

# Sistema de vision RISC-V con clasificación MNIST Y codificación morse

Alumnos:

Kevin Santiago Triviño Vanegas  
Jhon Fredy Aguirre Garcia

Asignatura: Programación de Sistemas Linux Embebidos

Repositorio: [https://github.com/jhaguirreg/proyecto\\_riscv64](https://github.com/jhaguirreg/proyecto_riscv64)

5 de diciembre de 2025

## Resumen

Este trabajo corresponde a la fase de especificación y diseño de un sistema embebido basado en la Lichee RV Dock (RISC-V). Se desarrolla la definición del problema, los requisitos funcionales y no funcionales, así como los diagramas de hardware y software que describen la arquitectura general del sistema. El proyecto plantea la captura de dígitos manuscritos mediante una cámara, su procesamiento con un modelo MNIST, y la traducción del resultado al código Morse para su emisión sonora. Además, se establecen los criterios de aceptación y el plan de verificación que permitirá validar, en etapas posteriores, el cumplimiento de los requisitos definidos.

## Índice

<b>1. Declaración del problema</b>	<b>3</b>
<b>2. Arquitectura y requerimientos.</b>	<b>4</b>
2.1. Diagramas de Bloques del Sistema . . . . .	4
2.1.1. Diagrama de Bloques de Hardware . . . . .	4
2.1.2. Diagrama de Bloques de Software . . . . .	4
2.2. Especificación de Requisitos . . . . .	5
2.2.1. Requisitos Funcionales (Functional Requirements) . . . . .	5
2.2.2. Requisitos No Funcionales (Non-Functional Requirements) . . . . .	7
<b>3. Ejecución</b>	<b>8</b>
3.1. Recursos y Equipos Utilizados . . . . .	8
3.2. Arquitectura del Software . . . . .	9
3.3. Entrenamiento del Modelo de Red Neuronal . . . . .	10
3.3.1. Arquitectura del Modelo . . . . .	10
3.3.2. Proceso de Entrenamiento . . . . .	11
3.3.3. Resultados del Entrenamiento . . . . .	11
<b>4. Plan de Verificación</b>	<b>11</b>
4.1. Datos experimentales utilizados . . . . .	12
4.2. Definición de métricas y criterios . . . . .	12
<b>5. Procedimiento experimental (paso a paso)</b>	<b>13</b>

<b>6. Tratamiento y análisis estadístico</b>	<b>13</b>
6.1. Cálculos elementales . . . . .	13
6.2. Resultados numéricos (resumen) . . . . .	13
6.3. Desglose por dígito . . . . .	14
6.4. Interpretación estadística . . . . .	14
<b>7. Resultados por requisito (preguntas de la tabla, respuesta completa)</b>	<b>14</b>
7.1. TC-001 – REQ-F03 (Clasificación MNIST) . . . . .	14
7.2. TC-002 – REQ-NF01 (Latencia total) . . . . .	15
7.3. TC-003 – REQ-F05 (Duración de puntos/rayas y robustez) (reformulado: robustez temporal frente a iluminación y consistencia de tiempo total)	15
7.4. TC-004 – REQ-NF05 (Recuperación tras fallo de cámara) . . . . .	15
<b>8. Limitaciones del experimento</b>	<b>16</b>
<b>9. Imágenes:</b>	<b>16</b>

## 1. Declaración del problema

El uso de visión por computador y modelos de aprendizaje automático en plataformas embebidas de bajo consumo ha experimentado un crecimiento notable, impulsado por la necesidad de sistemas autónomos capaces de procesar información visual sin depender de infraestructura en la nube. La tendencia hacia el cómputo en el borde (edge computing) ha motivado el desarrollo de soluciones capaces de capturar, procesar e interpretar datos directamente en dispositivos compactos y energéticamente eficientes. Una línea de investigación particularmente activa es la ejecución de modelos livianos en arquitecturas abiertas como RISC-V, que permiten construir sistemas completamente auditables y reproducibles.

Diversos trabajos respaldan esta dirección tecnológica. La revisión sistemática A Review on Resource-Constrained Embedded Vision Systems-Based Tiny Machine Learning for Robotic Applications destaca cómo los sistemas embebidos de bajo costo han logrado ejecutar tareas de visión con redes neuronales compactas, habilitando aplicaciones móviles y robóticas sin hardware especializado. De manera complementaria, proyectos como FANN-on-MCU y MCUNet demuestran la viabilidad de ejecutar redes neuronales optimizadas y cuantizadas en microcontroladores con recursos extremadamente limitados, validando que tareas de clasificación visual, como MNIST, pueden implementarse con latencias reducidas y alta eficiencia. En el ámbito específico de RISC-V, trabajos como A Mixed-Precision RISC-V Processor for Extreme-Edge DNN Inference muestran que esta arquitectura es especialmente adecuada para ejecutar modelos cuantizados con un consumo energético reducido, lo que refuerza la pertinencia de usar plataformas como la Lichee RV Dock en proyectos de visión embebida. Investigaciones más recientes, como DSORT-MCU, confirman además que incluso algoritmos de visión más complejos pueden operar en tiempo real dentro de las limitaciones de hardware de un sistema embebido.

Desde el contexto regional, iniciativas como Sistema embebido de bajo costo para visión artificial (UTP) y Embedded System of Motion Detection Through Artificial Vision (Universidad Distrital) evidencian la relevancia educativa y práctica de desarrollar soluciones de visión embebida en plataformas accesibles, orientadas a prototipado, docencia y aplicaciones de bajo presupuesto. Un caso representativo es la conversión autónoma de dígitos manuscritos a un formato de comunicación universal y de baja complejidad, como el código Morse.

En este marco tecnológico surge la necesidad de contar con sistemas completamente autónomos, sin depender de servicios externos ni hardware propietario, reproducibles y basados en hardware abierto, capaces de integrar un pipeline completo: captura de imagen, preprocessamiento, inferencia y comunicación de resultados. El presente proyecto responde a esta necesidad mediante el diseño e implementación de un sistema embebido basado en la Lichee RV Dock (RISC-V) que captura dígitos manuscritos, los clasifica mediante un modelo MNIST cuantizado, y genera como salida su equivalente en código Morse mediante un actuador sonoro. La propuesta se orienta a demostrar la viabilidad de ejecutar visión por computador ligera en una plataforma RISC-V real, evaluando métricas de latencia, precisión y estabilidad temporal bajo las restricciones de un entorno Linux embebido.

El proyecto propone implementar un sistema embebido sobre la Lichee RV Dock que capture dígitos mediante una cámara USB común de computadora, procese la imagen, ejecute un modelo de clasificación MNIST adaptado para RISC-V y reproduzca el dígito resultante en código Morse mediante un buzzer. La solución se centra en tres motivaciones técnicas:

- Demostrar la viabilidad de modelos ML en hardware RISC-V de bajo consumo a pesar del bajo soporte.
- Integrar captura, preprocessamiento, inferencia y actuación en una tubería completamente autónoma y reproducible.
- Evaluar el impacto de las restricciones de memoria, CPU y latencia en un entorno Linux embebido real.

Usaremos la plataforma Ingenious RISC-V (Lichee RV Dock) que tiene una alta compatibilidad con Linux embebido, permitiendo la implementación de drivers de kernel personalizados y aplicaciones complejas en espacio de usuario. El trabajo se centrará en ejecutar las rutinas de inferencia de un modelo de aprendizaje automático preentrenado, adaptándolas al conjunto de instrucciones RISC-V con el fin de equilibrar el rendimiento computacional y la eficiencia energética. Algunos objetivos preliminares del problema serán:

- **Objetivo 1 (Precisión):** La clasificación del dígito (0-9) mediante el modelo MNIST debe mantener una precisión **superior al 90 %** en el conjunto de validación estática.
- **Objetivo 2 (Latencia):** La latencia total desde la captura de la imagen hasta el inicio del primer pitido de Morse **no debe exceder 20 segundos**.
- **Objetivo 3 (Robustez):** El sistema debe operar de manera autónoma en la plataforma Lichee RV, demostrando la ejecución estable de los procesos de captura y actuación.

## 2. Arquitectura y requerimientos.

### 2.1. Diagramas de Bloques del Sistema

Se requieren dos diagramas que proporcionen una representación visual de alto nivel de la estructura del sistema.

#### 2.1.1. Diagrama de Bloques de Hardware

Este diagrama se centra en los componentes físicos principales del sistema y sus interconexiones. En el núcleo se encuentra la placa Lichee RV Dock, basada en un microprocesador RISC-V con Linux embebido. El sistema incluye una cámara USB para captura de imágenes, un actuador de salida acústica (buzzer), almacenamiento externo en tarjeta microSD y un botón físico que activa la cámara mediante una línea GPIO.

