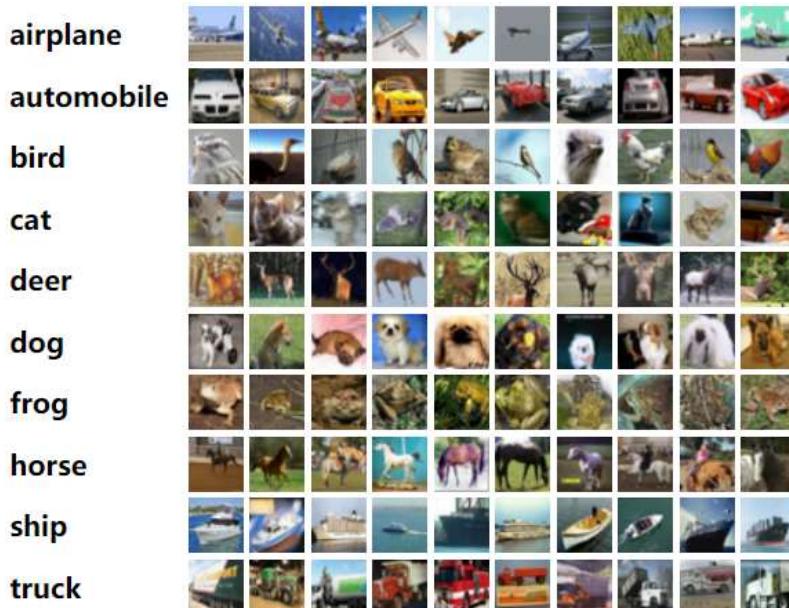


ACTIVIDAD 2: REDES NEURONALES CONVOLUCIONALES

En esta actividad, vamos a trabajar con Convolutional Neural Networks para resolver un problema de clasificación de imágenes. En particular, vamos a clasificar diez clases que incluyen fundamentalmente animales y vehículos.

Como las CNN profundas son un tipo de modelo bastante avanzado y computacionalmente costoso, se recomienda hacer la práctica en Google Colaboratory con soporte para GPUs. En [este enlace](#) se explica cómo activar un entorno con GPUs. *Nota: para leer las imágenes y estandarizarlas al mismo tamaño se usa la librería opencv. Esta librería está ya instalada en el entorno de Colab, pero si trabajáis de manera local tendréis que instalarla.*



El dataset a utilizar consiste en 60000 imágenes a color de 10 clases de animales y vehículos. El dataset en cuestión se denomina [CIFAR-10](#) y es más complejo que el dataset MNIST que hemos utilizado en la actividad 1. Aunque tiene las mismas clases (10), los animales y vehículos pueden aparecer en distintas poses, en distintas posiciones de la imagen o con otros animales/ vehículos en pantalla (si bien el elemento a clasificar siempre aparece en la posición predominante).

Carga de los datos

```
In [1]: import numpy as np
import keras
import matplotlib.pyplot as plt
import pandas as pd
import keras.datasets.cifar10 as cifar10

from tensorflow import keras
from keras.utils import to_categorical
```

2024-11-24 23:57:44.143184: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-11-24 23:57:44.143725: I external/local_tsl/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
2024-11-24 23:57:44.145884: I external/local_tsl/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
2024-11-24 23:57:44.151623: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:479] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
2024-11-24 23:57:44.163494: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:10575] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
2024-11-24 23:57:44.163520: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1442] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
2024-11-24 23:57:44.171346: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-11-24 23:57:44.650359: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT

```
In [ ]: import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
```

```
    horizontal_flip=True,
    zoom_range=0.2
)
```

```
In [3]: # Esta variable contiene un mapeo de número de clase a elemento (animal o vehículo).
# La incluimos para ayudarte con la identificación de clases. De ti depende
# siquieres utilizar esta variable o no
MAP_ELEMENTS = {
    0: 'avion', 1: 'coche', 2: 'ave',
    3: 'gato', 4: 'ciervo', 5: 'perro', 6: 'rana',
    7: 'caballo', 8: 'barco', 9: 'camion'
}
```

```
In [4]: from sklearn.model_selection import train_test_split

# Primero, definimos los datos de entrenamiento, validación y prueba
(X, Y), (x_test, y_test) = cifar10.load_data()
# Dividir en entrenamiento y validación (80%-20%)
x_train, x_valid, y_train, y_valid = train_test_split(
    X, Y, test_size=0.2, random_state=42
)

print("Dimensiones del nuevo conjunto de entrenamiento:")
print(f"x_train_combined: {x_train.shape}")
print(f"y_train_combined: {y_train.shape}")

Dimensiones del nuevo conjunto de entrenamiento:
x_train_combined: (40000, 32, 32, 3)
y_train_combined: (40000, 1)
```

