    def __init__(self, input_shape=(224, 224, 3), num_classes=4):
        self.input_shape = input_shape
        self.num_classes = num_classes
        self.model = None
        self.training_params = None

    def create_model(self):
        """Creates a MobileNet-inspired model architecture for can defect classification
    """
        model = Sequential([
            # Input Layer
            layers.Input(shape=self.input_shape),

            # Initial Feature Extraction
            layers.Conv2D(32, (3, 3), padding='same', activation='relu'),
            layers.BatchNormalization(),
            layers.MaxPooling2D((2, 2)),

            # MobileNet-style Blocks
            self._create_mobilenet_block(64, strides=1),
            self._create_mobilenet_block(128, strides=2),
            self._create_mobilenet_block(256, strides=2),
            self._create_mobilenet_block(512, strides=2),

            # Classification Head
            layers.GlobalAveragePooling2D(),
            layers.Dropout(0.5),
            layers.Dense(128, activation='relu'),
            layers.Dense(self.num_classes, activation='softmax')
        ])

        self.model = model
        return model

    def _create_mobilenet_block(self, filters, strides):
        """Creates a MobileNet-style convolution block"""
        return Sequential([
            # Depthwise Convolution
            layers.DepthwiseConv2D(kernel_size=(3, 3), strides=strides, padding='same'),
            layers.BatchNormalization(),
            layers.ReLU(),

            # Pointwise Convolution
            layers.Conv2D(filters, (1, 1), padding='same'),
            layers.BatchNormalization(),
            layers.ReLU()
        ])

    def setup_training_params(self):
        """Sets up training parameters with reasonable defaults"""
        self.training_params = {
```

```
            'optimizer': tf.keras.optimizers.Adam(learning_rate=0.001),
            'loss': 'categorical_crossentropy',
            'metrics': ['accuracy'],
            'batch_size': 32,
            'epochs': 50,
            'validation_split': 0.2
        }
        return self.training_params

    def compile_model(self):
        """Compiles the model with specified parameters"""
        if self.model is None:
            self.create_model()

        self.model.compile(
            optimizer=self.training_params['optimizer'],
            loss=self.training_params['loss'],
            metrics=self.training_params['metrics']
        )

def setup_data_pipeline(data_dir, img_height=224, img_width=224, batch_size=32):
    """Sets up the data pipeline for training"""
    data_dir = pathlib.Path(data_dir)

    train_ds = tf.keras.utils.image_dataset_from_directory(
        data_dir,
        validation_split=0.2,
        subset="training",
        seed=123,
        image_size=(img_height, img_width),
        batch_size=batch_size
    )

    val_ds = tf.keras.utils.image_dataset_from_directory(
        data_dir,
        validation_split=0.2,
        subset="validation",
        seed=123,
        image_size=(img_height, img_width),
        batch_size=batch_size
    )

    # Configure datasets for performance
    AUTOTUNE = tf.data.AUTOTUNE
    train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
    val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

    return train_ds, val_ds

def main():
    # Initialize classifier
    classifier = CanDefectClassifier()

    # Create and compile model
    model = classifier.create_model()
    training_params = classifier.setup_training_params()
    classifier.compile_model()

    # Print model summary
    print("\nModel Summary:")
    print("-" * 50)
    model.summary()

```

```
    print("\nTraining Parameters:")
    print("-" * 50)
    for key, value in training_params.items():
        print(f"{key}: {value}")

    # Setup data pipeline
    data_dir = "path/to/your/data/directory"  # Update this path
    train_ds, val_ds = setup_data_pipeline(
        data_dir,
        img_height=224,
        img_width=224,
        batch_size=training_params['batch_size']
    )

    # Train model
    history = model.fit(
        train_ds,
        validation_data=val_ds,
        epochs=training_params['epochs']
    )

    # Save model
    model.save('can_defect_classifier.h5')

if __name__ == "__main__":
    main()
```

ANEXO C
ESTRUCTURA DEL MODELO DE RED NEURONAL CONVOLUCIONAL V1

La arquitectura del modelo implementado consiste en una red neuronal convolucional (CNN) con la siguiente estructura y parámetros. Todas las capas utilizan el tipo de dato `float32` para su representación numérica.

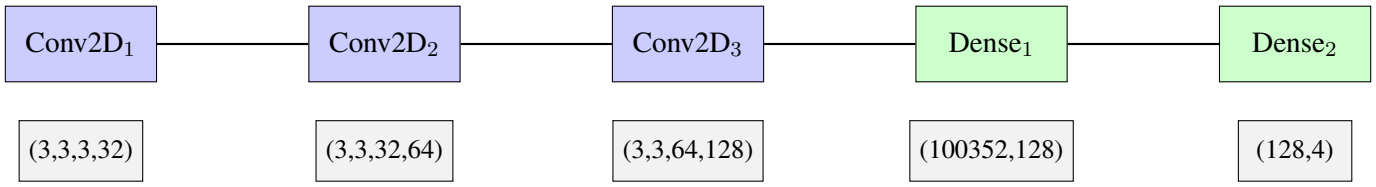


Figura 63: Arquitectura de la Red Neuronal Convolucional

Capas Convolucionales:

$$\begin{aligned} \text{Conv2D}_1 &: \mathbb{R}^{3\times3\times3} \to \mathbb{R}^{32} \\ \text{kernel}_1 &: (3,3,3,32) \\ \text{bias}_1 &: (32) \end{aligned} \tag{25}$$

$$\begin{aligned} \text{Conv2D}_2 &: \mathbb{R}^{3\times3\times32} \to \mathbb{R}^{64} \\ \text{kernel}_2 &: (3,3,32,64) \\ \text{bias}_2 &: (64) \end{aligned} \tag{26}$$

$$\begin{aligned} \text{Conv2D}_3 &: \mathbb{R}^{3\times3\times64} \to \mathbb{R}^{128} \\ \text{kernel}_3 &: (3,3,64,128) \\ \text{bias}_3 &: (128) \end{aligned} \tag{27}$$

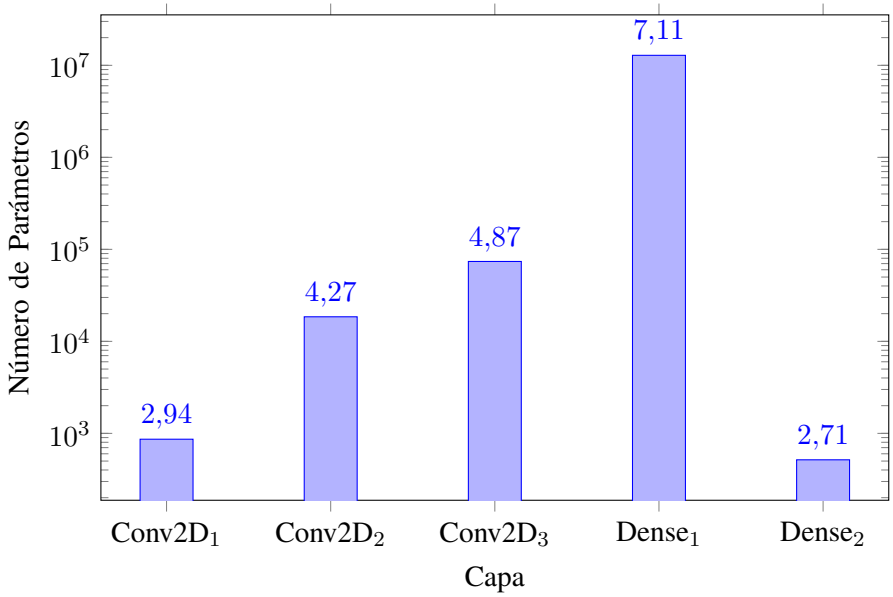


Figura 64: Distribución de Parámetros por Capa (escala logarítmica)

Capas Densas:

$$\begin{aligned} \text{Dense}_1 &: \mathbb{R}^{100352} \to \mathbb{R}^{128} \\ \text{kernel} &: (100352,128) \\ \text{bias} &: (128) \end{aligned} \tag{28}$$

$$\begin{aligned} \text{Dense}_2 &: \mathbb{R}^{128} \to \mathbb{R}^4 \\ \text{kernel} &: (128, 4) \\ \text{bias} &: (4) \end{aligned} \tag{29}$$

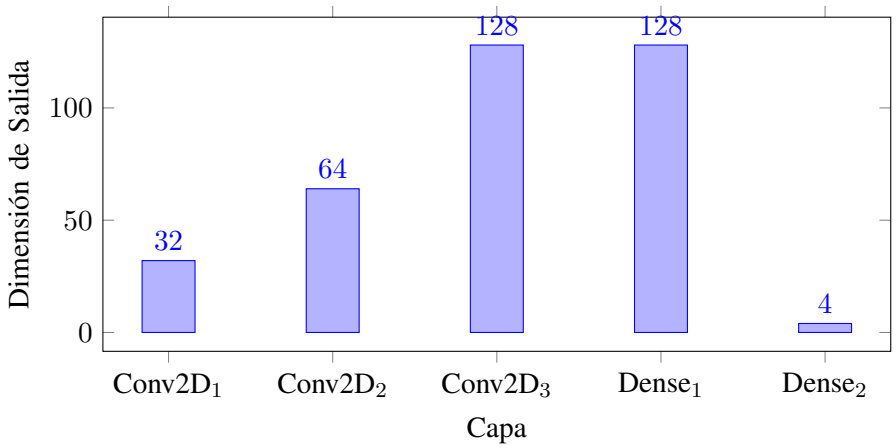


Figura 65: Dimensiones de Salida por Capa

Optimizador Adam:
El modelo utiliza el optimizador Adam con los siguientes momentos:
Momentos de Primer Orden (m):

- Conv2D_1: kernel $(3, 3, 3, 32)$, bias (32)
- Conv2D_2: kernel $(3, 3, 32, 64)$, bias (64)
- Conv2D_3: kernel $(3, 3, 64, 128)$, bias (128)
- Dense_1: kernel $(100352, 128)$, bias (128)
- Dense_2: kernel $(128, 4)$, bias (4)

Momentos de Segundo Orden (v):

- Conv2D_1: kernel $(3, 3, 3, 32)$, bias (32)
- Conv2D_2: kernel $(3, 3, 32, 64)$, bias (64)
- Conv2D_3: kernel $(3, 3, 64, 128)$, bias (128)
- Dense_1: kernel $(100352, 128)$, bias (128)
- Dense_2: kernel $(128, 4)$, bias (4)

La arquitectura implementada sigue un diseño secuencial con tres capas convolucionales seguidas de dos capas densas. La capa final produce una salida de dimensión 4, correspondiente a las clases de clasificación del modelo.

ANEXO D

CÓDIGO DE PYTHON 3 PARA PRUEBA DE CARGA DE IMAGENES PARA EVALUACIÓN DE RED NEURONAL PARA RECONOCIMIENTO POR VISIÓN ARTIFICIAL

A continuación se observa el código empleado:

```
import tensorflow as tf
import cv2
import numpy as np
import os


class PredictorLatas:
    def __init__(self, ruta_modelo):
        self.modelo = tf.keras.models.load_model(ruta_modelo)
        self.clases = ['Lata Correcta', 'Lata Hinchada', 'Sin Objetos', 'Lata Rasgada']
        self.img_height = 224
        self.img_width = 224


    def predecir(self, ruta_imagen):
        # Cargar y preprocesar la imagen
        imagen = cv2.imread(ruta_imagen)
        if imagen is None:
            return "Error: No se pudo cargar la imagen"

        # Redimensionar y normalizar
        imagen = cv2.resize(imagen, (self.img_width, self.img_height))
        imagen = imagen / 255.0
        imagen = np.expand_dims(imagen, axis=0)


        # Realizar predicci n
        prediccion = self.modelo.predict(imagen, verbose=0)
        clase_predicha = self.clases[np.argmax(prediccion[0])]
        confianza = np.max(prediccion[0]) * 100

        return clase_predicha, confianza


def main():
    # Ruta al modelo guardado
    RUTA_MODELO = r"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"


    # Crear el predictor
    predictor = PredictorLatas(RUTA_MODELO)


    while True:
        print("\n=== Sistema de Predicci n de Latas ===")
        print("1. Predecir una imagen")
        print("2. Salir")

        opcion = input("\nSeleccione una opci n (1-2): ")


        if opcion == "1":
            ruta_imagen = input("\nIngrese la ruta completa de la imagen a predecir: ")
            if os.path.exists(ruta_imagen):
                try:
                    clase_predicha, confianza = predictor.predecir(ruta_imagen)
                    print("\nResultado:")
```

