Universidad de Alicante

Ingeniería Informática

Sistemas Inteligentes

Práctica 2:

Visión artificial y aprendizaje

Autor:

Jaime Hernández Delgado (jhd3)

Curso 2024/2025

Índice

1. Introducción

1.1. Objetivos

Esta práctica tiene como objetivo principal el desarrollo de capacidades en el ámbito del aprendizaje automático y la visión artificial, centrándose específicamente en la implementación y optimización de redes neuronales para la clasificación de imágenes. Los objetivos específicos son:

1.2. Herramientas y tecnologías utilizadas

Para el desarrollo de esta práctica se han utilizado diversas herramientas y tecnologías del ecosistema de Python orientadas al aprendizaje automático y la ciencia de datos:

1.2.1. Biblioteca principal

- **Keras**: API de alto nivel que se ejecuta sobre TensorFlow, que ofrece:
  - Interfaz intuitiva para la construcción de redes neuronales
  - Implementación rápida de modelos complejos
  - Amplia variedad de capas predefinidas y funciones de activación
  - Herramientas de preprocesamiento de datos

1.2.2. Bibliotecas de apoyo

- **NumPy**: Biblioteca fundamental para la computación científica en Python:
  - Manipulación eficiente de arrays multidimensionales
  - Operaciones matemáticas vectorizadas
  - Funciones para el preprocesamiento de datos
- **Matplotlib**: Biblioteca para la creación de visualizaciones:
  - Generación de gráficas de evolución del entrenamiento
  - Visualización de imágenes del dataset
  - Creación de gráficas comparativas de resultados

1.2.3. Dataset

- **CIFAR-10**: Conjunto de datos estándar que contiene:
  - 60.000 imágenes en color de 32x32 píxeles
  - 10 clases diferentes (avión, automóvil, pájaro, gato, ciervo, perro, rana, caballo, barco y camión)
  - 50.000 imágenes para entrenamiento y 10.000 para pruebas
  - Distribución equilibrada de clases

1.2.4. Control de versiones y documentación

- **LaTeX**: Sistema de composición de textos utilizado para la generación de esta memoria
- **Git**: Sistema de control de versiones para el seguimiento de cambios en el código

1.3. Preprocesamiento de datos

El preprocesamiento de los datos es un paso fundamental para el correcto funcionamiento de las redes neuronales. En esta práctica, trabajamos con el dataset CIFAR10, que requiere cierta preparación antes de poder ser utilizado efectivamente por nuestros modelos.

1.3.1. Carga del dataset

El primer paso consiste en cargar el dataset CIFAR10 utilizando las utilidades proporcionadas por Keras:

```
(X_train, Y_train), (X_test, Y_test) = keras.datasets.cifar10.load_data
    ()
```

Tras la carga inicial, los datos presentan las siguientes características:

- **X_train**: Array de 50.000 imágenes de entrenamiento de dimensiones (32, 32, 3)
- **Y_train**: Array de 50.000 etiquetas de entrenamiento
- **X_test**: Array de 10.000 imágenes de prueba de dimensiones (32, 32, 3)
- **Y_test**: Array de 10.000 etiquetas de prueba

1.3.2. Normalización de las imágenes

Las imágenes originales tienen valores de píxeles en el rango [0, 255]. Para mejorar el entrenamiento de las redes neuronales, es necesario normalizar estos valores al rango [0, 1]:

```
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

Esta normalización es importante por varios motivos:

- Evita problemas de escala en los cálculos numéricos
- Ayuda a que el proceso de entrenamiento sea más estable
- Facilita la convergencia del algoritmo de optimización

1.3.3. Preprocesamiento específico para MLP

Para las redes neuronales tipo MLP, es necesario aplanar las imágenes ya que estas redes esperan datos de entrada unidimensionales:

```
X_train_mlp = X_train.reshape((X_train.shape[0], -1))  # (50000, 3072)
X_test_mlp = X_test.reshape((X_test.shape[0], -1))     # (10000, 3072)
```

Esta transformación convierte cada imagen de una matriz 32x32x3 en un vector de 3.072 elementos (32 * 32 * 3).

1.3.4. Codificación de etiquetas

Las etiquetas originales están en formato de índice (números del 0 al 9). Para el entrenamiento, necesitamos convertirlas a formato one-hot encoding:

```
Y_train = keras.utils.to_categorical(Y_train, 10)
Y_test = keras.utils.to_categorical(Y_test, 10)
```

Esta transformación convierte cada etiqueta en un vector de 10 elementos donde:

- El valor 1 indica la clase correcta
- El resto de valores son 0
- Por ejemplo, la clase 3 se convierte en [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

1.3.5. Consideraciones para CNN

Para las redes neuronales convolucionales (CNN), no es necesario aplanar las imágenes, ya que estas redes están diseñadas para trabajar directamente con datos bidimensionales. Por lo tanto, mantendremos la estructura original de las imágenes (32, 32, 3) cuando trabajemos con CNNs.

1.3.6. Validación de los datos

Después del preprocesamiento, es importante verificar:

- Que los valores de los píxeles estén en el rango [0, 1]
- Que las dimensiones de los arrays sean correctas
- Que las etiquetas estén correctamente codificadas

