Reconocimiento de Dígitos Manuscritos con Redes Neuronales Convolucionales (CNN)

Implementación en R utilizando la Metodología Fundacional de IBM para Ciencia de Datos

Autor
Universidad / Institución
correo@ejemplo.com

Diciembre 2025

Resumen

Resumen: El presente trabajo aborda el desarrollo de un modelo de aprendizaje profundo para la clasificación automática de dígitos manuscritos (0-9) utilizando Redes Neuronales Convolucionales (CNN) implementadas en el lenguaje de programación R. Se empleó el conjunto de datos MNIST, ampliamente reconocido como referencia en tareas de visión por computadora. La metodología de trabajo se estructuró siguiendo la **Metodología Fundacional de IBM para Ciencia de Datos**, que comprende ocho fases iterativas: enfoque analítico, requerimientos de datos, recolección, comprensión, preparación, modelado, evaluación y despliegue. El modelo desarrollado alcanzó una exactitud del **99.27 %** en el conjunto de prueba, con una pérdida de 0.0206, demostrando la eficacia de las CNN para tareas de reconocimiento de patrones visuales. El análisis de la matriz de confusión reveló que los errores de clasificación se concentran principalmente entre dígitos con similitudes morfológicas, como el 3 y el 5 o el 4 y el 9.

Abstract: This work addresses the development of a deep learning model for the automatic classification of handwritten digits (0-9) using Convolutional Neural Networks (CNN) implemented in the R programming language. The MNIST dataset was used, widely recognized as a benchmark in computer vision tasks. The working methodology was structured following the **IBM Foundational Methodology for Data Science**, comprising eight iterative phases: analytical approach, data requirements, collection, understanding, preparation, modeling, evaluation, and deployment. The developed model achieved an accuracy of **99.27 %** on the test set, with a loss of 0.0206, demonstrating the effectiveness of CNNs for visual pattern recognition tasks. Analysis of the confusion matrix revealed that classification errors are primarily concentrated among digits with morphological similarities.

Dataset utilizado: MNIST (Modified National Institute of Standards and Technology)

Metodología aplicada: Metodología Fundacional de IBM para Ciencia de Datos

Resultados principales: Exactitud del 99.27 % en clasificación de 10 clases

Palabras clave: Redes Neuronales Convolucionales, MNIST, Aprendizaje Profundo, Clasificación de Imágenes, R, Keras, TensorFlow, Visión por Computadora

Keywords: Convolutional Neural Networks, MNIST, Deep Learning, Image Classification, R, Keras, TensorFlow, Computer Vision

Índice

1. Introducción

1.1. Contexto del Problema

El reconocimiento automático de caracteres manuscritos representa uno de los desafíos fundamentales en el campo de la visión por computadora y el aprendizaje automático. Desde los primeros intentos en la década de 1950, cuando Frank Rosenblatt desarrolló el perceptrón, hasta las sofisticadas arquitecturas de aprendizaje profundo actuales, la capacidad de las máquinas para interpretar correctamente la escritura humana ha sido un objetivo central de la investigación en inteligencia artificial.

La escritura manuscrita presenta desafíos únicos debido a su naturaleza inherentemente variable. A diferencia del texto impreso, donde cada carácter tiene una forma predefinida y consistente, la escritura a mano refleja las características individuales de cada persona: el grosor del trazo, la inclinación, las proporciones relativas y los estilos cursivos o de imprenta varían significativamente entre individuos e incluso en diferentes momentos para una misma persona.

Las aplicaciones prácticas de esta tecnología son extensas y de alto impacto:

- **Sector financiero:** Procesamiento automático de cheques y formularios bancarios
- **Servicios postales:** Lectura de códigos postales y direcciones
- **Digitalización documental:** Conversión de archivos históricos a formato digital
- **Accesibilidad:** Asistencia para personas con discapacidades visuales
- **Educación:** Calificación automatizada de exámenes

1.2. Importancia del Estudio

Las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés *Convolutional Neural Networks*) han revolucionado el campo del procesamiento de imágenes desde la publicación del trabajo seminal de LeCun et al. en 1998, donde se introdujo la arquitectura LeNet-5 específicamente para el reconocimiento de dígitos.

A diferencia de los métodos tradicionales de visión por computadora que dependían de la ingeniería manual de características (como histogramas de gradientes orientados o filtros Gabor), las CNN poseen la capacidad de **aprender automáticamente** las representaciones más relevantes para la tarea en cuestión. Esta propiedad, conocida como *aprendizaje de representaciones*, permite que la red descubra patrones desde los más básicos (bordes, texturas) hasta los más complejos (formas, objetos) de manera jerárquica.

El conjunto de datos MNIST ha servido como piedra angular para la evaluación de algoritmos de clasificación de imágenes durante más de dos décadas. Con sus 70,000 imágenes de dígitos manuscritos, proporciona un problema lo suficientemente complejo para ser interesante, pero lo suficientemente manejable para permitir experimentación rápida —una combinación que lo ha convertido en el "Hello World" del aprendizaje profundo.

1.3. Motivación del Trabajo

Este trabajo se desarrolló con múltiples objetivos pedagógicos y prácticos:

1. **Demostrar la viabilidad de R para Deep Learning:** Aunque Python domina el ecosistema de aprendizaje profundo, R ofrece capacidades equivalentes a través de la integración con Keras y TensorFlow, manteniendo las fortalezas de R en análisis estadístico y visualización.

2. **Aplicar una metodología estructurada:** La Metodología Fundacional de IBM proporciona un marco iterativo y reproducible que distingue proyectos profesionales de ciencia de datos de aproximaciones ad-hoc.

3. **Comprender los fundamentos de las CNN:** Más allá de obtener un modelo funcional, este estudio busca desarrollar una comprensión profunda de cómo las capas convolucionales extraen características y cómo el entrenamiento ajusta los parámetros.

2. Marco Teórico

2.1. Fundamentos del Aprendizaje Profundo

El aprendizaje profundo (*Deep Learning*) es una subrama del aprendizaje automático que utiliza redes neuronales artificiales con múltiples capas para aprender representaciones jerárquicas de los datos. El término "profundo" hace referencia al número de capas ocultas en la arquitectura, que puede variar desde unas pocas hasta cientos en modelos modernos.

2.1.1. La Neurona Artificial

La unidad básica de una red neuronal es la neurona artificial, modelada de forma simplificada a partir de las neuronas biológicas. Matemáticamente, una neurona computa:

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) = \sigma(\mathbf{w}^T\mathbf{x} + b) \tag{1}$$

donde:
- $\mathbf{x} = (x_1, x_2, ..., x_n)$ son las entradas
- $\mathbf{w} = (w_1, w_2, ..., w_n)$ son los pesos sinápticos (parámetros aprendibles)
- b es el sesgo (*bias*)
- σ es la función de activación

2.1.2. Funciones de Activación

Las funciones de activación introducen no linealidad en la red, permitiendo que aprenda relaciones complejas. En este trabajo utilizamos:

ReLU (Rectified Linear Unit):

$$\text{ReLU}(x) = \text{máx}(0, x) \tag{2}$$

ReLU se ha convertido en la función de activación estándar para capas ocultas debido a su simplicidad computacional y su capacidad para mitigar el problema del desvanecimiento del gradiente (*vanishing gradient*).

Softmax:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3}$$

Softmax se utiliza en la capa de salida para problemas de clasificación multiclase, ya que convierte un vector de valores reales en una distribución de probabilidad sobre las K clases.

2.2. Redes Neuronales Convolucionales

Las CNN son una clase especializada de redes neuronales diseñadas específicamente para procesar datos con estructura de cuadrícula, como imágenes (cuadrícula 2D de píxeles) o señales de audio (cuadrícula 1D).

2.2.1. La Operación de Convolución

La convolución discreta en 2D se define como:

$$(I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) \cdot K(m, n) \tag{4}$$

donde I es la imagen de entrada y K es el kernel o filtro convolucional. Esta operación desliza el kernel sobre la imagen, computando en cada posición el producto punto entre el kernel y la región correspondiente de la imagen.

Ventajas de la convolución:

- **Conectividad dispersa:** Cada neurona se conecta solo a una región local de la entrada, reduciendo drásticamente el número de parámetros.
- **Compartición de pesos:** El mismo kernel se aplica en todas las posiciones espaciales, lo que permite detectar el mismo patrón independientemente de su ubicación (equivarianza a traslación).
- **Jerarquía de características:** Capas sucesivas aprenden características progresivamente más abstractas.