```
                    print(f"Clasificaci n: {clase_predicha}")
                    print(f"Confianza: {confianza:.2f}%")
                except Exception as e:
                    print(f"Error al procesar la imagen: {str(e)}")


            else:
                print("Error: La ruta de la imagen no existe")


        elif opcion == "2":
            print("\n Hasta luego!")
            break


        else:
            print("\nOpci n no v lida. Por favor, intente de nuevo.")


if __name__ == "__main__":
    main()
```

ANEXO E
CÓDIGO DE PYTHON 3 PARA PRUEBA DE CÁMARA V1 PARA EVALUACIÓN DE RED NEURONAL PARA RECONOCIMIENTO POR VISIÓN ARTIFICIAL

A continuación se observa el código empleado:

```
import tensorflow as tf
import cv2
import numpy as np
import os

class DetectorLatasRealTime:
    def __init__(self, ruta_modelo):
        self.modelo = tf.keras.models.load_model(ruta_modelo)
        self.clases = ['Lata Correcta', 'Lata Hinchada', 'Sin Objetos', 'Lata Rasgada']
        self.img_height = 224
        self.img_width = 224

    def procesar_frame(self, frame):
        # Preprocesar el frame
        frame_procesado = cv2.resize(frame, (self.img_width, self.img_height))
        frame_procesado = frame_procesado / 255.0
        frame_procesado = np.expand_dims(frame_procesado, axis=0)

        # Realizar predicci n
        prediccion = self.modelo.predict(frame_procesado, verbose=0)
        clase_predicha = self.clases[np.argmax(prediccion[0])]
        confianza = np.max(prediccion[0]) * 100

        return clase_predicha, confianza

    def iniciar_camara(self):
        # Inicializar la c mara
        cap = cv2.VideoCapture(0)

        # Verificar si la c mara se abri  correctamente
        if not cap.isOpened():
            print("Error: No se pudo acceder a la c mara")
            return

        print("Presiona 'q' para salir")

        while True:
            # Leer frame de la c mara
            ret, frame = cap.read()
            if not ret:
                print("Error: No se pudo leer el frame")
                break

            # Hacer una copia del frame para mostrar
            frame_mostrar = frame.copy()

            # Obtener predicci n
            clase_predicha, confianza = self.procesar_frame(frame)

            # Dibujar un rect ngulo semi-transparente para el texto
            overlay = frame_mostrar.copy()
            cv2.rectangle(overlay, (0, 0), (400, 60), (0, 0, 0), -1)
            cv2.addWeighted(overlay, 0.6, frame_mostrar, 0.4, 0, frame_mostrar)

            # Mostrar predicci n en el frame
            texto = f"{clase_predicha} ({confianza:.1f}%)"
            cv2.putText(frame_mostrar, texto, (10, 30),
```

```
                        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)

            # Agregar instrucciones
            cv2.putText(frame_mostrar, "Presiona 'q' para salir", (10, 50),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1)

            # Mostrar el frame
            cv2.imshow('Detector de Latas en Tiempo Real', frame_mostrar)

            # Salir si se presiona 'q'
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        # Liberar recursos
        cap.release()
        cv2.destroyAllWindows()

def main():
    # Ruta al modelo guardado
    RUTA_MODELO = r"xxxxxxxxxxxxxxxxxxxxxx"

    if not os.path.exists(RUTA_MODELO):
        print(f"Error: No se encuentra el modelo en {RUTA_MODELO}")
        return

    try:
        # Crear el detector
        detector = DetectorLatasRealTime(RUTA_MODELO)

        print("Iniciando c mara...")
        detector.iniciar_camara()

    except Exception as e:
        print(f"Error: {str(e)}")

if __name__ == "__main__":
    main()
```

ANEXO F

CÓDIGO DE PYTHON 3 PARA DISEÑO DE LA INTERFAZ GRÁFICA (GUI) DE PC DE LA RED NEURONAL PARA RECONOCIMIENTO POR VISIÓN ARTIFICIAL

A continuación se observa el código empleado:

```
import cv2
import numpy as np
import tensorflow as tf
import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk
import time
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from collections import deque
import threading

class EngineeringDetectionSystem:
    def __init__(self, window):
        self.window = window
        self.window.title("Sistema avanzado de detecci n neuronal v2.0")
        self.window.configure(bg='#1E1E1E')
        self.window.state('zoomed')

        # Theme configuration
        self.colors = self.configure_styles()

        # Initialize components
        self.initialize_data_structures()
        self.setup_model()
        self.create_main_layout()

        # Start camera
        self.camera = cv2.VideoCapture(0)
        self.camera_active = True

        # Start processing threads
        self.start_processing_threads()
        self.update()

    def configure_styles(self):
        style = ttk.Style()
        style.theme_use('default')

        colors = {
            'bg': '#1E1E1E',
            'accent': '#00FF9D',
            'secondary': '#2D2D2D',
            'text': '#FFFFFF',
            'highlight': '#007ACC',
            'error': '#FF4444',
            'success': '#44FF44'
        }

        # Configure custom styles
        style.configure('Tech.TFrame', background=colors['bg'])
        style.configure('Tech.TLabelframe', background=colors['bg'], foreground=colors['
    text'])
        style.configure('Tech.TLabelframe.Label', background=colors['bg'], foreground=
    colors['accent'])

```

```
        style.configure('Tech.TLabel',
                        background=colors['bg'],
                        foreground=colors['text'],
                        font=('Consolas', 10))

        style.configure('Header.TLabel',
                        background=colors['bg'],
                        foreground=colors['accent'],
                        font=('Consolas', 14, 'bold'))

        style.configure('Stats.TLabel',
                        background=colors['bg'],
                        foreground=colors['accent'],
                        font=('Consolas', 12))

        style.configure('Control.TButton',
                        background=colors['secondary'],
                        foreground=colors['text'],
                        padding=5)

        style.configure('Tech.TNotebook',
                        background=colors['bg'],
                        foreground=colors['text'])

        style.configure('Tech.TNotebook.Tab',
                        background=colors['secondary'],
                        foreground=colors['text'],
                        padding=[10, 5])

        return colors

    def create_main_layout(self):
        # Main container with configurable split
        self.main_container = ttk.PanedWindow(self.window, orient=tk.HORIZONTAL)
        self.main_container.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Left panel (camera and controls)
        self.left_panel = ttk.Frame(self.main_container, style='Tech.TFrame')
        self.left_panel.configure(padding="10 10 10 10")

        # Right panel (data and graphs)
        self.right_panel = ttk.Frame(self.main_container, style='Tech.TFrame')
        self.right_panel.configure(padding="10 10 10 10")

        self.main_container.add(self.left_panel, weight=45)
        self.main_container.add(self.right_panel, weight=55)

        self.setup_left_panel()
        self.setup_right_panel()

    def setup_left_panel(self):
        # System header with status indicator
        header_frame = ttk.Frame(self.left_panel, style='Tech.TFrame')
        header_frame.pack(fill='x', pady=(0, 10))

        header = ttk.Label(header_frame,
                           text="NEURAL DETECTION SYSTEM",
                           style='Header.TLabel')
        header.pack(side='left')

        self.status_indicator = ttk.Label(header_frame,
                                          text="    ACTIVO",
                                          style='Header.TLabel',
```

```
                                        foreground=self.colors['success'])
        self.status_indicator.pack(side='right')

        # Camera frame with technical border
        camera_container = ttk.LabelFrame(self.left_panel,
                                        text="Alimentaci n de la c mara",
                                        style='Tech.TLabelframe')
        camera_container.pack(fill='both', expand=True, pady=10)

        self.camera_frame = ttk.Frame(camera_container, style='Tech.TFrame')
        self.camera_frame.pack(padx=5, pady=5)

        self.camera_label = ttk.Label(self.camera_frame, style='Tech.TLabel')
        self.camera_label.pack()

        # Control section
        self.create_control_panel()

    def create_control_panel(self):
        control_container = ttk.LabelFrame(self.left_panel,
                                         text="Controles del sistema",
                                         style='Tech.TLabelframe')
        control_container.pack(fill='x', pady=10)

        # Metrics grid
        metrics_frame = ttk.Frame(control_container, style='Tech.TFrame')
        metrics_frame.pack(fill='x', padx=10, pady=5)

        # Create 2x2 grid for metrics
        for i in range(2):
            metrics_frame.grid_columnconfigure(i, weight=1)

        # FPS Counter
        self.fps_label = ttk.Label(metrics_frame,
                                 text="FPS: 0",
                                 style='Stats.TLabel')
        self.fps_label.grid(row=0, column=0, sticky='w', pady=2)

        # Process Time
        self.process_time_label = ttk.Label(metrics_frame,
                                          text="Process Time: 0ms",
                                          style='Stats.TLabel')
        self.process_time_label.grid(row=0, column=1, sticky='w', pady=2)

        # Detection Confidence
        confidence_frame = ttk.Frame(metrics_frame, style='Tech.TFrame')
        confidence_frame.grid(row=1, column=0, columnspan=2, sticky='ew', pady=2)

        self.confidence_label = ttk.Label(confidence_frame,
                                        text="Detecci n: None (0%)",
                                        style='Stats.TLabel')
        self.confidence_label.pack(side='left')

        self.confidence_bar = ttk.Progressbar(confidence_frame,
                                            length=200,
                                            mode='determinate',
                                            style='Tech.Horizontal.TProgressbar')
        self.confidence_bar.pack(side='left', padx=10)

    def setup_right_panel(self):
        # Notebook for multiple analysis views
        self.notebook = ttk.Notebook(self.right_panel, style='Tech.TNotebook')
        self.notebook.pack(fill='both', expand=True)
```

```
182
183         # Real-time analysis tab
184         self.realtime_tab = ttk.Frame(self.notebook, style='Tech.TFrame')
185         self.notebook.add(self.realtime_tab, text='An lisis en tiempo real')
186
187         # History tab
188         self.history_tab = ttk.Frame(self.notebook, style='Tech.TFrame')
189         self.notebook.add(self.history_tab, text='Datos hist ricos')
190
191         self.setup_realtime_graphs()
192         self.setup_history_graphs()
193
194     def setup_realtime_graphs(self):
195         # Configure matplotlib style for dark theme
196         plt.style.use('dark_background')
197
198         # Real-time confidence graph
199         self.confidence_fig, self.confidence_ax = plt.subplots(figsize=(8, 4))
200         self.confidence_fig.patch.set_facecolor(self.colors['bg'])
201         self.confidence_canvas = FigureCanvasTkAgg(self.confidence_fig, self.realtime_tab)
202         self.confidence_canvas.get_tk_widget().pack(fill='both', expand=True, padx=5, pady
    =5)
203
204         self.confidence_ax.set_title('Confianza en la detecci n en tiempo real')
205         self.confidence_ax.set_ylim(0, 100)
206         self.confidence_ax.grid(True, alpha=0.3)
207         self.confidence_line, = self.confidence_ax.plot([], [], color=self.colors['accent'
    ])
208
209         # Detection distribution
210         self.hist_fig, self.hist_ax = plt.subplots(figsize=(8, 4))
211         self.hist_fig.patch.set_facecolor(self.colors['bg'])
212         self.hist_canvas = FigureCanvasTkAgg(self.hist_fig, self.realtime_tab)
213         self.hist_canvas.get_tk_widget().pack(fill='both', expand=True, padx=5, pady=5)
214
215         self.hist_ax.set_title('Distribuci n de detecci n')
216         self.hist_ax.grid(True, alpha=0.3)
217
218     def setup_history_graphs(self):
219         # System performance trend
220         self.trend_fig, self.trend_ax = plt.subplots(figsize=(8, 4))
221         self.trend_fig.patch.set_facecolor(self.colors['bg'])
222         self.trend_canvas = FigureCanvasTkAgg(self.trend_fig, self.history_tab)
223         self.trend_canvas.get_tk_widget().pack(fill='both', expand=True, padx=5, pady=5)
224
225         self.trend_ax.set_title('Tendencia del rendimiento del sistema')
226         self.trend_ax.grid(True, alpha=0.3)
227
228         # Add performance metrics
229         metrics_frame = ttk.LabelFrame(self.history_tab,
230                                        text="M tricas de rendimiento",
231                                        style='Tech.TLabelframe')
232         metrics_frame.pack(fill='x', padx=5, pady=5)
233
234         self.avg_fps_label = ttk.Label(metrics_frame,
235                                        text="FPS promedio: 0",
236                                        style='Stats.TLabel')
237         self.avg_fps_label.pack(side='left', padx=10)
238
239         self.max_conf_label = ttk.Label(metrics_frame,
240                                         text="Peak Confidence: 0%",
241                                         style='Stats.TLabel')
242         self.max_conf_label.pack(side='left', padx=10)
```