```
print(f"Rango de valores X_train: [{X_train.min()}, {X_train.max()}]")
print(f"Forma X_train MLP: {X_train_mlp.shape}")
print(f"Forma Y_train: {Y_train.shape}")
```

Este preprocesamiento es crucial para el correcto funcionamiento de nuestros modelos y nos permitirá obtener mejores resultados durante el entrenamiento.

1.4. Tarea A: Implementación básica de MLP

1.4.1. Fundamentos teóricos

El Perceptrón Multicapa (Multi-Layer Perceptron, MLP) es una de las arquitecturas más fundamentales de redes neuronales artificiales. Se compone de múltiples capas de neuronas artificiales organizadas de manera jerárquica, donde cada neurona en una capa está conectada con todas las neuronas de la capa siguiente.

Estructura básica: Un MLP típico consta de tres tipos principales de capas:

- **Capa de entrada**: Recibe los datos de entrada (en nuestro caso, los píxeles de la imagen)
- **Capas ocultas**: Realizan transformaciones no lineales de los datos
- **Capa de salida**: Produce la predicción final (en nuestro caso, la clasificación de la imagen)

Funcionamiento: El proceso de una neurona artificial se puede describir matemáticamente como:

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

Donde:

- x_i son las entradas
- w_i son los pesos asociados a cada entrada
- b es el sesgo (bias)
- f es la función de activación
- y es la salida de la neurona

Función de activación: En nuestra implementación inicial utilizamos la función sigmoid para las capas ocultas:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Y softmax para la capa de salida:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Proceso de aprendizaje: El aprendizaje en un MLP ocurre mediante el algoritmo de retropropagación (backpropagation):

1. **Propagación hacia adelante**: Los datos atraviesan la red, generando una predicción
2. **Cálculo del error**: Se compara la predicción con el valor real

3. **Retropropagación**: El error se propaga hacia atrás en la red

4. **Actualización de pesos**: Se ajustan los pesos para minimizar el error

1.5. Tarea B: Ajuste del número de épocas

1.5.1. Fundamentos teóricos

En el contexto del aprendizaje automático, una época representa una pasada completa a través de todo el conjunto de datos de entrenamiento. El número de épocas es un hiperparámetro crucial que determina cuántas veces el algoritmo procesará el conjunto de datos completo durante el entrenamiento.

El sobreentrenamiento (overfitting) es un fenómeno que ocurre cuando un modelo aprende con demasiado detalle los datos de entrenamiento, hasta el punto de memorizar sus particularidades y ruido, en lugar de aprender patrones generales. Esto resulta en:

- Un rendimiento excelente en los datos de entrenamiento
- Un rendimiento deficiente en datos nuevos (baja capacidad de generalización)
- Una brecha creciente entre el rendimiento en entrenamiento y validación

La relación entre el número de épocas y el sobreentrenamiento es directa:

- Pocas épocas pueden resultar en subentrenamiento (underfitting), donde el modelo no ha aprendido suficientemente los patrones de los datos
- Demasiadas épocas pueden llevar al sobreentrenamiento, donde el modelo memoriza los datos de entrenamiento
- El desafío está en encontrar el punto óptimo entre estos dos extremos

1.5.2. Experimentación

Para encontrar el número óptimo de épocas, se realizaron experimentos con diferentes configuraciones:

Configuración del experimento:

- Valores de épocas probados: [5, 10, 20, 50, 100]
- Arquitectura del modelo: MLP con una capa oculta de 32 neuronas
- Función de activación: sigmoid en la capa oculta, softmax en la capa de salida
- Optimizador: Adam
- División de datos: 90 % entrenamiento, 10 % validación

Métricas monitorizadas:

- Precisión (accuracy) en entrenamiento y validación
- Tiempo de entrenamiento
- Pérdida (loss) en entrenamiento y validación

1.5.3. Resultados y análisis

Análisis de la precisión: La precisión del modelo evoluciona de la siguiente manera:

