Clasificación de Dígitos MNIST

Redes Neuronales y Aprendizaje Profundo

Xavi Far

Programación de inteligencia artificial

10 de febrero de 2026

Informe de Práctica: Red Neuronal MLP para Clasificación de Dígitos MNIST

Nombre del estudiante

10 de febrero de 2026

Índice

1. Objetivo

El objetivo de esta práctica es diseñar e implementar una red neuronal completamente conectada (**Multilayer Perceptron - MLP**) para la clasificación de imágenes de dígitos manuscritos del dataset MNIST utilizando TensorFlow y Keras. Además, se analizará la arquitectura de la red, se justificarán las técnicas de preprocesamiento y se evaluará si el modelo presenta overfitting.

2. Descripción del dataset

El dataset *MNIST* está compuesto por imágenes en escala de grises de tamaño 28×28 píxeles, donde cada imagen representa un dígito manuscrito comprendido entre 0 y 9. Se trata de un problema de clasificación multiclase con un total de 10 categorías posibles.

El conjunto de datos se divide en 60 000 imágenes destinadas al entrenamiento del modelo y 10 000 imágenes reservadas para la fase de prueba, lo que permite evaluar el rendimiento del modelo sobre datos no vistos durante el entrenamiento.

2.1. Carga del dataset

Para la carga del dataset se utiliza directamente la implementación proporcionada por la librería `Keras`, lo que garantiza una fuente fiable y estandarizada de los datos. El procedimiento de carga se realiza mediante el siguiente fragmento de código:

```
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Este código descarga el dataset, lo carga en memoria y lo separa automáticamente en los conjuntos de entrenamiento y prueba.

3. Preprocesamiento de los datos

Antes de entrenar una red neuronal, es necesario aplicar una serie de técnicas de preprocesamiento que faciliten el aprendizaje del modelo y mejoren su estabilidad y rendimiento.

3.1. Normalización de las imágenes

Las imágenes del dataset presentan valores de intensidad de píxel en el rango [0, 255]. Para favorecer una convergencia más rápida y estable durante el entrenamiento, estos valores se normalizan escalándolos a un rango comprendido entre 0 y 1.

3.2. Conversión de etiquetas a *one-hot encoding*

Dado que el problema abordado es de clasificación multiclase (dígitos del 0 al 9), las etiquetas originales, representadas como valores enteros, se transforman a formato *one-hot encoding*. Para ello se emplea la función `to_categorical()` de la librería `Keras`.

Esta representación permite que cada clase sea tratada de forma independiente durante el proceso de entrenamiento.

4. Análisis del dataset

En primer lugar, se analiza la distribución del número de imágenes correspondientes a cada dígito. Esta información se representa mediante una gráfica que muestra la frecuencia de aparición de cada clase.

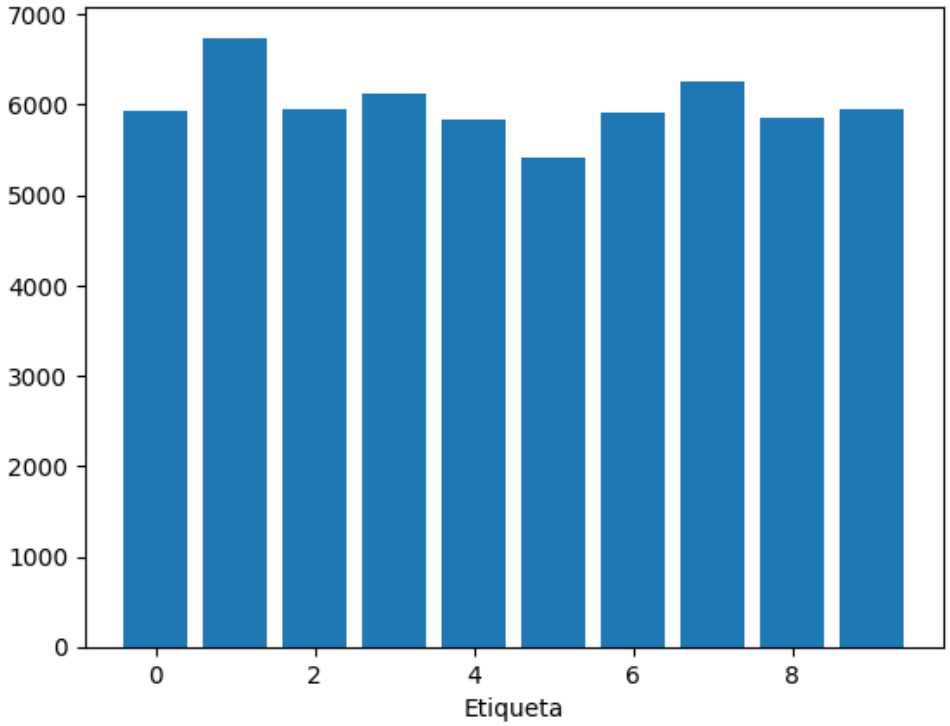

Figura 1: Gráfica de barras de etiquetas

A partir de dicha visualización se observa que el dígito que aparece con mayor frecuencia es el número 1, mientras que el menos frecuente es el número 5. No obstante, esta ligera desproporción no resulta significativa y no se espera que tenga un impacto relevante en el proceso de entrenamiento del modelo.

Asimismo, se realiza una visualización de algunas imágenes del dataset con el objetivo de comprender mejor la naturaleza de los datos. En esta representación se muestran distintos dígitos manuscritos junto con sus respectivas etiquetas, lo que permite verificar visualmente la correspondencia entre las imágenes y sus clases.

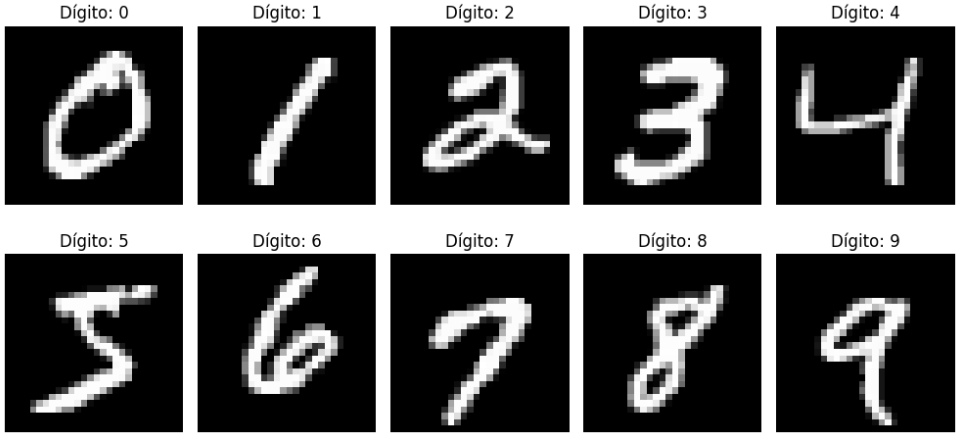

Figura 2: Ejemplos de imágenes del dataset MNIST junto con sus etiquetas

Al analizar visualmente parte del dataset, se observa que los dígitos 1 y 7, así como los dígitos 4 y 9, presentan similitudes que pueden provocar confusión.

5. Redes neuronales

Una red neuronal artificial es un modelo computacional inspirado en la organización y el funcionamiento básico de las redes neuronales biológicas del cerebro. Su objetivo principal es aprender patrones a partir de los datos para realizar tareas como clasificación, regresión, reconocimiento de imágenes, procesamiento del lenguaje natural y predicción.

5.1. Capas de las redes neuronales

Una red neuronal está compuesta por un conjunto de unidades denominadas neuronas artificiales, organizadas en capas. Estas capas permiten estructurar el flujo de información desde los datos de entrada hasta la generación del resultado final.