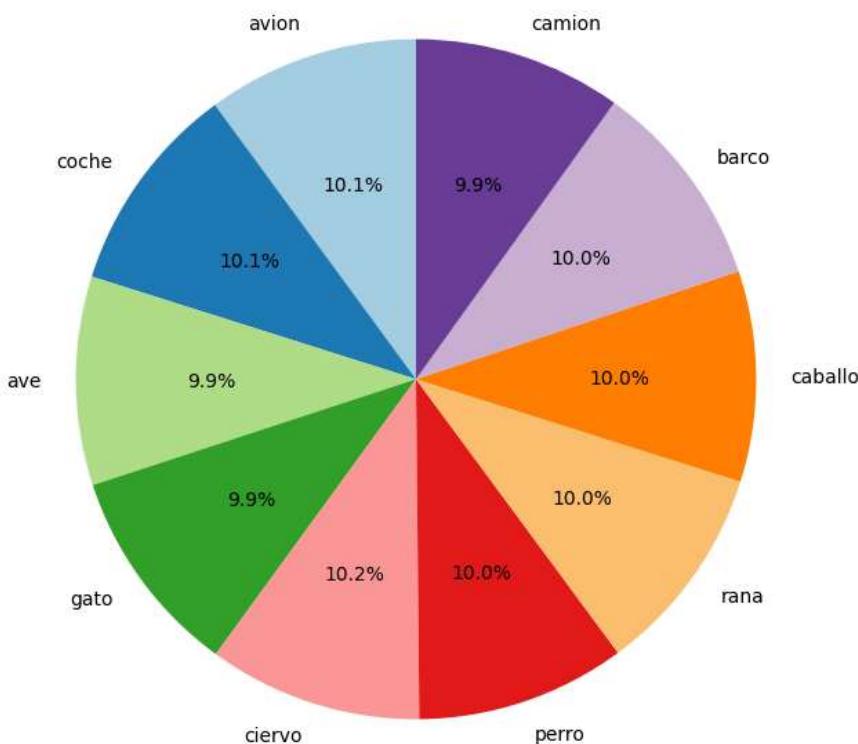
```
In [5]: # Normalizamos como de costumbre
x_train = x_train / 255.
x_valid = x_valid / 255.
x_test = x_test / 255.
```

```
In [6]: clases, conteo = np.unique(y_train, return_counts=True)

# Obtener etiquetas de clase a partir de MAP_ELEMENTS
etiquetas_clases = [MAP_ELEMENTS[clase] for clase in clases]

# Crear gráfico de pastel
plt.figure(figsize=(10, 8))
plt.pie(conteo, labels=etiquetas_clases, autopct='%1.1f%%', startangle=90, colors=plt.cm.Paired.colors)
plt.title("Proporción de Clases en el Conjunto de Entrenamiento")
plt.show()
```

Proporción de Clases en el Conjunto de Entrenamiento



- Las clases estan balanceadas entorno al 10%.

```
In [7]: # Función auxiliar para convertir las etiquetas a codificación one-hot
def convert_to_one_hot(labels, num_classes):
    return np.squeeze(np.array([to_categorical(label, num_classes) for label in labels]))

# Convertimos las etiquetas de entrenamiento, validación y prueba
num_classes = 10
y_train_one_hot = convert_to_one_hot(y_train, num_classes)
y_valid_one_hot = convert_to_one_hot(y_valid, num_classes)
y_test_one_hot = convert_to_one_hot(y_test, num_classes)

# Verificamos las conversiones
print(y_train_one_hot.shape)
print(y_valid_one_hot.shape)
print(y_test_one_hot.shape)

(40000, 10)
(10000, 10)
(10000, 10)
```

Ejercicio

Utilizando Convolutional Neural Networks con Keras, entrenar un clasificador que sea capaz de reconocer una imagen de las incluidas en CIFAR-10 con la mayor accuracy posible. Redactar un informe analizando varias de las alternativas probadas y los resultados obtenidos.

A continuación se detallan una serie de aspectos orientativos que podrían ser analizados en vuestro informe (no es necesario tratar todos ellos ni mucho menos, esto son ideas orientativas de aspectos que podéis explorar):

- Análisis de los datos a utilizar.
- Análisis de resultados, obtención de métricas de *precision* y *recall* por clase y análisis de qué clases obtienen mejores o peores resultados.
- Análisis visual de los errores de la red. ¿Qué tipo de imágenes dan más problemas a nuestro modelo?
- Comparación de modelos CNNs con un modelo de Fully Connected para este problema.
- Utilización de distintas arquitecturas CNNs, comentando aspectos como su profundidad, hiperparámetros utilizados, optimizador, uso de técnicas de regularización, *batch normalization*, etc.
- [algo más difícil] Utilización de *data augmentation*. Esto puede conseguirse con la clase `ImageDataGenerator` de Keras.

Notas:

- Te recomendamos mantener los conjuntos de entrenamiento, test y prueba que se crean en el Notebook. No obstante, si crees que modificando tales conjuntos puedes lograr mejores resultados (o que puedes lograr los mismos resultados con menos datos, lo cual es también un logro), eres libre de hacerlo.
- No es necesario mostrar en el notebook las trazas de entrenamiento de todos los modelos entrenados, si bien una buena idea sería guardar gráficas de esos entrenamientos para el análisis. Sin embargo, **se debe mostrar el entrenamiento completo del mejor modelo obtenido y la evaluación de los datos de test con este modelo.**

```
In [ ]: import tensorflow as tf
from datetime import datetime
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dropout, Flatten, Dense, BatchNormalization
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.initializers import HeUniform, GlorotUniform
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

modelo = Sequential([
    Input((32, 32, 3)),
    # Primer bloque convolucional: comenzamos por 64 filtros, mantenemos el padding para no perder información de los bordes
    Conv2D(64, (3, 3), activation='relu', padding='same', kernel_initializer=HeUniform()),
    BatchNormalization(),
    Dropout(0.2),
    Conv2D(64, (3, 3), activation='relu', padding='same', kernel_initializer=HeUniform()),
    BatchNormalization(),
    Dropout(0.2),
    MaxPooling2D((2, 2)),

    # Segundo bloque convolucional:
    Conv2D(128, (3, 3), activation='relu', padding='same', kernel_initializer=HeUniform()),
    BatchNormalization(),
    Conv2D(128, (3, 3), activation='relu', padding='same', kernel_initializer=HeUniform()),
    BatchNormalization(),
    Dropout(0.3),
    MaxPooling2D((2, 2)),
```

```

# Tercer bloque convolucional:
Conv2D(256, (3, 3), activation='relu', padding='same',kernel_initializer=HeUniform()),
BatchNormalization(),
Conv2D(256, (3, 3), activation='relu', padding='same',kernel_initializer=HeUniform()),
BatchNormalization(),
Dropout(0.3),
MaxPooling2D((2, 2)),

# Capa final
Flatten(),
Dropout(0.3),
Dense(1024, activation='relu',kernel_initializer=GlorotUniform()),
# Salida
Dense(num_classes, activation='softmax',kernel_initializer=GlorotUniform())
])

reducir_aprendizaje = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=1e-6,
    verbose=1
)

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

sgd = SGD(learning_rate=0.05, momentum=0.9)

adam = Adam(learning_rate=1e-4)

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

history = modelo.fit(datagen.flow(x_train, y_train_one_hot), validation_data=(x_valid, y_valid_one_hot),epochs=40, callbacks=[reducir_aprendizaje, early_stopping])

loss, accuracy = modelo.evaluate(x_test, y_test_one_hot)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")