- 5 épocas: 0.39 (insuficiente entrenamiento)
- 20 épocas: 0.43 (mejora significativa)
- 50 épocas: 0.453 (mejor rendimiento)
- 100 épocas: 0.44 (ligera degradación)

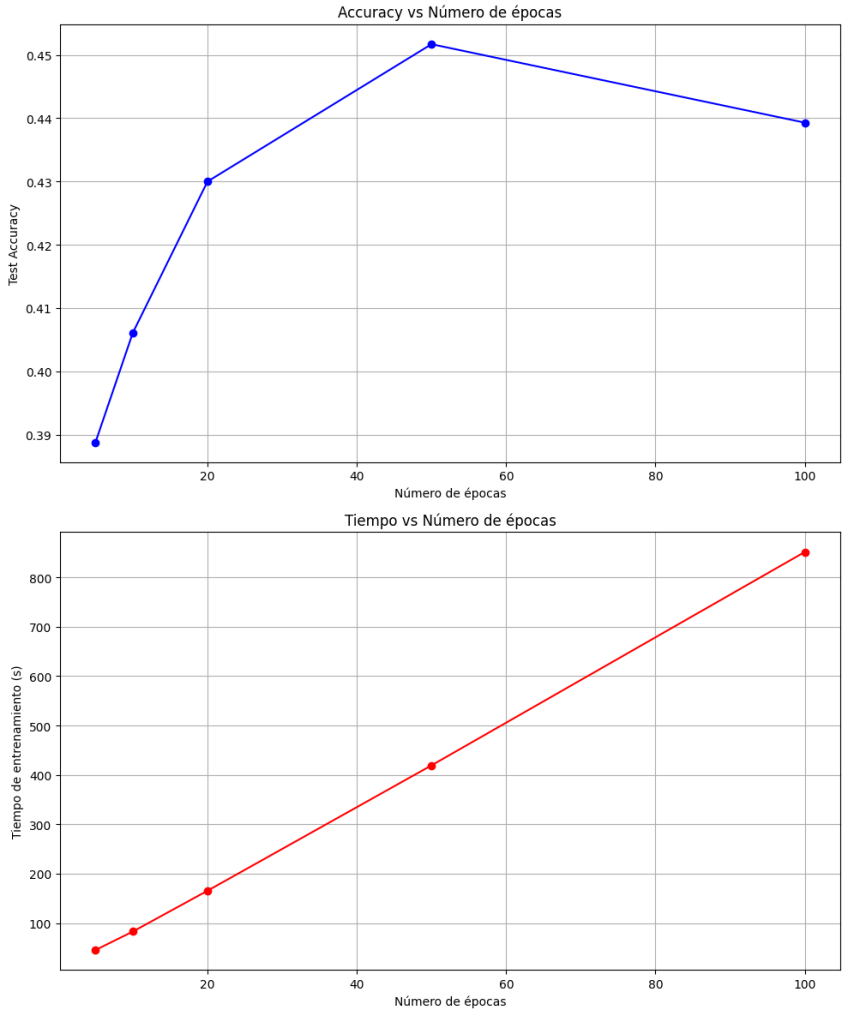

Figura 1: Evolución de la precisión según el número de épocas

Análisis del coste computacional: El tiempo de entrenamiento muestra una relación lineal con el número de épocas:

- 5 épocas: $\approx$ 50 segundos
- 50 épocas: $\approx$ 400 segundos
- 100 épocas: $\approx$ 800 segundos

Análisis de las curvas de aprendizaje: Las curvas de aprendizaje del mejor modelo (50 épocas) revelan:

- Una mejora constante en la precisión de entrenamiento hasta alcanzar 0.50
- Una estabilización de la precisión de validación alrededor de 0.43-0.45
- Una brecha creciente entre entrenamiento y validación, indicando inicio de sobreentrenamiento

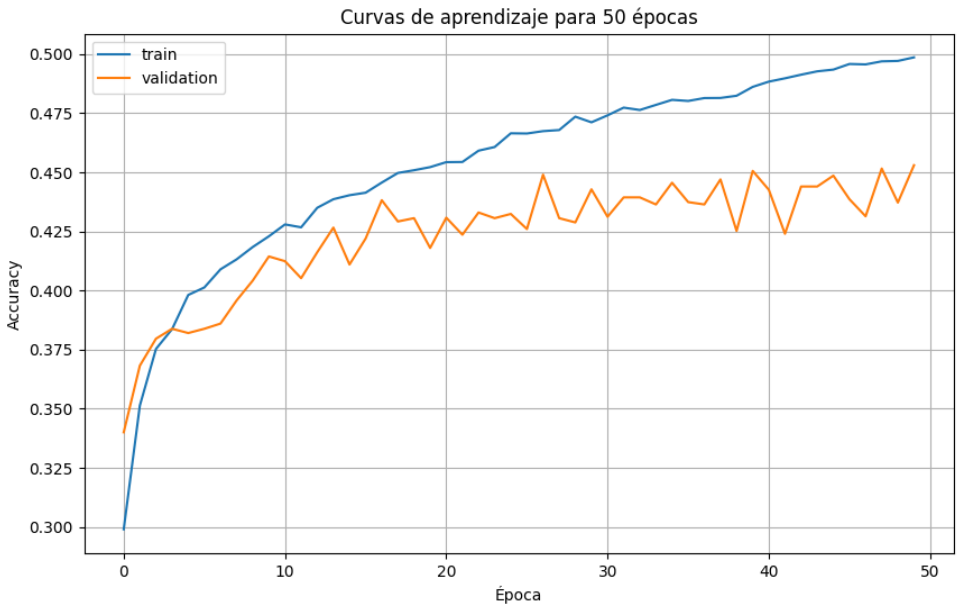

Figura 2: Curvas de aprendizaje para el modelo de 50 épocas

Conclusiones:

- El número óptimo de épocas para este modelo es 50, proporcionando el mejor balance entre precisión y tiempo de entrenamiento
- Entrenar más allá de 50 épocas resulta en:
  - Mayor tiempo de computación
  - Inicio de sobreentrenamiento

    - Degradación de la capacidad de generalización

- Se observa una clara relación entre el número de épocas y el riesgo de sobreentrenamiento

1.6. Tarea C: Ajuste del tamaño de batch

1.6.1. Fundamentos teóricos

El tamaño de batch (batch size) es un hiperparámetro que define cuántas muestras procesa el modelo antes de actualizar sus pesos. Este parámetro tiene un impacto significativo tanto en el proceso de aprendizaje como en el rendimiento computacional del modelo.

Aspectos clave del batch size:

- **Impacto en el aprendizaje**:
  - Afecta la estimación del gradiente
  - Influye en la capacidad de generalización
  - Determina la estabilidad del entrenamiento
- **Consideraciones computacionales**:
  - Define el uso de memoria durante el entrenamiento
  - Afecta la velocidad de convergencia
  - Impacta en el aprovechamiento del hardware

1.6.2. Experimentación

Para encontrar el tamaño de batch óptimo, se realizaron experimentos con las siguientes configuraciones:

Configuración del experimento:

- Valores de batch size: [16, 32, 64, 128, 256]
- Número fijo de épocas: 50
- Arquitectura: MLP con 32 neuronas en la capa oculta
- Función de activación: sigmoid
- División de datos: 90 % entrenamiento, 10 % validación

1.6.3. Resultados y análisis

Análisis del rendimiento: Los resultados muestran una clara relación entre el tamaño del batch y el rendimiento del modelo:

- **Batch size 16**:
  - Accuracy: 0.418

    - Tiempo: 775 segundos
    - Mayor variabilidad en el entrenamiento
- **Batch size 64**:
    - Accuracy: 0.449
    - Tiempo: 420 segundos
    - Mejor balance precisión/tiempo
- **Batch size 256**:
    - Accuracy: 0.457
    - Tiempo: 140 segundos
    - Mayor eficiencia computacional

Análisis del coste computacional: Se observa una clara relación inversa entre el tamaño del batch y el tiempo de entrenamiento:

- Reducción significativa del tiempo al aumentar el batch size
- Mejora de eficiencia más pronunciada entre 16 y 128
- Saturación de la mejora en tiempos a partir de 128

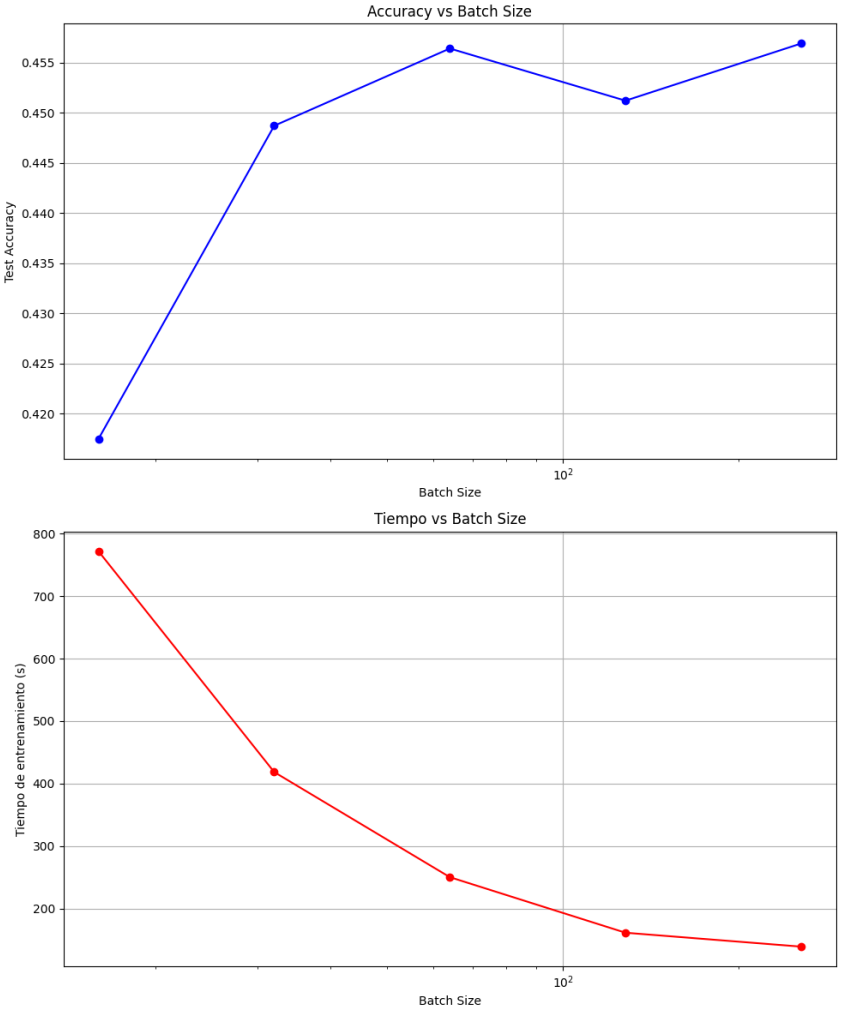

Figura 3: Tiempo de entrenamiento según el tamaño de batch

Análisis de las curvas de aprendizaje: Para el batch size de 256 (mejor rendimiento):

- Convergencia estable y consistente
- Diferencia moderada entre train y validation
- Mejora continua sin signos claros de sobreentrenamiento

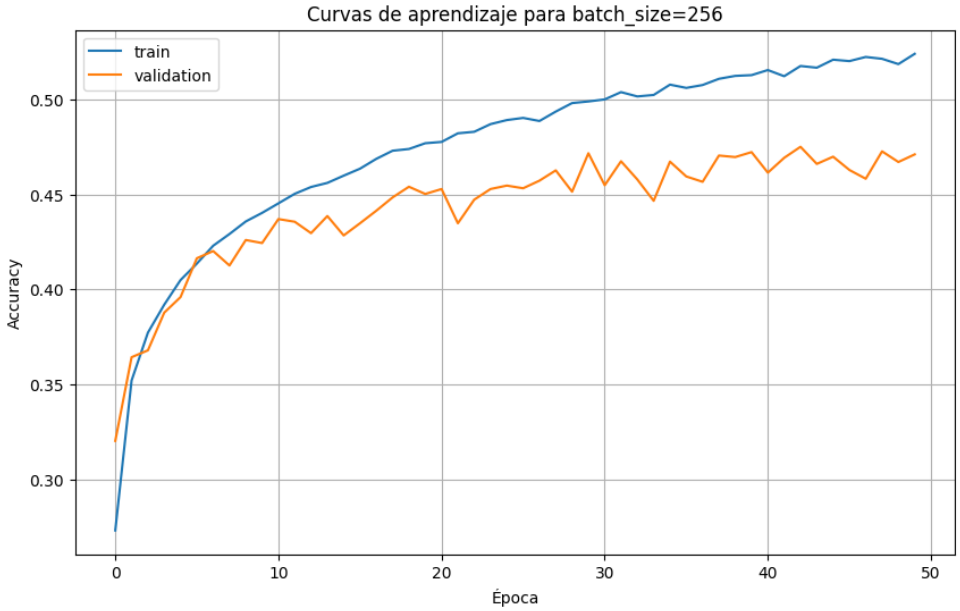

Figura 4: Curvas de aprendizaje para batch size=256

Conclusiones:

- El tamaño de batch óptimo es 256, proporcionando:
  - Mejor accuracy (0.457)
  - Tiempo de entrenamiento reducido (140s)
  - Estabilidad en el aprendizaje
- Se observa que:
  - Batches pequeños resultan en entrenamiento más lento
  - Batches grandes mejoran la eficiencia sin comprometer el rendimiento