```
243
244     def initialize_data_structures(self):
245         self.confidence_history = deque(maxlen=100)
246         self.detection_history = {}
247         self.fps_history = deque(maxlen=50)
248         self.last_time = time.time()
249         self.frame_count = 0
250
251     def setup_model(self):
252         try:
253             self.interpreter = tf.lite.Interpreter(model_path="model_unquant.tflite")
254             self.interpreter.allocate_tensors()
255             self.input_details = self.interpreter.get_input_details()
256             self.output_details = self.interpreter.get_output_details()
257             self.class_names = open("labels.txt", "r").readlines()
258         except Exception as e:
259             print(f"Error al cargar el modelo: {e}")
260             self.status_indicator.configure(text="    ERROR", foreground=self.colors['
        error'])
261
262     def start_processing_threads(self):
263         self.processing_thread = threading.Thread(target=self.process_data, daemon=True)
264         self.processing_thread.start()
265
266     def process_data(self):
267         while self.camera_active:
268             if hasattr(self, 'current_prediction'):
269                 if hasattr(self, 'current_class'):
270                     if self.current_class not in self.detection_history:
271                         self.detection_history[self.current_class] = 0
272                     self.detection_history[self.current_class] += 1
273                 self.update_graphs()
274             time.sleep(0.1)
275
276     def update_graphs(self):
277         try:
278             # Update confidence graph
279             self.confidence_line.set_data(range(len(self.confidence_history)),
280                                           list(self.confidence_history))
281             self.confidence_ax.set_xlim(0, len(self.confidence_history))
282             self.confidence_canvas.draw()
283
284             # Update histogram with improved label handling
285             self.hist_ax.clear()
286             if self.detection_history:
287                 classes = list(self.detection_history.keys())
288                 values = list(self.detection_history.values())
289
290                 # Create bars
291                 bars = self.hist_ax.bar(classes, values, color=self.colors['accent'])
292                 self.hist_ax.set_title('Distribuci n de detecci n')
293
294                 # Improve label rotation and positioning
295                 plt.setp(self.hist_ax.xaxis.get_majorticklabels(),
296                         rotation=45,           # Rotate labels 45 degrees
297                         ha='right',           # Horizontal alignment
298                         rotation_mode='anchor' # Rotate around the right side
299                 )
300
301                 # Adjust subplot parameters to give specified padding
302                 self.hist_fig.subplots_adjust(bottom=0.2)  # Add padding at the bottom
303
304                 # Add value labels on top of bars
```

```
                for bar in bars:
                    height = bar.get_height()
                    self.hist_ax.text(bar.get_x() + bar.get_width()/2., height,
                                      f'{int(height)}',
                                      ha='center', va='bottom')

            self.hist_canvas.draw()

            # Update trend graph
            self.trend_ax.clear()
            self.trend_ax.plot(list(self.fps_history), color=self.colors['accent'])
            self.trend_ax.set_title('Rendimiento del sistema (FPS)')
            self.trend_ax.grid(True, alpha=0.3)
            self.trend_canvas.draw()

            # Update performance metrics
            if self.fps_history:
                avg_fps = sum(self.fps_history) / len(self.fps_history)
                self.avg_fps_label.configure(text=f"FPS promedio: {avg_fps:.1f}")

            if self.confidence_history:
                max_conf = max(self.confidence_history)
                self.max_conf_label.configure(text=f"M xima confianza: {max_conf}%")

        except Exception as e:
            print(f"Error updating graphs: {e}")

    def update(self):
        start_time = time.time()

        try:
            ret, frame = self.camera.read()
            if ret:
                # Process frame
                frame = cv2.resize(frame, (224, 224))
                cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

                # Prepare image for model
                image = np.asarray(frame, dtype=np.float32).reshape(1, 224, 224, 3)
                image = (image / 127.5) - 1

                # Make prediction
                self.interpreter.set_tensor(self.input_details[0]['index'], image)
                self.interpreter.invoke()
                self.current_prediction = self.interpreter.get_tensor(self.output_details
    [0]['index'])

                # Process result
                index = np.argmax(self.current_prediction)
                self.current_class = self.class_names[index][2:].strip()
                confidence_score = float(self.current_prediction[0][index])

                # Update interface
                confidence_percentage = int(confidence_score * 100)
                self.confidence_history.append(confidence_percentage)
                self.confidence_label.configure(
                    text=f"Detection: {self.current_class} ({confidence_percentage}%)",
                    foreground=self.colors['accent'] if confidence_percentage > 50 else
    self.colors['text']
                )
                self.confidence_bar['value'] = confidence_percentage

                # Update camera display
```

```
                display_size = (640, 480)  # Larger display size
                display_frame = cv2.resize(cv2image, display_size)

                # Add detection overlay
                cv2.putText(
                    display_frame,
                    f"{self.current_class}: {confidence_percentage}%",
                    (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1,
                    (0, 255, 0) if confidence_percentage > 50 else (255, 255, 255),
                    2
                )

                # Convert to PhotoImage
                img = Image.fromarray(display_frame)
                imgtk = ImageTk.PhotoImage(image=img)
                self.camera_label.imgtk = imgtk
                self.camera_label.configure(image=imgtk)

                # Calculate and update FPS
                self.frame_count += 1
                if self.frame_count % 30 == 0:
                    current_time = time.time()
                    fps = 30 / (current_time - self.last_time)
                    self.fps_history.append(fps)

                    # Update FPS display with color coding
                    fps_color = self.colors['success'] if fps > 25 else (
                        self.colors['accent'] if fps > 15 else self.colors['error']
                    )
                    self.fps_label.configure(
                        text=f"FPS: {fps:.1f}",
                        foreground=fps_color
                    )
                    self.last_time = current_time

                # Calculate and display processing time
                process_time = (time.time() - start_time) * 1000
                self.process_time_label.configure(
                    text=f"Tiempo de proceso: {process_time:.1f}ms",
                    foreground=self.colors['accent'] if process_time < 50 else self.colors
    ['error']
                )

                # Update status indicator
                if not hasattr(self, '_status_toggle'):
                    self._status_toggle = True
                self._status_toggle = not self._status_toggle
                self.status_indicator.configure(
                    text="    ACTIVO" if self._status_toggle else "    ACTIVE"
                )

            except Exception as e:
                print(f"Error en el bucle de actualizaci n: {e}")
                self.status_indicator.configure(
                    text="    ERROR",
                    foreground=self.colors['error']
                )

            finally:
                # Schedule next update
                if self.camera_active:
```

```
                self.window.after(10, self.update)

    def __del__(self):
        self.camera_active = False
        if hasattr(self, 'camera'):
            self.camera.release()

if __name__ == "__main__":
    root = tk.Tk()
    app = EngineeringDetectionSystem(root)
    root.mainloop()
```

ANEXO G
CÓDIGO DE PYTHON 3 PARA DISEÑO DE LA INTERFAZ GRÁFICA (GUI) DE RASPBERRY PI 4B OPTIMIZADA DE LA RED NEURONAL PARA RECONOCIMIENTO POR VISIÓN ARTIFICIAL

A continuación se observa el código empleado:

```
import cv2
import numpy as np
import tensorflow as tf
import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk
import threading
import queue
import time
import RPi.GPIO as GPIO
from collections import deque

class DetectionGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Detector de Objetos")

        # Configurar GPIO
        self.RELAY_PIN = 17
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.RELAY_PIN, GPIO.OUT)
        GPIO.output(self.RELAY_PIN, GPIO.HIGH)  # Inicialmente apagado

        # Variables para verificaci n temporal
        self.detection_window = 0.70  # Ventana de tiempo en segundos
        self.detection_history = deque()  # Historial de detecciones
        self.last_activation_time = 0  #  ltimo  tiempo de activaci n

        # Configurar estilo minimalista
        style = ttk.Style()
        style.configure('TFrame', background='#f0f0f0')
        style.configure('TLabel', background='#f0f0f0', font=('Arial', 10))

        # Frame principal
        self.main_frame = ttk.Frame(root, padding="5")
        self.main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        #  rea  de video
        self.video_label = ttk.Label(self.main_frame)
        self.video_label.grid(row=0, column=0, columnspan=2, pady=5)

        # Etiquetas de resultados
        self.class_label = ttk.Label(self.main_frame, text="Clase: -")
        self.class_label.grid(row=1, column=0, pady=2, padx=5, sticky=tk.W)

        self.conf_label = ttk.Label(self.main_frame, text="Confianza: -")
        self.conf_label.grid(row=2, column=0, pady=2, padx=5, sticky=tk.W)

        # Etiqueta estado del  rel
        self.relay_frame = ttk.Frame(self.main_frame)
        self.relay_frame.grid(row=3, column=0, pady=5, padx=5, sticky=tk.W)

        self.relay_label = ttk.Label(self.relay_frame, text="Estado del  Rel  : ")
        self.relay_label.grid(row=0, column=0)

        self.relay_status = ttk.Label(self.relay_frame, text="APAGADO", background='red')
        self.relay_status.grid(row=0, column=1)
```

```
58
59          # Bot n de salida
60          self.exit_button = ttk.Button(self.main_frame, text="Salir", command=self.quit_app
        )
61          self.exit_button.grid(row=4, column=0, columnspan=2, pady=5)
62
63          # Variables de control
64          self.running = False
65          self.queue = queue.Queue()
66
67          # Inicializar modelo y c mara
68          self.setup_model()
69          self.camera = None
70
71          # Iniciar procesamiento
72          self.running = True
73          self.video_thread = threading.Thread(target=self.video_loop, daemon=True)
74          self.video_thread.start()
75
76          # Actualizar GUI
77          self.update_gui()
78
79      def verify_detection_window(self, class_name, confidence):
80          current_time = time.time()
81
82          # Limpiar detecciones antiguas
83          while self.detection_history and (current_time - self.detection_history[0][2]) >
        self.detection_window:
84              self.detection_history.popleft()
85
86          # Agregar nueva detecci n
87          self.detection_history.append((class_name, confidence, current_time))
88
89          # Si no hay suficientes detecciones en la ventana de tiempo, retornar False
90          if len(self.detection_history) < 10:  # Requiere al menos 10 detecciones
91              return False
92
93          # Contar detecciones v lidas en la ventana de tiempo
94          valid_detections = [(c, conf) for c, conf, t in self.detection_history
95                              if c.strip() in ['Lata Hinchada', 'Lata Rasgada']
96                              and conf >= 95]
97
98          # Calcular porcentaje de detecciones v lidas
99          detection_percentage = len(valid_detections) / len(self.detection_history)
100
101         # Retornar True si m s del 80% de las detecciones son v lidas
102         return detection_percentage > 0.80
103
104     def control_relay(self, class_name, confidence):
105         # Lista de clases que activan el rel
106         activating_classes = ['Lata Hinchada', 'Lata Rasgada']
107
108         # Verificar si la clase y confianza son v lidas
109         if class_name.strip() in activating_classes and confidence >= 95:
110             # Verificar la ventana de tiempo
111             if self.verify_detection_window(class_name, confidence):
112                 GPIO.output(self.RELAY_PIN, GPIO.LOW)
113                 return True
114
115         GPIO.output(self.RELAY_PIN, GPIO.HIGH)
116         return False
117
118     def setup_model(self):
```

```
        try:
            self.interpreter = tf.lite.Interpreter(
                model_path="/home/raspberrypi/Desktop/uiyuihjk/model_unquant.tflite"
            )
            self.interpreter.allocate_tensors()
            self.input_details = self.interpreter.get_input_details()
            self.output_details = self.interpreter.get_output_details()
            self.class_names = open(
                "/home/raspberrypi/Desktop/uiyuihjk/labels.txt", "r"
            ).readlines()
        except Exception as e:
            print(f"Error al cargar el modelo: {e}")
            self.quit_app()

    def video_loop(self):
        try:
            self.camera = cv2.VideoCapture(0)
            while self.running:
                ret, frame = self.camera.read()
                if ret:
                    # Redimensionar para mostrar
                    display_frame = cv2.resize(frame, (320, 240))

                    # Preparar imagen para el modelo
                    model_frame = cv2.resize(frame, (224, 224))
                    image = np.asarray(model_frame, dtype=np.float32).reshape(1, 224, 224,
 3)
                    image = (image / 127.5) - 1

                    # Hacer predicci n
                    self.interpreter.set_tensor(self.input_details[0]['index'], image)
                    self.interpreter.invoke()
                    prediction = self.interpreter.get_tensor(self.output_details[0]['index
']) 

                    # Obtener resultado
                    index = np.argmax(prediction)
                    class_name = self.class_names[index][2:]
                    confidence = np.round(prediction[0][index] * 100)