- **Capa de entrada:** recibe los datos iniciales, como píxeles de una imagen, señales o variables numéricas, y los transmite a las capas posteriores sin realizar transformaciones complejas.
- **Capas ocultas:** realizan transformaciones intermedias sobre la información mediante operaciones matemáticas y funciones de activación. Una red que contiene múltiples capas ocultas se denomina red neuronal profunda (*deep neural network*).
- **Capa de salida:** produce el resultado final del modelo, que puede corresponder a una etiqueta de clasificación, una probabilidad o un valor numérico continuo.

Cada neurona recibe valores de entrada, los pondera mediante pesos, suma estos valores y aplica una función de activación no lineal, como ReLU, sigmoide o tangente hiperbólica. Esta no linealidad es fundamental, ya que permite a la red modelar relaciones complejas y no lineales presentes en los datos.

5.2. Multilayer Perceptron Neuron

El perceptrón multicapa (*Multilayer Perceptron*, MLP) es una de las arquitecturas fundamentales de las redes neuronales artificiales. Está formado por múltiples capas de neuronas completamente conectadas (*fully connected layers*), organizadas en una capa de entrada, una o varias capas ocultas y una capa de salida. A diferencia del perceptrón simple, el MLP es capaz de modelar relaciones no lineales complejas gracias al uso de funciones de activación no lineales y múltiples niveles de transformación.

Cada neurona dentro de un MLP realiza una operación matemática básica que consiste en calcular una combinación lineal de sus entradas y aplicar posteriormente una función de activación. Formalmente, la salida de una neurona puede expresarse como:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{1}$$

donde x_i representan las entradas, w_i los pesos sinápticos asociados, b el término de sesgo (*bias*), $f(\cdot)$ la función de activación y y la salida de la neurona.

El entrenamiento de un MLP se realiza generalmente mediante el algoritmo de retropropagación del error (*backpropagation*), combinado con métodos de optimización basados en descenso por gradiente. Este procedimiento ajusta iterativamente los pesos de la red minimizando una función de pérdida que cuantifica la diferencia entre la salida predicha y el valor real.

El perceptrón multicapa se utiliza ampliamente en tareas de clasificación, regresión y reconocimiento de patrones, debido a su capacidad de aproximar funciones no lineales. De hecho, está demostrado teóricamente que un MLP con al menos una capa oculta y un número suficiente de neuronas puede aproximar cualquier función continua bajo condiciones generales, resultado conocido como el teorema de aproximación universal.

6. Diseño del Modelo

La arquitectura propuesta para la red neuronal está diseñada para abordar un problema de clasificación multiclase de imágenes de tamaño 28×28 píxeles. El modelo sigue una estructura secuencial compuesta por una etapa de preprocesamiento de datos, capas densas ocultas para la extracción de características y una capa de salida para la generación de probabilidades de clase.

- **Capa de entrada con Flatten:** se utiliza la operación `Flatten()` para transformar la imagen bidimensional de tamaño 28×28 en un vector unidimensional de longitud 784. Este paso es necesario para adaptar el formato de los datos a las capas densas, que requieren entradas vectoriales.
- **Capas ocultas densas:** el modelo incorpora dos capas densas (`Dense`) con función de activación ReLU. Estas capas permiten aprender representaciones no lineales de los datos mediante combinaciones ponderadas de las características de entrada.
- **Capa de salida:** la última capa está compuesta por 10 neuronas, correspondientes al número de clases del problema. Se emplea la función de activación `softmax` para convertir las salidas del modelo en una distribución de probabilidad normalizada sobre las clases.

6.1. Análisis de los componentes del modelo

1. **Función de la capa `Flatten()`:** la capa `Flatten()` cumple la función de reestructurar los datos de entrada desde un formato matricial bidimensional a un vector unidimensional sin modificar los valores originales. Este proceso no introduce parámetros adicionales ni realiza aprendizaje, sino que actúa como un paso de adaptación estructural para permitir el procesamiento posterior mediante capas completamente conectadas.

