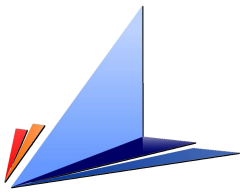

- **Universidad de Alicante**

Ingeniería Informática

Sistemas Inteligentes

Visión artificial y aprendizaje

Autor:
Jaime Hernández Delgado (jhd3)

Cusro 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:

- Comprender en profundidad el funcionamiento de las redes neuronales artificiales como técnicas de aprendizaje supervisado, con especial énfasis en:
  - Perceptrones multicapa (MLP)
  - Redes neuronales convolucionales (CNN)
- Adquirir conocimientos fundamentales sobre el procesamiento de imágenes:
  - Entender el concepto de imagen digital y píxel
  - Comprender cómo las técnicas de aprendizaje supervisado pueden aplicarse para la clasificación de imágenes
- Dominar el algoritmo de retropropagación (backpropagation) y sus componentes esenciales:
  - Funciones de activación
  - Optimizadores
  - Funciones de pérdida
  - Métricas de evaluación
- Desarrollar habilidades prácticas en el uso de Keras:
  - Implementación eficiente de redes neuronales
  - Configuración y ajuste de hiperparámetros
  - Evaluación de modelos
- Aprender a optimizar el rendimiento de redes neuronales mediante:
  - Ajuste del número de épocas de entrenamiento
  - Configuración del tamaño de batch
  - Selección de funciones de activación apropiadas
  - Diseño de arquitecturas de red efectivas
- Desarrollar competencias en visualización y análisis de resultados:
  - Utilización de Matplotlib para la generación de gráficas
  - Interpretación de métricas de rendimiento
  - Análisis de matrices de confusión
- Adquirir experiencia en la automatización de procesos:
  - Desarrollo de scripts para pruebas automáticas
  - Generación automatizada de resultados
  - Creación eficiente de contenido para el informe

A través de estos objetivos, se busca desarrollar una comprensión integral de las redes neuronales y su aplicación práctica en problemas de visión artificial, desde la implementación básica hasta la optimización avanzada de modelos.

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. Bibliotecaa 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
- **Seaborn**: Biblioteca de visualización estadística basada en Matplotlib:
  - Generación de matrices de confusión
  - Mejora visual de las gráficas estadísticas
- **Scikit-learn**: Biblioteca para aprendizaje automático:
  - Cálculo de métricas de evaluación
  - Funciones de utilidad para el procesamiento de datos

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

Todas estas herramientas y tecnologías han sido seleccionadas por su madurez, amplia documentación disponible y gran comunidad de usuarios, lo que facilita la resolución de problemas y el acceso a recursos de aprendizaje. Además, representan el estado del arte en el desarrollo de aplicaciones de aprendizaje automático y son ampliamente utilizadas tanto en entornos académicos como profesionales.

2. Desarrollo

2.1. 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.

2.1.1. 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

2.1.2. 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

2.1.3. 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).

2.1.4. 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]

2.1.5. 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.

2.1.6. 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.

2.2. Tarea A: Implementación básica de MLP

2.2.1. 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

Referencias

[1] Keras documentation, `https://keras.io/`

A. Anexos