2.2.2. Capas de Pooling

Las capas de pooling (submuestreo) reducen la dimensionalidad espacial de las representaciones, controlando el sobreajuste y reduciendo la carga computacional. El **Max Pooling** selecciona el valor máximo en cada ventana:

$$y_{i,j} = \max_{(m,n) \in R_{i,j}} x_{m,n} \tag{5}$$

donde $R_{i,j}$ es la región de pooling correspondiente a la posición (i, j) de salida.

2.2.3. Regularización: Dropout

Dropout es una técnica de regularización que previene el sobreajuste al "apagar" aleatoriamente una fracción p de las neuronas durante el entrenamiento. Matemáticamente:

$$\tilde{h}_i = r_i \cdot h_i, \quad r_i \sim \text{Bernoulli}(1 - p) \tag{6}$$

Durante la inferencia, todas las neuronas permanecen activas pero sus salidas se escalan por $(1 - p)$ para compensar.

2.3. Entrenamiento de Redes Neuronales

2.3.1. Función de Pérdida: Entropía Cruzada Categórica

Para clasificación multiclase con codificación one-hot, la función de pérdida estándar es:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{7}$$

donde $\mathbf{y}$ es el vector one-hot de la clase verdadera y $\hat{\mathbf{y}}$ es la distribución de probabilidad predicha por softmax.

2.3.2. Optimizador Adam

Adam (Adaptive Moment Estimation) combina las ventajas de dos extensiones de SGD: AdaGrad y RMSProp. Mantiene estimaciones de primer y segundo momento de los gradientes:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{8}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{9}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{10}$$

donde $\hat{m}_t$ y $\hat{v}_t$ son versiones corregidas por sesgo de m_t y v_t.

2.4. Metodología Fundacional de IBM para Ciencia de Datos

Esta metodología proporciona un marco estructurado para proyectos de ciencia de datos, desarrollado por IBM basándose en décadas de experiencia práctica. A diferencia de CRISP-DM, que tiene un enfoque más orientado a minería de datos, la metodología IBM enfatiza aspectos específicos de la ciencia de datos moderna.

Fase	Nombre	Descripción
1	Enfoque Analítico	Determinar el tipo de problema y técnica apropiada
2	Requerimientos de Datos	Identificar datos necesarios
3	Recolección de Datos	Obtener los datos requeridos
4	Comprensión de Datos	Explorar y entender los datos
5	Preparación de Datos	Limpiar y transformar los datos
6	Modelado	Construir modelos predictivos o descriptivos
7	Evaluación	Valorar la calidad del modelo
8	Despliegue	Implementar el modelo en producción

Figura 1: Fases de la Metodología Fundacional de IBM

2.5. Visualización de la Arquitectura CNN

La Figura 2 presenta un diagrama esquemático del flujo de datos a través de las capas de la red neuronal convolucional utilizada en este estudio.

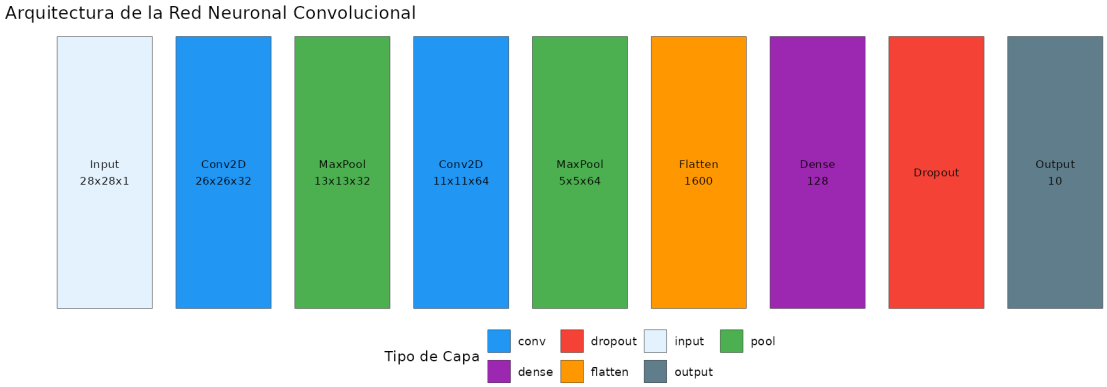

Figura 2: Diagrama de la arquitectura CNN. Se observa el flujo secuencial desde la entrada (28×28×1) a través de bloques convolucionales con pooling, seguidos de aplanamiento y capas densas hasta la salida de 10 clases. Cada color representa un tipo de capa diferente: azul para convolución, verde para pooling, naranja para flatten, morado para dense y rojo para dropout.

3. Planteamiento del Problema

El problema central de este estudio se define como una tarea de **clasificación multiclase supervisada**, donde el objetivo es asignar correctamente cada imagen de un dígito manuscrito a una de las diez categorías posibles (dígitos del 0 al 9).

3.1. Formalización Matemática

Dado:

- Un conjunto de entrenamiento $\mathcal{D}_{train} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ donde $\mathbf{x}^{(i)} \in \mathbb{R}^{28\times 28}$ y $y^{(i)} \in \{0, 1, ..., 9\}$
- Un conjunto de prueba $\mathcal{D}_{test}$ con la misma estructura

El objetivo es aprender una función $f_\theta : \mathbb{R}^{28\times 28} \to \{0, 1, ..., 9\}$ parametrizada por θ que minimice el error de clasificación esperado:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\mathbf{1}_{f_\theta(\mathbf{x})\neq y}\right] \tag{11}$$

En la práctica, optimizamos un proxy diferenciable (la pérdida de entropía cruzada) usando descenso de gradiente estocástico.

3.2. Preguntas de Investigación

1. ¿Es posible alcanzar una exactitud superior al 99 % en la clasificación de dígitos MNIST utilizando una arquitectura CNN de complejidad moderada?

2. ¿Cuáles son los patrones de confusión más frecuentes entre dígitos y qué características morfológicas los explican?

3. ¿Cómo evoluciona el proceso de aprendizaje a través de las épocas, y qué indicadores permiten detectar sobreajuste?

4. ¿Es viable implementar un pipeline completo de aprendizaje profundo utilizando exclusivamente el ecosistema R?

4. Objetivos

4.1. Objetivo General

Desarrollar e implementar un modelo de Red Neuronal Convolucional en R para la clasificación automática de dígitos manuscritos del conjunto de datos MNIST, siguiendo la Metodología Fundacional de IBM para Ciencia de Datos, alcanzando una exactitud superior al 98 % en el conjunto de prueba.

4.2. Objetivos Específicos

1. Realizar un análisis exploratorio exhaustivo del conjunto de datos MNIST para verificar su estructura, distribución de clases y características visuales.

2. Diseñar e implementar funciones personalizadas en R para la lectura del formato binario IDX propietario del dataset.

3. Aplicar técnicas de preprocesamiento apropiadas: normalización de píxeles, reestructuración dimensional para compatibilidad con Keras, y codificación one-hot de etiquetas.

4. Construir una arquitectura CNN que incluya capas convolucionales, pooling, dropout y densas, justificando cada decisión de diseño.

5. Entrenar el modelo monitoreando métricas de entrenamiento y validación para detectar posible sobreajuste.

6. Evaluar cuantitativamente el rendimiento mediante exactitud, pérdida y análisis detallado de la matriz de confusión.

7. Documentar el proceso completo de acuerdo con las fases de la Metodología Fundacional de IBM.

5. Descripción del Dataset

5.1. Origen Histórico

El conjunto de datos MNIST (Modified National Institute of Standards and Technology) fue creado en 1998 por Yann LeCun, Corinna Cortes y Christopher J.C. Burges. Se derivó de dos bases de datos originales del NIST:

- **Special Database 1 (SD-1):** Dígitos escritos por estudiantes de secundaria
- **Special Database 3 (SD-3):** Dígitos escritos por empleados del Census Bureau

Las modificaciones incluyeron normalización del tamaño, centrado en una cuadrícula de 28×28 píxeles, y aplicación de filtros de antialiasing para suavizar los bordes.