                    # Controlar rel  basado en la detecci n y confianza
                    relay_active = self.control_relay(class_name, confidence)

                    # Convertir frame para tkinter
                    rgb_frame = cv2.cvtColor(display_frame, cv2.COLOR_BGR2RGB)
                    img = Image.fromarray(rgb_frame)
                    img_tk = ImageTk.PhotoImage(image=img)

                    # Enviar a cola
                    self.queue.put({
                        'image': img_tk,
                        'class': class_name,
                        'confidence': confidence,
                        'relay_active': relay_active
                    })

                time.sleep(0.03)  # Limitar FPS para reducir uso de CPU

        except Exception as e:
            print(f"Error en video_loop: {e}")
        finally:
            if self.camera is not None:
                self.camera.release()
```

```
180
181     def update_gui(self):
182         try:
183             while not self.queue.empty():
184                 data = self.queue.get()
185
186                 # Actualizar imagen
187                 self.video_label.configure(image=data['image'])
188                 self.video_label.image = data['image']
189
190                 # Actualizar etiquetas
191                 self.class_label.configure(text=f"Clase: {data['class']}")
192                 self.conf_label.configure(text=f"Confianza: {data['confidence']}%")
193
194                 # Actualizar estado del rel
195                 if data['relay_active']:
196                     self.relay_status.configure(text="ENCENDIDO", background='green')
197                 else:
198                     self.relay_status.configure(text="APAGADO", background='red')
199
200         except Exception as e:
201             print(f"Error en update_gui: {e}")
202
203         if self.running:
204             self.root.after(10, self.update_gui)
205
206     def quit_app(self):
207         self.running = False
208         if self.camera is not None:
209             self.camera.release()
210         GPIO.cleanup()  # Limpiar GPIO al salir
211         self.root.quit()
212
213 if __name__ == "__main__":
214     root = tk.Tk()
215     app = DetectionGUI(root)
216     root.mainloop()
```

ANEXO H
CÓDIGO DE PYTHON 3 PARA CONVERSIÓN DE ARCHIVOS .H5 DE IA A PDF PARA ANÁLISIS MATEMÁTICO DE RENDIMIENTO

A continuación se observa el código empleado:

```
1
2  import h5py
3  import matplotlib.pyplot as plt
4  from fpdf import FPDF
5  import numpy as np
6  import os
7  import shutil
8
9  def h5_to_pdf(h5_file_path, output_pdf_path):
10     """
11     Convierte un archivo H5 a PDF, visualizando sus datasets
12
13     Par metros:
14     h5_file_path (str): Ruta al archivo .h5
15     output_pdf_path (str): Ruta donde guardar el PDF resultante
16     """
17
18     # Crear un directorio temporal para las im genes
19     temp_dir = 'temp_images'
20
21     # Limpiar el directorio temporal si existe
22     if os.path.exists(temp_dir):
23         shutil.rmtree(temp_dir)
24
25     # Crear nuevo directorio temporal
26     os.makedirs(temp_dir)
27
28     try:
29         # Abrir el archivo H5
30         with h5py.File(h5_file_path, 'r') as f:
31             # Crear PDF
32             pdf = FPDF()
33             pdf.set_auto_page_break(auto=True, margin=15)
34
35             # A adir p gina de t tulo
36             pdf.add_page()
37             pdf.set_font('Arial', 'B', 16)
38             pdf.cell(190, 10, f'Contenido del archivo: {os.path.basename(h5_file_path)}',
39                      ln=True, align='C')
40
41             def process_group(name, obj):
42                 if isinstance(obj, h5py.Dataset):
43                     try:
44                         # A adir nueva p gina para cada dataset
45                         pdf.add_page()
46                         pdf.set_font('Arial', 'B', 12)
47                         pdf.cell(190, 10, f'Dataset: {name}', ln=True)
48
49                         # Obtener y visualizar datos
50                         data = obj[()]
51
52                         if len(data.shape) <= 2:  # Para datos 1D o 2D
53                             plt.figure(figsize=(10, 6))
54                             if len(data.shape) == 1:
55                                 plt.plot(data)
56                             else:
57                                 plt.imshow(data)
```

```
                                plt.title(f'Visualizaci n de {name}')

                                # Guardar figura temporalmente
                                temp_image = f'{temp_dir}/{name.replace("/", "_")}.png'
                                plt.savefig(temp_image)
                                plt.close()

                                # A adir imagen al PDF
                                pdf.image(temp_image, x=10, w=190)

                            # A adir informaci n sobre el dataset
                            pdf.set_font('Arial', '', 10)
                            pdf.cell(190, 10, f'Forma: {data.shape}', ln=True)
                            pdf.cell(190, 10, f'Tipo de datos: {data.dtype}', ln=True)

                        except Exception as e:
                            pdf.set_font('Arial', '', 10)
                            pdf.cell(190, 10, f'No se pudo visualizar este dataset: {str(e)}',
     ln=True)

                # Procesar todos los grupos y datasets
                f.visititems(process_group)

                # Guardar PDF
                pdf.output(output_pdf_path)

        finally:
            # Limpiar el directorio temporal al finalizar
            if os.path.exists(temp_dir):
                shutil.rmtree(temp_dir)

# Usar la ruta espec fica del archivo
ruta_h5 = r"xxxxxxxxxxxxxxxxxxxxxx"
# Crear la ruta de salida en la misma carpeta
ruta_salida = r"xxxxxxxxxxxxxxxxxxxxxx"

# Ejecutar la conversi n
if __name__ == "__main__":
    try:
        h5_to_pdf(ruta_h5, ruta_salida)
        print(f"PDF creado exitosamente en: {ruta_salida}")
    except Exception as e:
        print(f"Error al procesar el archivo: {str(e)}")
```

ANEXO I
CÓDIGO DE PYTHON 3 PARA CONVERSIÓN DE ARCHIVOS .TFLITE DE IA A PDF PARA ANÁLISIS MATEMÁTICO DE RENDIMIENTO

A continuación se observa el código empleado:

```
import tensorflow as tf
import numpy as np
import os
from fpdf import FPDF
from fpdf.enums import XPos, YPos

def analyze_tflite_model(model_path, pdf_path):
    """
    Analiza un modelo TFLite y genera un PDF con informaci n detallada
    """
    interpreter = tf.lite.Interpreter(model_path=model_path)
    interpreter.allocate_tensors()

    # Obtener detalles del modelo
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    tensor_details = interpreter.get_tensor_details()

    # Crear PDF
    pdf = FPDF()
    pdf.set_auto_page_break(auto=True, margin=15)
    pdf.add_page()

    # T tulo
    pdf.set_font("Helvetica", "B", 16)
    pdf.cell(0, 10, "An lisis Detallado del Modelo TFLite", new_x=XPos.LMARGIN, new_y=
    YPos.NEXT, align='C')
    pdf.ln(10)

    # Informaci n b sica
    pdf.set_font("Helvetica", "B", 12)
    pdf.cell(0, 10, "1. Informaci n B sica del Modelo", new_x=XPos.LMARGIN, new_y=YPos.
    NEXT)
    pdf.set_font("Helvetica", "", 10)
    pdf.cell(0, 10, f"Nombre del archivo: {os.path.basename(model_path)}", new_x=XPos.
    LMARGIN, new_y=YPos.NEXT)
    pdf.cell(0, 10, f"Tama o del archivo: {os.path.getsize(model_path)/1024:.2f} KB",
    new_x=XPos.LMARGIN, new_y=YPos.NEXT)

    # Capas de entrada
    pdf.set_font("Helvetica", "B", 12)
    pdf.cell(0, 10, "2. Capas de Entrada", new_x=XPos.LMARGIN, new_y=YPos.NEXT)
    pdf.set_font("Helvetica", "", 10)

    for input_detail in input_details:
        pdf.multi_cell(0, 10,
            f"""Nombre: {input_detail['name']}
            Forma: {input_detail['shape']}
            Tipo de datos: {input_detail['dtype']}""")
        pdf.ln(5)

    # Capas de salida
    pdf.set_font("Helvetica", "B", 12)
    pdf.cell(0, 10, "3. Capas de Salida", new_x=XPos.LMARGIN, new_y=YPos.NEXT)
    pdf.set_font("Helvetica", "", 10)

    for output_detail in output_details:
```

```
        pdf.multi_cell(0, 10,
            f"""Nombre: {output_detail['name']}
            Forma: {output_detail['shape']}
            Tipo de datos: {output_detail['dtype']}""")
        pdf.ln(5)

    # An lisis de tensores
    pdf.add_page()
    pdf.set_font("Helvetica", "B", 12)
    pdf.cell(0, 10, "4. An lisis de Tensores", new_x=XPos.LMARGIN, new_y=YPos.NEXT)
    pdf.set_font("Helvetica", "", 10)

    total_params = 0
    for tensor in tensor_details:
        try:
            tensor_name = tensor['name']
            tensor_shape = tensor['shape']
            tensor_type = tensor['dtype']

            # Calcular par metros de manera segura
            params = int(np.prod(tensor_shape)) if tensor_shape.size > 0 else 0
            total_params += params

            # Informaci n b sica del tensor
            info_text = f"""
            Tensor: {tensor_name}
            Forma: {tensor_shape} (Par metros: {params:,})
            Tipo: {tensor_type}
            """

            try:
                # Intentar obtener estad sticas del tensor si es posible
                if params > 0:
                    tensor_data = interpreter.tensor(tensor['index'])()
                    if tensor_data is not None and tensor_data.size > 0:
                        min_val = float(np.min(tensor_data))
                        max_val = float(np.max(tensor_data))
                        mean_val = float(np.mean(tensor_data))
                        std_val = float(np.std(tensor_data))

                        info_text += f"""
                        Estad sticas:
                        Min={min_val:.4f}, Max={max_val:.4f}
                        Media={mean_val:.4f}, Desv. Est.={std_val:.4f}
                        """
            except Exception as e:
                info_text += f"\nNo se pudieron calcular estad sticas para este tensor"

            pdf.multi_cell(0, 10, info_text)
            pdf.ln(5)

        except Exception as e:
            pdf.multi_cell(0, 10, f"Error al analizar tensor: {str(e)}")

    # Resumen final
    pdf.add_page()
    pdf.set_font("Helvetica", "B", 12)
    pdf.cell(0, 10, "5. Resumen del Modelo", new_x=XPos.LMARGIN, new_y=YPos.NEXT)
    pdf.set_font("Helvetica", "", 10)

    summary_text = f"""
    Total de par metros: {total_params:,}
    N mero de tensores: {len(tensor_details)}
```

```
    N mero de capas de entrada: {len(input_details)}
    N mero de capas de salida: {len(output_details)}
    """
    pdf.multi_cell(0, 10, summary_text)

    # Guardar PDF
    try:
        pdf.output(pdf_path)
        print(f"An lisis completo guardado en: {pdf_path}")
    except Exception as e:
        print(f"Error al guardar el PDF: {str(e)}")

if __name__ == "__main__":
    model_path = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    pdf_path = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

    try:
        analyze_tflite_model(model_path, pdf_path)
    except Exception as e:
        print(f"Error durante el an lisis: {str(e)}")
```

ANEXO J
CÓDIGO DE MATLAB PARA ANÁLISIS MATEMÁTICO DE RENDIMIENTO DE LA IA V2 FINAL

A continuación se observa el código empleado:

```
%% 1. An lisis de Arquitectura y Complejidad
figure(1)
% Datos de la arquitectura
param_counts = [150528, 100, 2, 8, 16, 16, 48, 48, 48, 96, 96, 96, 144, 144, 144, 192,
    192, 192, 336, 336, 336];
layer_names = {'Input', 'Dense1', 'Const1', 'Const2', 'Const3', 'Const4', 'Const5', '
    Const6', 'Const7', 'Const8', 'Const9', 'Const10', 'Const11', 'Const12', 'Const13', '
    Const14', 'Const15', 'Const16', 'Const17', 'Const18', 'Const19'};

% Subplot 1: Distribuci n de par metros
subplot(2,2,1)
bar(param_counts)
title('Distribuci n de Par metros')
xlabel('Capa')
ylabel('Par metros')
xticks(1:length(layer_names))
xticklabels(layer_names)
xtickangle(45)
grid on

% Subplot 2: Distribuci n logar tmica
subplot(2,2,2)
semilogy(param_counts, '-o')
title('Distribuci n Logar tmica de Par metros')
xlabel(' ndice  de Capa')
ylabel('log(Par metros)')
grid on

% Subplot 3: Complejidad acumulativa
subplot(2,2,3)
cumulative_params = cumsum(param_counts);
plot(cumulative_params, '-s')
title('Complejidad Acumulativa')
xlabel('Profundidad de Red')
ylabel('Par metros Acumulados')
grid on