  - Existe un punto de equilibrio entre precisión y tiempo

1.7. Tarea D: Funciones de activación

1.7.1. Fundamentos teóricos

Las funciones de activación son componentes críticos en las redes neuronales que introducen no linealidad en el modelo. Cada función tiene características específicas:

- **Sigmoid**: Rango [0,1], tradicionalmente usada pero propensa a desvanecimiento del gradiente
- **ReLU**: $f(x) = max(0, x)$, reduce el problema del desvanecimiento del gradiente
- **Tanh**: Similar a sigmoid pero con rango [-1,1]
- **ELU**: Variante de ReLU que puede manejar valores negativos

1.7.2. Experimentación

Se realizaron pruebas manteniendo la arquitectura base del MLP (32 neuronas en la capa oculta) y modificando únicamente la función de activación. Parámetros:

- Épocas: 50
- Batch size: 256
- Optimizador: Adam

1.7.3. Resultados y análisis

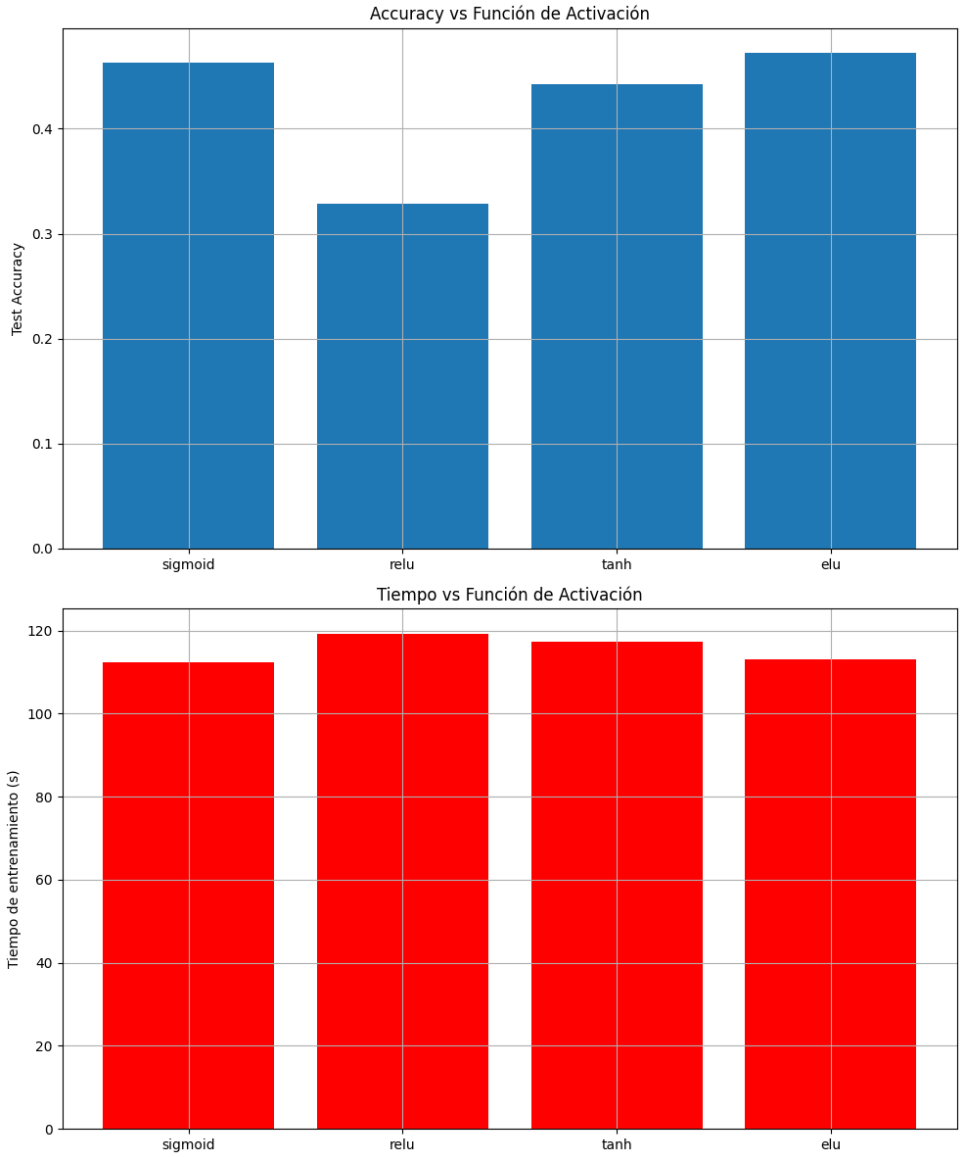

Figura 5: Comparación de funciones de activación

Las funciones sigmoid y elu mostraron el mejor rendimiento (accuracy 0.45), mientras que relu tuvo el peor desempeño. Los tiempos de entrenamiento fueron similares para todas las funciones, con diferencias menores al 10

1.8. Tarea E: Ajuste del número de neuronas

1.8.1. Fundamentos teóricos

El número de neuronas en la capa oculta determina la capacidad del modelo para aprender patrones complejos:

- Pocas neuronas: Underfitting (capacidad insuficiente)
- Demasiadas neuronas: Posible overfitting y mayor coste computacional

1.8.2. Experimentación

Se evaluaron diferentes configuraciones con [16, 32, 64, 128, 256] neuronas en la capa oculta, manteniendo:

- Función de activación: ELU (mejor resultado de tarea D)
- Épocas: 50
- Batch size: 256

1.8.3. Resultados y análisis

Los resultados muestran una mejora significativa en accuracy al aumentar el número de neuronas hasta 128, donde se alcanza un accuracy de aproximadamente 0.48. Sin embargo, el tiempo de entrenamiento aumenta exponencialmente con el número de neuronas. La configuración de 128 neuronas ofrece el mejor balance entre rendimiento y coste computacional.

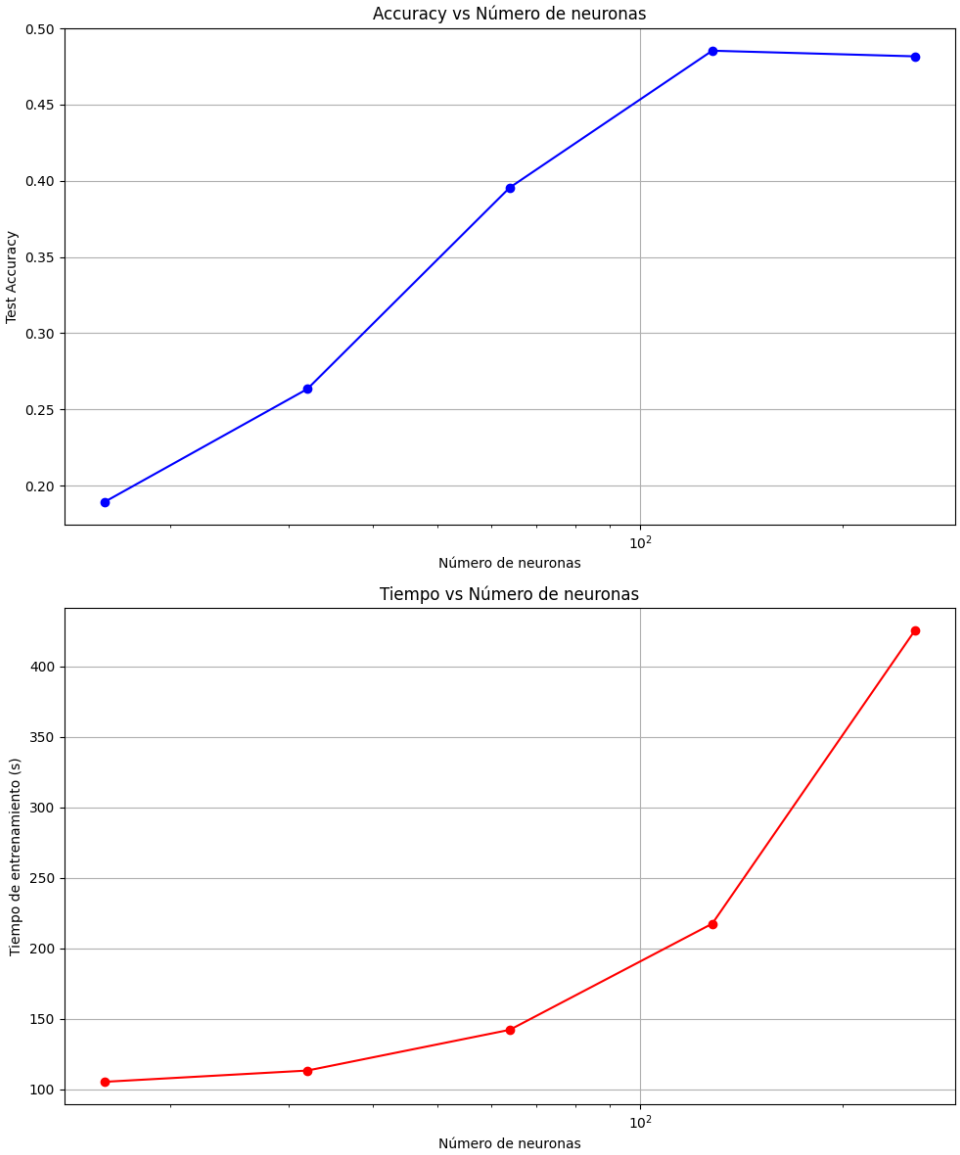

Figura 6: Número de neuronas

1.9. Tarea F: Optimización de MLP multicapa

1.9.1. Fundamentos teóricos

Las arquitecturas multicapa permiten aprender representaciones jerárquicas de los datos, donde cada capa puede especializarse en diferentes niveles de abstracción. El diseño de la arquitectura debe considerar:

- Profundidad (número de capas)
- Anchura (neuronas por capa)
- Patrón de conectividad entre capas

1.9.2. Experimentación

Se probaron diferentes arquitecturas:

- Una capa: [64]
- Dos capas: [32→32], [64→32], [32→64]
- Tres capas: [128→64→32]

1.9.3. Resultados y análisis

La arquitectura [128→64→32] mostró el mejor rendimiento (accuracy 0.50), con un tiempo de entrenamiento aceptable. Esta configuración permite una reducción gradual de dimensionalidad que parece beneficiar el aprendizaje. Las curvas de aprendizaje muestran una convergencia estable con una brecha moderada entre entrenamiento y validación, sugiriendo un buen balance entre capacidad de ajuste y generalización.

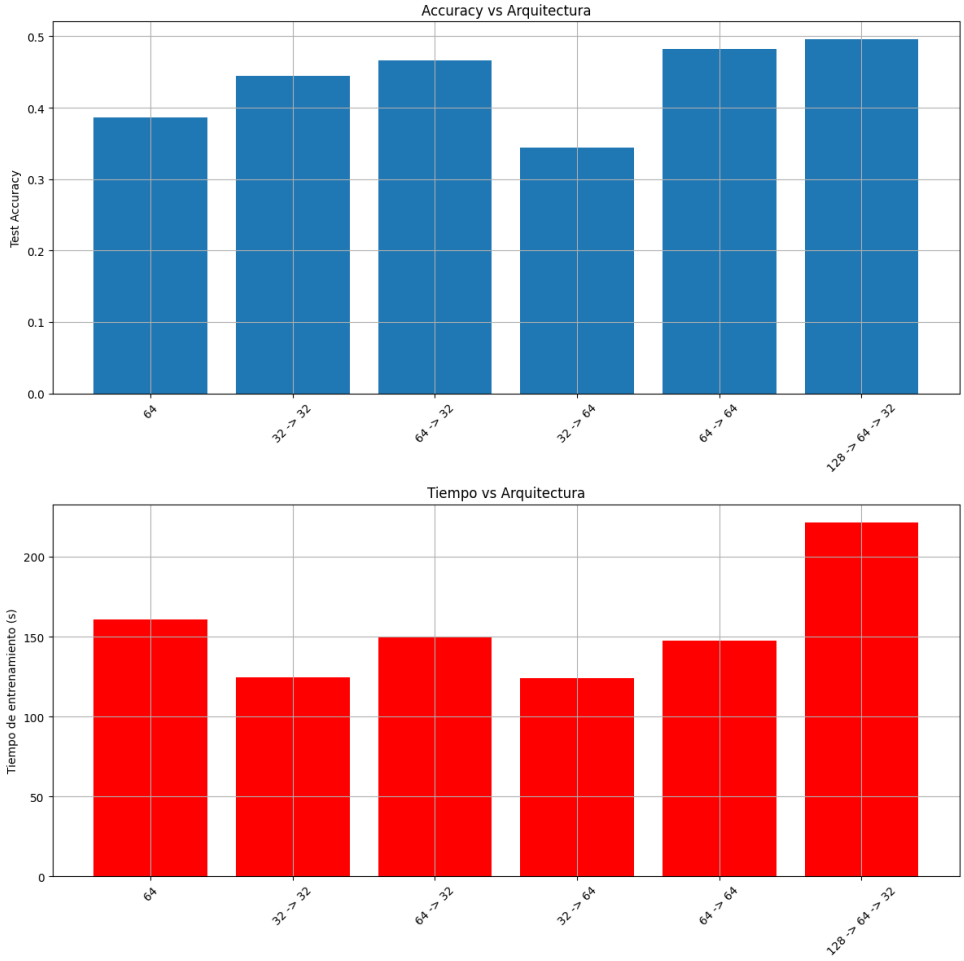

Figura 7: Arquitectura [128→64→32]

1.10. Tarea G: Implementación de CNN

1.10.1. Fundamentos teóricos

Las CNN se distinguen de otras redes neuronales por su mejor desempeño con entradas de señal de imagen, voz o audio. Tienen tres tipos principales de capas, que son:

1. Capa convolucional
2. Capa de agrupamiento
3. Capa totalmente conectada (FC)

La capa convolucional es la primera capa de una red convolucional. Con cada capa, la CNN aumenta su complejidad, identificando mayores porciones de la imagen. Las primeras capas se enfocan en características simples, como colores y bordes. A medida que los datos de la imagen avanzan a través de las capas de CNN, se comienzan a reconocer elementos o formas más grandes del objeto, hasta que finalmente se identifica el objeto previsto.