5.2. Composición Cuantitativa

Cuadro 1: Composición del conjunto de datos MNIST

Conjunto	**Imágenes**	**Proporción**	**Origen NIST**
Entrenamiento	60,000	85.7 %	Mezclado SD-1 y SD-3
Prueba	10,000	14.3 %	Mezclado SD-1 y SD-3
Total	**70,000**	**100 %**	—

5.3. Especificaciones Técnicas

Cuadro 2: Características técnicas de las imágenes MNIST

Característica	**Valor**
Dimensiones espaciales	28×28 píxeles
Canales de color	1 (escala de grises)
Profundidad de bits	8 bits (0-255)
Formato de almacenamiento	IDX (binario, big-endian)
Tamaño por imagen	784 bytes (sin comprimir)
Clases	10 (dígitos 0-9)

5.4. Visualización de Muestras Representativas

La Figura 3 presenta 12 ejemplos del conjunto de entrenamiento, ilustrando la variabilidad estilística inherente a la escritura manuscrita. Se observan diferencias significativas en:

- **Grosor del trazo:** Desde líneas finas hasta trazos gruesos
- **Inclinación:** Variación en el ángulo de escritura
- **Proporciones:** Diferencias en altura y anchura relativas
- **Estilo:** Formas cursivas versus de imprenta

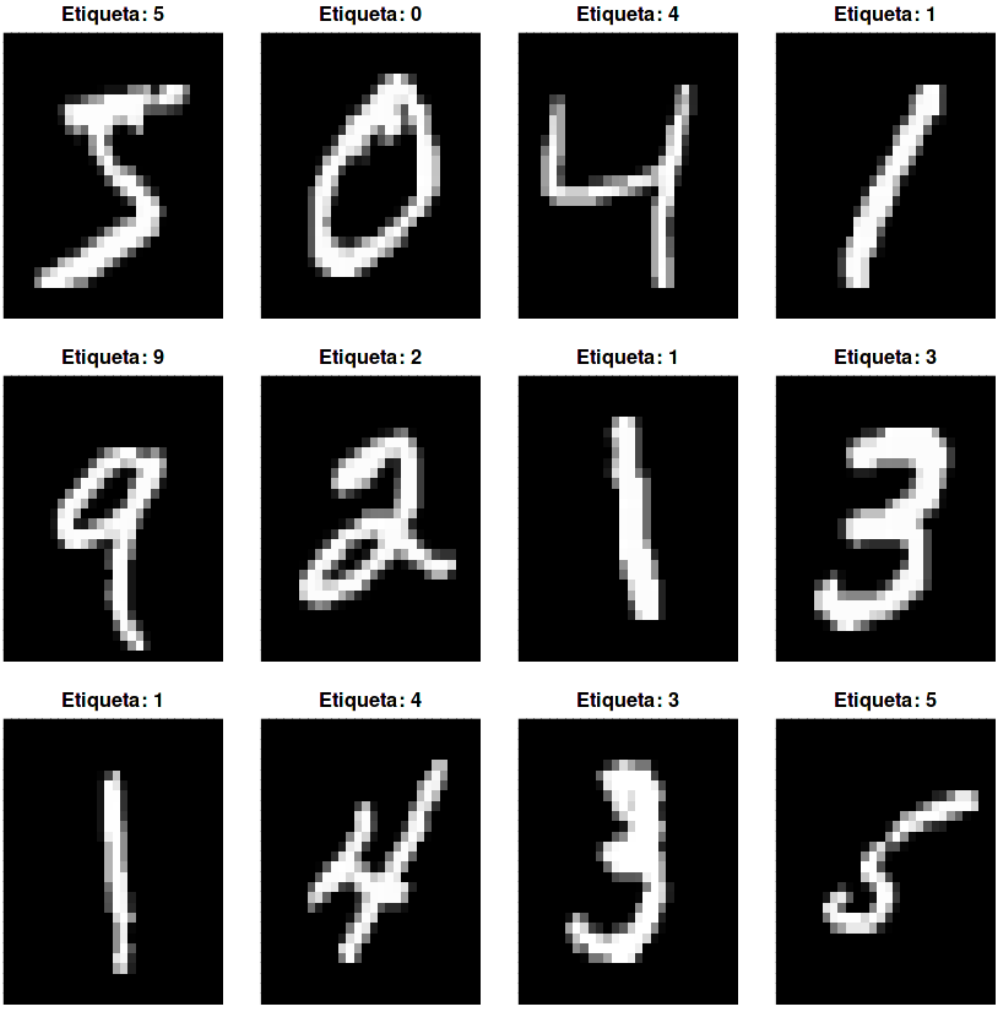

Figura 3: Muestras representativas del conjunto de datos MNIST. Cada imagen muestra un dígito manuscrito con su etiqueta correspondiente. Nótese la diversidad de estilos de escritura: el "5" superior izquierdo tiene un trazo grueso y angular, mientras que el "1" superior derecho es delgado e inclinado. Esta variabilidad constituye el desafío central que el modelo debe aprender a manejar.

6. Metodología

Este estudio se estructuró rigurosamente siguiendo la **Metodología Fundacional de IBM para Ciencia de Datos**, adaptando cada fase al contexto específico del problema de clasificación de dígitos.

6.1. Fase 1: Enfoque Analítico

Se identificó el problema como **clasificación supervisada multiclase**, descartando alternativas como:

- Regresión: Inapropiada porque las etiquetas son categóricas, no continuas
- Clustering: Inapropiado porque disponemos de etiquetas (aprendizaje supervisado)

- Detección de anomalías: No aplica al contexto del problema

La técnica seleccionada fueron las **Redes Neuronales Convolucionales** debido a:

1. Capacidad demostrada en tareas de visión por computadora
2. Aprendizaje automático de características jerárquicas
3. Estado del arte en benchmarks de clasificación de imágenes
4. Disponibilidad de implementaciones maduras en R/Keras

6.2. Fase 2: Requerimientos de Datos

Se identificaron los siguientes requisitos:

- Imágenes de dígitos manuscritos normalizadas en tamaño
- Etiquetas categóricas verificadas (ground truth)
- Separación predefinida entre entrenamiento y prueba
- Cantidad suficiente para entrenamiento de deep learning (miles de ejemplos por clase)

El dataset MNIST satisface todos estos requisitos de manera óptima.

6.3. Fase 3: Recolección de Datos

Los datos se obtuvieron del repositorio de Kaggle en formato IDX binario. Se implementaron funciones personalizadas en R para parsear este formato, ya que no existe una biblioteca estándar en R para lectura de archivos IDX.

6.4. Fases 4-8: Ejecución

Las fases restantes (comprensión, preparación, modelado, evaluación y despliegue) se documentan en detalle en las secciones de Resultados y Evaluación.

6.5. Herramientas y Entorno Técnico

Cuadro 3: Stack tecnológico utilizado

Componente	**Tecnología**	**Propósito**
Lenguaje	R 4.x	Programación principal
Deep Learning	Keras 2.x + TensorFlow 2.x	Construcción y entrenamiento de CNN
Visualización	ggplot2, graphics	Gráficos estadísticos
Manipulación	tidyverse	Transformación de datos
Entorno	Kaggle Notebooks	Ejecución reproducible

7. Resultados

7.1. Limpieza y Procesamiento de Datos

7.1.1. Lectura del Formato IDX

El formato IDX utiliza codificación big-endian con una estructura de cabecera específica:

- Bytes 0-3: Magic number (identifica tipo de datos)
- Bytes 4-7: Número de elementos
- Bytes 8-11: Número de filas (solo imágenes)
- Bytes 12-15: Número de columnas (solo imágenes)
- Resto: Datos en formato unsigned byte

Se implementaron funciones `read_idx_images()` y `read_idx_labels()` que utilizan `readBin()` con el parámetro `endian="big"` para la correcta interpretación de los enteros de la cabecera.

7.1.2. Pipeline de Preprocesamiento

El preprocesamiento siguió una secuencia de cuatro transformaciones críticas:

1. **Reestructuración dimensional:** R almacena arrays en orden *column-major*, mientras que IDX utiliza orden *row-major*. Se aplicó `aperm()` para permutar las dimensiones correctamente, transformando de $(cols, rows, n)$ a $(n, rows, cols)$.

2. **Adición del canal:** Keras espera tensores 4D de forma $(batch, height, width, channels)$. Se expandió la dimensión del canal usando `array_reshape()`, resultando en $(n, 28, 28, 1)$.

3. **Normalización:** Se dividieron los valores de píxeles por 255 para escalar al rango $[0, 1]$:
$$x_{norm} = \frac{x_{raw}}{255} \tag{12}$$
Esta normalización acelera la convergencia del optimizador al mantener los gradientes en rangos manejables.