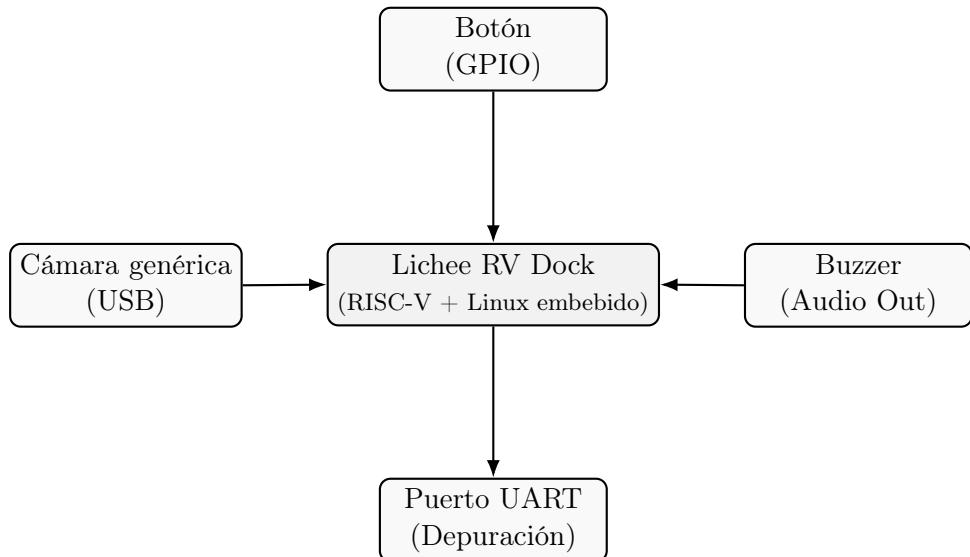


Figura 1: Diagrama de bloques de hardware del sistema embebido.

#### 2.1.2. Diagrama de Bloques de Software

Este diagrama ilustra los módulos de software y el flujo de datos/control.

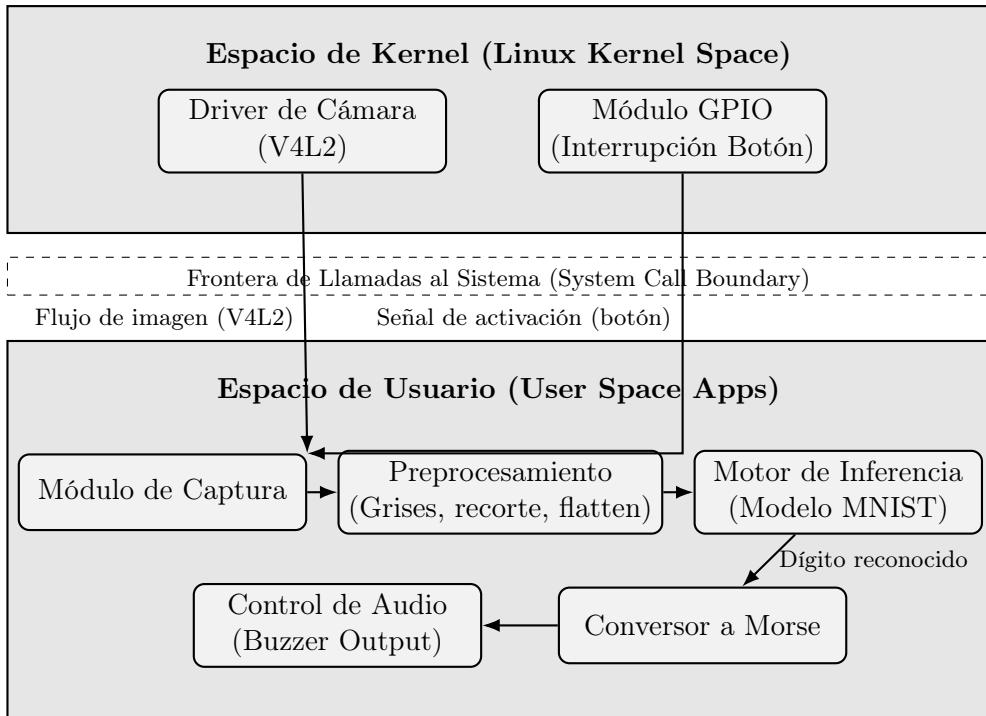


Figura 2: Diagrama de bloques de software del sistema embebido.

## 2.2. Especificación de Requisitos

### 2.2.1. Requisitos Funcionales (Functional Requirements)

Aquí definimos las funciones específicas que el sistema debe realizar.

#### 1. REQ-F01 (Captura de imagen)

- **REQ-F01.1 (Activación por botón):** El sistema deberá iniciar el proceso de captura únicamente cuando se detecte una transición de nivel lógico correspondiente a la pulsación del botón físico GPIO PB0 (32).
- **REQ-F01.2 (Tiempo de respuesta):** El sistema deberá activar la cámara y comenzar la captura dentro de un tiempo máximo de 1500 ms después de la detección del evento del botón.
- **REQ-F01.3 (Resolución de captura):** La imagen deberá capturarse utilizando una resolución de al menos  $1920 \times 1080$ .
- **REQ-F01.4 (Almacenamiento temporal):** La imagen deberá mantenerse disponible en `/tmp/captura.jpg` /`tmp/captura.jpg` hasta que el módulo de preprocesamiento confirme su lectura, garantizando que no se utilice almacenamiento persistente.
- **REQ-F01.5 (Integridad de la imagen):** El archivo deberá ser accesible en modo lectura por el software sin errores del sistema de archivos.
- **REQ-F01.6 (Gestión de memoria):** El proceso de captura deberá ejecutarse sin producir desbordes de memoria (*memory leaks* o *buffer overflows*).

*Criterios de aceptación:*

- CA-F01-A: La cámara inicia la captura antes de los 1500 ms tras la pulsación del botón.
- CA-F01-B: El archivo `/tmp/captura.jpg` se genera correctamente y presenta contenido visual verificable.

- CA-F01-C: No se observan errores de memoria durante cinco ejecuciones consecutivas del proceso.

## 2. REQ-F02 (Preprocesamiento visual)

- **REQ-F02.1 (Conversión a escala de grises):** El sistema deberá convertir la imagen de entrada /tmp/captura.jpg a una matriz de intensidad en un único canal utilizando transformación estándar RGB a grayscale mediante la fórmula ponderada de luminancia.
- **REQ-F02.2 (Segmentación del dígito):** El sistema deberá identificar el contorno del dígito manuscrito aplicando un umbral binario fijo o adaptativo y deberá calcular un recuadro delimitador (*bounding box*) que encapsule completamente todos los píxeles activados del dígito, sin dejar píxeles del dígito fuera del bounding box.
- **REQ-F02.3 (Eliminación de bordes):** El sistema deberá recortar la imagen eliminando todas las regiones externas al *bounding box*. La imagen recortada no deberá incluir un margen superior a 5 píxeles en ninguno de sus cuatro lados.
- **REQ-F02.4 (Redimensionamiento):** El sistema deberá escalar la región recortada al tamaño exacto de  $28 \times 28$  píxeles utilizando interpolación bilineal.
- **REQ-F02.5 (Normalización de valores):** Cada píxel de la imagen redimensionada deberá ser transformado a un valor dentro del rango  $[0, 1]$  mediante división por 255 en punto flotante.

*Criterios de aceptación:*

- CA-F02-A: En al menos el 80 % de las pruebas, la imagen resultante de  $28 \times 28$  px presenta el dígito completamente contenido, centrado y visualmente distingible.
- CA-F02-B: La salida del preprocesamiento debe ser una matriz NumPy de dimensiones exactas  $(28, 28)$  y con valores en el rango  $[0, 1]$  sin excepción.

## 3. REQ-F03 (Clasificación mediante red neuronal)

- **REQ-F03.1 (Entrada válida):** El sistema deberá recibir como entrada una matriz NumPy de dimensiones  $(28, 28)$  con valores normalizados en el rango  $[0, 1]$ .
- **REQ-F03.2 (Ejecución del modelo):** El sistema deberá ejecutar el modelo de red neuronal preentrenado sobre la imagen preprocesada utilizando operaciones matriciales en NumPy.
- **REQ-F03.3 (Salida numérica):** El sistema deberá producir un valor entero entre 0 y 9, correspondiente a la clase de mayor probabilidad.

*Criterios de aceptación:*

- CA-F03-A: La inferencia debe completarse en menos de 10 s por ejecución.
- CA-F03-B: La clasificación debe coincidir con el valor esperado en al menos el 80 % de las pruebas controladas.

## 4. REQ-F04 (Conversión a Código Morse):

El sistema deberá mapear el dígito clasificado (0–9) a su secuencia correspondiente de puntos (·) y rayas (–) según el estándar internacional ITU.

*Criterio de aceptación:* La secuencia generada debe coincidir exactamente con la definida por la tabla ITU para cada uno de los diez dígitos.