1.10.2. Resultados y análisis

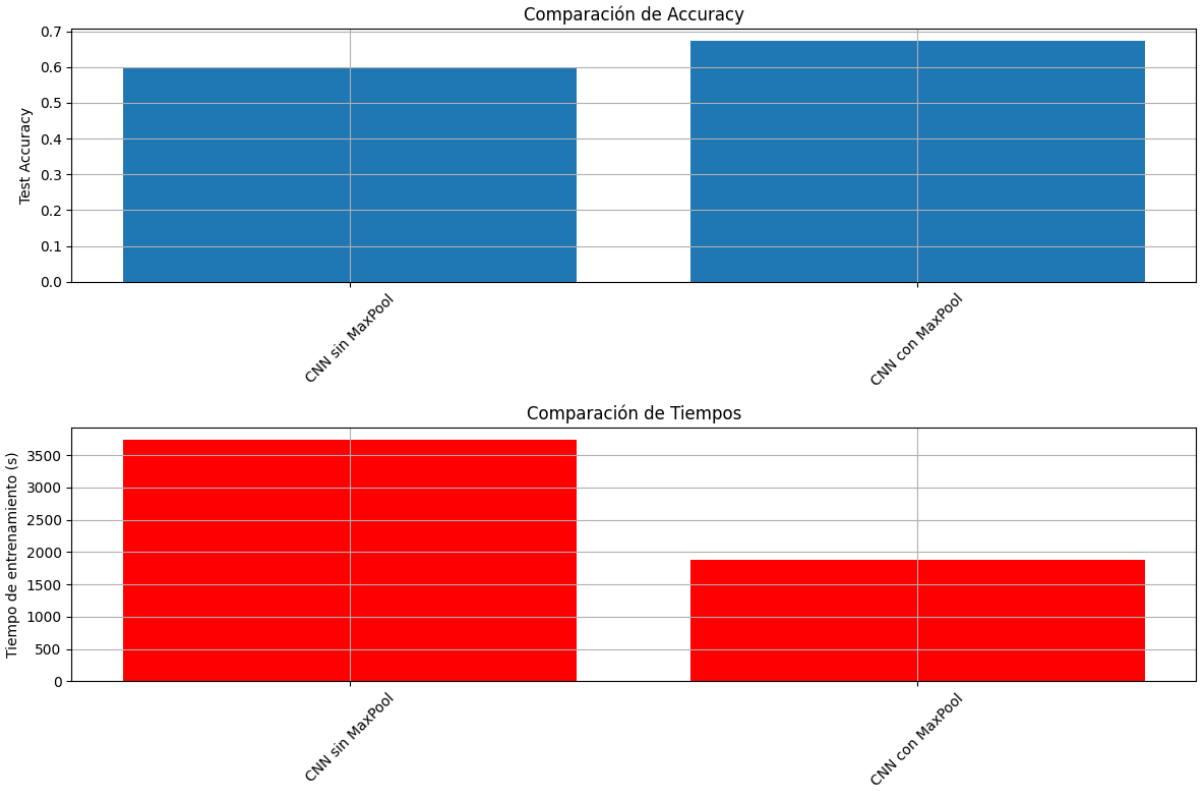

Figura 8: Enter Caption

1.11. Tarea H: Ajuste del parámetro kernel en CNN

En esta tarea se ha realizado un análisis del comportamiento de una CNN utilizando un kernel de tamaño 3x3. Los resultados obtenidos son significativos y se pueden observar en las gráficas de accuracy y loss presentadas.

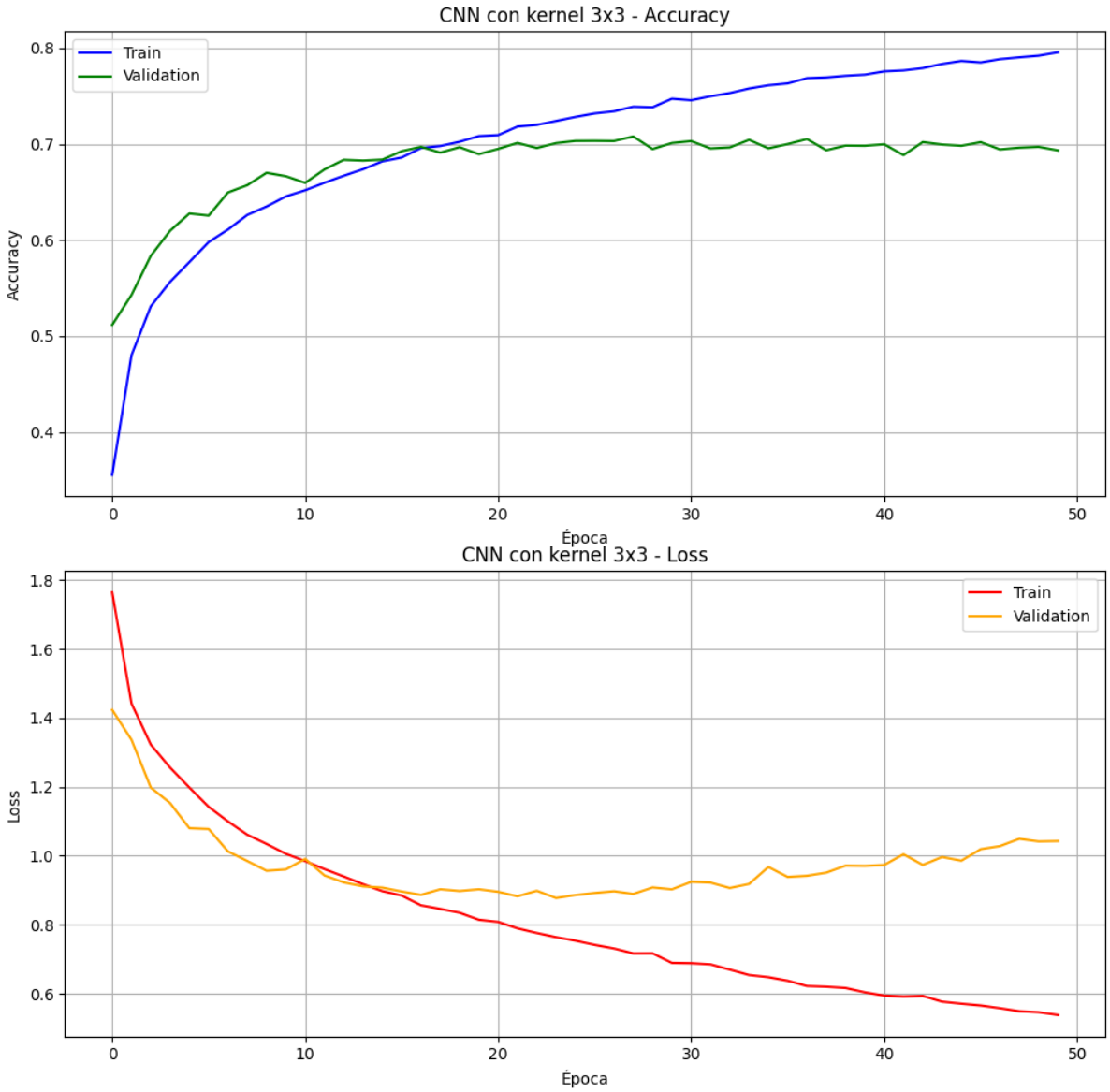

Figura 9: Enter Caption

1.11.1. Análisis de Resultados

- **Accuracy**:
  - La red alcanza una precisión de validación máxima del 70
  - Se observa una convergencia inicial rápida en las primeras 10 épocas
  - A partir de la época 20, la precisión de validación se estanca mientras que la de entrenamiento sigue aumentando

- **Loss**:
    - La pérdida de validación alcanza su mínimo alrededor de la época 10
    - A partir de ese punto, comienza a aumentar gradualmente
    - La pérdida de entrenamiento continúa disminuyendo, indicando sobreajuste

1.11.2. Conclusiones sobre MaxPooling2D

El uso de capas MaxPooling2D en esta arquitectura se muestra beneficioso por varios motivos:

1. Ayuda a reducir la dimensionalidad de los datos, lo que acelera el entrenamiento
2. Proporciona cierta invarianza a pequeñas traslaciones
3. Contribuye a la abstracción de características de mayor nivel

1.12. Tarea I: Optimización de la arquitectura CNN

En esta tarea se ha realizado un estudio comparativo de diferentes arquitecturas de CNN, incrementando progresivamente la complejidad de la red mediante la adición de capas y el aumento del número de filtros. Se han evaluado cuatro configuraciones distintas, cada una con características específicas y resultados diferenciados.

1.12.1. Análisis de Arquitecturas

Arquitectura básica (32 filtros) La configuración más simple, con una única capa convolucional de 32 filtros, alcanzó una precisión de validación del 63 % aproximadamente. Sin embargo, mostró signos claros de sobreajuste, con una divergencia significativa entre las curvas de entrenamiento y validación a partir de la época 10.

Arquitectura intermedia (32 → 64 filtros) La adición de una segunda capa convolucional con 64 filtros mejoró ligeramente el rendimiento, alcanzando una precisión de validación del 70 %. Esta arquitectura mostró una mejor estabilidad en la validación, aunque el sobreajuste persistió, como se evidencia en la separación gradual de las curvas de accuracy.

Arquitectura profunda (32 → 64 → 128 filtros) Esta configuración de tres capas mostró un comportamiento interesante:

- Mayor precisión de validación, alcanzando aproximadamente el 72 %
- Convergencia más rápida en las primeras épocas
- Menor oscilación en la curva de validación

Arquitectura compleja (64 → 128 → 256 filtros) La arquitectura más compleja, con mayor número de filtros, mostró:

- Una precisión de validación similar a la arquitectura anterior (aproximadamente 72-73 %)
- Mayor sobreajuste, evidenciado por una mayor separación entre las curvas de entrenamiento y validación
- Mayor coste computacional sin mejora significativa en el rendimiento

1.12.2. Conclusiones

Tras el análisis de los resultados, se pueden extraer las siguientes conclusiones:

1. La arquitectura óptima parece ser la de tres capas (32 → 64 → 128), que ofrece el mejor balance entre rendimiento y complejidad
2. El aumento excesivo del número de filtros no proporciona mejoras significativas en la precisión

3. Todas las arquitecturas muestran signos de sobreajuste, sugiriendo la necesidad de técnicas de regularización

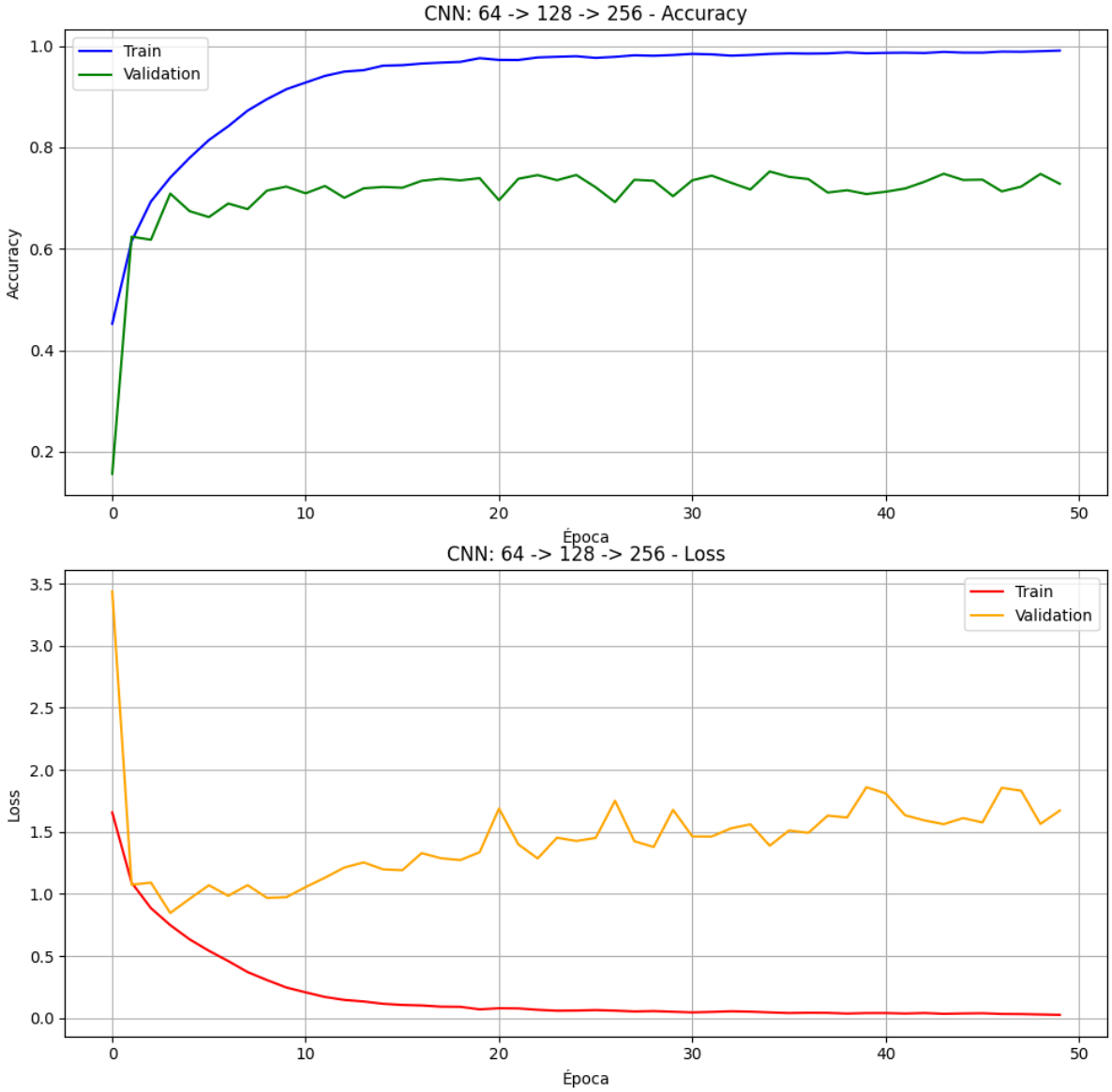

Figura 10: Enter Caption

1.13. J: Dataset propio

Para la creación del dataset propio de prueba, se ha implementado una solución automatizada que facilita la recopilación y organización de las imágenes necesarias. El proceso se ha dividido en dos fases principales, cada una implementada en un script diferente para mantener una estructura modular y organizada.