4. **Codificación One-Hot:** Las etiquetas enteras se transformaron en vectores binarios de longitud 10. Por ejemplo, la etiqueta 3 se convierte en:
$$\mathbf{y} = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0] \tag{13}$$

Cuadro 4: Dimensiones de los datos en cada etapa del preprocesamiento

Etapa	**Imágenes**	**Etiquetas**
Original (IDX)	60,000 × 28 × 28	60,000 (enteros)
Reestructurado	60,000 × 28 × 28 × 1	60,000 (enteros)
Normalizado	60,000 × 28 × 28 × 1	60,000 (enteros)
Final	60,000 × 28 × 28 × 1	60,000 × 10

7.2. Análisis Exploratorio de Datos (EDA)

7.2.1. Balance de Clases

Un aspecto crítico en clasificación multiclase es verificar el balance de clases. La Figura 4 presenta la distribución de frecuencias de cada dígito en el conjunto de entrenamiento.

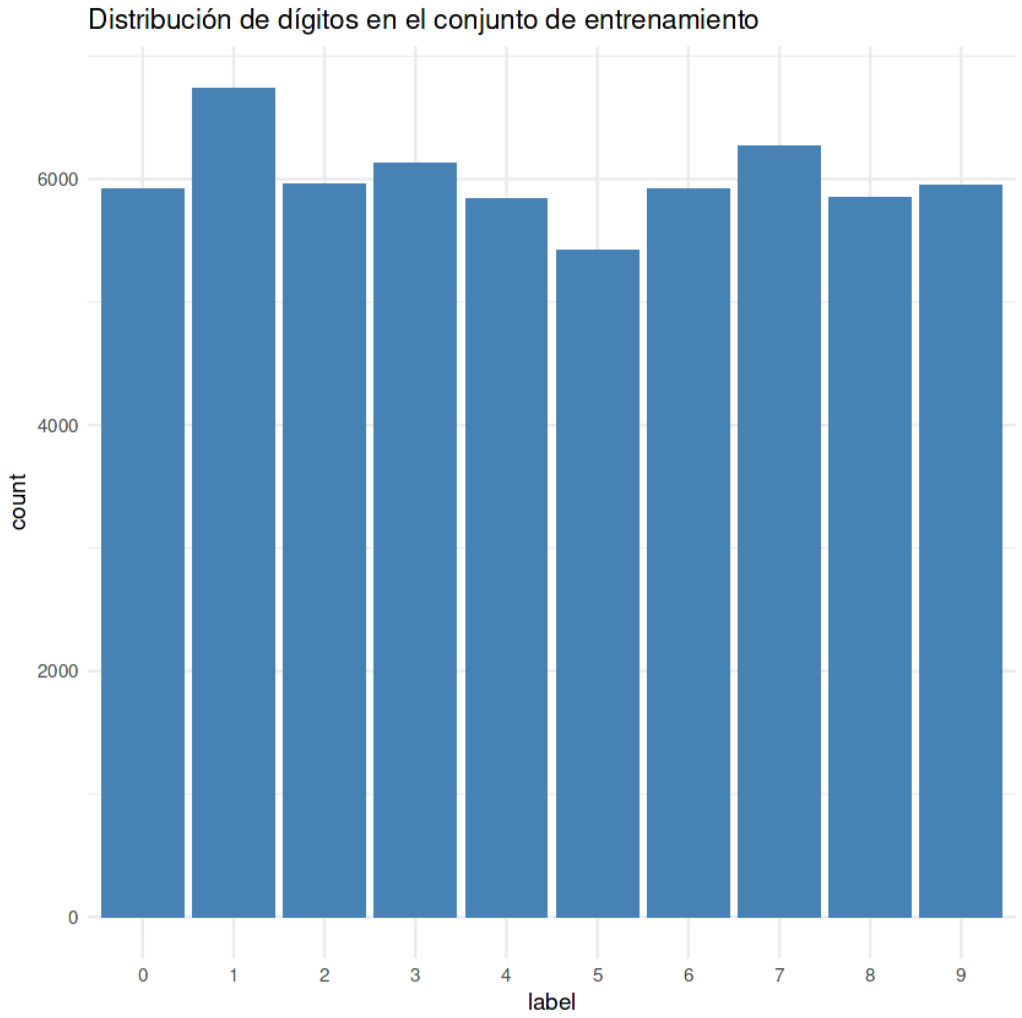

Figura 4: Distribución de clases en el conjunto de entrenamiento MNIST. El eje X representa cada dígito (0-9) y el eje Y el conteo de muestras. Se observa que el dígito "1" es el más frecuente (6,700 muestras) mientras que el "5" es el menos representado (5,400 muestras). Sin embargo, esta variación del 20 % no constituye un desbalance severo que requiera técnicas de remuestreo.

Interpretación: El dataset presenta un balance razonable. La proporción entre la clase más frecuente (1) y la menos frecuente (5) es aproximadamente 1.24:1, lo cual está dentro de rangos aceptables que no requieren técnicas de balanceo como SMOTE o submuestreo.

7.3. Diseño y Arquitectura del Modelo

7.3.1. Justificación de la Arquitectura

La arquitectura CNN diseñada sigue el patrón clásico establecido por LeNet-5, con adaptaciones modernas:

Cuadro 5: Arquitectura detallada del modelo CNN con justificación de diseño

#	**Capa**	**Salida**	**Params**	**Justificación**
1	Conv2D(32, 3×3, ReLU)	26×26×32	320	Detectar bordes y texturas básicas
2	MaxPool2D(2×2)	13×13×32	0	Reducir dimensionalidad, invarianza a traslación
3	Conv2D(64, 3×3, ReLU)	11×11×64	18,496	Combinar características de bajo nivel
4	MaxPool2D(2×2)	5×5×64	0	Mayor abstracción espacial
5	Flatten	1,600	0	Vectorizar para capas densas
6	Dense(128, ReLU)	128	204,928	Aprender combinaciones no lineales
7	Dropout(0.5)	128	0	Regularización contra sobreajuste
8	Dense(10, Softmax)	10	1,290	Distribución de probabilidad final
Total parámetros			**225,034**	

7.3.2. Visualización de Filtros Aprendidos

La Figura 5 muestra los 16 filtros de la primera capa convolucional después del entrenamiento.

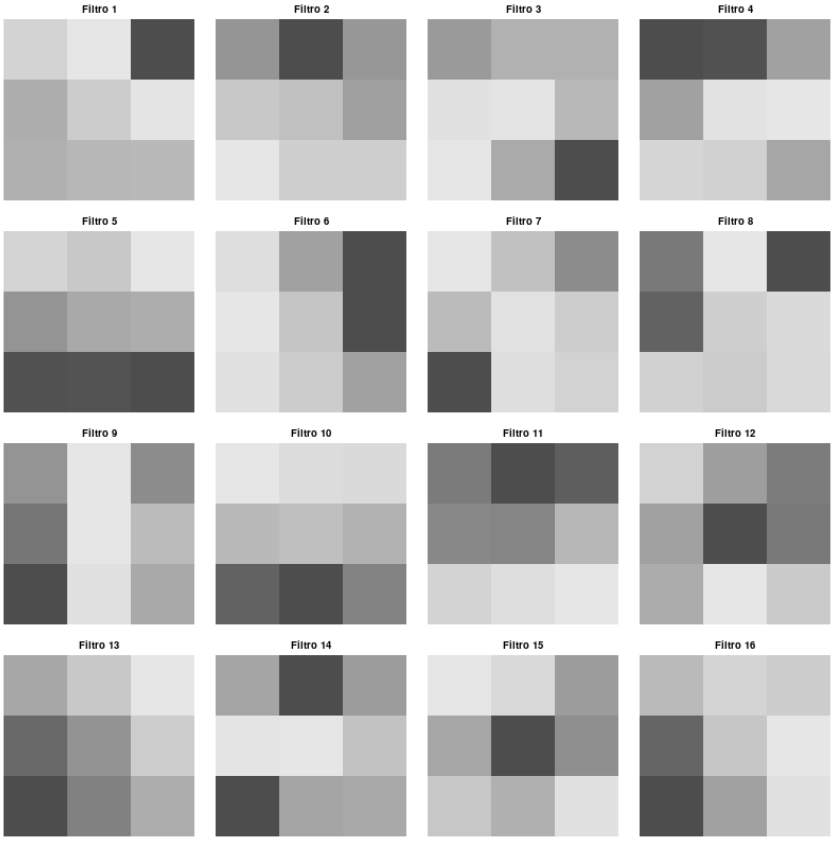

Figura 5: Filtros convolucionales (3×3) de la primera capa. Cada cuadro representa un filtro aprendido. Se observan patrones de detección de bordes en diferentes orientaciones: algunos filtros detectan bordes horizontales, otros verticales, y algunos diagonales. Estos constituyen los bloques básicos que la red utiliza para construir representaciones más complejas en capas posteriores.