2. **Uso de la activación ReLU en las capas ocultas:** la función ReLU (*Rectified Linear Unit*) se utiliza debido a su eficiencia computacional y a su capacidad para mitigar el problema del desvanecimiento del gradiente. Al introducir no linealidad en el modelo, ReLU permite que la red aprenda representaciones complejas, además de favorecer una convergencia más rápida durante el entrenamiento en comparación con funciones de activación tradicionales como la sigmoide o la tangente hiperbólica.

3. **Uso de `softmax` en la capa de salida:** la función `softmax` se emplea en la última capa porque transforma las salidas del modelo en valores comprendidos entre 0 y 1 cuya suma es igual a 1. Esto permite interpretar cada salida como la probabilidad asociada a una clase específica, lo cual resulta especialmente adecuado para problemas de clasificación multiclase y para el uso conjunto con funciones de pérdida como *categorical cross-entropy*.

6.2. Funciones

Las funciones de activación desempeñan un papel fundamental en el comportamiento de las redes neuronales, ya que introducen no linealidad en el modelo y permiten aprender relaciones complejas entre las variables de entrada y salida. En este trabajo se emplean las funciones ReLU en las capas ocultas y Softmax en la capa de salida, debido a sus propiedades teóricas y prácticas.

6.2.1. Función ReLU

La función de activación ReLU (*Rectified Linear Unit*) se define matemáticamente como:

$$f(x) = \text{máx}(0, x) \tag{2}$$

Esta función devuelve el valor de entrada si este es positivo y cero en caso contrario. Su simplicidad computacional la convierte en una de las funciones de activación más utilizadas en redes neuronales profundas.

Entre sus principales ventajas se encuentran su capacidad para acelerar la convergencia durante el entrenamiento y su contribución a la reducción del problema del desvanecimiento del gradiente, frecuente en funciones sigmoides tradicionales. Además, ReLU induce esparsidad en las activaciones, ya que muchas neuronas permanecen inactivas (salida cero), lo que puede mejorar la eficiencia computacional y la capacidad de generalización del modelo.

No obstante, ReLU también presenta limitaciones, como el denominado problema de las "neuronas muertas", que ocurre cuando una neurona deja de activarse de forma permanente debido a actualizaciones desfavorables de los pesos. Este fenómeno puede mitigarse mediante variantes como Leaky ReLU o Parametric ReLU.

6.2.2. Función Softmax

La función Softmax se utiliza habitualmente en la capa de salida de modelos de clasificación multiclase. Su formulación matemática es la siguiente:

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

donde z_i representa la salida lineal asociada a la clase i y K es el número total de clases.

Softmax transforma un vector de valores reales en una distribución de probabilidad discreta, asegurando que todos los valores sean positivos y que su suma sea igual a uno. Esta propiedad permite interpretar directamente la salida del modelo como la probabilidad de pertenencia a cada clase.

El uso de Softmax resulta especialmente adecuado cuando se combina con la función de pérdida de entropía cruzada categórica (*categorical cross-entropy*), ya que esta combinación proporciona gradientes estables y una optimización eficiente durante el entrenamiento. Además, Softmax facilita la toma de decisiones finales mediante la selección de la clase con mayor probabilidad estimada.

7. Implementación del modelo

En esta sección se describe la implementación del modelo MLP en TensorFlow/Keras, siguiendo la arquitectura propuesta y utilizando el dataset MNIST preprocesado. El modelo se construye con la API secuencial, empleando una capa `Flatten` y capas densas con activación ReLU. Finalmente, se compila y entrena sobre el conjunto de entrenamiento.

7.1. Código de implementación

El siguiente fragmento de código muestra la implementación completa del modelo, su compilación, el proceso de entrenamiento y la obtención de las predicciones sobre el conjunto de prueba:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
import numpy as np

# Definicion del modelo
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

# Compilacion del modelo
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Entrenamiento del modelo
history = model.fit(
    x_train,
    y_train_oh,
    validation_split=0.2,
    epochs=10,
    batch_size=32
)

# Prediccion sobre el conjunto de prueba
y_pred_prob = model.predict(x_test)
y_pred = np.argmax(y_pred_prob, axis=1)
```