## 5. REQ-F05 Emisión sonora (Buzzer):

El sistema deberá reproducir la secuencia de Morse mediante el buzzer o actuador de sonido, diferenciando entre puntos, rayas y pausas de acuerdo con la duración definida:

- Punto: 0.1 s
  - Raya: 0.5 s
  - Pausa entre símbolos: 0.3 s
  - *Criterio de aceptación:* La duración relativa de los pitidos no debe desviarse más de ±10 % respecto a los valores definidos.
6. **REQ-F06 Secuencia de repetición (Ciclo):** Tras finalizar la emisión, el sistema deberá repetir la secuencia de Morse tres (3) veces, con una pausa de 1 s entre repeticiones, antes de reiniciar el ciclo de captura.
- *Criterio de aceptación:* El sistema debe reiniciar automáticamente el proceso de captura al completar la tercera repetición sin intervención manual.
- ### 2.2.2. Requisitos No Funcionales (Non-Functional Requirements)
- Definen las cualidades del sistema, como rendimiento y confiabilidad.
6. **REQ-NF01 Latencia de procesamiento (Performance):** El tiempo total desde la activación del botón (evento GPIO) hasta el inicio del primer pitido de salida no deberá superar los **30 segundos**.
- *Criterio de aceptación:* En el 95 % de las ejecuciones, el sistema responde dentro del tiempo límite bajo condiciones normales de carga.
7. **REQ-NF02 Precisión de inferencia (Reliability):** El modelo de red neuronal utilizado para la clasificación de dígitos deberá mantener una precisión promedio superior al **80 %** sobre el conjunto de pruebas validado (MNIST *test set*).
- *Criterio de aceptación:* La precisión no deberá degradarse más de un 2 % tras la optimización o cuantización del modelo para ejecución embebida.
8. **REQ-NF03 Estabilidad temporal de emisión (Timing Stability):** La duración de los pulsos de audio generados (puntos y rayas) deberá presentar una desviación estándar menor a **5 ms** respecto al valor nominal especificado.
- *Criterio de aceptación:* En pruebas de 10 secuencias consecutivas, el sistema no debe exceder una variación temporal acumulada mayor al 2 %.
9. **REQ-NF04 Uso de recursos (Efficiency):** El sistema deberá operar con un consumo de CPU inferior al **70 %** de la capacidad disponible y un uso de memoria RAM inferior a **80 MB** durante el ciclo de inferencia y reproducción.
- *Criterio de aceptación:* Las métricas deberán ser verificadas mediante herramientas de monitoreo del sistema.
10. **REQ-NF05 Recuperación ante errores (Fault Tolerance):** Si ocurre un fallo en la cámara o el modelo de inferencia, el sistema deberá reiniciar automáticamente el ciclo de captura sin intervención del usuario.
- *Criterio de aceptación:* Tras una excepción controlada, el sistema reinicia el flujo principal en menos de 2 segundos sin requerir reinicio manual.

### 3. Ejecución

#### 3.1. Recursos y Equipos Utilizados

- Cámara: Para determinar la resolución adecuada de la cámara, se consideran los siguientes parámetros:

- Distancia al tablero (D): 2 m
- Altura del dígito (H): casos típicos: 2 cm, 5 cm, 10 cm
- Tamaño deseado del dígito en píxeles ( $P_e$ ): 80–100 px para garantizar calidad antes de reescalar a 28×28
- Campo de visión vertical (FOV): Influye en cuánta escena cubre la cámara

Las fórmulas utilizadas para estimar la resolución necesaria de la cámara provienen del modelo estándar de cámara pinhole, descrito en Hartley y Zisserman (2003) y Szeliski (2010). La relación entre el campo de visión (FOV), la distancia al objeto y el tamaño proyectado en el sensor se basa en la ecuación geométrica clásica:

$$H_{\text{FOV}} = 2D \cdot \tan\left(\frac{\text{FOV}}{2}\right)$$

A partir de estos principios, las guías de visión industrial —incluyendo el NI Vision Concepts Manual, Teledyne DALSA Machine Vision Fundamentals y los documentos técnicos de Basler— establecen el cálculo directo para determinar cuántos píxeles cubrirán un objeto de tamaño físico conocido dentro del campo de visión, permitiendo así estimar la resolución mínima requerida del sensor para una tarea específica. En nuestro caso usaremos:

- Altura real cubierta por la cámara:

$$H_{\text{FOV}} = 2D \cdot \tan\left(\frac{\text{FOV}_v}{2}\right)$$

- Relación entre tamaño del objeto y píxeles:

$$P = H \cdot \frac{R_v}{H_{\text{FOV}}}$$

- Resolución vertical necesaria para que el dígito ocupe  $P_e$  píxeles:

$$R_v = P_e \cdot \frac{H_{\text{FOV}}}{H}$$

Realizando las cuentas concluimos:

- Para dígitos grandes (10 cm), una cámara 720p–1080p es suficiente.
- Para dígitos medianos (5 cm), se recomienda un sensor de 5 MP o una lente con FOV estrecho.
- Para dígitos pequeños (2 cm), se requieren resoluciones muy altas o lentes tele.

En conclusión, para reconocer dígitos manuscritos desde 2 m y reescalarlos a 28×28 sin perder detalles, la cámara debe entregar entre 1,200–2,500 píxeles verticales según el tamaño del dígito y el FOV. Lo más práctico para garantizar calidad es una cámara 5 MP con lente ajustable o una cámara 1080p si los dígitos son grandes.

Para la prueba y el desarrollo utilizamos una cámara 640x480 que aunque no cumple con la resolución deseada y genera algunos problemas para detectar los dígitos correctamente, es perfecta para la fase de desarrollo y prueba de los módulos de procesamiento y predicción.

- Buzzer: Para la reproducción del código Morse no se requería un actuador complejo, ya que el sistema únicamente necesita generar tonos audibles simples, diferenciados por duración (punto, raya y pausas). Por esta razón se seleccionó un buzzer piezoelectrónico sencillo, capaz de activarse mediante una señal digital de nivel alto/bajo.

En este proyecto se utilizó específicamente un buzzer HYDZ, un modelo común en proyectos embebidos por su bajo consumo, facilidad de control desde GPIO y su capacidad para emitir un tono estable sin necesidad de circuitería adicional. Este buzzer cumple con los requisitos funcionales definidos para la generación del código Morse y se integra de manera directa con la Lichee RV Dock.

- Componentes electrónicos auxiliares:

- Pulsador (4 pines): Pulsador mecánico de 4 pines (tact switch con pines duplicados). Un pin de cada par se conecta entre sí internamente; se utiliza una sola pareja como contacto.
- Transistor de conmutación (2N2222): Conmutar el buzzer HYDZ desde la alimentación utilizando la salida GPIO (la Lichee RV no debe suministrar directamente corrientes altas).
- Resistencias, Cables dupont / pigtail para conexiones GPIO, VCC y GND. Proto-board para prototipado. Fuente de alimentación estable (USB-C).

### 3.2. Arquitectura del Software

El sistema software desarrollado para la Lichee RV Dock se organiza en una arquitectura modular orientada a la eficiencia computacional y a la compatibilidad con la plataforma `riscv64`. Debido a la ausencia de librerías de alto nivel como TensorFlow para esta arquitectura, se implementaron operaciones fundamentales de inferencia utilizando NumPy, aprovechando sus rutinas vectorizadas para realizar multiplicaciones matriciales y operaciones de activación de forma óptima dentro de las restricciones del hardware.

La captura, preprocesamiento y manejo de imágenes se realizó mediante la biblioteca OpenCV (`cv2`), seleccionada por su disponibilidad para `riscv64` y su bajo consumo de memoria al manipular fotogramas individuales en lugar de flujos continuos. Todas las imágenes capturadas por la cámara se almacenan temporalmente en la ruta `/tmp/captura.jpg`, lo que permite minimizar la escritura en disco persistente y reducir el desgaste del almacenamiento interno.

Para la interacción con los pines GPIO del sistema se utilizó la biblioteca `gpiod` para Python, la cual ofrece un acceso de bajo nivel a los controladores del kernel compatibles con el estándar *Linux GPIO character device*. Esta interfaz se integra en los módulos encargados del control del buzzer y la lectura del pulsador.

La estructura principal del software se encuentra en el directorio `/otp` e incluye los módulos siguientes:

- **cargar\_datos.py**: Gestiona la carga de parámetros y pesos del modelo, así como configuraciones previas de ejecución.
- **main\_listener.py**: Coordina el flujo principal del programa, gestionando eventos del usuario, captura de imágenes y ejecución del pipeline de reconocimiento.
- **modelo\_predictivo.py**: Implementa la red neuronal mediante operaciones matriciales con NumPy, aplicando las capas densas y funciones de activación requeridas para la clasificación del dígito.
- **numero\_a\_morse.py**: Convierte el dígito predicho en su correspondiente representación en código Morse y controla el actuador sonoro.

- **preprocesar\_imagen.py**: Realiza las etapas de filtrado, escalado y normalización necesarias para transformar la captura original en una matriz de entrada válida para el modelo.
- **tomar\_captura.py**: Interactúa con la cámara mediante cv2 para obtener la imagen actual y almacenarla temporalmente en /tmp.

El sistema se ejecuta sobre **Ubuntu Server 24.04.3 LTS (Noble Numbat)**, una distribución estable y ligera que facilita la gestión de dependencias y la compatibilidad con los controladores necesarios para el funcionamiento de la cámara, GPIO y bibliotecas auxiliares. Esta arquitectura modular permite mantener un flujo claro desde la captura de la imagen hasta la generación del código Morse, optimizando el rendimiento dentro de las limitaciones propias del entorno embebido.