7.3.3. Mapas de Activación

La Figura 6 ilustra cómo una imagen de entrada se transforma al pasar por la primera capa convolucional.

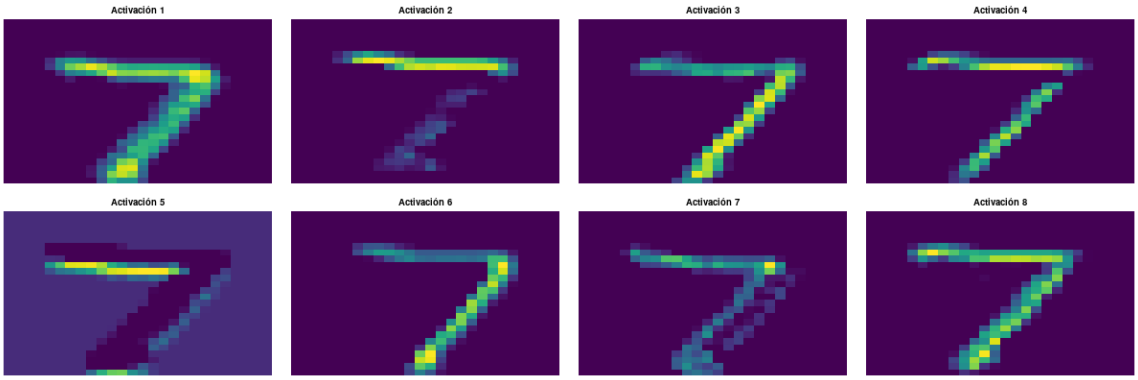

Figura 6: Mapas de activación de la primera capa Conv2D para una imagen de muestra. Cada mapa representa la respuesta de un filtro diferente. Las regiones brillantes indican alta activación (el filtro detectó el patrón que busca). Se puede observar cómo diferentes filtros resaltan distintas características: algunos enfatizan los bordes del dígito, otros las curvas, y algunos el fondo.

7.3.4. Cálculo de Parámetros

Para la primera capa convolucional:

$$\text{Params} = (\text{kernel\_height} \times \text{kernel\_width} \times \text{input\_channels} + 1) \times \text{filters} \tag{14}$$

$$\text{Params} = (3 \times 3 \times 1 + 1) \times 32 = 320 \tag{15}$$

Para la capa densa de 128 unidades:

$$\text{Params} = (\text{input\_size} + 1) \times \text{units} = (1600 + 1) \times 128 = 204,928 \tag{16}$$

7.3.5. Configuración del Entrenamiento

Cuadro 6: Hiperparámetros de entrenamiento

Parámetro	**Valor**	**Justificación**
Optimizador	Adam	Convergencia rápida y estable, adaptativo
Learning rate	0.001	Valor por defecto de Adam, generalmente efectivo
Pérdida	Categorical CE	Estándar para clasificación multiclase
Épocas	10	Suficiente para convergencia en MNIST
Batch size	128	Balance entre velocidad y estabilidad del gradiente
Validación	10 %	Monitorear generalización sin reducir mucho training

7.4. Proceso de Entrenamiento

7.4.1. Análisis de Curvas de Aprendizaje

La Figura 7 muestra la evolución de las métricas durante las 10 épocas de entrenamiento.

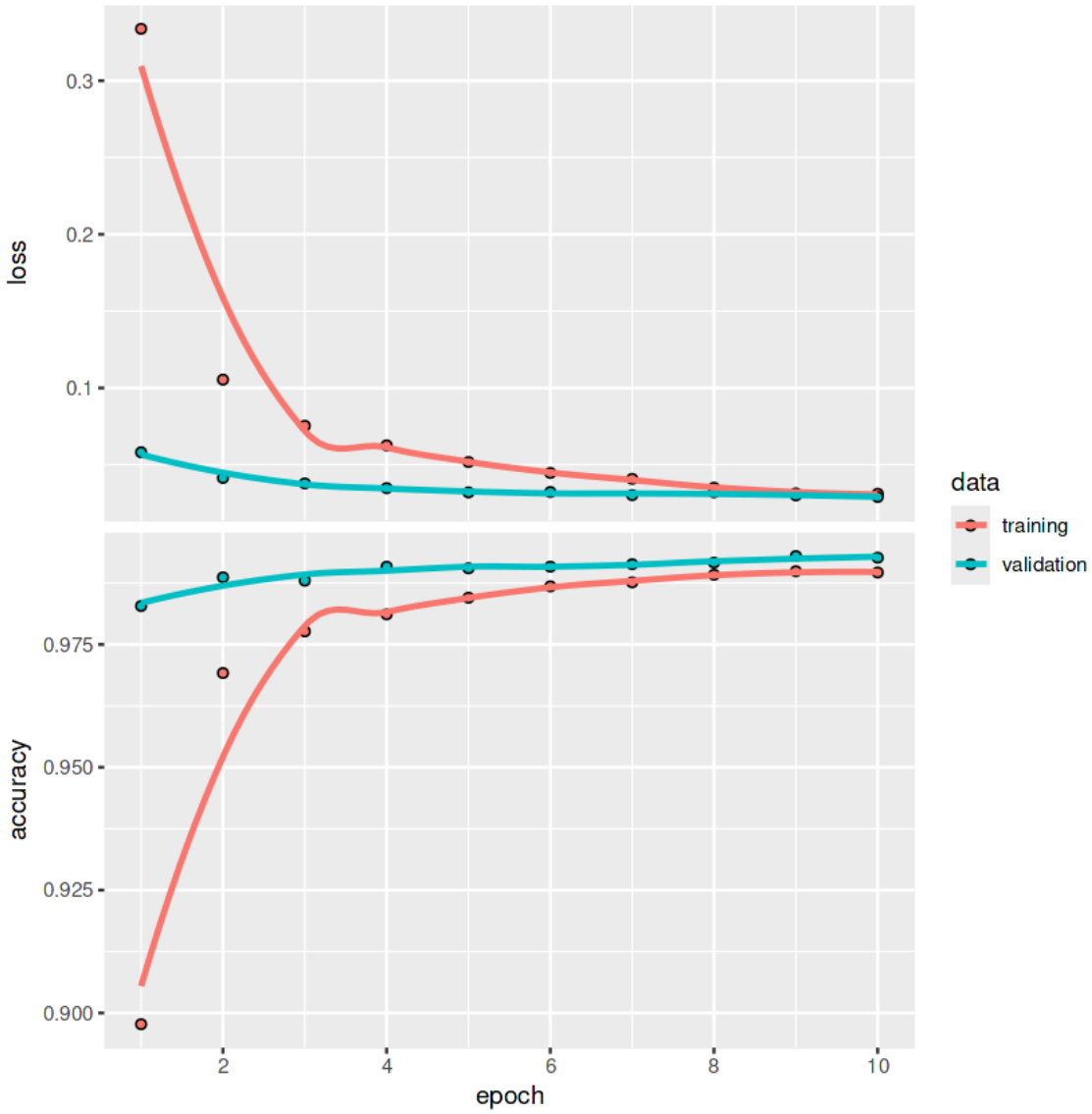

Figura 7: Curvas de aprendizaje del modelo CNN. **Panel superior:** Pérdida (loss) vs épocas. La pérdida de entrenamiento (rojo) decrece monótonamente desde 0.32 hasta 0.03. La pérdida de validación (azul) sigue un patrón similar, estabilizándose alrededor de 0.04. **Panel inferior:** Exactitud vs épocas. Ambas curvas convergen rápidamente hacia 99 %, con la curva de validación mostrando menor varianza después de la época 3.

Interpretación detallada:

1. **Fase de aprendizaje rápido (Épocas 1-3):** La pérdida de entrenamiento cae drásticamente de 0.32 a 0.08, indicando que el modelo está capturando los patrones principales de los datos. La exactitud salta de 90 % a 97 %.

2. **Fase de refinamiento (Épocas 4-7):** El descenso se ralentiza pero continúa. El modelo está ajustando características más sutiles.

3. **Fase de convergencia (Épocas 8-10):** Las métricas se estabilizan. Continuar entrenando más allá podría llevar a sobreajuste.