Durante el entrenamiento, se reserva el 20 % del conjunto de entrenamiento para validación, lo que permite monitorizar el comportamiento del modelo y detectar posibles fenómenos de sobreajuste. Tras el entrenamiento, el modelo genera probabilidades de pertenencia a cada clase para las imágenes del conjunto de prueba. La clase predicha se obtiene seleccionando aquella con mayor probabilidad mediante la operación `argmax`.

8. Entrenamiento y Evaluación

El proceso de entrenamiento del modelo consiste en el ajuste iterativo de los parámetros internos de la red neuronal con el objetivo de minimizar el error de predicción. Para ello, se definen una serie de hiperparámetros que controlan el comportamiento del algoritmo de optimización y el procedimiento de aprendizaje. A continuación, se describen los valores utilizados y su justificación.

8.1. Parámetros del entrenamiento

- **Optimizador: Adam.** El optimizador Adam (*Adaptive Moment Estimation*) es un método de optimización basado en descenso por gradiente que combina las ventajas de los algoritmos AdaGrad y RMSProp. Adam adapta automáticamente la tasa de aprendizaje para cada parámetro utilizando estimaciones del primer y segundo momento del gradiente. Esto permite una convergencia más rápida y estable, especialmente en problemas con grandes volúmenes de datos o funciones de pérdida complejas.

- **Función de pérdida: categorical cross-entropy.** La función de pérdida de entropía cruzada categórica (*categorical cross-entropy*) se utiliza en problemas de clasificación multiclase donde las etiquetas están codificadas en formato *one-hot*. Esta función mide la discrepancia entre la distribución de probabilidad predicha por el modelo y la distribución real de las clases. Matemáticamente, se expresa como:

$$L = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{4}$$

  donde y_i representa el valor real de la clase i, $\hat{y}_i$ la probabilidad predicha por el modelo y K el número total de clases. Minimizar esta función equivale a maximizar la probabilidad asignada a la clase correcta.