```

Epoch 1/40

```

/home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages/keras/src/trainers/data_adapters/pydataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()

```

1250/1250 ━━━━━━━━ 118s 93ms/step - accuracy: 0.2955 - loss: 2.3003 - val_accuracy: 0.4650 - val_loss: 1.4992 - 1
earning_rate: 1.0000e-04
Epoch 2/40
1250/1250 ━━━━━━━━ 116s 93ms/step - accuracy: 0.4461 - loss: 1.5262 - val_accuracy: 0.5117 - val_loss: 1.3849 - 1
earning_rate: 1.0000e-04
Epoch 3/40
1250/1250 ━━━━━━━━ 118s 94ms/step - accuracy: 0.5090 - loss: 1.3570 - val_accuracy: 0.5867 - val_loss: 1.1738 - 1
earning_rate: 1.0000e-04
Epoch 4/40
1250/1250 ━━━━━━━━ 117s 94ms/step - accuracy: 0.5576 - loss: 1.2384 - val_accuracy: 0.6021 - val_loss: 1.1249 - 1
earning_rate: 1.0000e-04
Epoch 5/40
1250/1250 ━━━━━━━━ 118s 95ms/step - accuracy: 0.5943 - loss: 1.1243 - val_accuracy: 0.6499 - val_loss: 1.0170 - 1
earning_rate: 1.0000e-04
Epoch 6/40
1250/1250 ━━━━━━━━ 117s 94ms/step - accuracy: 0.6250 - loss: 1.0477 - val_accuracy: 0.6790 - val_loss: 0.9169 - 1
earning_rate: 1.0000e-04
Epoch 7/40
1250/1250 ━━━━━━━━ 120s 96ms/step - accuracy: 0.6534 - loss: 0.9712 - val_accuracy: 0.7030 - val_loss: 0.8693 - 1
earning_rate: 1.0000e-04
Epoch 8/40
1250/1250 ━━━━━━━━ 118s 95ms/step - accuracy: 0.6743 - loss: 0.9195 - val_accuracy: 0.7081 - val_loss: 0.8438 - 1
earning_rate: 1.0000e-04
Epoch 9/40
1250/1250 ━━━━━━━━ 120s 96ms/step - accuracy: 0.6949 - loss: 0.8650 - val_accuracy: 0.7276 - val_loss: 0.8076 - 1
earning_rate: 1.0000e-04
Epoch 10/40
1250/1250 ━━━━━━━━ 117s 94ms/step - accuracy: 0.7098 - loss: 0.8208 - val_accuracy: 0.7234 - val_loss: 0.8386 - 1
earning_rate: 1.0000e-04
Epoch 11/40
1250/1250 ━━━━━━━━ 119s 95ms/step - accuracy: 0.7241 - loss: 0.7788 - val_accuracy: 0.7531 - val_loss: 0.7754 - 1
earning_rate: 1.0000e-04
Epoch 12/40
1250/1250 ━━━━━━━━ 116s 93ms/step - accuracy: 0.7341 - loss: 0.7476 - val_accuracy: 0.7651 - val_loss: 0.7191 - 1
earning_rate: 1.0000e-04
Epoch 13/40
1250/1250 ━━━━━━━━ 117s 93ms/step - accuracy: 0.7468 - loss: 0.7196 - val_accuracy: 0.7616 - val_loss: 0.7465 - 1
earning_rate: 1.0000e-04
Epoch 14/40
1250/1250 ━━━━━━━━ 116s 92ms/step - accuracy: 0.7571 - loss: 0.6888 - val_accuracy: 0.7717 - val_loss: 0.7250 - 1
earning_rate: 1.0000e-04
Epoch 15/40
1250/1250 ━━━━━━━━ 117s 94ms/step - accuracy: 0.7655 - loss: 0.6674 - val_accuracy: 0.7901 - val_loss: 0.6737 - 1
earning_rate: 1.0000e-04
Epoch 16/40
1250/1250 ━━━━━━━━ 115s 92ms/step - accuracy: 0.7716 - loss: 0.6460 - val_accuracy: 0.7760 - val_loss: 0.7392 - 1
earning_rate: 1.0000e-04
Epoch 17/40
1250/1250 ━━━━━━━━ 118s 95ms/step - accuracy: 0.7802 - loss: 0.6205 - val_accuracy: 0.7926 - val_loss: 0.6846 - 1
earning_rate: 1.0000e-04
Epoch 18/40
1250/1250 ━━━━━━━━ 139s 92ms/step - accuracy: 0.7897 - loss: 0.5959 - val_accuracy: 0.7987 - val_loss: 0.6557 - 1
earning_rate: 1.0000e-04
Epoch 19/40
1250/1250 ━━━━━━━━ 115s 92ms/step - accuracy: 0.8001 - loss: 0.5685 - val_accuracy: 0.7937 - val_loss: 0.6834 - 1
earning_rate: 1.0000e-04
Epoch 20/40
1250/1250 ━━━━━━━━ 117s 93ms/step - accuracy: 0.7996 - loss: 0.5776 - val_accuracy: 0.8149 - val_loss: 0.6173 - 1
earning_rate: 1.0000e-04
Epoch 21/40
1250/1250 ━━━━━━━━ 116s 92ms/step - accuracy: 0.8068 - loss: 0.5443 - val_accuracy: 0.8190 - val_loss: 0.6164 - 1
earning_rate: 1.0000e-04
Epoch 22/40
1250/1250 ━━━━━━━━ 117s 94ms/step - accuracy: 0.8147 - loss: 0.5285 - val_accuracy: 0.8254 - val_loss: 0.5831 - 1
earning_rate: 1.0000e-04
Epoch 23/40
1250/1250 ━━━━━━━━ 116s 93ms/step - accuracy: 0.8186 - loss: 0.5130 - val_accuracy: 0.8125 - val_loss: 0.6406 - 1
earning_rate: 1.0000e-04
Epoch 24/40
1250/1250 ━━━━━━━━ 116s 93ms/step - accuracy: 0.8247 - loss: 0.5010 - val_accuracy: 0.8126 - val_loss: 0.6413 - 1
earning_rate: 1.0000e-04
Epoch 25/40
1250/1250 ━━━━━━━━ 119s 95ms/step - accuracy: 0.8277 - loss: 0.4894 - val_accuracy: 0.8287 - val_loss: 0.5886 - 1
earning_rate: 1.0000e-04
Epoch 26/40
1250/1250 ━━━━━━━━ 117s 93ms/step - accuracy: 0.8305 - loss: 0.4754 - val_accuracy: 0.8241 - val_loss: 0.6247 - 1
earning_rate: 1.0000e-04
Epoch 27/40
1250/1250 ━━━━━━━━ 0s 89ms/step - accuracy: 0.8349 - loss: 0.4647
Epoch 27: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
1250/1250 ━━━━━━━━ 117s 94ms/step - accuracy: 0.8349 - loss: 0.4647 - val_accuracy: 0.8131 - val_loss: 0.6768 - 1
earning_rate: 1.0000e-04
Epoch 28/40
1250/1250 ━━━━━━━━ 116s 93ms/step - accuracy: 0.8503 - loss: 0.4300 - val_accuracy: 0.8439 - val_loss: 0.5423 - 1
earning_rate: 5.0000e-05
Epoch 29/40
1250/1250 ━━━━━━━━ 118s 95ms/step - accuracy: 0.8523 - loss: 0.4179 - val_accuracy: 0.8414 - val_loss: 0.5632 - 1

```

earning_rate: 5.0000e-05
Epoch 30/40
1250/1250  117s 94ms/step - accuracy: 0.8604 - loss: 0.3980 - val_accuracy: 0.8368 - val_loss: 0.5863 - 1
earning_rate: 5.0000e-05
Epoch 31/40
1250/1250  117s 94ms/step - accuracy: 0.8603 - loss: 0.4003 - val_accuracy: 0.8415 - val_loss: 0.5534 - 1
earning_rate: 5.0000e-05
Epoch 32/40
1250/1250  116s 93ms/step - accuracy: 0.8665 - loss: 0.3789 - val_accuracy: 0.8357 - val_loss: 0.5986 - 1
earning_rate: 5.0000e-05
Epoch 33/40
1250/1250  0s 90ms/step - accuracy: 0.8685 - loss: 0.3718
Epoch 33: ReduceLROnPlateau reducing learning rate to 2.49999936844688e-05.
1250/1250  118s 95ms/step - accuracy: 0.8685 - loss: 0.3718 - val_accuracy: 0.8377 - val_loss: 0.5842 - 1
earning_rate: 5.0000e-05
Epoch 34/40
1250/1250  116s 93ms/step - accuracy: 0.8726 - loss: 0.3625 - val_accuracy: 0.8443 - val_loss: 0.5623 - 1
earning_rate: 2.5000e-05
Epoch 35/40
1250/1250  117s 94ms/step - accuracy: 0.8712 - loss: 0.3542 - val_accuracy: 0.8485 - val_loss: 0.5521 - 1
earning_rate: 2.5000e-05
Epoch 36/40
1250/1250  116s 93ms/step - accuracy: 0.8781 - loss: 0.3444 - val_accuracy: 0.8502 - val_loss: 0.5452 - 1
earning_rate: 2.5000e-05
Epoch 37/40
1250/1250  117s 94ms/step - accuracy: 0.8754 - loss: 0.3503 - val_accuracy: 0.8435 - val_loss: 0.5902 - 1
earning_rate: 2.5000e-05
Epoch 38/40
1250/1250  0s 88ms/step - accuracy: 0.8786 - loss: 0.3374
Epoch 38: ReduceLROnPlateau reducing learning rate to 1.24999968422344e-05.
1250/1250  116s 92ms/step - accuracy: 0.8787 - loss: 0.3374 - val_accuracy: 0.8417 - val_loss: 0.5864 - 1
earning_rate: 2.5000e-05
313/313  6s 19ms/step - accuracy: 0.8348 - loss: 0.5748
Test Loss: 0.5693228244781494, Test Accuracy: 0.8343999981880188