% Subplot 4: Proporci n de par metros
subplot(2,2,4)
pie(param_counts(1:5))
title('Proporci n de Par metros (Top 5 Capas)')
legend(layer_names(1:5), 'Location', 'bestoutside')

%% 2. An lisis Estad stico de Tensores
figure(2)
% Datos estad sticos de tensores
means = [0.0034, 1.5000, 0.2500, 0.9672, 1.1389, 1.5995, -0.0224, 1.2497, 1.8569, 1.0121];
stds = [0.0063, 0.5000, 0.4330, 3.0305, 1.3124, 1.5313, 4.0888, 1.5905, 3.1758, 1.6619];
mins = [-0.0091, 1.0000, 0.0000, -4.8292, -0.8916, -0.2823, -18.2299, -1.8225, -2.4831,
    -1.4633];
maxs = [0.0182, 2.0000, 1.0000, 7.3783, 4.0017, 4.4879, 6.5887, 5.7262, 18.5597, 5.9454];

% Subplot 1: Box plot
subplot(2,2,1)
boxplot([mins', means', maxs'])
title('Distribuci n de Valores por Tensor')
xlabel('Estad stica')
ylabel('Valor')
xticklabels({'Min', 'Media', 'Max'})
```

```
grid on

% Subplot 2: Dispersi n estad stica
subplot(2,2,2)
scatter(means, stds, 50, 'filled')
title('Dispersi n Media vs Desviaci n Est ndar')
xlabel('Media')
ylabel('Desviaci n Est ndar')
grid on
text(means, stds, cellstr(num2str((1:length(means))')))

% Subplot 3: Rango din mico
subplot(2,2,3)
dynamic_range = maxs - mins;
bar(dynamic_range)
title('Rango Din mico por Tensor')
xlabel(' ndice  del Tensor')
ylabel('Rango (Max - Min)')
grid on

% Subplot 4: Coeficiente de variaci n
subplot(2,2,4)
cv = abs(stds ./ means);
bar(cv)
title('Coeficiente de Variaci n')
xlabel(' ndice  del Tensor')
ylabel('CV (|std/mean|)')
grid on

%% 3. An lisis de Eficiencia y Rendimiento
figure(3)
% C lculos de eficiencia
memory_per_layer = param_counts .* 4; % 4 bytes por par metro float32
total_memory = sum(memory_per_layer);
efficiency_ratio = param_counts ./ memory_per_layer;
theoretical_ops = param_counts .* 2; % Operaciones aproximadas

% Subplot 1: Uso de memoria
subplot(2,2,1)
bar(memory_per_layer./1024) % Convertir a KB
title('Uso de Memoria por Capa')
xlabel('Capa')
ylabel('Memoria (KB)')
grid on

% Subplot 2: Eficiencia de memoria
subplot(2,2,2)
plot(cumsum(memory_per_layer)./1024, '-o')
title('Uso Acumulativo de Memoria')
xlabel('Capa')
ylabel('Memoria Acumulada (KB)')
grid on

% Subplot 3: Operaciones te ricas
subplot(2,2,3)
bar(theoretical_ops)
title('Operaciones Te ricas por Capa')
xlabel('Capa')
ylabel('N mero de Operaciones')
grid on

% Subplot 4: Eficiencia computacional
subplot(2,2,4)
```

```
plot(theoretical_ops./memory_per_layer, '-s')
title('Eficiencia Computacional')
xlabel('Capa')
ylabel('Ops/Byte')
grid on

%% 4. An lisis de Distribuci n y Normalizaci n
figure(4)
% Generar m s datos para an lisis de distribuci n
expanded_weights = [mins maxs means];
scaled_weights = (expanded_weights - min(expanded_weights)) / (max(expanded_weights) - min
    (expanded_weights));

% Subplot 1: Histograma de pesos
subplot(2,2,1)
histogram(expanded_weights, 30, 'Normalization', 'probability')
title('Distribuci n de Pesos')
xlabel('Valor del Peso')
ylabel('Frecuencia Normalizada')
grid on

% Subplot 2: Q-Q Plot
subplot(2,2,2)
qqplot(expanded_weights)
title('Q-Q Plot de Pesos')
grid on

% Subplot 3: Distribuci n acumulativa
subplot(2,2,3)
cdfplot(expanded_weights)
title('Distribuci n Acumulativa')
grid on

% Subplot 4: An lisis de normalizaci n
subplot(2,2,4)
bar(scaled_weights)
title('Pesos Normalizados')
xlabel(' ndice ')
ylabel('Valor Normalizado')
grid on

%% 5. An lisis de Correlaci n y Dependencias
figure(5)
% Matrices de correlaci n simuladas
tensor_correlations = corrcoef([means; stds; mins; maxs]');

% Subplot 1: Mapa de calor de correlaciones
subplot(2,2,1)
heatmap(tensor_correlations)
title('Correlaciones entre Estad sticas')

% Subplot 2: An lisis de dependencia
subplot(2,2,2)
scatter(means, maxs, 50, 'filled')
hold on
p = polyfit(means, maxs, 1);
plot(means, polyval(p, means), 'r--')
hold off
title('Dependencia Media-M ximo')
xlabel('Media')
ylabel('M ximo')
grid on

```

```
% Subplot 3: An lisis de varianza
subplot(2,2,3)
variance_ratio = stds.^2 ./ abs(means);
bar(variance_ratio)
title('Ratio de Varianza')
xlabel('Tensor')
ylabel('Varianza/|Media|')
grid on

% Subplot 4: Clustering jer rquico
subplot(2,2,4)
Z = linkage([means' stds' mins' maxs'], 'complete');
dendrogram(Z)
title('Clustering de Tensores')
xlabel('Tensor')
ylabel('Distancia')

%% 6. M tricas de Rendimiento Global
figure(6)
% Calcular m tricas globales
total_params = sum(param_counts);
avg_params_per_layer = mean(param_counts);
std_params_per_layer = std(param_counts);
compression_ratio = total_params / max(param_counts);
efficiency_score = mean(efficiency_ratio);

% Crear gr fico de m tricas
metrics = [total_params/1e3, avg_params_per_layer, std_params_per_layer, compression_ratio
    *10, efficiency_score*1e3];
metric_names = {'Total Params (K)', 'Avg Params/Layer', 'Std Params/Layer', 'Compression
    x10', 'Efficiency x1000'};

bar(metrics)
title('M tricas Globales de Rendimiento')
xlabel('M trica')
ylabel('Valor')
xticklabels(metric_names)
xtickangle(45)
grid on

% A adir texto con valores exactos
for i = 1:length(metrics)
    text(i, metrics(i), sprintf('%.2f', metrics(i)), 'HorizontalAlignment', 'center', '
    VerticalAlignment', 'bottom')
end

%% 7. An lisis de Complejidad Temporal
figure(7)
% Estimaci n de tiempos de ejecuci n basados en complejidad
layer_complexity = param_counts .* log2(param_counts);
relative_complexity = layer_complexity / sum(layer_complexity);

% Subplot 1: Complejidad por capa
subplot(2,2,1)
bar(layer_complexity)
title('Complejidad Temporal por Capa')
xlabel('Capa')
ylabel('Complejidad (O(n log n))')
grid on