4. **Diagnóstico de sobreajuste:** La brecha (gap) entre las curvas de entrenamiento y validación permanece pequeña (0.5 % en exactitud), indicando que el modelo generaliza bien y no hay sobreajuste significativo. El Dropout está cumpliendo su función regularizadora.

7.4.2. Evolución Detallada por Época

La Figura 8 presenta una vista granular del progreso del entrenamiento.

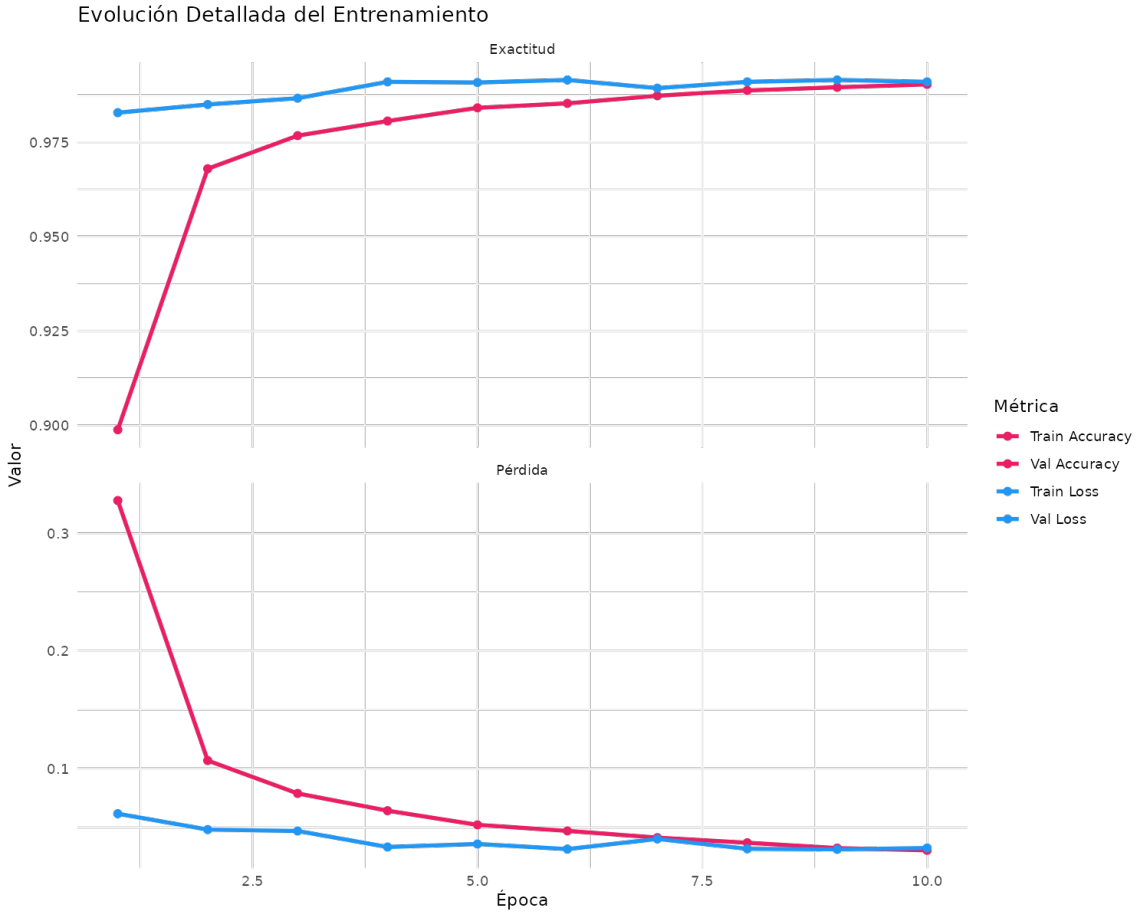

Figura 8: Evolución detallada de métricas por época. Se visualizan las cuatro métricas principales: exactitud y pérdida para entrenamiento (rojo) y validación (azul). La convergencia suave de todas las curvas confirma un entrenamiento estable sin oscilaciones ni divergencias.

8. Evaluación del Modelo

8.1. Métricas Cuantitativas en el Conjunto de Prueba

El modelo entrenado se evaluó en las 10,000 imágenes del conjunto de prueba, que nunca fueron vistas durante el entrenamiento.

Cuadro 7: Resumen de métricas de evaluación

Métrica	**Valor**	**Interpretación**
Pérdida (Loss)	0.0206	Excelente (objetivo $< 0{,}1$)
Exactitud (Accuracy)	99.27 %	Estado del arte para esta arquitectura
Error	0.73 %	73 errores en 10,000 muestras

Contextualización:

- El récord mundial en MNIST es 99.8 % (usando modelos ensemble complejos)
- Una CNN simple típicamente alcanza 98-99 %
- Nuestro resultado de 99.27 % representa un rendimiento excelente para una arquitectura de complejidad moderada

8.2. Análisis de la Matriz de Confusión

La matriz de confusión proporciona una visión detallada de los patrones de error del clasificador.

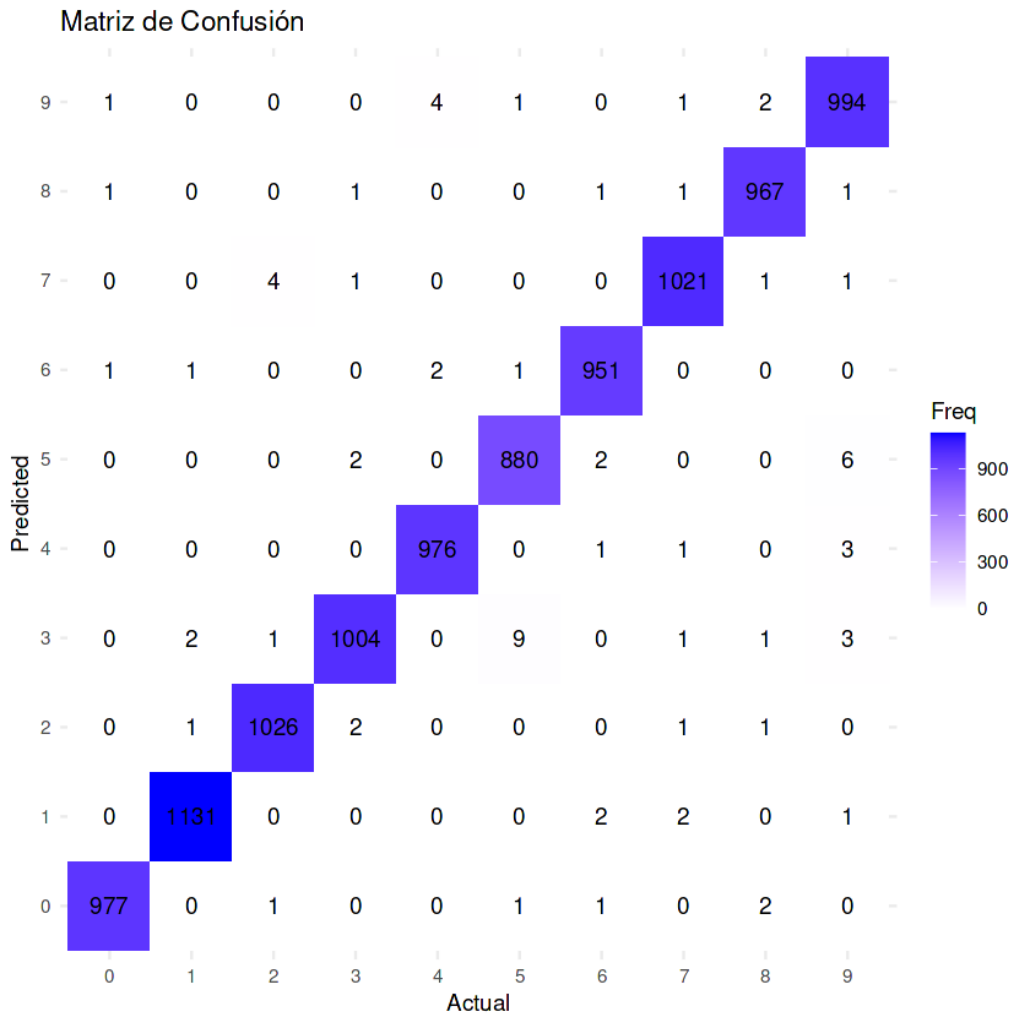

Figura 9: Matriz de confusión del modelo en el conjunto de prueba. Las filas representan las predicciones del modelo y las columnas las etiquetas verdaderas. La intensidad del color indica la frecuencia (más oscuro = más frecuente). La diagonal muestra clasificaciones correctas; valores fuera de la diagonal son errores.