```

```

In [9]: numero_conv2d = sum(1 for layer in modelo.layers if isinstance(layer, Conv2D))
timestamp = datetime.now().strftime("%Y%m%d-%H%M%S")
nombre_modelo = f"{timestamp}-{numero_conv2d}_Conv2D_model.keras"

modelo.save(nombre_modelo)
print(f"Modelo guardado en el directorio: {nombre_modelo}")

```

Modelo guardado en el directorio: 20241125-011226-6_Conv2D_model.keras

```

In [10]: import plotly.graph_objects as go
import numpy as np
import pandas as pd

def plot_history(history, model_name):
    # Extraer la información del objeto history
    acc = history.history['accuracy']
    val_acc = history.history.get('val_accuracy')
    loss = history.history['loss']
    val_loss = history.history.get('val_loss')

    # Definir las épocas en base a la Longitud de los datos
    epochs = np.arange(1, len(acc) + 1)

    # Calcular diferencias absolutas entre entrenamiento y validación
    acc_diff = np.abs(np.array(acc) - np.array(val_acc)) if val_acc is not None else None
    loss_diff = np.abs(np.array(loss) - np.array(val_loss)) if val_loss is not None else None

    # Umbral para detectar sobreajuste
    acc_threshold = 0.05 # Diferencia del 5% en precisión
    loss_threshold = 0.1 # Diferencia del 10% en pérdida

    # Detectar sobreajuste en precisión y pérdida
    acc_overfit = (acc_diff > acc_threshold) if acc_diff is not None else None
    loss_overfit = (loss_diff > loss_threshold) if loss_diff is not None else None

    # Función para encontrar intersecciones
    def encontrar_intersecciones(x, y1, y2):
        intersecciones_x = []
        intersecciones_y = []
        for i in range(1, len(x)):
            if (y1[i-1] - y2[i-1]) * (y1[i] - y2[i]) <= 0: # Cambio de signo
                intersecciones_x.append(x[i])
                intersecciones_y.append((y1[i] + y2[i]) / 2)
        return intersecciones_x, intersecciones_y

    # Encontrar intersecciones entre las curvas de accuracy
    acc_inter_x, acc_inter_y = encontrar_intersecciones(epochs, acc, val_acc)

    # Encontrar intersecciones entre las curvas de loss
    loss_inter_x, loss_inter_y = encontrar_intersecciones(epochs, loss, val_loss)

    # --- Primer gráfico: curvas de exactitud y pérdida ---
    fig_curvas = go.Figure()