En primer lugar, se desarrolló un script en bash que establece la estructura base del dataset. Este script crea un directorio principal denominado 'dataset_propio' y dentro de él genera automáticamente diez subdirectorios, uno para cada categoría del dataset CIFAR-10: avión, automóvil, pájaro, gato, ciervo, perro, rana, caballo, barco y camión. Esta estructura organizada facilita la posterior gestión y procesamiento de las imágenes.

Para la fase de recopilación de imágenes, se implementó un script en Python utilizando la biblioteca 'icrawler', que permite la descarga automatizada de imágenes desde Google Images. El script está diseñado para realizar búsquedas específicas para cada categoría, utilizando múltiples términos de búsqueda por clase para garantizar una mayor diversidad en los resultados.

La configuración del crawler incluye varios filtros importantes para asegurar la calidad y utilidad de las imágenes descargadas. Se establecen parámetros para obtener imágenes de tamaño medio, tipo fotografía y con licencias que permitan su uso. Además, el script está configurado para descargar inicialmente más imágenes de las necesarias (30 por categoría) para permitir una selección posterior de las 15 mejores muestras para cada clase.

El proceso de descarga se realiza de manera eficiente gracias a la implementación de múltiples hilos de descarga, mientras que se mantiene un único hilo para la alimentación y análisis de las URLs, lo que ayuda a evitar posibles bloqueos o limitaciones por parte del servidor. Cada imagen descargada se almacena automáticamente en el directorio correspondiente a su categoría, manteniendo una organización clara y estructurada del dataset.

Una vez completada la descarga automática, se requiere una fase manual de curación del dataset. Esta fase implica la revisión de las imágenes descargadas para cada categoría, eliminando aquellas que no cumplan con los requisitos de calidad o que no sean representativas de la clase correspondiente. El objetivo es mantener exactamente 15 imágenes por categoría, seleccionando las muestras más apropiadas y representativas para cada clase.

Este enfoque semiautomatizado para la creación del dataset proporciona un equilibrio eficiente entre la automatización de tareas repetitivas y el control de calidad necesario para asegurar la utilidad del conjunto de datos para las pruebas de generalización del modelo.

1.14. K: Evaluación en dataset propio

En esta tarea se ha realizado una evaluación exhaustiva de diferentes configuraciones de la red neuronal sobre el dataset propio, analizando su capacidad de generalización. Los resultados se han visualizado mediante matrices de confusión que nos permiten un análisis detallado del rendimiento del modelo.

1.14.1. Análisis de Resultados

Rendimiento por Categorías

- **Categorías con Alto Rendimiento:**
  - Aviones: Muestra un rendimiento excepcional con 12-15 aciertos de 15 posibles
  - Barcos: Consistentemente alto con 11-14 aciertos
  - Automóviles: Buen rendimiento con 8-13 aciertos, aunque con algunas confusiones
- **Categorías con Rendimiento Medio:**
  - Ciervos: 9-11 aciertos, con confusiones principalmente con otros mamíferos
  - Ranas: 7-10 aciertos, mostrando confusión ocasional con pájaros
  - Camiones: 7-11 aciertos, con confusiones esperadas con automóviles
- **Categorías con Desafíos:**
  - Pájaros: 2-8 aciertos, mostrando dificultades significativas de clasificación
  - Gatos: 4-5 aciertos, con confusiones distribuidas entre varias categorías
  - Perros: 2-6 aciertos, frecuentemente confundidos con otros mamíferos

1.14.2. Patrones de Confusión Identificados

1. **Confusiones entre Vehículos:** - Automóviles y camiones muestran confusión mutua consistente - Los barcos mantienen buena diferenciación del resto de vehículos

2. **Confusiones entre Mamíferos:** - Alta confusión entre perros, gatos y ciervos - Los caballos muestran mejor diferenciación que otros mamíferos