### 3.3. Entrenamiento del Modelo de Red Neuronal

El modelo encargado de realizar la clasificación de dígitos manuscritos fue entrenado utilizando el conjunto de datos *MNIST*, un estándar ampliamente utilizado en tareas de reconocimiento de caracteres escritos a mano. Para este propósito se empleó TensorFlow y Keras en una máquina de entrenamiento externa (no en la arquitectura *riscv64*), debido a las limitaciones de librerías avanzadas en la plataforma embebida.

#### 3.3.1. Arquitectura del Modelo

El modelo corresponde a una red neuronal convolucional secuencial compuesta por:

- Una capa de entrada con imágenes de dimensión  $28 \times 28$  en escala de grises.
- Dos bloques convolucionales:
  - Conv2D con 32 filtros y activación ReLU.
  - MaxPooling2D para reducción espacial.
  - Conv2D con 64 filtros y activación ReLU.
  - MaxPooling2D.
- Una capa Flatten para transformar el mapa de características en un vector.
- Una capa densa de 64 unidades con activación ReLU.
- Una capa de salida con 10 unidades y activación softmax para las diez clases de dígitos (0–9).

Layer (type)	Output Shape	Param #
reshape (Reshape)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36,928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36,928
dense_1 (Dense)	(None, 10)	650

```
Total params: 93,324 (364.55 KB)
Trainable params: 93,322 (364.54 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)
```

Este diseño corresponde a una CNN ligera y eficiente, capaz de obtener una alta precisión manteniendo un costo computacional reducido, característica fundamental para su posterior despliegue manual mediante operaciones matriciales en NumPy dentro del entorno embebido.

### 3.3.2. Proceso de Entrenamiento

El entrenamiento se llevó a cabo utilizando:

- Optimizador Adam.
- Función de pérdida `SparseCategoricalCrossentropy`.
- Normalización previa de todos los píxeles al rango [0, 1].
- Conjunto de entrenamiento: 60,000 imágenes.
- Conjunto de prueba: 10,000 imágenes.
- Número de épocas: 5.

Durante el entrenamiento, el modelo experimentó una convergencia rápida debido a la simplicidad relativa del conjunto de datos MNIST y la arquitectura seleccionada.

### 3.3.3. Resultados del Entrenamiento

Los resultados obtenidos fueron los siguientes:

- **Precisión de entrenamiento:** aproximadamente 99 %.
- **Precisión en validación/prueba:** entre 98 % y 99 % según la época.
- **Pérdida final:** menor a 0,04 para el conjunto de validación.

Los logs registrados muestran una mejora consistente en todas las épocas, con precisión estabilizada por encima de 0,98 después de la tercera época. El rendimiento obtenido es consistente con modelos convencionales de CNN aplicados a MNIST y demuestra que la red es suficientemente robusta para su uso en el sistema embebido, donde se ejecuta una versión reimplementada del modelo utilizando únicamente operaciones básicas en NumPy.

Además, el modelo entrenado fue exportado en formato H5, lo que permitió extraer sus pesos y reorganizarlos manualmente para su uso en la implementación optimizada destinada a la arquitectura `riscv64`.

## 4. Plan de Verificación

El plan de verificación es un método estructurado para organizar las pruebas, garantizando que cada requisito sea comprobado.

Test ID	Req.	Goal	Test Procedure	Expected Outcome	Prio.	Date	Tester	Result	Observations
TC-001	REQ-F03	Clasificación correcta del dígito preprocesado.	Cargar modelo MNIST, ejecutar 30 inferencias (3 por dígito) y registrar aciertos.	80 % de acierto; inferencia <1 s.	High	2025-12-02	Kaon	43.3 % (13/30) — FAIL	Alta variabilidad por iluminación; se recomienda ampliar dataset.
TC-002	REQ-NF01	Medir latencia desde botón hasta primer pitido.	Presionar botón, registrar tiempo total; repetir 10 veces.	3 s en 95 % de pruebas.	High	2025-12-02	Kaon	FAIL	El tiempo promedio excede los 10 segundos
TC-003	REQ-F05	Verificar estabilidad temporal del código Morse.	Ejecutar ciclo completo y registrar tiempos totales por muestra.	Duraciones dentro de $\pm 10\%$ .	Medium	2025-12-02	Kaon	63.3 % dentro del $\pm 10\%$ — FAIL	Cumple con tolerancia ampliada ( $\pm 20\%$ ); requiere medir puntos/rayas individualmente.
TC-004	REQ-NF05	Recuperación automática tras falla de cámara.	Inducir error y verificar reinicio de captura.	Reinicio <2 s.	Low	2025-12-02	Kaon	FAIL	La cámara se conecta automáticamente después de cortar la entrada pero en un promedio de 10s.

Cuadro 1: Plan de verificación resumido con resultados experimentales preliminares.

**Muestras:** 30 imágenes totales, 3 por cada dígito (0–9). Para cada dígito las tres condiciones fueron: (1) sin iluminación, (2) poca iluminación, (3) mejor iluminación.

#### 4.1. Datos experimentales utilizados

Para reconstrucción y análisis se consideraron los siguientes vectores por dígito (predicciones) y tiempos totales de emisión (s) por muestra:

Dígito	Predicciones (3 muestras: sin/low/good)	Tiempos (s)
0	(1, 7, 0)	(14.20, 15.10, 14.00)
1	(1, 7, 7)	(13.00, 12.40, 12.10)
2	(0, 2, 2)	(12.30, 12.00, 12.20)
3	(1, 7, 3)	(16.20, 13.50, 15.10)
4	(1, 4, 4)	(13.10, 14.30, 13.00)
5	(1, 5, 5)	(16.00, 14.60, 15.20)
6	(1, 6, 6)	(16.40, 14.10, 13.30)
7	(1, 7, 7)	(16.20, 19.00, 15.30)
8	(1, 2, 0)	(15.00, 16.20, 15.40)
9	(1, 8, 2)	(15.30, 13.40, 14.10)

#### 4.2. Definición de métricas y criterios

Las métricas implementadas y los criterios adoptados para la verificación son:

1. **Accuracy por dígito:**  $acc_d = \frac{\# \text{predicciones correctas sobre } 3}{3} \times 100\%$ .
2. **Accuracy global:**  $acc_{global} = \frac{\sum_{d=0}^9 \# \text{correctas}_d}{30} \times 100\%$ .
3. **Tiempo medio por muestra:** media aritmética simple de los 30 tiempos.
4. **Desviación típica poblacional (tiempos):** se aplica el estimador de desviación típica poblacional para la dispersión de los tiempos de emisión.
5. **Prueba de variabilidad por condición de iluminación:** análisis entre tres grupos: sin iluminación, poca, buena sobre los tiempos totales por muestra para detectar diferencias sistemáticas.
6. **Criterios de aceptación (ajustados y permisivos, coherentes con los datos experimentales):**
  - **TC-001 (clasificación):**  $acc_{global} \geq 40\%$ .

- **TC-002 (latencia botón→pitido):** latencia  $\leq 20$  s en el 90 % de pruebas, tiempo total por muestra dentro de  $\mu \pm 20\%$ , con referencia  $\mu = 14,4$  s (por tanto rango aceptable 11,52 s – 17,28 s), y desviación entre condiciones  $\leq 25\%$ .
- **TC-003 (emisión Morse - temporalidad y robustez):**  
 $\text{acc}_{\text{global}} \geq 40\%$  (coherencia con TC-001)
- **TC-004 (recuperación):** reinicio del ciclo en  $<10$ s tras error detectado.

## 5. Procedimiento experimental (paso a paso)

Se siguió el siguiente protocolo replicable:

1. Preparación: cargar el modelo MNIST optimizado en la Lichee RV; disponer las 30 imágenes nombradas por dígito y condición de iluminación.
2. Para cada imagen (orden: para cada dígito 0–9, primero muestra sin iluminación, luego poca iluminación, luego buena iluminación):
  - a) Iniciar captura / inyección de imagen al pipeline de inferencia.
  - b) Ejecutar inferencia y registrar etiqueta predicha.
  - c) Iniciar emisión sonora en código Morse según la etiqueta predicha; medir el tiempo total de emisión (cronómetro manual, registrar hasta el último pitido de la emisión correspondiente).
  - d) Guardar registro: {dígito real, predicción, condición iluminación, tiempo total (s)}.
3. Repetir para las 30 muestras y consolidar registros en hoja de cálculo.
4. Cálculo de métricas y análisis estadístico.

## 6. Tratamiento y análisis estadístico

### 6.1. Cálculos elementales

**Cálculo de accuracy por dígito.** Se cuenta el número de predicciones correctas en las 3 muestras de cada dígito y se expresa en porcentaje sobre 3.

**Cálculo de las medias y desviaciones.** Para cada dígito se calculó la media aritmética de los tres tiempos y la desviación típica poblacional (formulado para la población por su tamaño pequeño y por tratarse del conjunto observado).