```

```

# Agregar Líneas de accuracy
fig_curvas.add_trace(go.Scatter(x=epochs, y=acc, mode='lines', name='Exactitud de Entrenamiento', line=dict(color='blue'))
fig_curvas.add_trace(go.Scatter(x=epochs, y=val_acc, mode='lines', name='Exactitud de Validación', line=dict(color='green'))

# Agregar intersecciones de accuracy
fig_curvas.add_trace(go.Scatter(x=acc_inter_x, y=acc_inter_y, mode='markers', name='Intersección de Exactitud', marker=d

# Agregar Líneas de pérdida
fig_curvas.add_trace(go.Scatter(x=epochs, y=loss, mode='lines', name='Pérdida de Entrenamiento', line=dict(color='red'))
fig_curvas.add_trace(go.Scatter(x=epochs, y=val_loss, mode='lines', name='Pérdida de Validación', line=dict(color='orange'))

# Agregar intersecciones de loss
fig_curvas.add_trace(go.Scatter(x=loss_inter_x, y=loss_inter_y, mode='markers', name='Intersección de Pérdida', marker=d

# Configurar el primer gráfico con tamaño reducido
fig_curvas.update_layout(
    title=f'Curvas de Exactitud y Pérdida ({model_name})',
    xaxis_title='Épocas',
    yaxis_title='Valor',
    hovermode='x unified',
    template='seaborn',
    xaxis=dict(range=[epochs[0], epochs[-1]]), # Sincronizar el rango del eje x
    width=700, # Ancho reducido
    height=500 # Altura estándar
)

# --- Segundo gráfico: diferencias en barras ---
fig_diferencias = go.Figure()

if acc_diff is not None and loss_diff is not None:
    # Trazas para las diferencias de precisión
    fig_diferencias.add_trace(go.Bar(
        x=epochs,
        y=acc_diff,
        name='Diferencia de Exactitud',
        marker_color='lightblue',
        opacity=0.6
    ))

    # Trazas para las diferencias de pérdida
    fig_diferencias.add_trace(go.Bar(
        x=epochs,
        y=loss_diff,
        name='Diferencia de Pérdida',
        marker_color='lightcoral',
        opacity=0.6
    ))

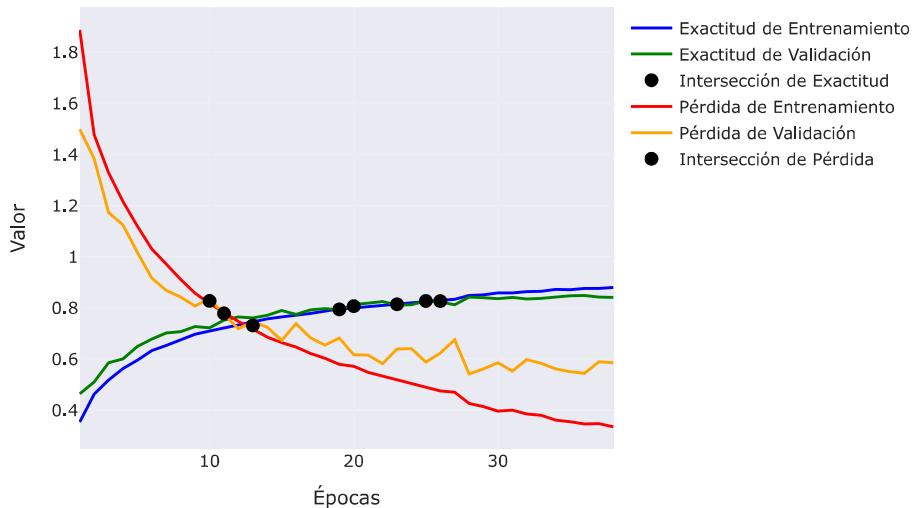
# Configurar el segundo gráfico con tamaño reducido
fig_diferencias.update_layout(
    title=f'Diferencias entre Entrenamiento y Validación ({model_name})',
    xaxis_title='Épocas',
    yaxis_title='Diferencia',
    hovermode='x unified',
    template='seaborn',
    xaxis=dict(range=[epochs[0], epochs[-1] + 1]), # Sincronizar el rango del eje x
    width=700, # Ancho reducido
    height=500 # Altura estándar
)

# Mostrar los gráficos
fig_curvas.show()
fig_diferencias.show()

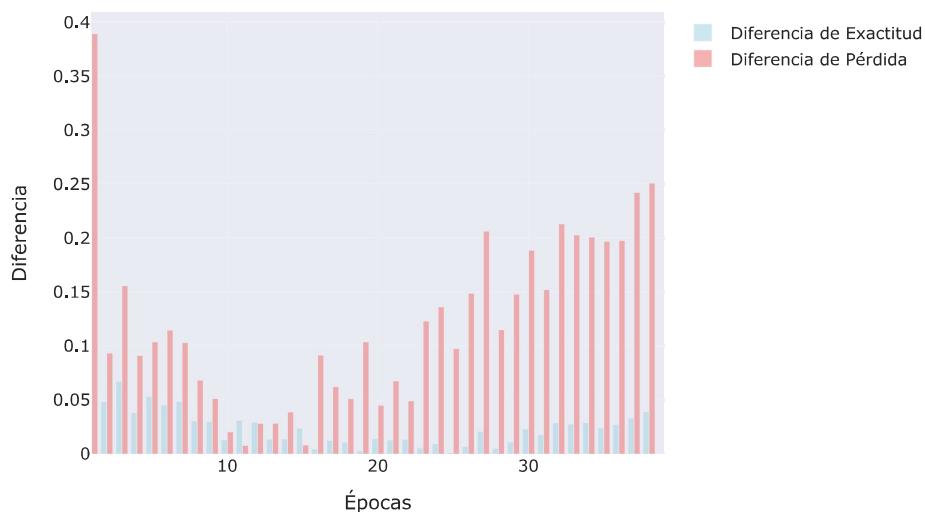
plot_history(history, 'CNN')