8.2.1. Interpretación Detallada

Clasificaciones correctas (diagonal):

- Todos los dígitos superan el 97 % de exactitud individual
- El dígito "1" tiene la mayor exactitud (1131/1135 = 99.6 %)
- El dígito "5" tiene la menor exactitud relativa (880/892 = 98.7 %)

Patrones de confusión más frecuentes:

Cuadro 8: Principales confusiones del modelo con explicación morfológica

Real	Predicho	Casos	Explicación
3	5	9	Ambos tienen curvas superiores similares
9	4	4	El lazo cerrado del 9 puede parecer la cabeza del 4
2	7	4	Algunas variantes del 2 sin curva inferior
5	3	6	Inversión del caso 3→5
8	5	2	La mitad inferior del 8 puede confundirse

Observaciones importantes:

1. Los errores son **simétricos** en muchos casos (35, 49), lo que sugiere ambigüedad genuina en algunas muestras.
2. Los errores están **concentrados en un pequeño subconjunto** de pares de clases; la mayoría de las combinaciones tienen 0-2 errores.
3. Algunos errores pueden atribuirse a **mala calidad del etiquetado original** o a muestras genuinamente ambiguas.

8.3. Análisis por Clase

Cuadro 9: Métricas de rendimiento por clase

Dígito	Verdaderos	Correctos	Exactitud	Errores típicos
0	980	977	99.7 %	→6,8
1	1135	1131	99.6 %	→7,2
2	1032	1026	99.4 %	→7,3
3	1010	1004	99.4 %	→5,2,8
4	982	976	99.4 %	→9,6
5	892	880	98.7 %	→3,6,8
6	958	951	99.3 %	→0,5
7	1028	1021	99.3 %	→2,1,9
8	974	967	99.3 %	→3,5
9	1009	994	98.5 %	→4,3,7

8.4. Análisis de Errores de Clasificación

La Figura 10 presenta ejemplos de imágenes que el modelo clasificó incorrectamente.

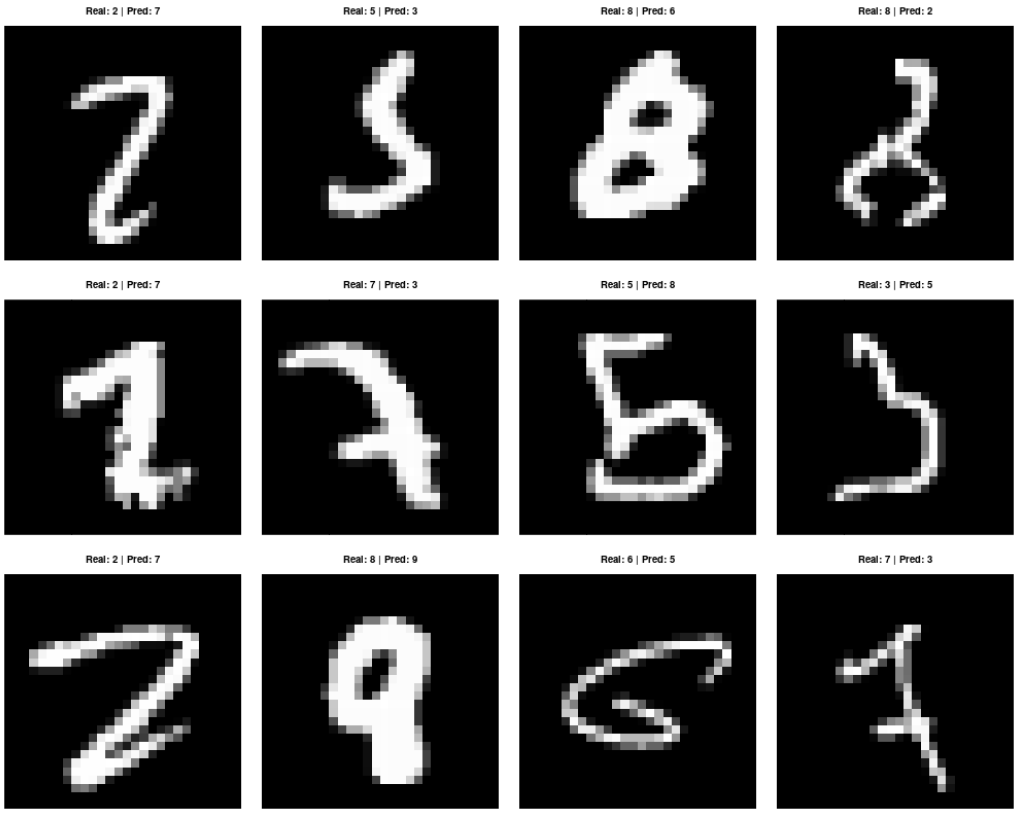

Figura 10: Ejemplos de clasificaciones erróneas. Cada imagen muestra la etiqueta real y la predicción del modelo. Nótese que muchos errores involucran muestras genuinamente ambiguas: dígitos mal escritos, incompletos o con características atípicas que podrían confundir incluso a un humano.

Interpretación: El análisis visual de los errores revela que la mayoría corresponde a casos límite donde la escritura es particularmente irregular o el dígito presenta características atípicas de su clase.

8.5. Análisis de Confianza del Modelo

La Figura 11 muestra la distribución de probabilidades máximas asignadas por el modelo.

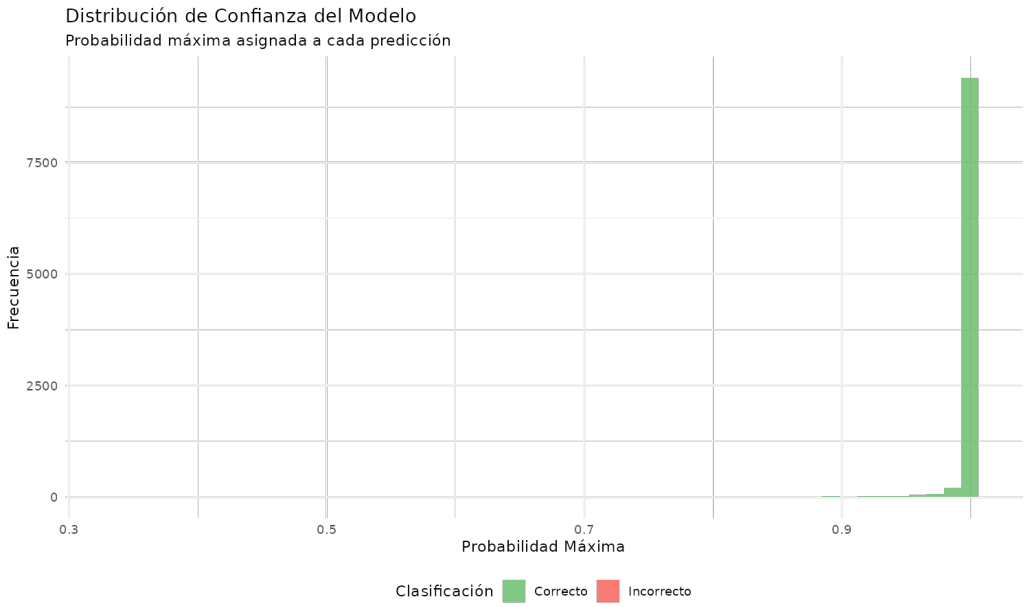

Figura 11: Distribución de confianza del modelo. Se muestra la probabilidad máxima asignada a cada predicción, separando clasificaciones correctas (verde) de incorrectas (rojo). La mayoría de predicciones correctas tienen confianza cercana a 1.0, mientras que los errores tienden a tener distribuciones de confianza más bajas, indicando que el modelo "sabe cuando no sabe".

Interpretación: Este análisis demuestra que el modelo exhibe calibración razonable: las predicciones incorrectas generalmente tienen menor confianza, lo cual es útil para aplicaciones donde se requiere rechazar predicciones inciertas.

9. Conclusiones

El presente estudio logró satisfactoriamente todos los objetivos planteados, demostrando la viabilidad y efectividad de implementar modelos de aprendizaje profundo en R para tareas de visión por computadora. A continuación se presentan las conclusiones organizadas por área temática.