- **Número de épocas: 10.** Una época corresponde a una pasada completa del conjunto de entrenamiento a través de la red neuronal. El uso de 10 épocas implica que el modelo procesa todos los datos de entrenamiento diez veces durante el aprendizaje. Este valor representa un compromiso entre tiempo de entrenamiento y capacidad de convergencia, evitando en muchos casos el sobreajuste que puede producirse con un número excesivo de iteraciones.

- **Tamaño de batch: 32.** El tamaño de *batch* determina el número de muestras utilizadas para calcular el gradiente antes de actualizar los pesos del modelo. Un valor de 32 es ampliamente utilizado en la práctica, ya que ofrece un equilibrio adecuado entre estabilidad del gradiente, eficiencia computacional y consumo de memoria. Además, el entrenamiento por mini-batches permite una convergencia más robusta en comparación con el descenso por gradiente estocástico puro.

- **Validación: 20 % del conjunto de entrenamiento.** Se reserva el 20 % del conjunto de entrenamiento para validación, con el objetivo de evaluar el rendimiento del modelo durante el proceso de aprendizaje. Este subconjunto no se utiliza para actualizar los pesos, sino para monitorizar métricas como la pérdida y la precisión, permitiendo detectar fenómenos como el sobreajuste y ajustar adecuadamente los hiperparámetros.

8.2. Evaluación del modelo

Una vez finalizado el entrenamiento, el modelo se evalúa sobre un conjunto de datos independiente, no utilizado durante aprendizaje ni validación. Esta evaluación permite estimar su capacidad de generalización mediante métricas como la precisión (*accuracy*), la pérdida (*loss*) y la matriz de confusión. Es fundamental para asegurar que el modelo no solo memoriza los datos de entrenamiento, sino que también realiza predicciones fiables sobre datos nuevos.

8.3. Visualización de resultados

Para analizar el comportamiento del modelo durante el entrenamiento, se presentan las gráficas de evolución de la pérdida y la precisión tanto para los conjuntos de entrenamiento como de validación.

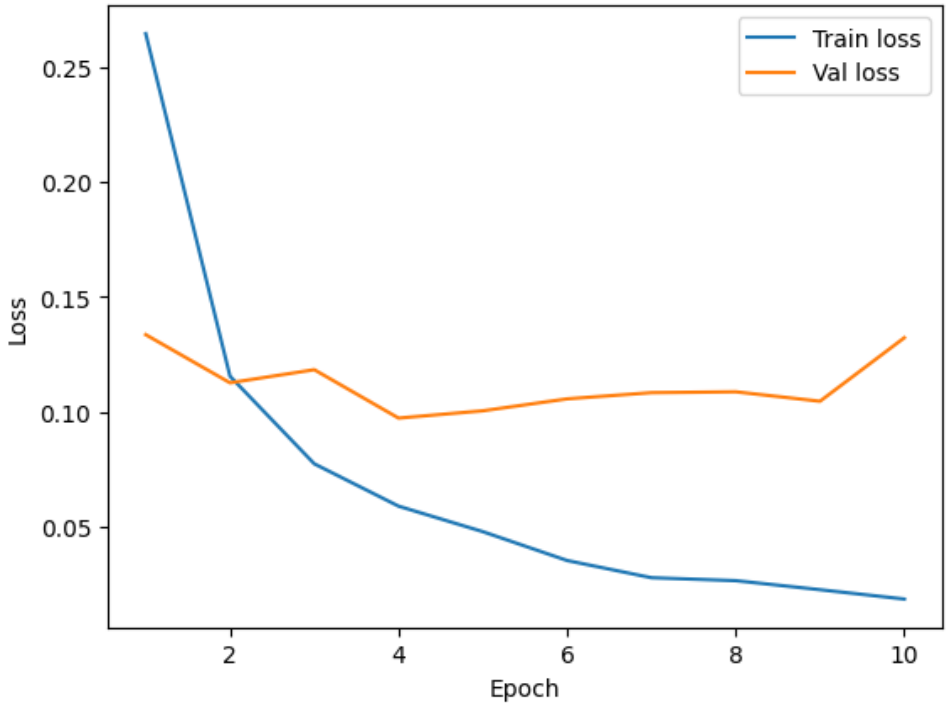

Figura 3: Evolución de la función de pérdida (*loss*) durante el entrenamiento y validación.