```

Curvas de Exactitud y Pérdida (CNN)



Diferencias entre Entrenamiento y Validación (CNN)



```
In [11]: def plot_comparacion_modelos(historiales, nombres_modelos, resultados_prueba):
    # Verificar que las listas tengan el mismo tamaño
    #if len(historiales) != len(nombres_modelos) or len(historiales) != len(resultados_prueba):
    #    raise ValueError("El número de historiales, nombres de modelos y resultados de prueba debe coincidir.")

    # Inicializar listas para almacenar los valores de la última época
    exactitud_entrenamiento = []
    exactitud_validacion = []
    perdida_entrenamiento = []
    perdida_validacion = []
    exactitud_prueba = []
    perdida_prueba = []

    # Recorrer los historiales y extraer los valores de la última época y los resultados de prueba
    for historial, resultado_prueba in zip(historiales, resultados_prueba):
        prueba_perdida, prueba_exactitud = resultado_prueba[0], resultado_prueba[1]

        exactitud = historial.history['accuracy']
        exactitud_val = historial.history.get('val_accuracy')
        perdida = historial.history['loss']
        perdida_val = historial.history.get('val_loss')

        ultima_epoca = len(exactitud) - 1

        # Guardar los valores de la última época y redondearlos al tercer decimal
        exactitud_entrenamiento.append(round(np.mean(exactitud), 3))
        exactitud_validacion.append(round(np.mean(exactitud_val), 3))
        perdida_entrenamiento.append(round(np.mean(perdida), 3))
        perdida_validacion.append(round(np.mean(perdida_val), 3))
        exactitud_prueba.append(prueba_exactitud)
        perdida_prueba.append(prueba_perdida)
```

```

exactitud_validacion.append(round(np.mean(exactitud_val), 3) if exactitud_val is not None else 'N/A')
perdida_entrenamiento.append(round(np.mean(perdida), 3))
perdida_validacion.append(round(np.mean(perdida_val), 3) if perdida_val is not None else 'N/A')
# Guardar los valores de precisión y pérdida en prueba
exactitud_prueba.append(round(prueba_exactitud, 3))
perdida_prueba.append(round(prueba_perdida, 3))

df = pd.DataFrame({
    'Modelo':nombres_modelos,
    'Media(Exactitud_entrenamiento)': exactitud_entrenamiento,
    'Media(Exactitud_validacion)': exactitud_validacion,
    'Media(Exactitud_prueba)': exactitud_prueba,
    'Media(Perdida_entrenamiento)': perdida_entrenamiento,
    'Media(Perdida_validacion)': perdida_validacion,
    'Media(Perdida_prueba)': perdida_prueba
})
)

# Crear el gráfico de barras
fig = go.Figure()

# Añadir las barras en el orden especificado
fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=exactitud_entrenamiento,
    name='Exactitud de Entrenamiento',
    marker_color='blue',
    text=exactitud_entrenamiento,
    textposition='outside',
    hoverinfo='none'
))

fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=exactitud_validacion,
    name='Exactitud de Validación',
    marker_color='green',
    text=exactitud_validacion,
    textposition='outside',
    hoverinfo='none'
))

fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=perdida_entrenamiento,
    name='Pérdida de Entrenamiento',
    marker_color='red',
    text=perdida_entrenamiento,
    textposition='outside',
    hoverinfo='none'
))

fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=perdida_validacion,
    name='Pérdida de Validación',
    marker_color='orange',
    text=perdida_validacion,
    textposition='outside',
    hoverinfo='none'
))

fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=exactitud_prueba,
    name='Exactitud de Prueba',
    marker_color='purple',
    text=exactitud_prueba,
    textposition='outside',
    hoverinfo='none'
))

fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=perdida_prueba,
    name='Pérdida de Prueba',
    marker_color='darkred',
    text=perdida_prueba,
    textposition='outside',
    hoverinfo='none'
))

# Configurar el layout del gráfico
fig.update_layout(
    title='Comparativa de Modelos',
    barmode='group',
)