% Subplot 2: Complejidad relativa
subplot(2,2,2)
pie(relative_complexity)
```

```
241 title('Distribuci n de Complejidad')
242
243 % Subplot 3: Complejidad acumulativa
244 subplot(2,2,3)
245 plot(cumsum(layer_complexity), '-o')
246 title('Complejidad Acumulativa')
247 xlabel('Profundidad')
248 ylabel('Complejidad Total')
249 grid on
250
251 % Subplot 4: Eficiencia temporal
252 subplot(2,2,4)
253 efficiency_temporal = param_counts ./ layer_complexity;
254 bar(efficiency_temporal)
255 title('Eficiencia Temporal')
256 xlabel('Capa')
257 ylabel('Par metros/Complejidad')
258 grid on
```

ANEXO K
PLANOS DE PIEZAS ACOTADAS EN SOLIDWORK:

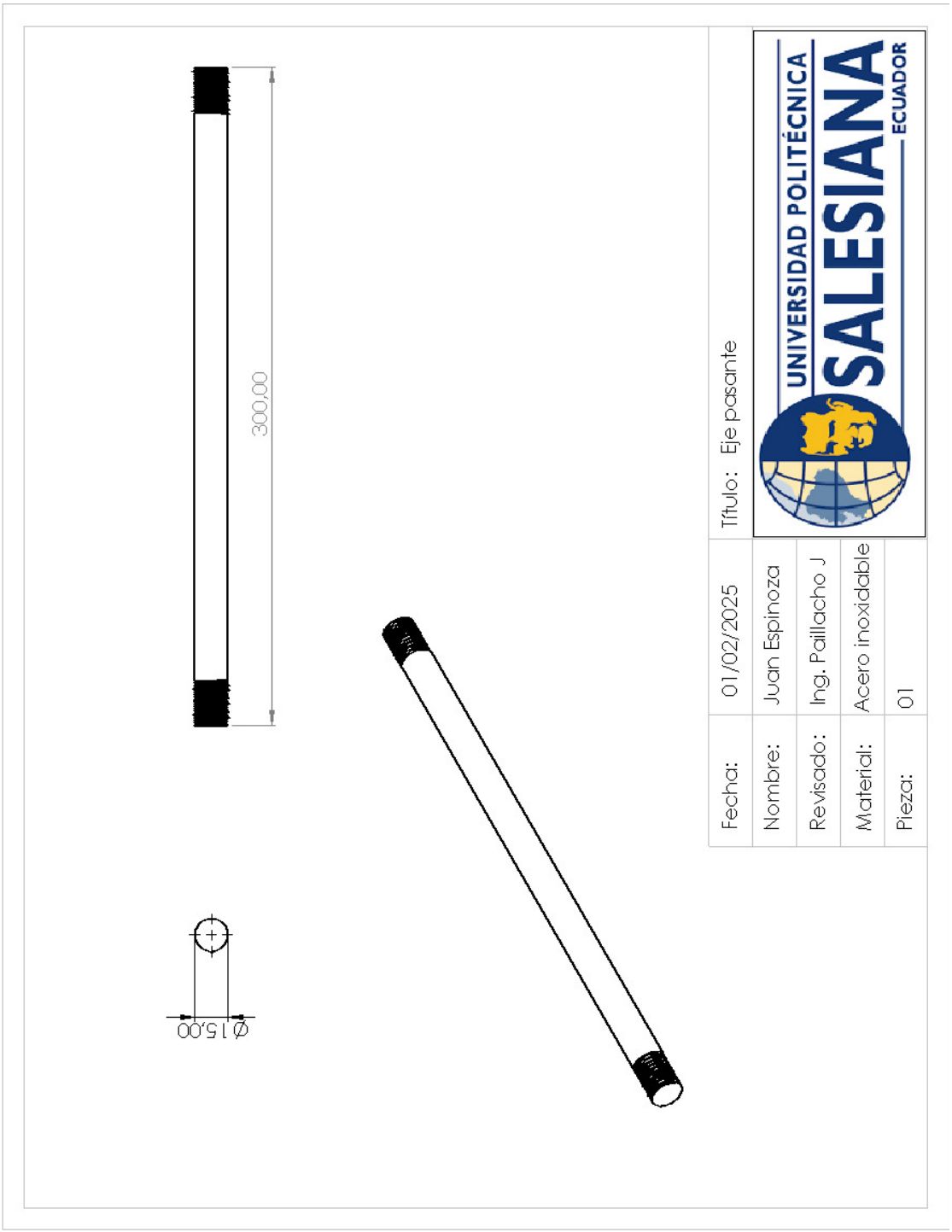


Figura 66: Eje pasante, por Juan Espinoza.

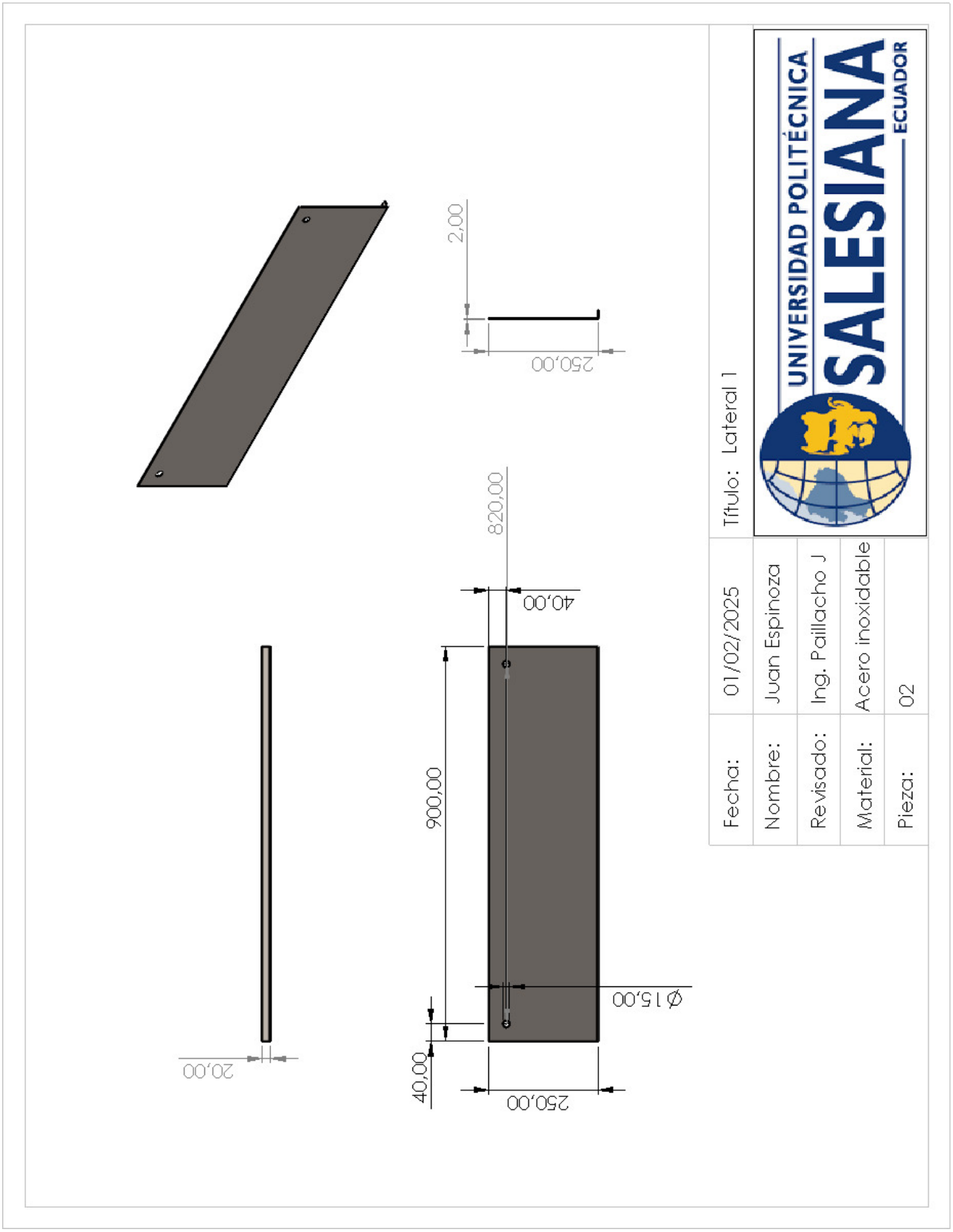


Figura 67: Lateral 1, por Juan Espinoza.

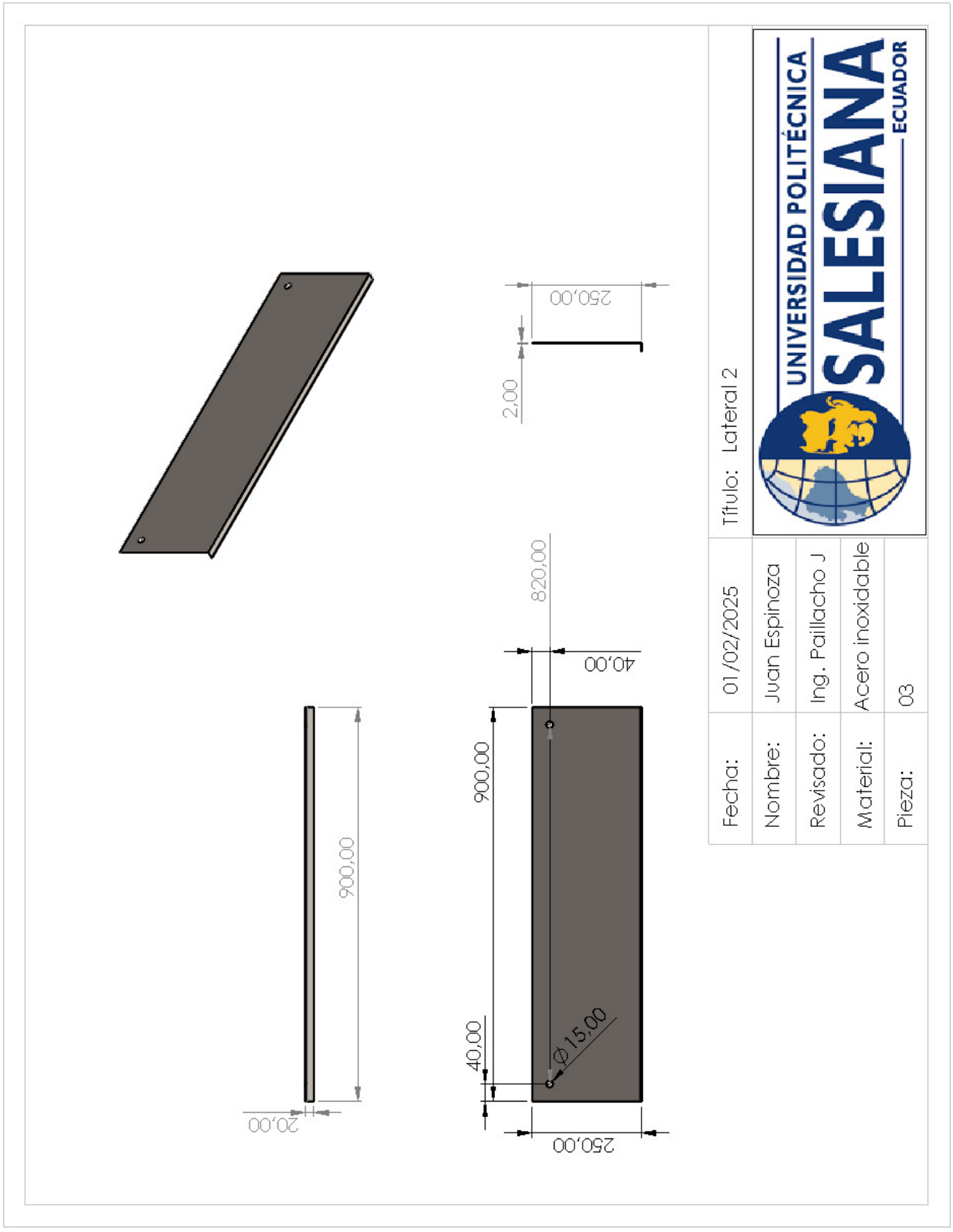


Figura 68: Lateral 2, por Juan Espinoza.

Figura 69: Tubo de acero, por Juan Espinoza.

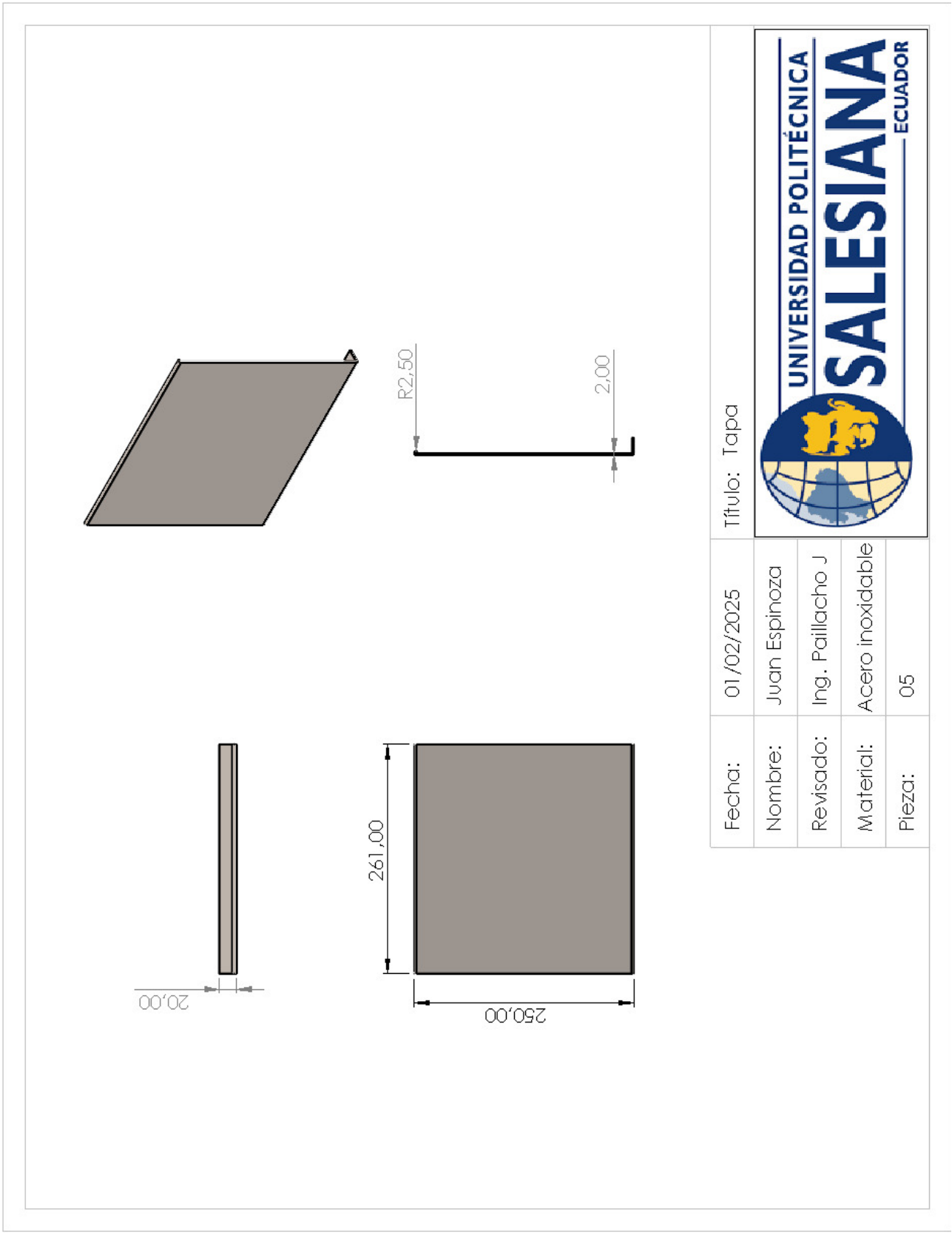


Figura 70: Tapa, por Juan Espinoza.

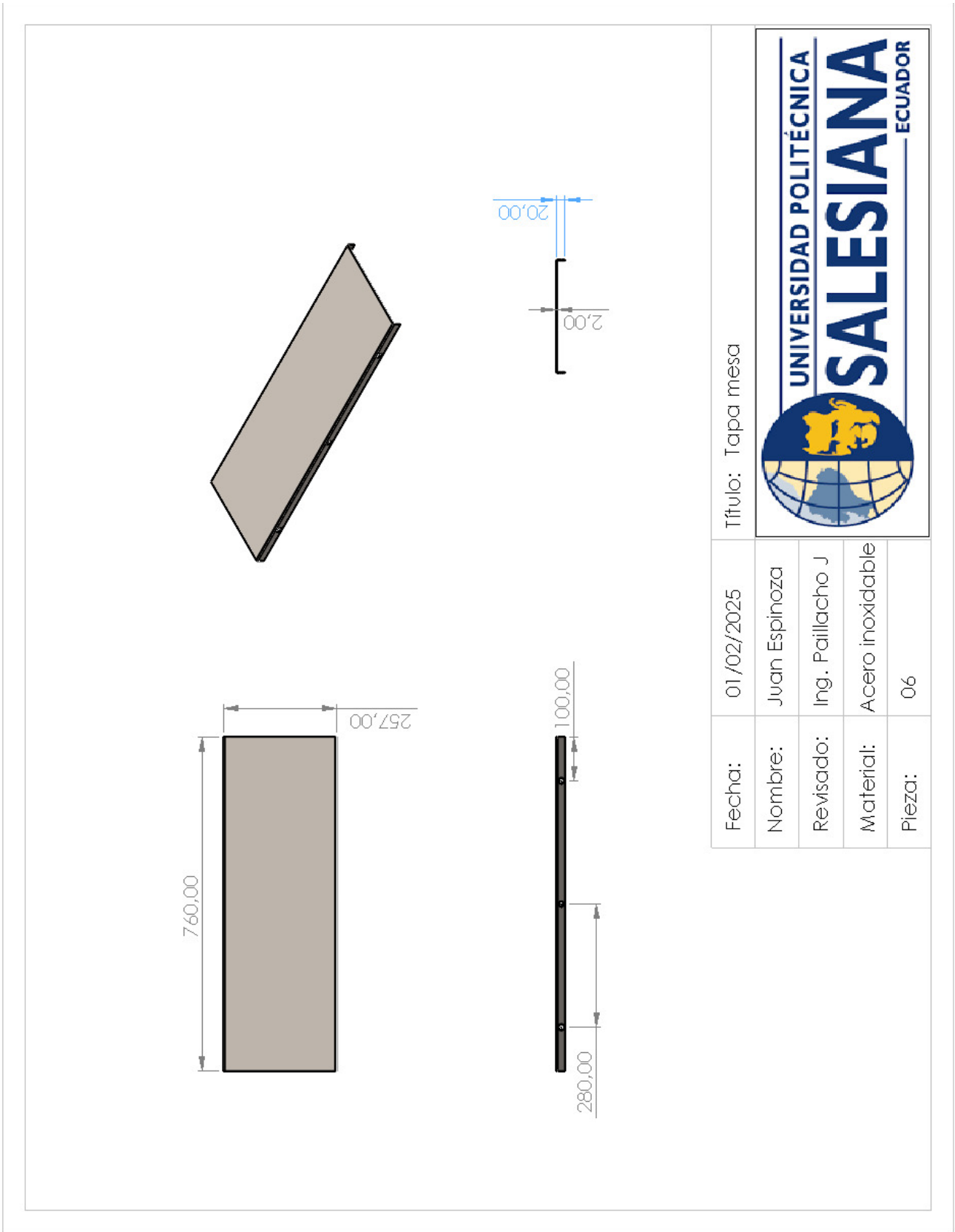


Figura 71: Tapa mesa, por Juan Espinoza.

Figura 72: Rodamiento, por Juan Espinoza.

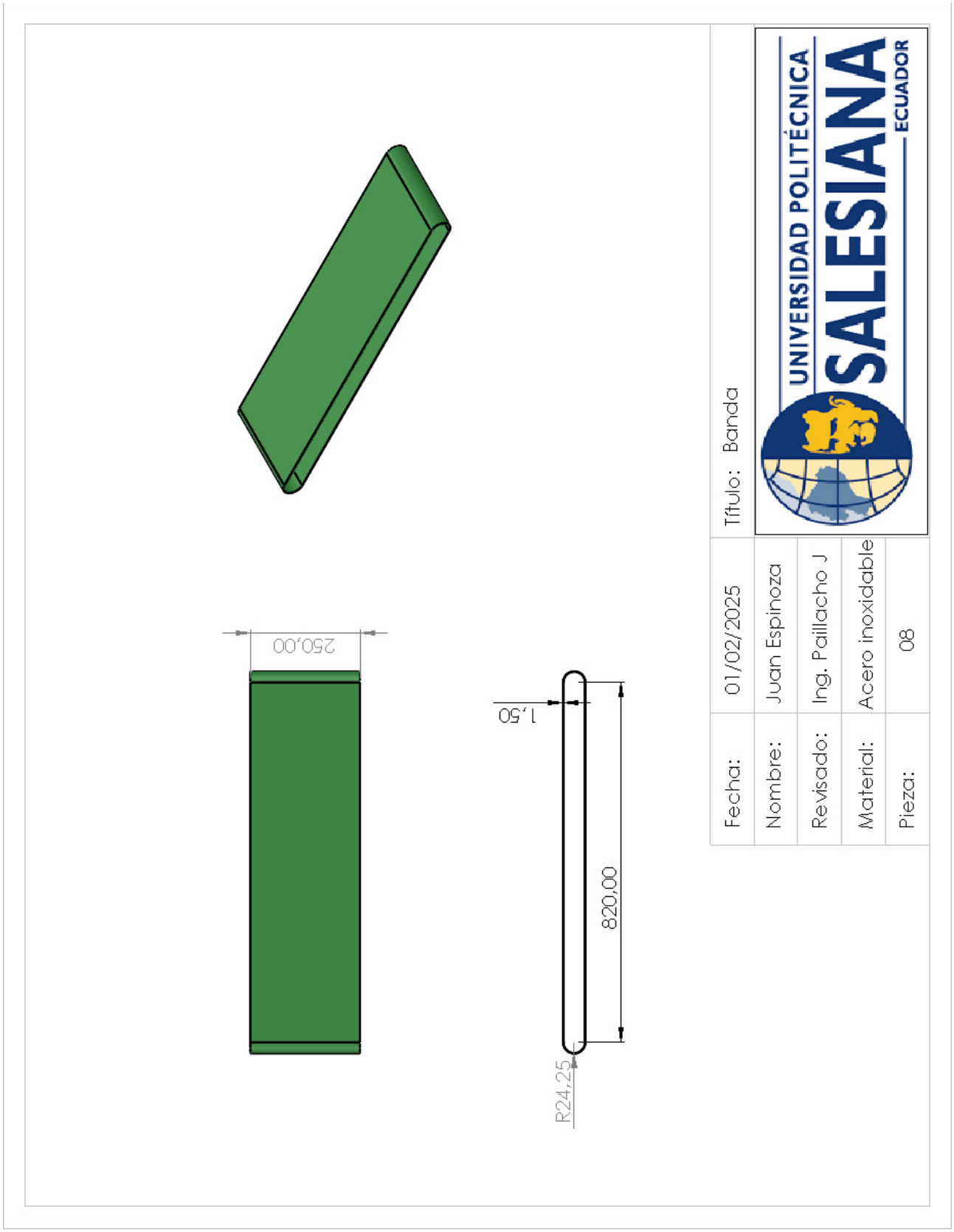


Figura 73: Banda, por Juan Espinoza.