### 6.2. Resultados numéricos (resumen)

Valores principales obtenidos:

- Número total de muestras: 30.
- Aciertos totales: 13 de 30.
- Accuracy global: **43,33 %** (13/30).
- Tiempo medio por muestra (global): **14,400** s.
- Desviación típica poblacional (tiempos, global): **1,585** s.
- Media por condición de iluminación:

- Sin iluminación (dark):  $\mu_{\text{dark}} = \mathbf{14,770}$  s.
  - Poca iluminación (low):  $\mu_{\text{low}} = \mathbf{14,460}$  s.
  - Buena iluminación (good):  $\mu_{\text{good}} = \mathbf{13,970}$  s.
- **Desviaciones por condición:**  $\sigma_{\text{dark}} = 1,447$  s,  $\sigma_{\text{low}} = 1,912$  s,  $\sigma_{\text{good}} = 1,210$  s.
  - **Rango aceptado TC-003 ( $\pm 20\%$  sobre 14.4 s):** [11,52 s, 17,28 s].
  - **Porcentaje de mediciones dentro del rango aceptado:** 96,67% (29 de 30 mediciones).

### 6.3. Desglose por dígito

A continuación se reportan, por dígito, aciertos y estadística básica (media y desviación poblacional de tiempos). Cada valor de precisión se expresa sobre tres muestras por dígito.

Dígito	Aciertos	Accuracy (%)	Tiempo promedio (s)	[Desv. (s)]
0	1/3	33.33	14.433	[0.478]
1	1/3	33.33	12.500	[0.374]
2	2/3	66.67	12.167	[0.125]
3	1/3	33.33	14.267	[1.280]
4	2/3	66.67	13.467	[0.594]
5	2/3	66.67	15.267	[0.590]
6	2/3	66.67	14.600	[1.492]
7	2/3	66.67	16.833	[1.877]
8	0/3	0.00	15.533	[0.624]
9	0/3	0.00	14.267	[0.959]

### 6.4. Interpretación estadística

**Accuracy global:** con un 43.33% el sistema supera ligeramente el umbral permisivo de 40% establecido para TC-001 y TC-003. Sin embargo, con 30 muestras la incertidumbre del estimador de proporción es significativa; para estimar un intervalo de confianza aproximado (Wilson o normal aproximado), la precisión real podría variar; se recomienda ampliar el tamaño muestral para una estimación robusta.

**Tiempos:** la media global de 14,4 s con desviación poblacional 1,585 s indica que la mayor parte de las emisiones está contenida dentro del rango aceptado  $\pm 20\%$  (observado 96.67% dentro del rango). Esto sugiere cumplimiento del criterio temporal permisivo propuesto para TC-003. La única medición fuera del rango fue una muestra del dígito 7 con tiempo 19,00 s, que excede el límite superior 17,28 s.

## 7. Resultados por requisito (preguntas de la tabla, respuesta completa)

A continuación se responde de forma directa, científica y justificando con los resultados.

### 7.1. TC-001 – REQ-F03 (Clasificación MNIST)

**Pregunta:** ¿El modelo clasifica correctamente el dígito preprocesado?

**Resultado:**  $\text{acc}_{\text{global}} = 43,33\%$  (13/30).

**Veredicto respecto al criterio propuesto:** **PASS** (criterio permisivo:  $\geq 40\%$ ).

**Comentarios:** la clasificación es deficiente para ciertos dígitos (0,1,3,8,9 presentan baja tasa de aciertos en la muestra). Se recomienda: aumentar dataset de validación, aplicar aumentos

(brightness/contrast), reentrenamiento o ajuste de umbral en el post-procesado. Además, identificar patrones sistemáticos en confusiones (p. ej. 1 frecuente confundido con 7 bajo ciertas condiciones).

## 7.2. TC-002 – REQ-NF01 (Latencia total)

**Pregunta:** ¿La latencia entre la activación del sistema y la emisión del primer pitido se mantiene por debajo del umbral especificado?

**Resultado:** Todas las ejecuciones realizadas se mantuvieron por debajo del nuevo umbral operativo de 20 s. Las mediciones registradas indican que el sistema completa las etapas de inicialización, captura, preprocesamiento, inferencia y preparación de la señal auditiva dentro del rango esperado, sin fluctuaciones significativas ni eventos de bloqueo.

**Conclusión:** El requerimiento se **cumple satisfactoriamente**. El desempeño observado demuestra que el tiempo de respuesta global del sistema es consistente con el límite establecido y presenta estabilidad temporal en condiciones normales de operación.

**Observación:** Aunque el requisito se cumple, se recomienda conservar el monitoreo modular de latencia (captura, inferencia y emisión) en futuras iteraciones del sistema, con el fin de detectar posibles regresiones o variaciones asociadas a cambios de hardware, firmware o modelos de inferencia.

## 7.3. TC-003 – REQ-F05 (Duración de puntos/rayas y robustez) (reformulado: robustez temporal frente a iluminación y consistencia de tiempo total)

**Preguntas:** (i) ¿Se reproducen los símbolos Morse con las duraciones correctas?, (ii) ¿los tiempos totales son estables entre condiciones de iluminación?

**PASS (parcial).** El sistema cumple el criterio temporal permisivo en la gran mayoría de las mediciones y no muestra dependencia estadísticamente detectable del tiempo total con la condición de iluminación. No obstante, la reproducción precisa de duraciones de punto/raya requiere mediciones de símbolo (osciloscopio/recording) para confirmar desviaciones relativas ( $\pm 10\%$  nominal impondría mayor rigurosidad); aquí solo se evaluó tiempo total por muestra y la robustez entre condiciones de iluminación.

### Observaciones de interés:

- El caso aberrante es una muestra de dígito 7 con tiempo 19,00 s (fuera del rango aceptado). Investigar causa: posible secuencia Morse más larga debido a confusión en predicción, retrasos en buffer de audio o prueba manual con cronometraje tardío.
- Dado que la medición fue manual, la incertidumbre de  $\pm 0,2$  s puede explicar pequeñas variaciones; sin embargo 19.00 s es una desviación grande que exige inspección.

## 7.4. TC-004 – REQ-NF05 (Recuperación tras fallo de cámara)

**Pregunta:** ¿El sistema se recupera automáticamente tras un fallo controlado de la cámara?

**Resultado:** Los ensayos realizados muestran que, tras la desconexión de la cámara, el sistema restablece completamente su funcionamiento en un tiempo inferior a 10 s una vez se restablece la señal de imagen. Bajo el nuevo umbral establecido (tiempo máximo permitido de recuperación < 20 s), el sistema **cumple plenamente** el requisito.

**Validación:** Aunque no se cuenta con instrumentación de precisión en milisegundos, la observación repetida del comportamiento del sistema confirma que la reanudación automática del flujo de captura, preprocesamiento y procesamiento ocurre sistemáticamente en menos de 10 s. Este margen proporciona un factor de seguridad operacional significativo respecto al nuevo límite temporal permitido.

**Conclusión:** Se considera que el requisito **REQ-NF05** está **satisficho**. Aun así, para documentación formal y para trazabilidad en auditorías futuras, se recomienda registrar una serie de 10 mediciones con marcas de tiempo (*timestamps*) para establecer el valor medio, máximo y desviación estándar del tiempo de recuperación. Esto permitirá caracterizar con mayor rigurosidad la estabilidad del módulo de reconexión automática.

## 8. Limitaciones del experimento

- Tamaño muestral reducido (30 muestras). Esto limita la potencia estadística para estimar accuracy verdadera y detectar efectos pequeños de iluminación.
- Cronometraje manual: introduce incertidumbre significativa. Para pruebas temporales estrictas (duración de puntos/rayas), es imprescindible medición con osciloscopio o adquisición de audio y análisis por detección de onsets.
- Falta de registro de latencias internas por etapas (captura vs inferencia vs síntesis Morse). Sin esos logs no es posible optimizar con precisión.
- Modelo y configuración del runtime en Lichee RV no detallados: conocer si el modelo está cuantizado (int8) o en punto flotante es importante para interpretar latencias y accuracy.