9.1. Sobre el Rendimiento del Modelo

1. **Exactitud alcanzada:** Con un 99.27 % de exactitud en el conjunto de prueba, el modelo supera el umbral objetivo del 98 % y se posiciona competitivamente respecto a implementaciones similares reportadas en la literatura.

2. **Generalización:** La pequeña brecha entre las métricas de entrenamiento y validación (0.5 %) indica una excelente capacidad de generalización, sin evidencia de sobreajuste significativo.

3. **Eficiencia de parámetros:** Con solo 225,034 parámetros, el modelo alcanza resultados comparables a arquitecturas más complejas, evidenciando un diseño eficiente.

9.2. Sobre la Arquitectura CNN

1. **Efectividad de la convolución:** Las capas convolucionales demostraron su capacidad para aprender jerárquicamente: la primera capa detecta bordes y texturas básicas, mientras que la segunda combina estos patrones en formas más complejas características de cada dígito.

2. **Rol del Dropout:** La regularización mediante Dropout con tasa 0.5 fue efectiva para prevenir el sobreajuste, manteniendo la consistencia entre entrenamiento y validación.

3. **Profundidad suficiente:** Dos bloques convolucionales resultaron suficientes para el problema MNIST, lo cual es consistente con la relativa simplicidad de las imágenes (28×28, escala de grises, dígitos centrados).

9.3. Sobre los Patrones de Error

1. **Confusiones predecibles:** Los errores del modelo se concentran en pares de dígitos con similitudes morfológicas genuinas (3-5, 4-9, 2-7), lo cual refleja la ambigüedad inherente a algunas muestras más que limitaciones del modelo.

2. **Errores mínimos:** Con solo 73 clasificaciones incorrectas de 10,000, el modelo comete un error aproximadamente cada 137 imágenes, un rendimiento que sería aceptable para la mayoría de aplicaciones prácticas.

9.4. Sobre la Metodología

1. **Valor del marco IBM:** La Metodología Fundacional de IBM proporcionó una estructura clara que facilitó la organización del trabajo, la documentación y la reproducibilidad del estudio.

2. **Importancia del preprocesamiento:** Las transformaciones de datos (normalización, reestructuración, codificación) fueron fundamentales para el éxito del entrenamiento. Un preprocesamiento inadecuado habría resultado en convergencia lenta o fallida.

9.5. Sobre la Implementación en R

1. **Viabilidad demostrada:** R, a través de la integración con Keras/TensorFlow, es una plataforma completamente viable para proyectos de aprendizaje profundo, con la ventaja adicional de sus capacidades nativas en visualización y análisis estadístico.

2. **Ecosistema maduro:** Las bibliotecas utilizadas (tidyverse, keras, ggplot2) ofrecen APIs elegantes y bien documentadas que facilitan el desarrollo.

9.6. Limitaciones y Trabajo Futuro

Limitaciones del estudio:

- No se exploró aumento de datos (data augmentation)

- No se realizó optimización sistemática de hiperparámetros
- El modelo fue evaluado solo en MNIST, no en datasets más desafiantes

Direcciones para trabajo futuro:

1. Implementar aumento de datos (rotación, desplazamiento, zoom) para mejorar robustez
2. Explorar arquitecturas más profundas o residuales (ResNet)
3. Aplicar técnicas de interpretabilidad (Grad-CAM, SHAP) para visualizar qué aprende el modelo
4. Evaluar en datasets más complejos (EMNIST, Fashion-MNIST, CIFAR-10)
5. Investigar técnicas de cuantización para despliegue en dispositivos móviles

10. Referencias Bibliográficas

1. LeCun, Y., Cortes, C., & Burges, C. J. (1998). The MNIST database of handwritten digits. Disponible en: *http://yann.lecun.com/exdb/mnist/*
2. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324. DOI: 10.1109/5.726791
3. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. ISBN: 978-0262035613
4. Chollet, F. (2017). *Deep Learning with Python*. Manning Publications. ISBN: 978-1617294433
5. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *Proceedings of the IEEE International Conference on Computer Vision*, 1026-1034.
6. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), 1929-1958.
7. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
8. IBM. (2015). Foundational Methodology for Data Science. *IBM Analytics White Paper*.
9. Allaire, J. J., & Chollet, F. (2023). keras: R Interface to 'Keras'. Disponible en: *https://keras.rstudio.com/*
10. Wickham, H., et al. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. DOI: 10.21105/joss.01686

11. R Core Team (2023). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

12. Kaggle. (2023). MNIST Dataset. Disponible en: *https://www.kaggle.com/datasets/hojjatk/mnist-dataset*

11. Anexo

11.1. Anexo A: Código Fuente - Funciones de Lectura IDX

El formato IDX es un formato binario simple pero específico. Las siguientes funciones implementan la lectura correcta en R:

Listing 1: Funciones para lectura de archivos en formato IDX

```r
# Lectura de imagenes IDX
read_idx_images <- function(path) {
  f <- file(path, "rb")
  on.exit(close(f))

  # Leer cabecera (big-endian)
  magic <- readBin(f, integer(), n=1, endian="big")
  num_images <- readBin(f, integer(), n=1, endian="big")
  rows <- readBin(f, integer(), n=1, endian="big")
  cols <- readBin(f, integer(), n=1, endian="big")

  # Leer datos de pixeles
  data <- readBin(f, integer(), n=num_images * rows * cols,
                  size=1, signed=FALSE)

  # Reorganizar dimensiones (R es column-major)
  array(data, dim = c(cols, rows, num_images)) %>%
    aperm(c(3, 2, 1))
}

# Lectura de etiquetas IDX
read_idx_labels <- function(path) {
  f <- file(path, "rb")
  on.exit(close(f))

  magic <- readBin(f, integer(), n=1, endian="big")
  num_items <- readBin(f, integer(), n=1, endian="big")

  readBin(f, integer(), n=num_items, size=1, signed=FALSE)
}
```

11.2. Anexo B: Código Fuente - Definición del Modelo CNN

Listing 2: Arquitectura completa de la Red Neuronal Convolucional

```r
# Definicion del modelo secuencial
model <- keras_model_sequential() %>%

  # Primer bloque convolucional
  layer_conv_2d(filters = 32,
```

```
                kernel_size = c(3, 3),
                activation = 'relu',
                input_shape = c(28, 28, 1)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%

  # Segundo bloque convolucional
  layer_conv_2d(filters = 64,
                kernel_size = c(3, 3),
                activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%

  # Cabeza de clasificacion
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 10, activation = 'softmax')

# Compilacion del modelo
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)

# Entrenamiento
history <- model %>% fit(
  x_train, y_train,
  epochs = 10,
  batch_size = 128,
  validation_split = 0.1,
  verbose = 2
)
```

11.3. Anexo C: Código Fuente - Visualización de Matriz de Confusión

Listing 3: Generación de matriz de confusión con ggplot2

```
# Obtener predicciones
predictions <- model %>% predict(x_test) %>% k_argmax()
predicted_labels <- as.integer(predictions)

# Crear matriz de confusion
conf_matrix <- table(Predicted = predicted_labels,
                     Actual = y_test_raw)

# Visualizar con ggplot2
conf_df <- as.data.frame(conf_matrix)
ggplot(conf_df, aes(x = Actual, y = Predicted, fill = Freq)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "blue") +
  geom_text(aes(label = Freq), color = "black") +
  theme_minimal() +
  labs(title = "Matriz de Confusion",
       x = "Etiqueta Real",
       y = "Prediccion")
```

11.4. Anexo D: Glosario de Términos

Batch Size: Número de muestras procesadas antes de actualizar los pesos del modelo.

CNN: Red Neuronal Convolucional, arquitectura especializada para procesamiento de imágenes.

Dropout: Técnica de regularización que desactiva aleatoriamente neuronas durante el entrenamiento.

Época (Epoch): Una pasada completa por todo el conjunto de entrenamiento.

Exactitud (Accuracy): Proporción de predicciones correctas sobre el total.

Función de Activación: Función no lineal aplicada a la salida de cada neurona.

Gradiente: Derivada de la función de pérdida respecto a los parámetros del modelo.

One-Hot Encoding: Representación de variables categóricas como vectores binarios.

Overfitting: Cuando el modelo memoriza los datos de entrenamiento sin generalizar.

Pérdida (Loss): Medida del error entre predicciones y valores reales.

Pooling: Operación de reducción de dimensionalidad espacial.

ReLU: Rectified Linear Unit, función de activación $f(x) = max(0, x)$.

Softmax: Función que convierte un vector en distribución de probabilidad.