La Figura 3 muestra que la pérdida de entrenamiento disminuye de manera constante, lo que indica que el modelo está ajustando correctamente sus parámetros. La pérdida de validación alcanza su mínimo alrededor de la cuarta época y, a partir de ahí, comienza a divergir ligeramente de la curva de entrenamiento, evidenciando un inicio de sobreajuste (*overfitting*). Esto sugiere que un *early stopping* óptimo se situaría en la cuarta época.

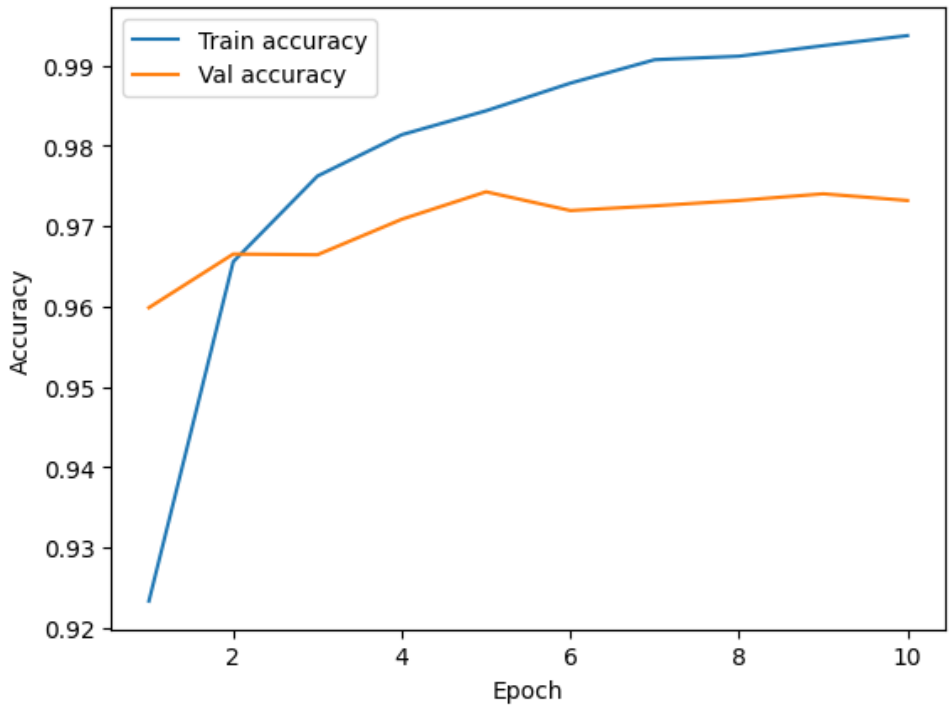

Figura 4: Evolución de la precisión (*accuracy*) durante el entrenamiento y validación.

La Figura 4 muestra que la precisión de entrenamiento crece de manera sostenida hasta casi 0.995 en la última época, mientras que la precisión de validación alcanza su pico alrededor de la quinta época (0.974) y luego se estabiliza. La divergencia creciente entre ambas curvas confirma el ligero sobreajuste: el modelo sigue mejorando sobre los datos conocidos sin que esto se refleje en los datos nuevos.