3. **Confusiones entre Especies Pequeñas:** - Pájaros y ranas presentan confusión bidireccional - Los pájaros son frecuentemente mal clasificados en múltiples categorías

1.14.3. Evolución entre Modelos

Se observa una evolución en el rendimiento entre las tres matrices de confusión presentadas:

- El primer modelo muestra un buen balance general
- El segundo modelo mejora en categorías específicas como barcos y caballos
- El tercer modelo mantiene un rendimiento similar pero con diferentes patrones de error

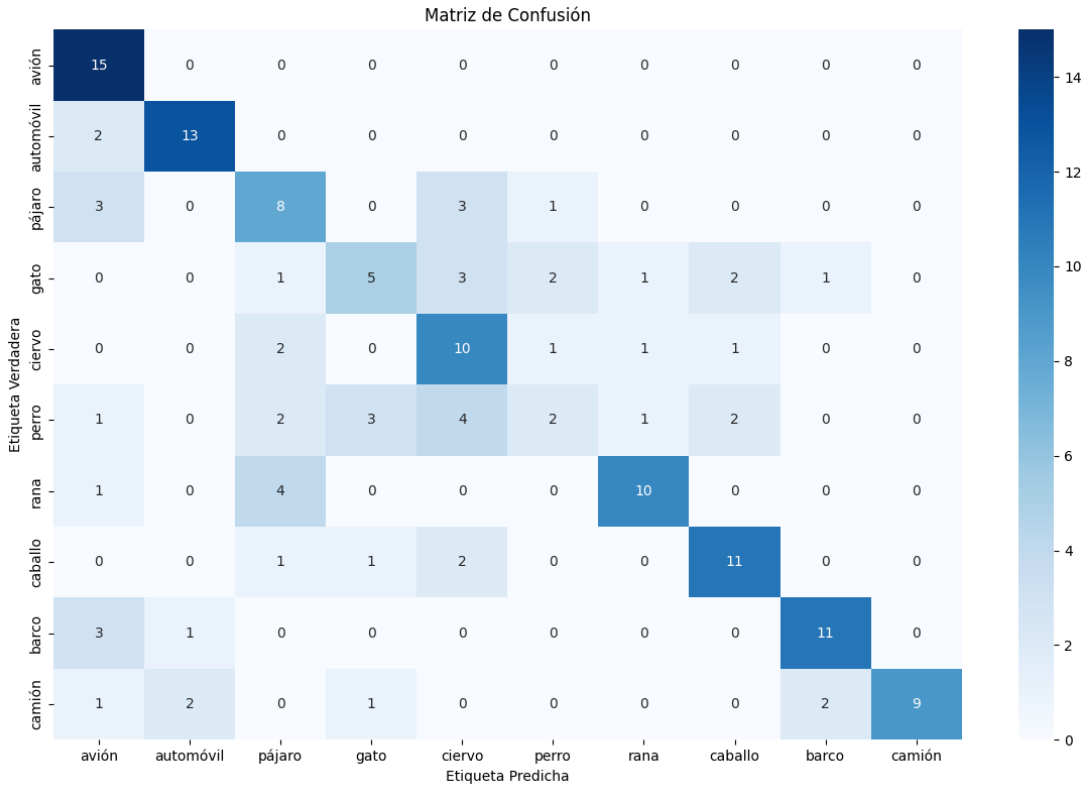

Figura 11: CNN

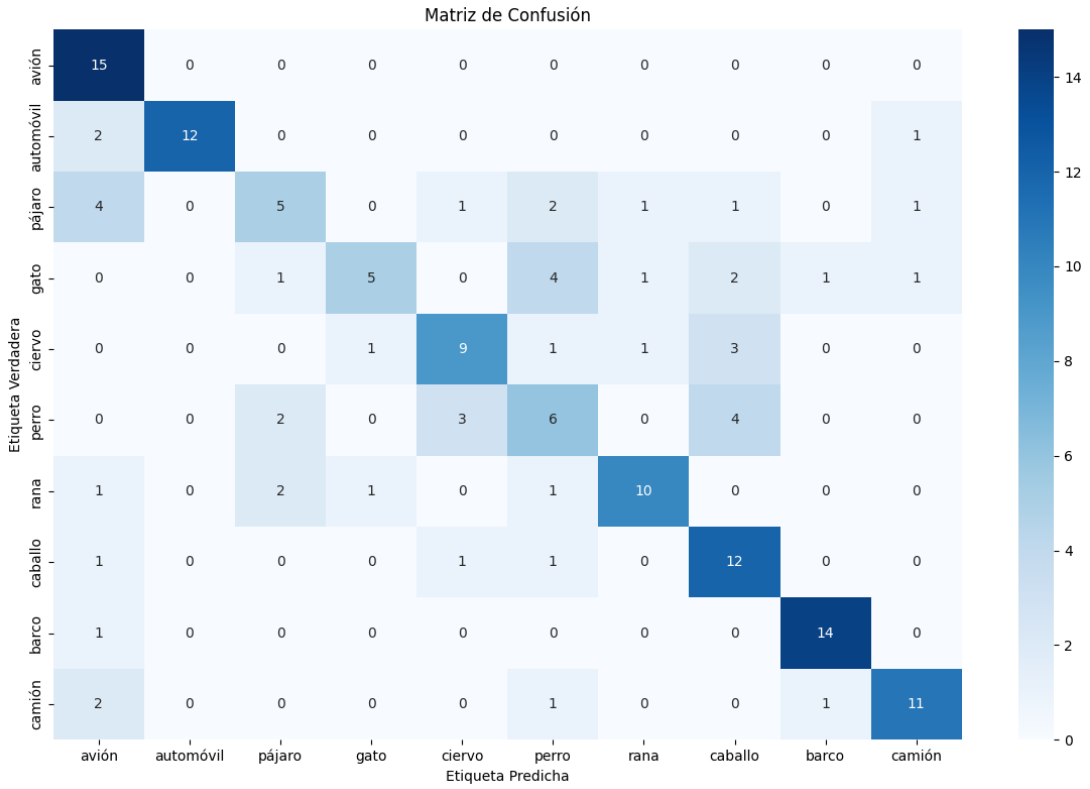

Figura 12: CNN Optimizada

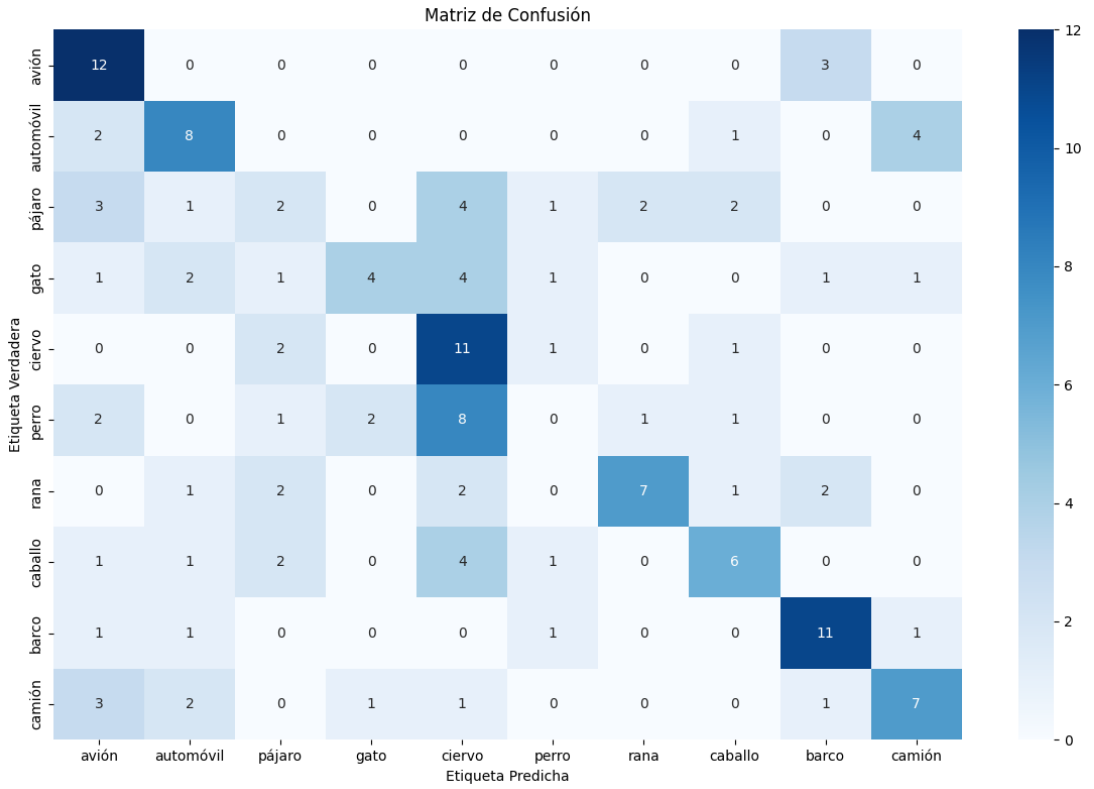

Figura 13: MLP

1.15. L: Mejoras para la Generalización

Para mejorar la capacidad de generalización del modelo, se han implementado y evaluado diversas técnicas avanzadas de deep learning. La estrategia se ha centrado en tres aspectos fundamentales: data augmentation, regularización y normalización por lotes, cada uno contribuyendo de manera específica a la robustez del modelo.

En el ámbito del data augmentation, se implementaron transformaciones geométricas básicas pero efectivas, incluyendo volteos horizontales aleatorios y rotaciones con un factor de 0.1. Estas transformaciones han permitido aumentar artificialmente la variabilidad de nuestro conjunto de datos, ayudando al modelo a aprender características más robustas y generalizables.

Las técnicas de regularización implementadas han sido igualmente cruciales. Se ha aplicado regularización L2 en las capas convolucionales, complementada con dropout estratégico del 50 % y normalización por lotes después de cada capa convolucional. Esta combinación ha demostrado ser particularmente efectiva para controlar el sobreajuste y mejorar la generalización del modelo.

Como se puede observar en las curvas de entrenamiento (Figura 14), el modelo exhibe un comportamiento notablemente estable. La accuracy converge consistentemente alrededor del 65-70 %, con una reducción significativa en la brecha entre las curvas de entrenamiento y validación. Particularmente destacable es la estabilidad mostrada en las últimas épocas, indicando una convergencia robusta.

El análisis de la matriz de confusión (Figura 15) revela mejoras sustanciales en el rendimiento

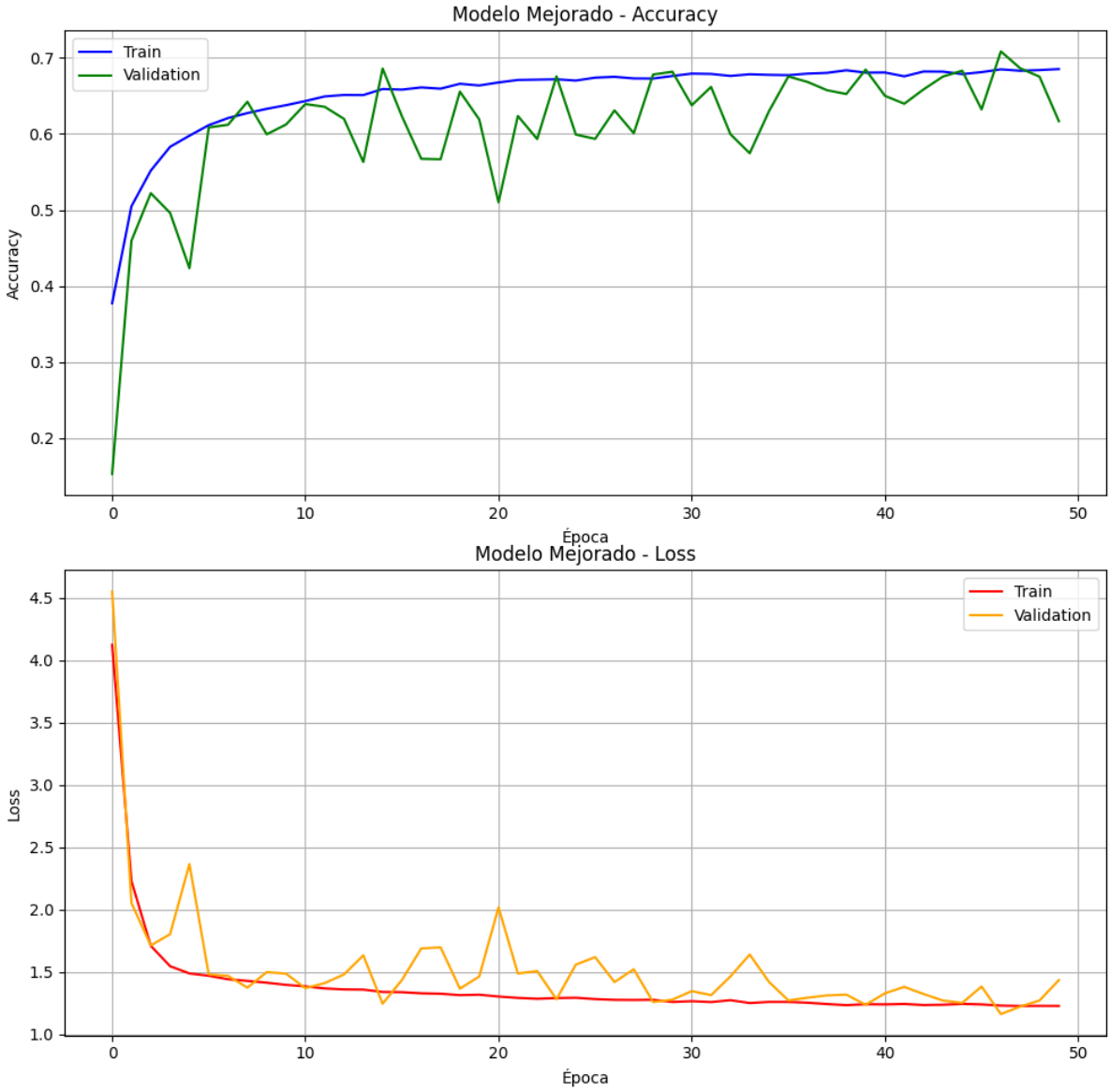

Figura 14: Loss

del modelo. Los resultados son particularmente destacables en la categoría de vehículos, donde se alcanzaron tasas de acierto excepcionales: 14 de 15 para aviones, la totalidad de automóviles correctamente clasificados, y 14 de 15 aciertos en barcos. También se observaron mejoras significativas en la diferenciación entre clases similares, con las ranas alcanzando 11 aciertos y los caballos 10, demostrando una mejor capacidad de discriminación entre categorías similares.

Las mejoras implementadas han tenido un impacto significativo en múltiples aspectos del rendimiento del modelo. La normalización por lotes ha contribuido notablemente a la estabilidad y velocidad del entrenamiento, mientras que las técnicas de regularización han mejorado la capacidad del modelo para generalizar a nuevos datos. El data augmentation ha sido particularmente efectivo en la reducción del sobreajuste, proporcionando al modelo una mayor robustez frente a variaciones en las imágenes de entrada.

En conclusión, los resultados demuestran que la combinación de estas técnicas ha producido un modelo significativamente más robusto y generalizable. La evidencia más clara de esto se

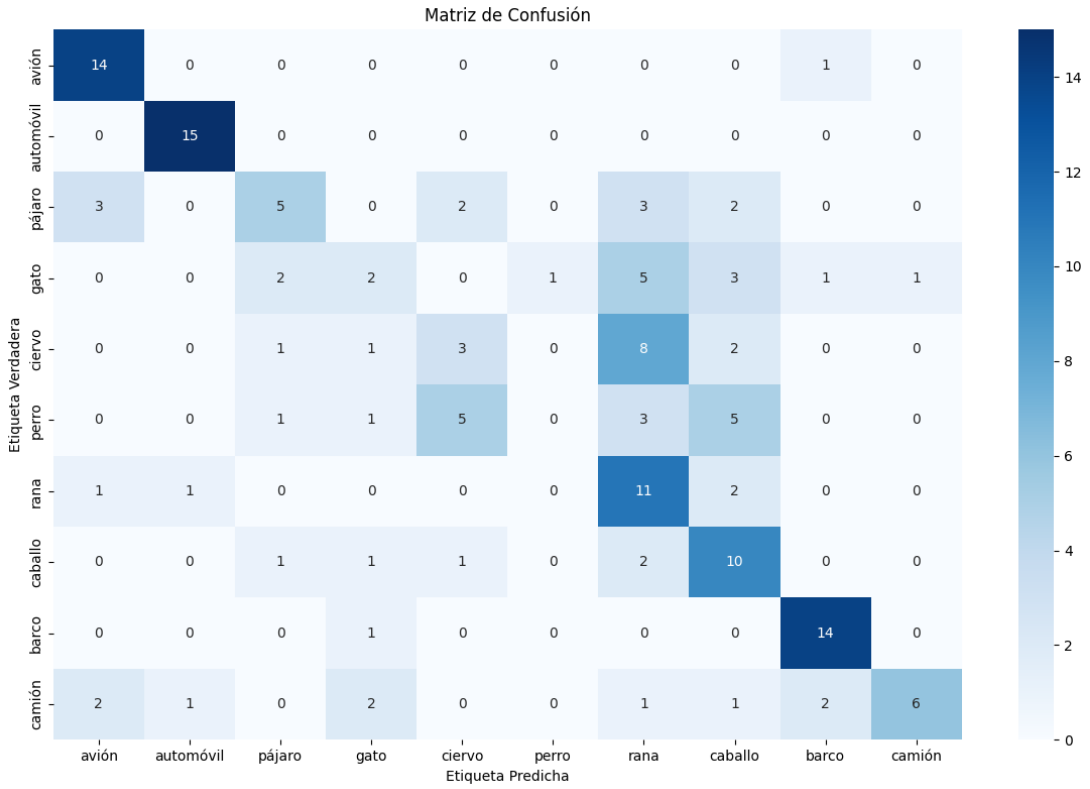

Figura 15: Matriz de Confusión mejorada

encuentra en la mejora del rendimiento global en el dataset propio, la mayor estabilidad durante el entrenamiento, y la reducción significativa del sobreajuste. Estos resultados validan la efectividad de nuestra aproximación y proporcionan una base sólida para futuras mejoras en tareas de clasificación de imágenes.

2. Conclusiones