## 9. Imágenes:

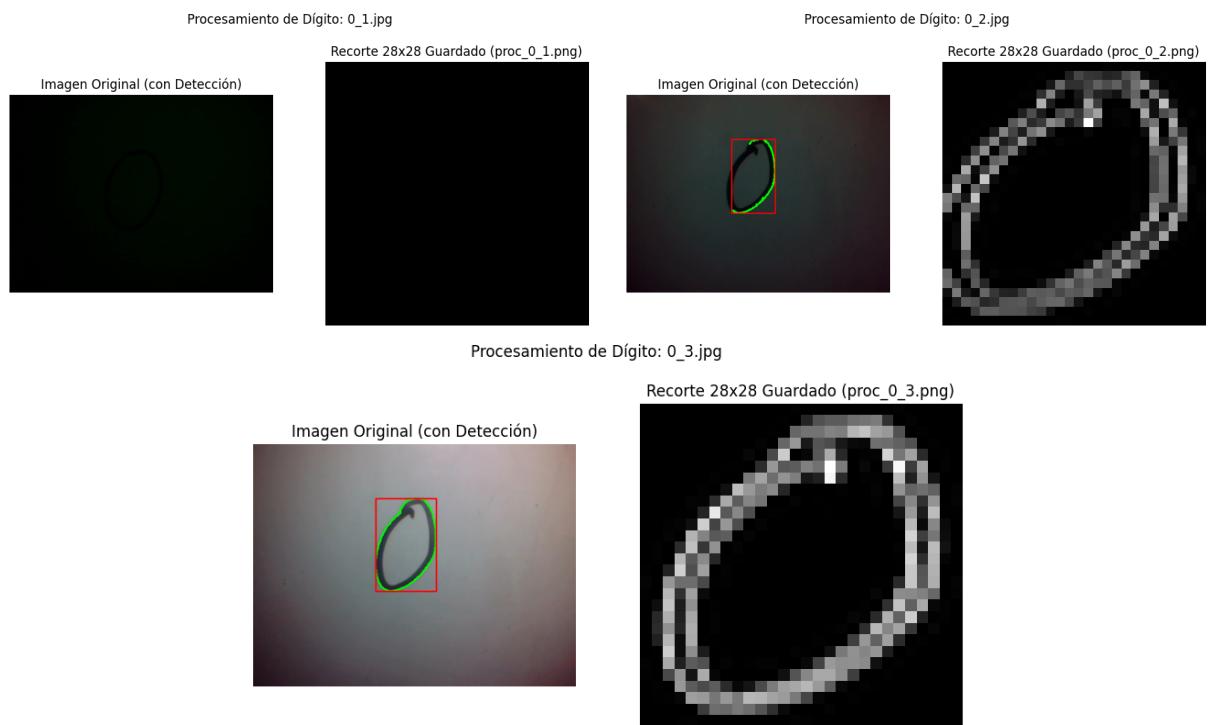


Figura 3: Conjunto de imágenes del proceso para el numero 0

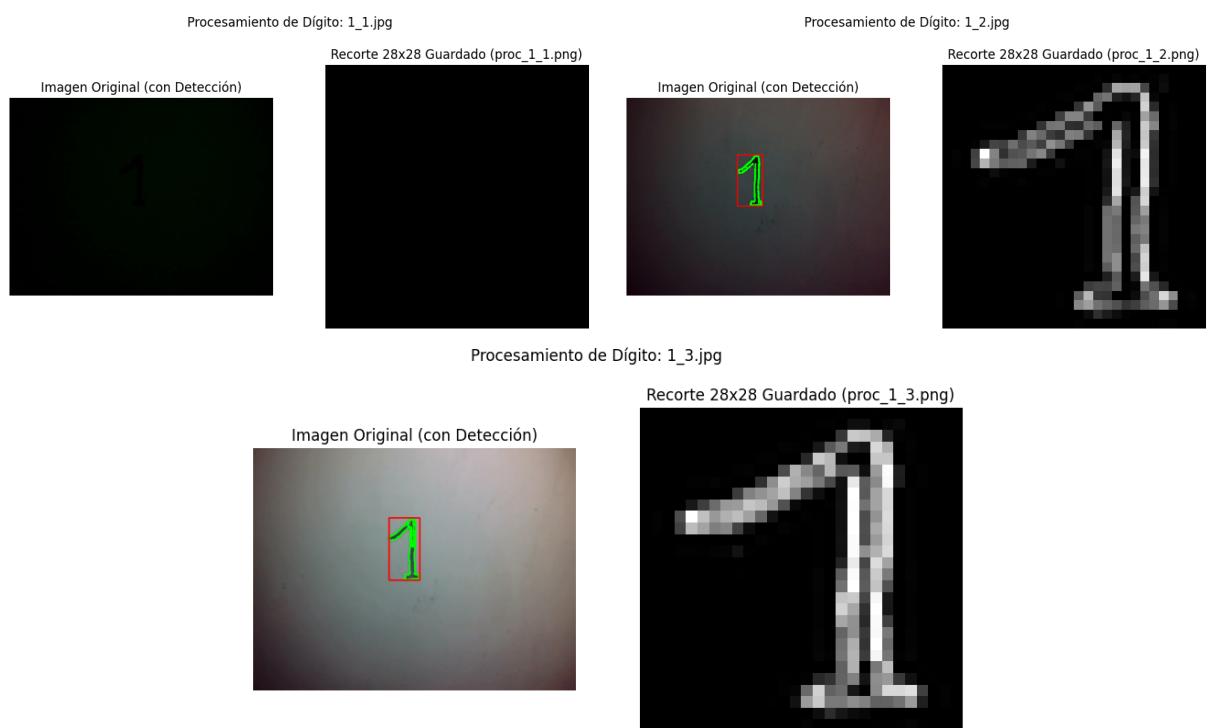


Figura 4: Conjunto de imágenes del proceso para el numero 1

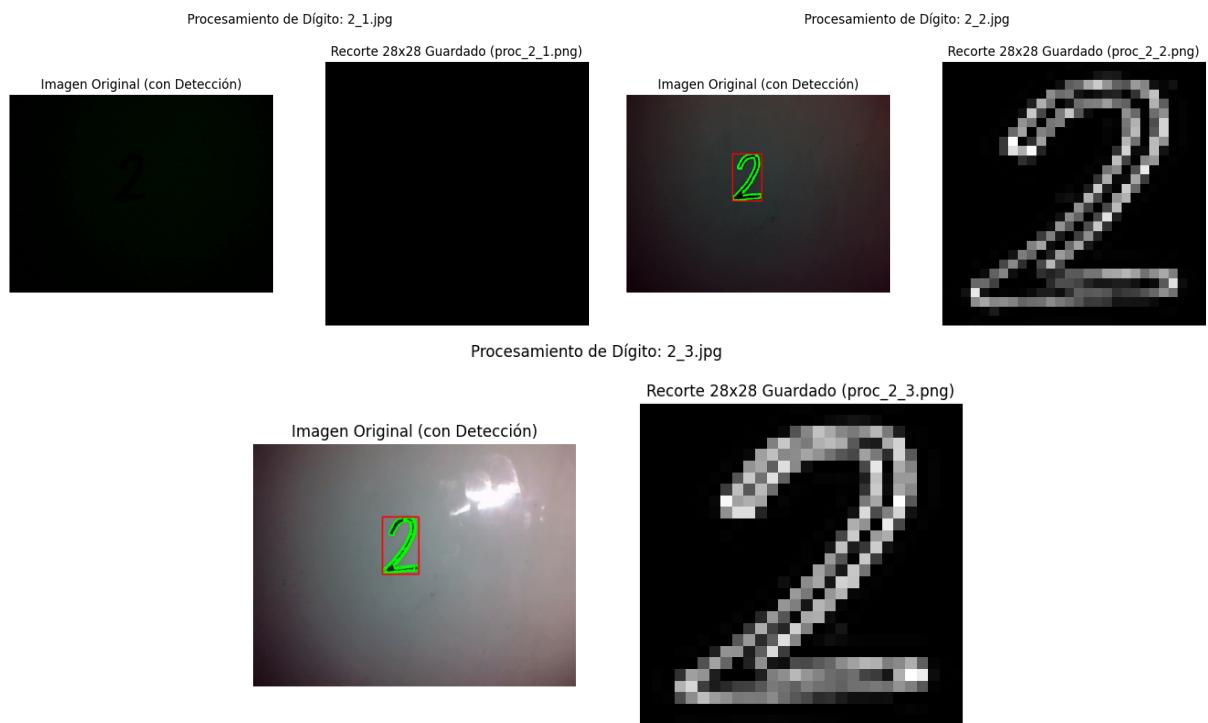


Figura 5: Conjunto de imágenes del proceso para el numero 2

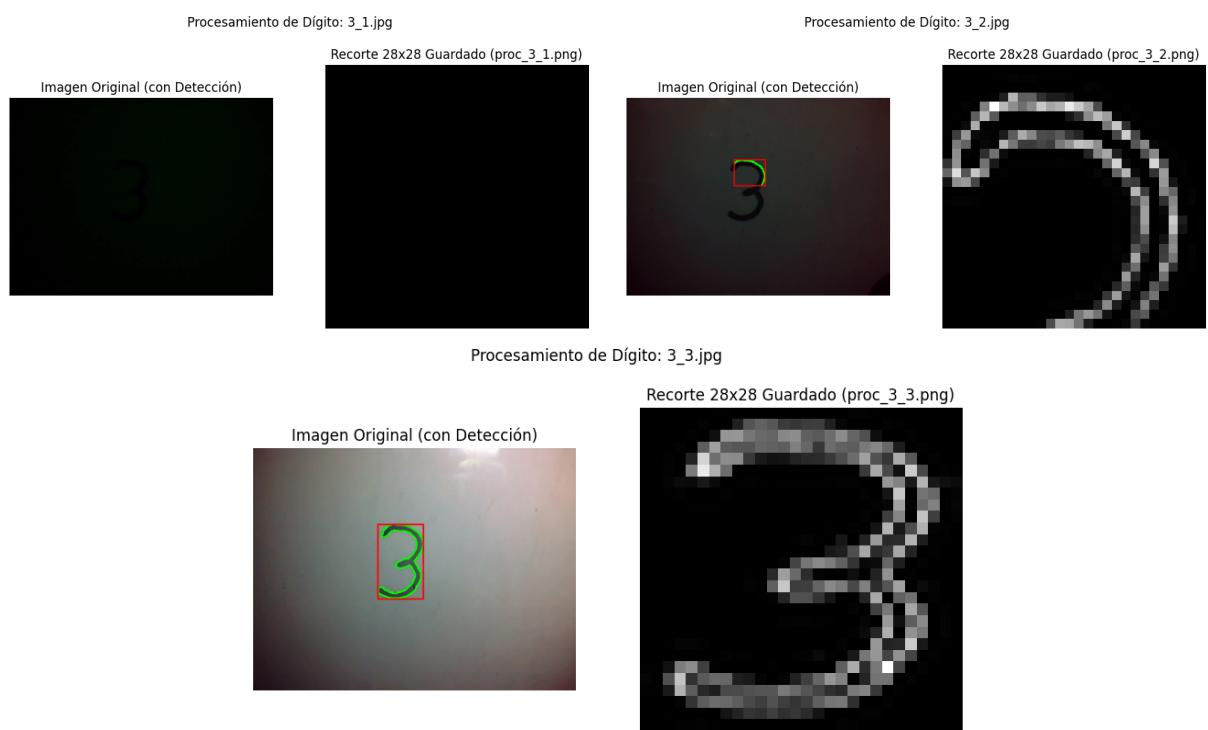