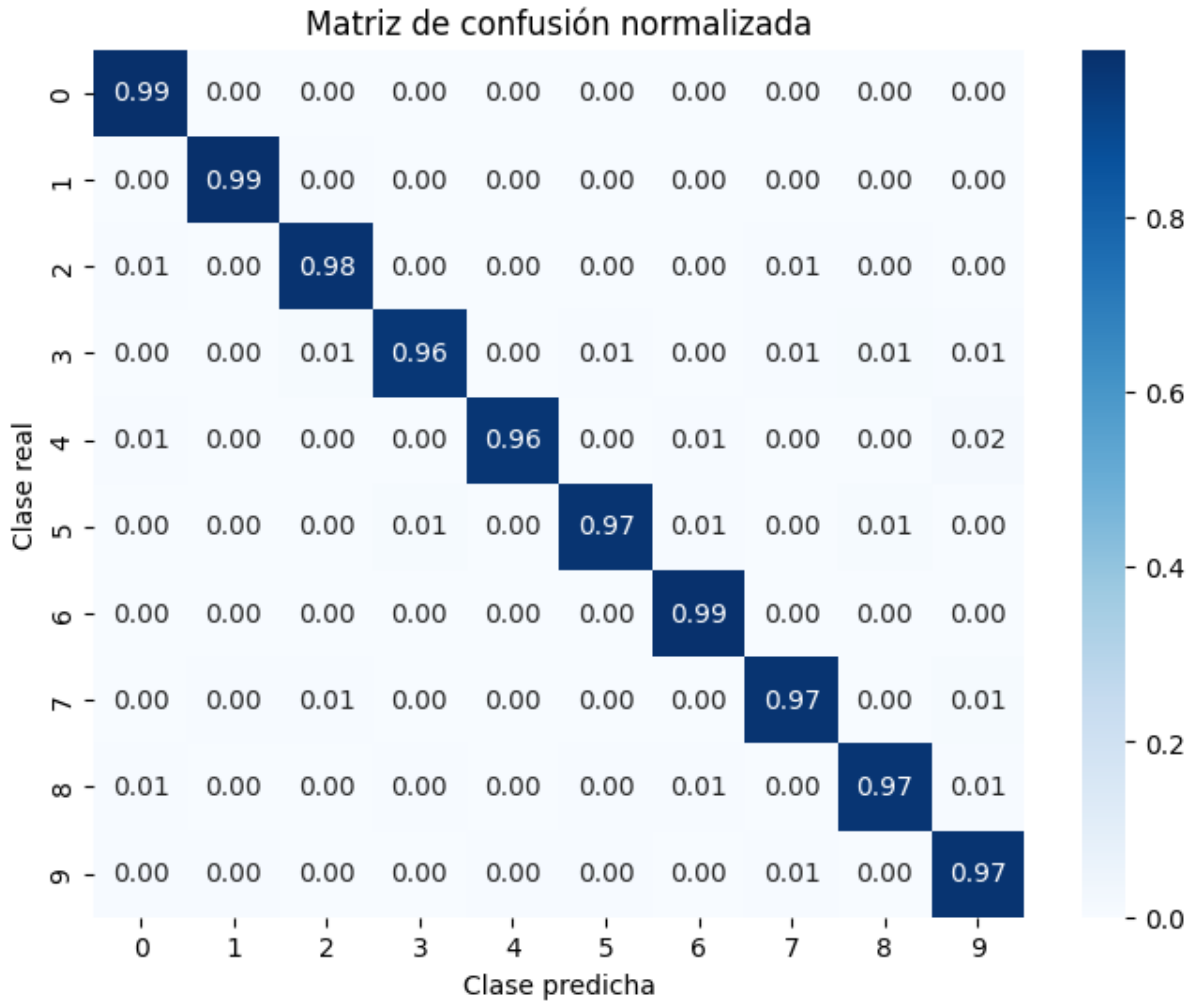

Figura 5: Matriz de confusión normalizada sobre los resultados del modelo.

La Figura 5 presenta la matriz de confusión normalizada, que permite evaluar el rendimiento del modelo por clase. Se observa un alto desempeño general, con la mayoría de clases superando el 96Algunas confusiones marginales aparecen, por ejemplo, la clase 4 se mezcla ligeramente con las clases 9 y 2, pero estos errores son mínimos. En conjunto, la matriz confirma que el modelo discrimina correctamente la mayoría de los dígitos, a pesar del ligero sobreajuste detectado en las métricas globales.

9. Preguntas de análisis

9.1. Preprocesamiento de Datos

¿Por qué es importante normalizar los datos antes de entrenar? Las redes neuronales utilizan el descenso de gradiente para optimizar los pesos. Si las características de entrada tienen rangos muy dispares (por ejemplo, píxeles entre 0 y 255), la superficie de la función de coste se vuelve irregular, dificultando la convergencia. La normalización (escalar los valores al rango $[0, 1]$) permite que todos los pesos se actualicen de manera proporcional, acelerando el entrenamiento y evitando inestabilidad numérica.

¿Por qué se usa one-hot encoding en lugar de enteros? Las etiquetas en formato entero (0, 1, ..., 9) implican una relación de orden o magnitud (ordinalidad) que no existe en una clasificación nominal. La red podría interpretar erróneamente que la clase 8 es .el doble"que la clase 4. El *one-hot encoding* transforma las etiquetas en vectores ortogonales equidistantes, eliminando cualquier jerarquía matemática falsa entre las clases.

9.2. Arquitectura de la Red

¿Cuál es la función de la capa `Flatten()`? Las capas densas (`Dense`) requieren vectores unidimensionales como entrada. Las imágenes del conjunto MNIST son matrices bidimensionales (28×28). La capa `Flatten` se encarga de .aplanar.esta estructura, transformando la matriz en un vector de 784 elementos ($28 \times 28 = 784$) para que pueda ser procesado por las neuronas.