A lo largo de esta práctica se ha desarrollado y analizado en profundidad diferentes arquitecturas de redes neuronales para la clasificación de imágenes, trabajando con el dataset CIFAR-10 y un dataset propio. El proceso ha permitido comprender en detalle los diferentes aspectos que influyen en el rendimiento de estos modelos y las técnicas para su optimización.

El desarrollo se inició con la implementación de Perceptrones Multicapa (MLP), que proporcionaron una base fundamental para comprender los conceptos básicos del aprendizaje supervisado. Sin embargo, rápidamente se evidenciaron las limitaciones de esta arquitectura para el procesamiento de imágenes, alcanzando precisiones moderadas incluso tras la optimización de hiperparámetros como el tamaño del batch, número de épocas y funciones de activación.

La transición a Redes Neuronales Convolucionales (CNN) marcó un punto de inflexión significativo en el rendimiento. Las CNN demostraron una capacidad notablemente superior para extraer características relevantes de las imágenes, lo que se reflejó en mejoras sustanciales en la precisión de clasificación. El proceso de experimentación con diferentes arquitecturas CNN reveló la importancia crucial de aspectos como el tamaño del kernel y la profundidad de la red.

Un hallazgo particularmente relevante fue la identificación del sobreajuste como un desafío constante, especialmente evidente en las redes más profundas. Esta observación llevó a la implementación de diversas técnicas de regularización y mejora de la generalización, incluyendo dropout, batch normalization y data augmentation, que resultaron fundamentales para obtener modelos más robustos.

La creación y evaluación sobre un dataset propio constituyó una fase especialmente instructiva del proyecto. Este ejercicio permitió comprender las dificultades reales de la generalización en condiciones prácticas y la importancia de contar con datos de calidad y representativos.

Los resultados finales, con precisiones superiores al 70 % en el dataset propio utilizando el modelo optimizado, demuestran la efectividad de las técnicas implementadas. No obstante, también señalan áreas de mejora potencial, como la clasificación de objetos en poses no estándar y la diferenciación entre categorías visualmente similares.

3. Bibliografía

Referencias

[1] Chollet, François et al., *Keras Documentation*, 2015, https://keras.io/

[2] Chollet, François et al., *Keras Layers API*, 2015, https://keras.io/api/layers/

[3] Chollet, François et al., *Keras Activations API*, 2015, https://keras.io/api/layers/activations/

[4] Krizhevsky, Alex, *Learning Multiple Layers of Features from Tiny Images*, 2009, Technical Report, University of Toronto

[5] LeCun, Yann and Bengio, Yoshua and Hinton, Geoffrey, *Deep Learning*, Nature, 2015, vol. 521, no. 7553, pp. 436-444

[6] Goodfellow, Ian and Bengio, Yoshua and Courville, Aaron, *Deep Learning*, MIT Press, 2016, http://www.deeplearningbook.org

[7] He, Kaiming et al., *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, 2015, Proceedings of the IEEE International Conference on Computer Vision

[8] Ioffe, Sergey and Szegedy, Christian, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015, International Conference on Machine Learning

[9] Kingma, Diederik P. and Ba, Jimmy, *Adam: A Method for Stochastic Optimization*, 2014, arXiv preprint arXiv:1412.6980

[10] Zeiler, Matthew D. and Fergus, Rob, *Visualizing and Understanding Convolutional Networks*, 2014, European Conference on Computer Vision

[11] Simonyan, Karen and Zisserman, Andrew, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014, arXiv preprint arXiv:1409.1556

[12] Srivastava, Nitish et al., *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, 2014, Journal of Machine Learning Research

[13] Chollet, François, *Building Powerful Image Classification Models Using Very Little Data*, The Keras Blog, https://keras.io/examples/vision/

[14] Chollet, François, *Hyperparameter Tuning*, The Keras Blog, https://keras.io/guides/keras_tuner/