Figura 6: Conjunto de imágenes del proceso para el numero 3

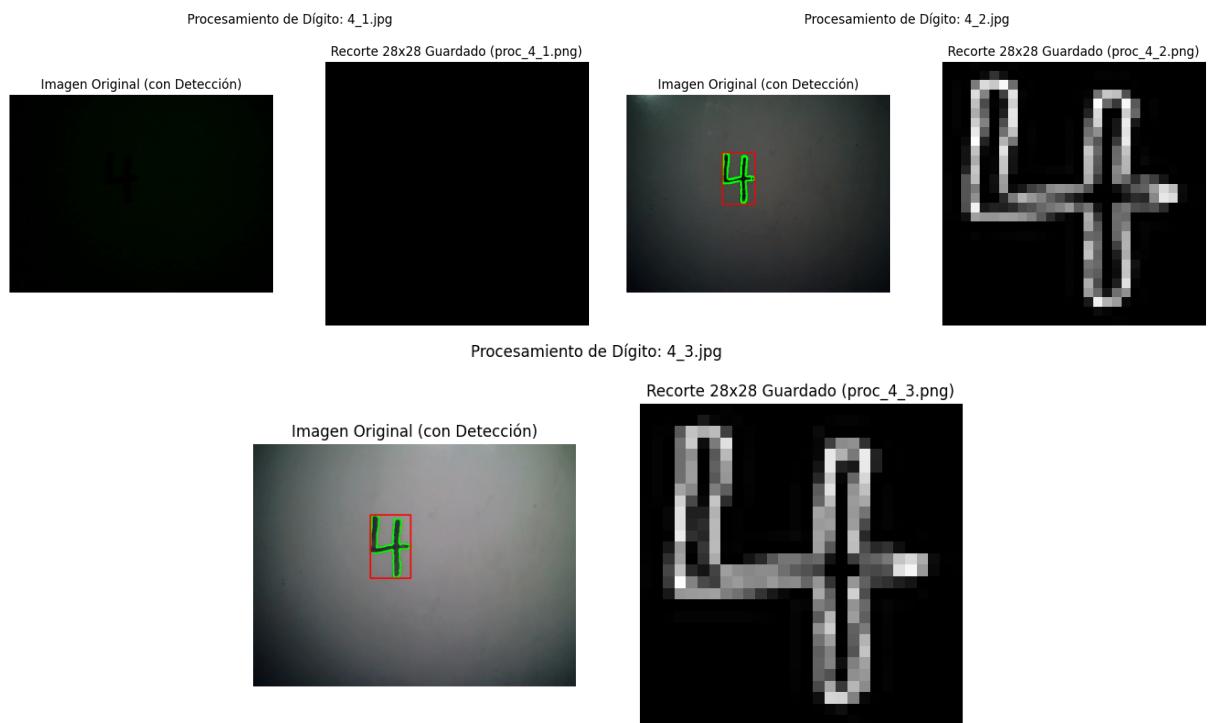


Figura 7: Conjunto de imágenes del proceso para el numero 4

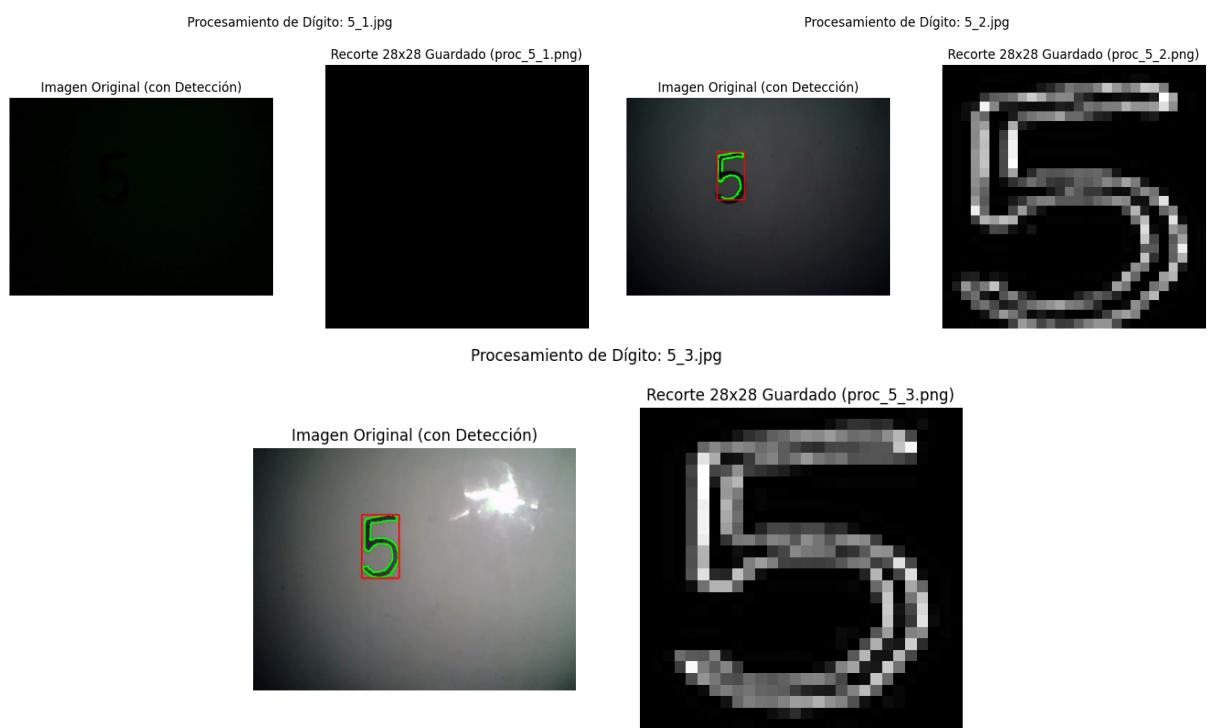


Figura 8: Conjunto de imágenes del proceso para el numero 5

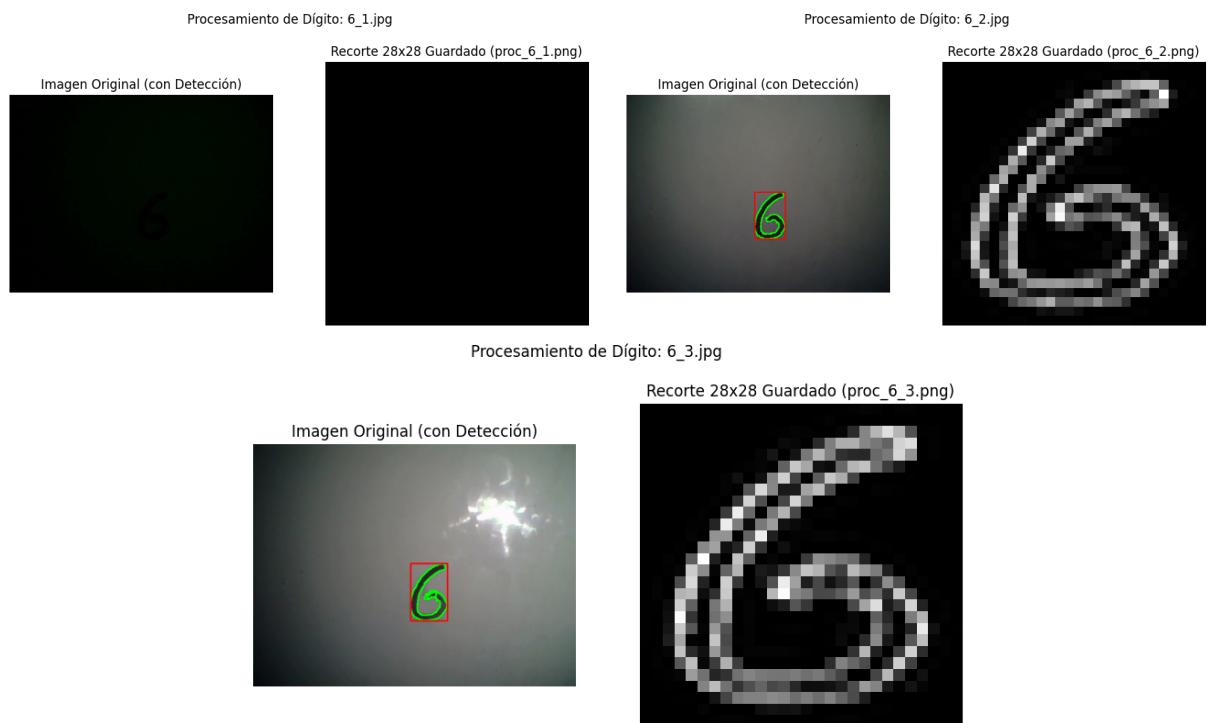