```

```

        xaxis_title='Modelos',
        yaxis_title='Valor',
        template='seaborn',
        width=1200,
        height=600
    )

    # Mostrar el gráfico
    fig.show()
    return df

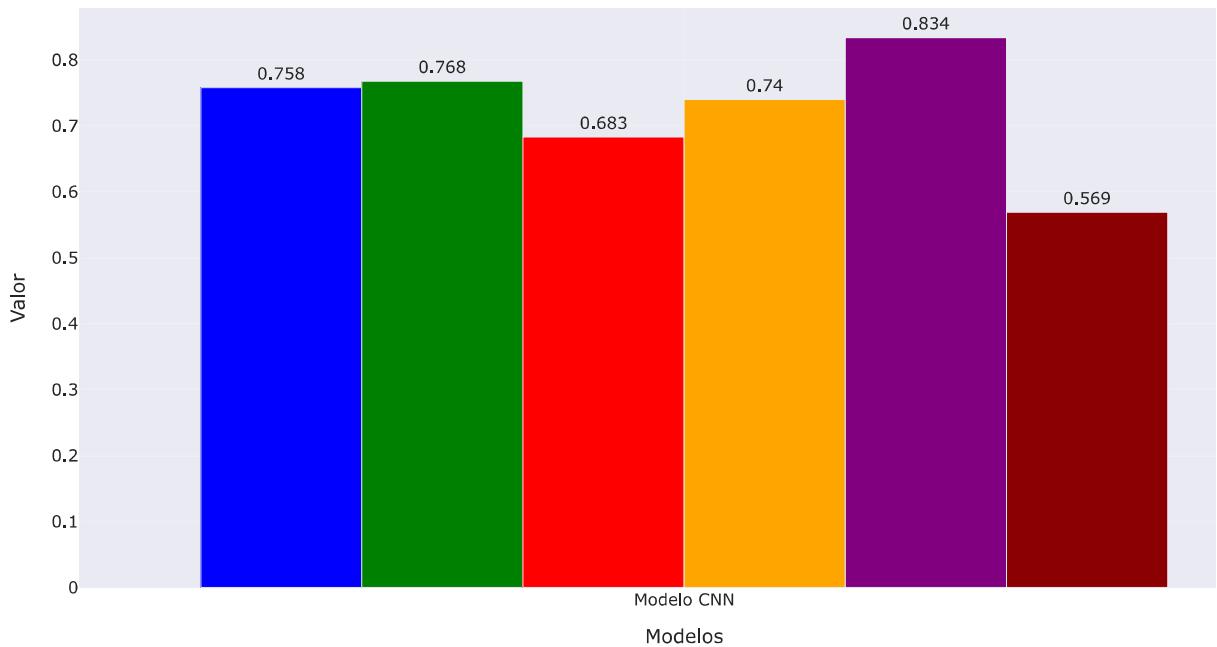
# Crear listas de histories, nombres de modelos y tests.
historias = [history]
modelos = ["Modelo CNN"]

tests = [modelo.evaluate(x_test, y_test_one_hot)]
# Llamar a la función para mostrar el gráfico
df=plot_comparacion_modelos(historias, modelos,tests)
df

```

313/313 ━━━━━━ 6s 19ms/step - accuracy: 0.8348 - loss: 0.5748

Comparativa de Modelos



Out[11]:	Modelo	Media(Exactitud_entrenamiento)	Media(Exactitud_validacion)	Media(Exactitud_prueba)	Media(Perdida_entrenamiento)	Media
0	Modelo CNN	0.758	0.768	0.834	0.683	

```

In [12]: import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report
import matplotlib.pyplot as plt

# Generar predicciones del modelo
predicciones = modelo.predict(x_test)
predicciones_clases = np.argmax(predicciones, axis=1) # Clases predichas

# Obtener las clases verdaderas
verdaderas_clases = np.argmax(y_test_one_hot, axis=1) # Etiquetas verdaderas

# Crear la matriz de confusión
matriz_confusion = confusion_matrix(verdaderas_clases, predicciones_clases)

# Usar MAP_ELEMENTS para los nombres de las clases
MAP_ELEMENTS = {
    0: 'avion', 1: 'coche', 2: 'ave',
    3: 'gato', 4: 'ciervo', 5: 'perro', 6: 'rana',
    7: 'caballo', 8: 'barco', 9: 'camion'
}
clases = [MAP_ELEMENTS[i] for i in range(len(MAP_ELEMENTS))]

# Visualizar la matriz de confusión como diagrama

```

```

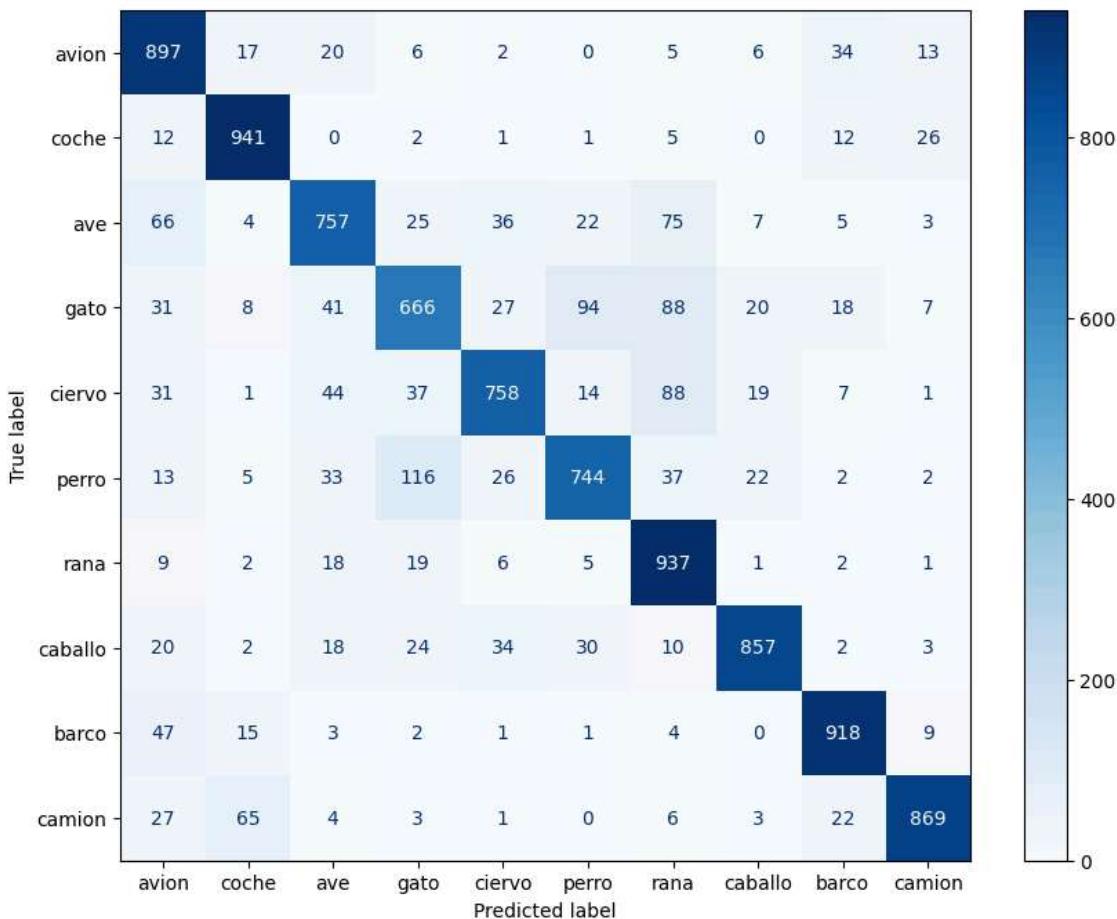
plt.figure(figsize=(10, 8))
ConfusionMatrixDisplay(confusion_matrix=matriz_confusion, display_labels=clases).plot(cmap='Blues', ax=plt.gca())
plt.title("Matriz de Confusión")
plt.show()

print("\nReporte de clasificación:")
print(classification_report(verdaderas_clases, predicciones_clases, target_names=clases))