¿Por qué se usa la activación ReLU en las capas ocultas? La función ReLU (*Rectified Linear Unit*), definida como $f(x) = \max(0, x)$, se utiliza por tres razones principales:

- Introduce no-linealidad, permitiendo a la red aprender patrones complejos.
- Es computacionalmente eficiente.
- Mitiga el problema del desvanecimiento del gradiente (*vanishing gradient*) presente en funciones como la sigmoide en redes profundas.

¿Por qué la última capa usa `softmax` en lugar de `ReLU`? La capa de salida debe representar una distribución de probabilidad sobre las 10 clases posibles. Mientras que ReLU devuelve valores positivos sin acotar, la función `softmax` normaliza las salidas (logits) para que sumen 1.0, permitiendo interpretar el resultado como la probabilidad de pertenencia a cada clase.

9.3. Entrenamiento y Evaluación

¿Por qué `categorical_crossentropy` y no `binary_crossentropy`? La `binary_crossentropy` se utiliza para clasificación binaria o multietiqueta independiente. Para la clasificación multiclase donde las clases son mutuamente excluyentes (un dígito no puede ser un 5 y un 3 a la vez), se utiliza la `categorical_crossentropy`, que penaliza la divergencia entre la distribución de probabilidad predicha y la etiqueta real (one-hot).

¿Qué función tiene la partición del 20 % de validación? Este subconjunto de datos, que el modelo no ve durante el ajuste de pesos, sirve para monitorear la capacidad de generalización en tiempo real. Permite detectar cuándo el modelo empieza a memorizar los datos de entrenamiento en lugar de aprender patrones generales.

¿Cómo identificas si hay overfitting? El sobreajuste se identifica observando las curvas de aprendizaje: ocurre cuando la pérdida de entrenamiento continúa disminuyendo, pero la pérdida de validación se estanca o comienza a aumentar. También se manifiesta como una gran discrepancia entre la precisión de entrenamiento (muy alta) y la de validación.

¿Qué técnicas podrías aplicar para reducir el overfitting? Existen varias estrategias de regularización:

- **Dropout:** Desactiva aleatoriamente un porcentaje de neuronas durante el entrenamiento.
- **Early Stopping:** Detiene el entrenamiento cuando la métrica de validación deja de mejorar.
- **Simplificación del modelo:** Reducir el número de capas o neuronas.
- **Regularización L1/L2:** Penalizar los pesos grandes en la función de coste.
- **Data Augmentation:** Aumentar sintéticamente el dataset mediante transformaciones.

10. Conclusiones

A partir del desarrollo, entrenamiento y evaluación del modelo de red neuronal para la clasificación de dígitos manuscritos (MNIST), se extraen las siguientes conclusiones principales:

- **Diagnóstico de Sobreajuste (Overfitting):** El análisis de las curvas de aprendizaje revela que el modelo alcanza su punto óptimo de generalización en la **cuarta época**. A partir de este momento, aunque la precisión de entrenamiento continúa mejorando (acercándose al 99.5 %), la pérdida de validación comienza a divergir y aumentar. Esto confirma que extender el entrenamiento a 10 épocas sin técnicas de regularización es contraproducente para este modelo específico.
- **Eficacia de la Arquitectura MLP:** La arquitectura de Perceptrón Multicapa (MLP) propuesta (entrada aplanada, capas ocultas con activación ReLU y salida Softmax) ha demostrado ser altamente efectiva para este problema, logrando una tasa de acierto global superior al 97 % en el conjunto de prueba. La matriz de confusión ratifica este desempeño, mostrando una discriminación casi perfecta en clases como el '0', '1' y '6'.
- **Importancia del Preprocesamiento:** Se ha verificado la criticidad de la normalización de los datos de entrada (escalado de píxeles a $[0, 1]$) para la convergencia del algoritmo. Asimismo, el uso de *one-hot encoding* para las etiquetas ha resultado fundamental para permitir el cálculo correcto de la función de pérdida `categorical_crossentropy`, evitando falsas relaciones ordinales entre las clases numéricas.

- **Estrategias de Optimización Futura:** Para mitigar el sobreajuste detectado y mejorar la robustez del modelo en futuras iteraciones, se recomienda implementar:

    1. **Early Stopping:** Configurar el entrenamiento para detenerse automáticamente cuando la pérdida de validación no mejore tras 2 o 3 épocas consecutivas.
    2. **Regularización:** Introducir capas de `Dropout` (con una tasa entre 0.2 y 0.5) después de las capas densas para reducir la co-adaptación de neuronas.