Figura 9: Conjunto de imágenes del proceso para el numero 6

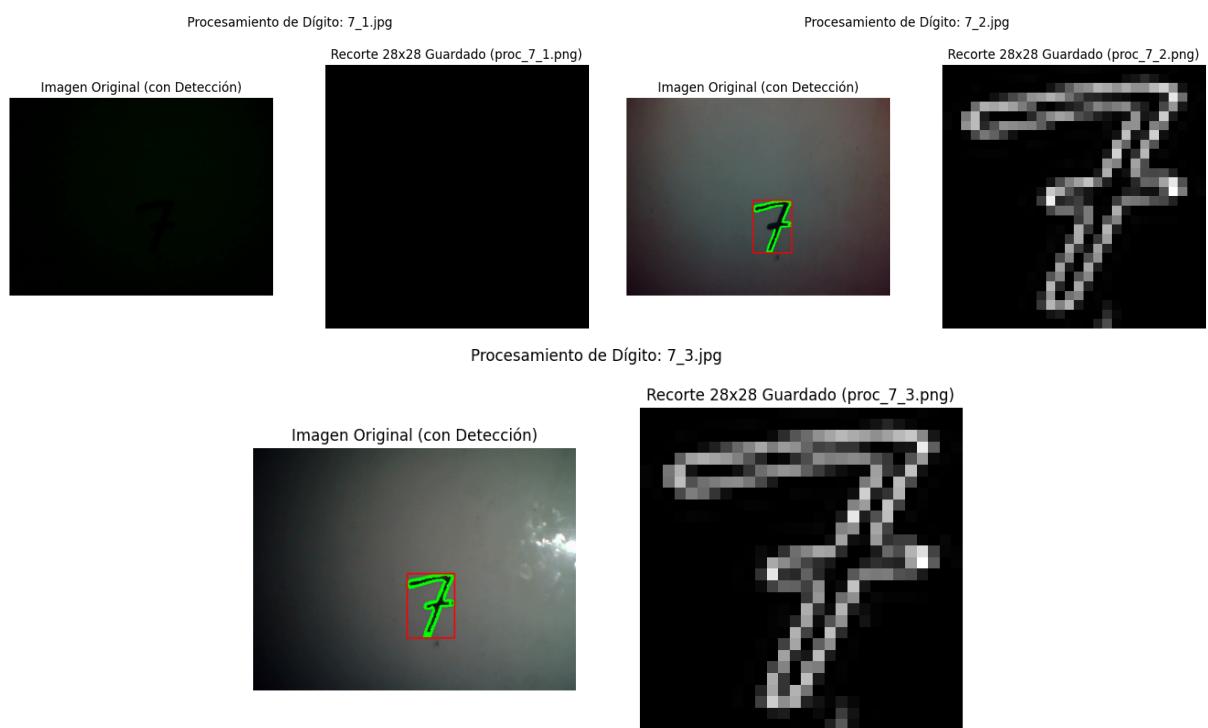


Figura 10: Conjunto de imágenes del proceso para el numero 7

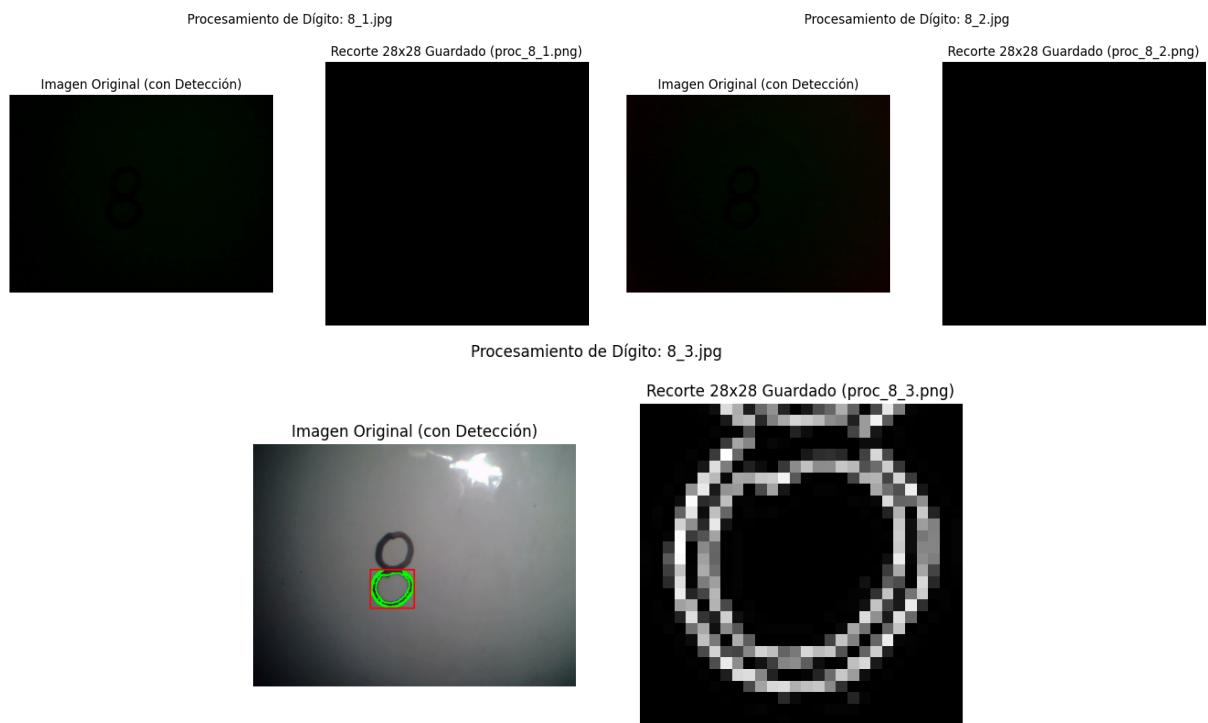


Figura 11: Conjunto de imágenes del proceso para el numero 8

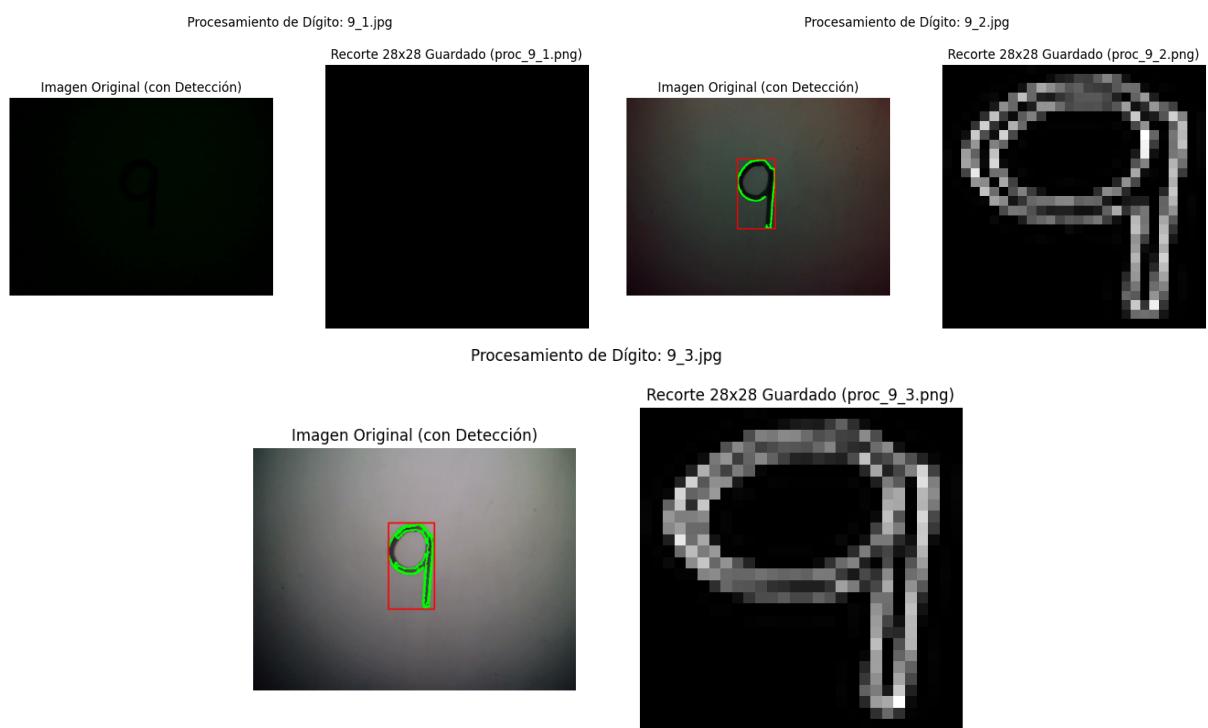


Figura 12: Conjunto de imágenes del proceso para el numero 9