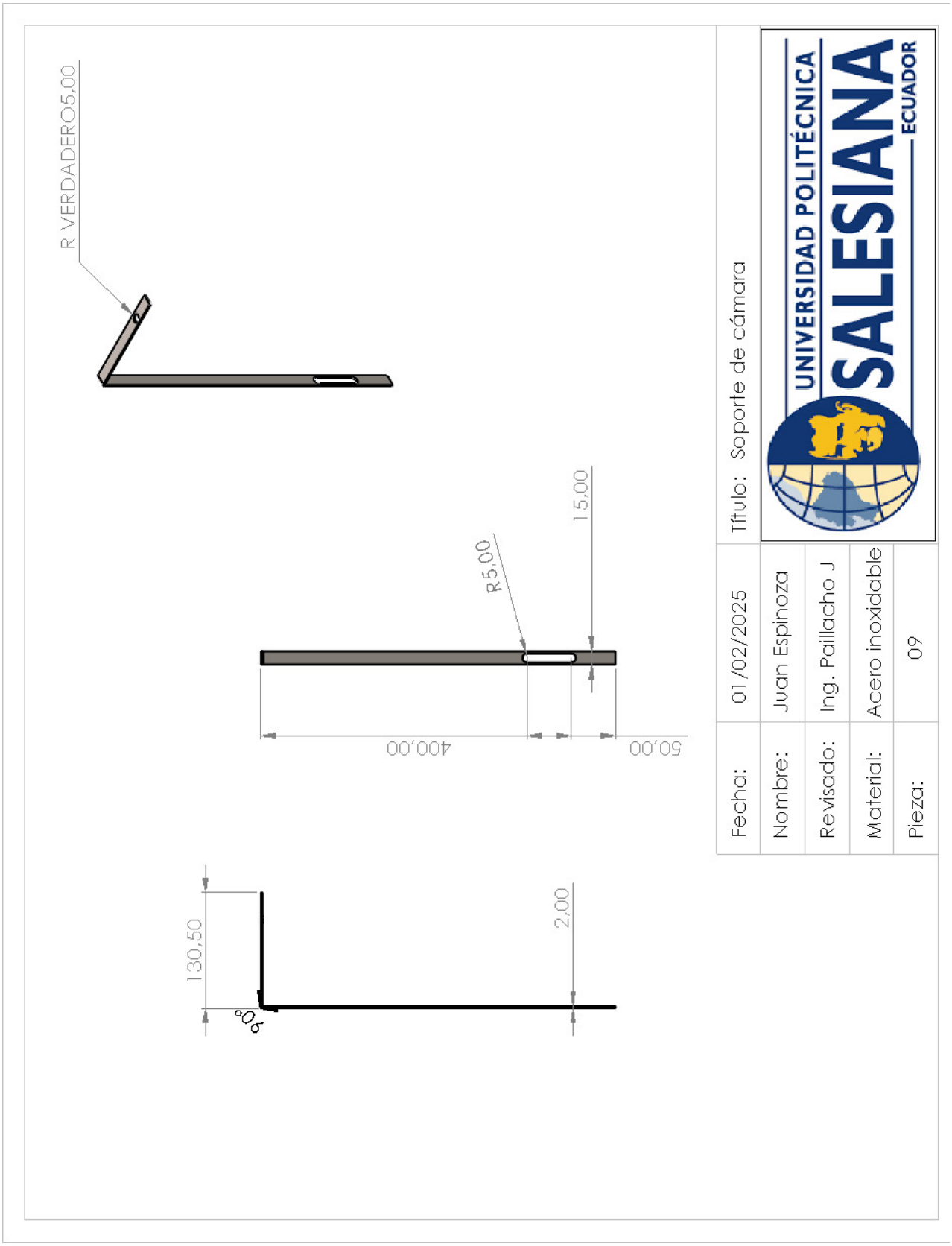


Figura 74: Soporte de cámara, por Juan Espinoza.

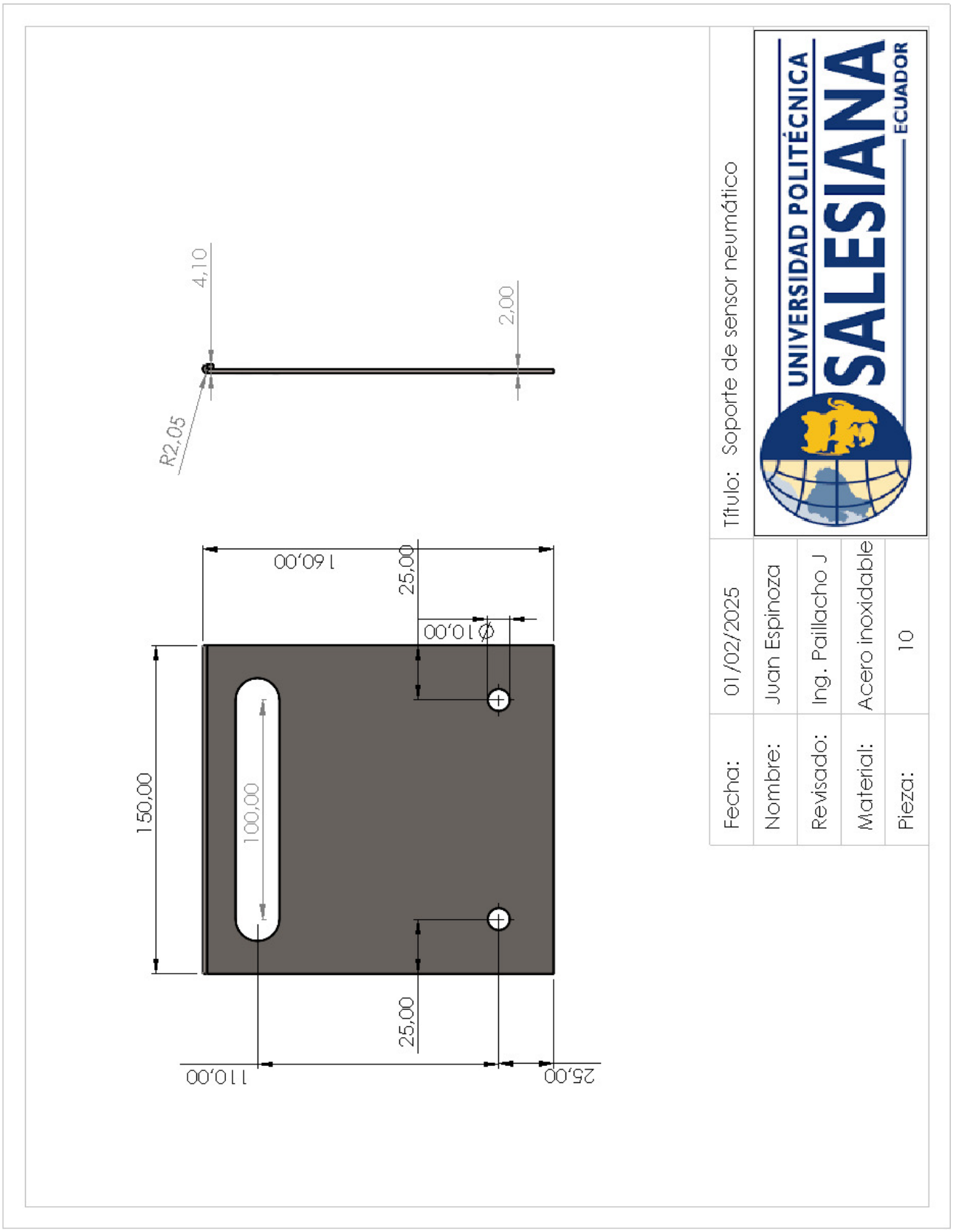


Figura 75: Soporte de sensor neumático, por Juan Espinoza.

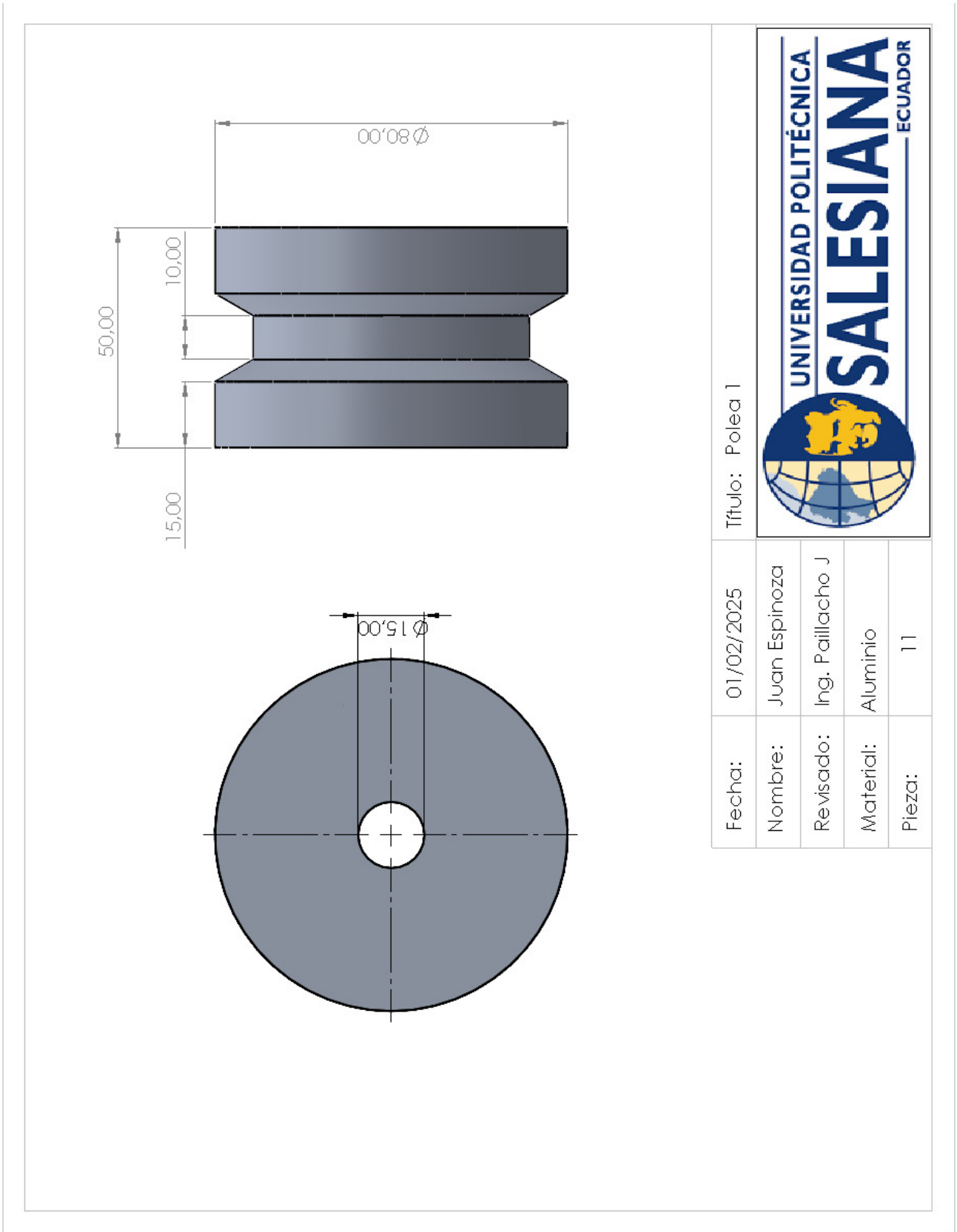


Figura 76: Polea 1, por Juan Espinoza.

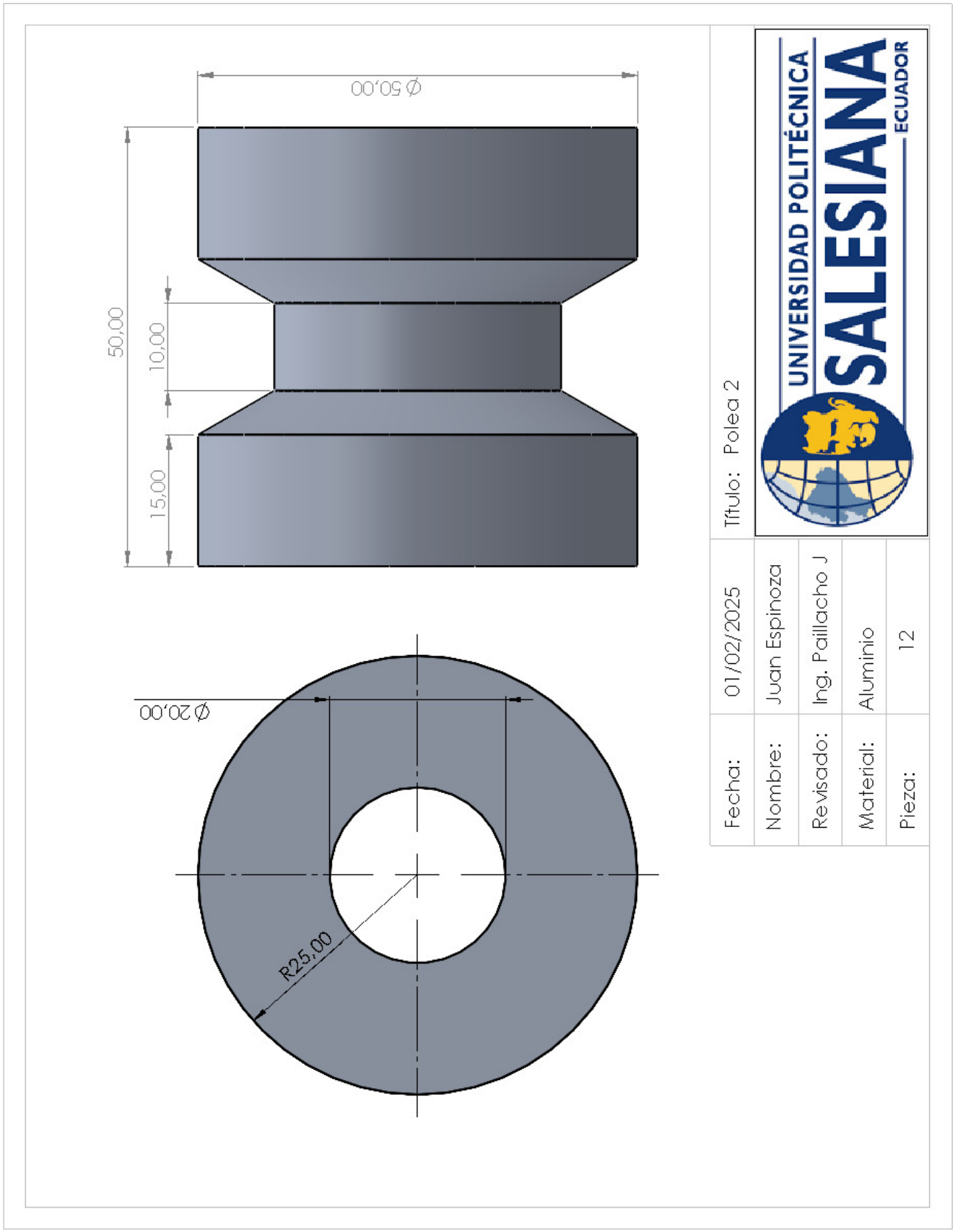


Figura 77: Polea 2, por Juan Espinoza.

Figura 78: Templador de rodillo, por Juan Espinoza.

Figura 79: Eje 2, por Juan Espinoza.

Figura 80: Arandela tipo perno, por Juan Espinoza.

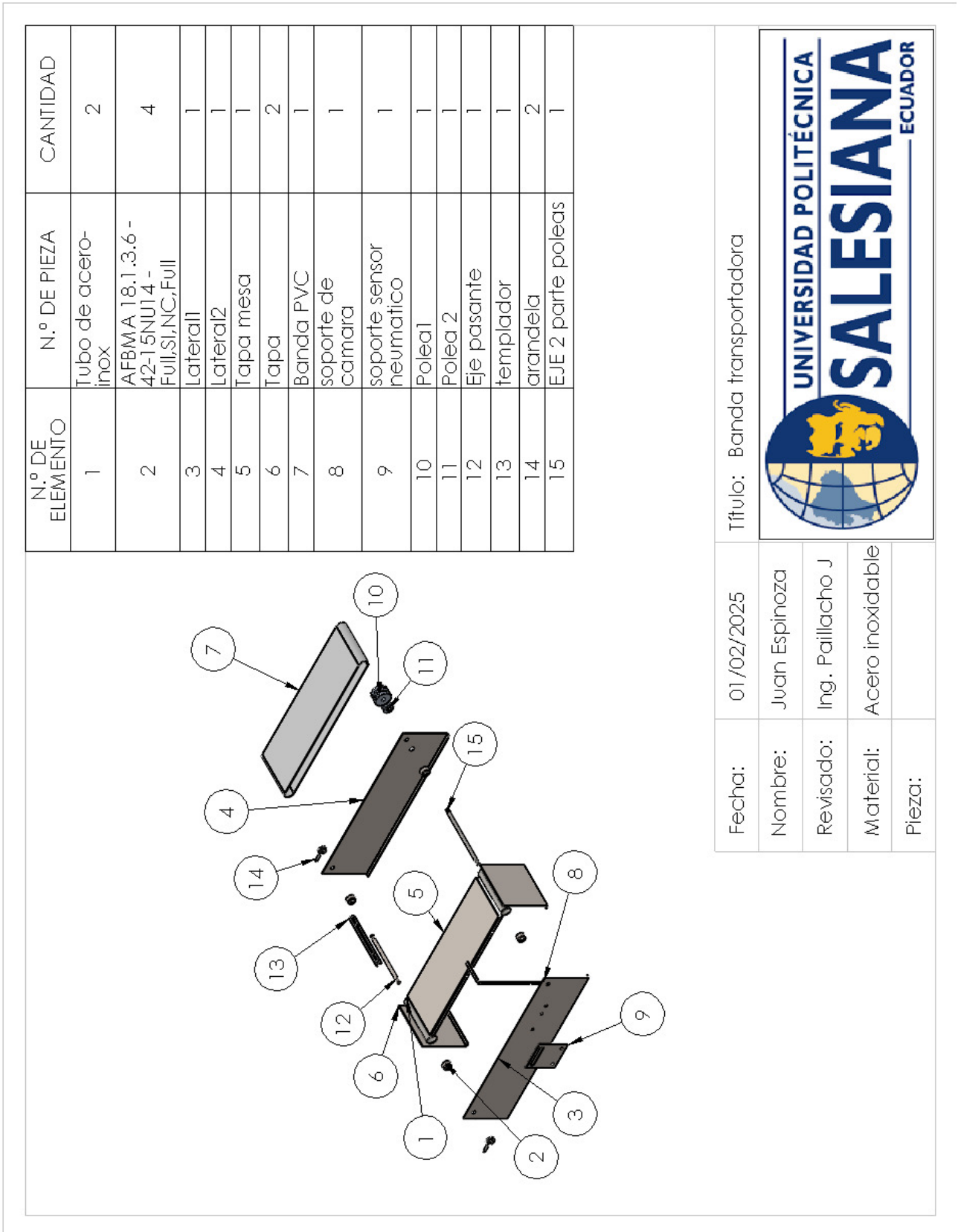


Figura 81: Ensamble total de banda transportadora, por Juan Espinoza.