```

313/313 ━━━━━━ 8s 24ms/step

Matriz de Confusión



Reporte de clasificación:

	precision	recall	f1-score	support
avion	0.78	0.90	0.83	1000
coche	0.89	0.94	0.91	1000
ave	0.81	0.76	0.78	1000
gato	0.74	0.67	0.70	1000
ciervo	0.85	0.76	0.80	1000
perro	0.82	0.74	0.78	1000
rana	0.75	0.94	0.83	1000
caballo	0.92	0.86	0.89	1000
barco	0.90	0.92	0.91	1000
camion	0.93	0.87	0.90	1000
accuracy			0.83	10000
macro avg	0.84	0.83	0.83	10000
weighted avg	0.84	0.83	0.83	10000

Práctica: Arquitectura VGG para CIFAR-10

Durante la práctica me he centrado en la construcción de una arquitectura **VGG** diseñada para trabajar con el dataset CIFAR-10. La red contiene **tres bloques convolucionales dobles** de profundidad creciente (64, 128, 256 filtros), ideales para capturar características jerárquicas en imágenes.

Bloques de la arquitectura

- **Bloque 1 (Conv2D):** Captura características básicas como bordes y texturas.
- **Bloque 2 (Conv2D):** Aprende relaciones más complejas y patrones locales.
- **Bloque 3 (Conv2D):** Detecta partes de objetos más avanzadas.

Consideraciones en la arquitectura

1. Padding = 'same':

- Se utilizó para no perder información en los bordes, especialmente importante dado el uso de técnicas de augmentación con desplazamientos.

2. BatchNormalization:

- Aplicado después de cada capa convolucional para estabilizar la salida de las activaciones **ReLU**.

3. Regularización:

- Se ensayó **L2**, pero finalmente se optó por **Dropout**:
 - **20% en los bloques iniciales** para evitar pérdida prematura de información.
 - **30% en bloques finales y en la capa densa**.

4. Inicialización de pesos:

- **HeUniform** para capas con activación **ReLU**.
- **GlorotUniform** para capas finales, asegurando un inicio de entrenamiento estable y eficiente.

5. Reducción dimensional:

- **MaxPooling2D** aplicado para reducir progresivamente las dimensiones de entrada:
 - $32 \times 32 \rightarrow 16 \times 16 \rightarrow 8 \times 8$.

6. Capa final densa:

- Una capa de **1024 unidades**, adecuada tras comprobar que el refinamiento jerárquico ya había sido realizado en los bloques convolucionales.

Data Augmentation aplicado

Se incluyó augmentación para aumentar la diversidad del dataset:

- **Rotación aleatoria:** ± 15 grados para robustez ante orientaciones variadas.
- **Desplazamiento horizontal:** Hasta un 10% del ancho para simular objetos descentrados.
- **Desplazamiento vertical:** Hasta un 10% de la altura con fines similares al punto anterior.
- **Volteo horizontal:** Útil para objetos o animales con simetría horizontal.
- **Zoom aleatorio:** $\pm 20\%$ para simular imágenes a diferentes escalas.

Función de optimización

- **SGD**: Se probó con diferentes valores de **learning rate** y **momentum**.
- **Adam**: Mostró una **convergencia más rápida**, pero también problemas de estancamiento, evidenciado en la falta de mejora en `val_loss`.

Soluciones:

1. **ReduceLROnPlateau**: Reducción progresiva del learning rate al detectar estancamiento.
2. **EarlyStopping**: Salvaguarda para detener el entrenamiento tras 10 épocas sin mejora en `val_loss`, recuperando los mejores pesos.

Resultados finales

El modelo alcanzó una **accuracy promedio de 83.4%**. A continuación, un análisis de la matriz de confusión:

Clases con mejor desempeño

• Coche:

- **Precision: 0.89 | Recall: 0.94 | F1-score: 0.91**
 - Alta precisión y recall, indicando que pocas imágenes reales de "coche" son clasificadas incorrectamente.

• Barco y Camión:

- Ambos tienen **F1-scores** en torno a 0.90.

• Caballo:

- **F1-score: 0.89**, sugiriendo que el modelo captura bien las características distintivas.

Clases con peor desempeño

• Gato:

- **Precision: 0.74 | Recall: 0.67 | F1-score: 0.70**

- Muchas imágenes reales de "gato" son clasificadas incorrectamente, posiblemente confundidas con "perro" debido a similitudes en texturas y colores.

- **Ave:**

- **Precision:** 0.81 | **Recall:** 0.76 | **F1-score:** 0.78
- Probablemente confundida con aviones debido a características comunes en los fondos.

- **Ciervo:**

- **Precision:** 0.85 | **Recall:** 0.76 | **F1-score:** 0.80
- Dificultades para identificar ciervos, posiblemente debido a similitudes con caballos.

Este análisis resalta fortalezas y áreas de mejora del modelo, especialmente en las clases con menor desempeño.