In [1]:
```
from tensorflow.keras.datasets import mnist
import math, time
import matplotlib.pyplot as plt
import numpy as np
#!pip install seaborn
import seaborn as sns
%matplotlib inline
import pandas as pd
from sklearn.model_selection import train_test_split
import pickle
from PIL import Image

from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from imblearn.metrics import specificity_score
from matplotlib import*
from matplotlib.cm import register_cmap
import matplotlib.pyplot as plt

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
#from keras.layers.normalization import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from keras.models import model_from_json
from keras.models import load_model

from sklearn.svm import SVC #SVR para regresión
from sklearn.metrics import classification_report
from keras import models
from keras.layers import BatchNormalization, MaxPool2D, GlobalMaxPool2D
#Arquitecturas de Transfer learning. Puedes configurar parámetros específicos de ca
from tensorflow.keras.applications.vgg16 import VGG16
#from keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.xception import Xception
from keras.applications.inception_resnet_v2 import InceptionResNetV2

import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Flatten, Dense, concatenate, Dropout
from tensorflow.keras.applications import VGG16, Xception
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Conv2D, MaxPooling2D
import os
from skimage import io
from sklearn.model_selection import train_test_split
import os
import cv2
import numpy as np
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, recall_score,
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical

print('Módulos importados')
```

```
D:\anaconda3\Lib\site-packages\paramiko\transport.py:219: CryptographyDeprecationW
arning: Blowfish has been deprecated
  "class": algorithms.Blowfish,
Módulos importados
```

In [2]:

```
# Funciones para guardar y cargar objetos pickle
def guardarObjeto(pipeline,nombreArchivo):
    print("Guardando Objeto en Archivo")
    with open(nombreArchivo+'.pickle', 'wb') as handle:
        pickle.dump(pipeline, handle, protocol=pickle.HIGHEST_PROTOCOL)
        print("Objeto Guardado en Archivo")
def cargarObjeto(nombreArchivo):
    with open(nombreArchivo+'.pickle', 'rb') as handle:
        pipeline = pickle.load(handle)
        print("Objeto Cargado desde Archivo")
    return pipeline
# Funciones para guardar y cargar la Red Neuronal (Arquitectura y Pesos)
def guardarNN(model,nombreArchivo):
    print("Guardando Red Neuronal en Archivo")
    model.save(nombreArchivo+'.h5')
    print("Red Neuronal Guardada en Archivo")

def cargarNN(nombreArchivo):
    model = load_model(nombreArchivo+'.h5')
    print("Red Neuronal Cargada desde Archivo")
    return model

# Función para medir la calidad de modelos
def obtenerResultados(y_test, y_pred):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')
    specificity = specificity_score(y_test, y_pred, average='macro')

    accuracy=str(round(accuracy, 4))
    precision=str(round(precision, 4))
    recall=str(round(recall, 4))
    f1=str(round(f1, 4))
    specificity=str(round(specificity, 4))

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall o Sensitivity:", recall)
    print("F1-Score:", f1)
    print("Specificity:", specificity)
```

```
    return accuracy, precision, recall, f1, specificity
print('Funciones para guardar y cargar modelos personalizados')
```

```
Funciones para guardar y cargar modelos personalizados
```

In [3]:
```
#Importar dataset
from keras.datasets import cifar100
# Cargar el conjunto de datos CIFAR-100
(x_train, y_train), (x_test, y_test) = cifar100.load_data(label_mode='fine')
```

In [4]:
```
import random

# Obtener las categorías únicas y convertirlas a una lista
categorias_unicas = list(set(y_train.flatten()))

# Seleccionar 5 categorías al azar
categorias_al_azar = random.sample(categorias_unicas, 5)

# Inicializar un diccionario para contar los datos en cada categoría
conteo_por_categoria = {categoria: 0 for categoria in categorias_al_azar}

# Contar los datos en cada categoría
for etiqueta in y_train.flatten():
    if etiqueta in categorias_al_azar:
        conteo_por_categoria[etiqueta] += 1

# Mostrar el conteo por categoría
print("Conteo de datos por categoría:")
for categoria, conteo in conteo_por_categoria.items():
    print(f"{categoria}: {conteo} datos")
```

```
Conteo de datos por categoría:
58: 500 datos
39: 500 datos
88: 500 datos
87: 500 datos
51: 500 datos
```

In [5]:
```
# Obtener nombres de las etiquetas
label_names = [
    'apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle', 'bi
    'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel', 'can', 'castle', 'caterpi
    'chair', 'chimpanzee', 'clock', 'cloud', 'cockroach', 'couch', 'crab', 'crocodi
    'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl', 'hamster', 'house',
    'lamp', 'lawn_mower', 'leopard', 'lion', 'lizard', 'lobster', 'man', 'maple_tre
    'mouse', 'mushroom', 'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree', 'pea
    'plain', 'plate', 'poppy', 'porcupine', 'possum', 'rabbit', 'raccoon', 'ray', '
    'rose', 'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake
    'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table', 'tank', 'telepho
    'train', 'trout', 'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf'
]

# Seleccionar aleatoriamente algunas imágenes para visualizar
num_images_to_display = 5
random_indices = random.sample(range(len(x_train)), num_images_to_display)

# Mostrar las imágenes seleccionadas
plt.figure(figsize=(15, 5))
for i, idx in enumerate(random_indices):
    plt.subplot(1, num_images_to_display, i + 1)
    plt.imshow(x_train[idx])
    plt.title(label_names[y_train[idx][0]])
    plt.axis('off')
```

```
plt.show()
```

In [6]:

```
import numpy as np
from sklearn.model_selection import train_test_split

# Fijar la semilla para reproducibilidad
np.random.seed(42)

# Realizar muestreo para obtener un total de 5000 datos en el conjunto de entrenami
x_train_sample, _, y_train_sample, _ = train_test_split(x_train, y_train, train_siz
x_test_sample, _, y_test_sample, _ = train_test_split(x_test, y_test, train_size=10

# Obtener las categorías únicas y convertirlas a una lista
categorias_unicas_train = list(set(y_train_sample.flatten()))

# Inicializar un diccionario para contar los datos en cada categoría en el conjunto
conteo_por_categoria_train = {categoria: 0 for categoria in categorias_unicas_train

# Contar los datos en cada categoría en el conjunto de entrenamiento
for etiqueta in y_train_sample.flatten():
    conteo_por_categoria_train[etiqueta] += 1

# Mostrar el conteo por categoría en el conjunto de entrenamiento
print("Conteo de datos por categoría en el conjunto de entrenamiento:")
for categoria, conteo in conteo_por_categoria_train.items():
    print(f"{label_names[categoria]}: {conteo} datos")

# Obtener las categorías únicas en el conjunto de prueba y convertirlas a una lista
categorias_unicas_test = list(set(y_test_sample.flatten()))

# Inicializar un diccionario para contar los datos en cada categoría en el conjunto
conteo_por_categoria_test = {categoria: 0 for categoria in categorias_unicas_test}

# Contar los datos en cada categoría en el conjunto de prueba
for etiqueta in y_test_sample.flatten():
    conteo_por_categoria_test[etiqueta] += 1

# Mostrar el conteo por categoría en el conjunto de prueba
print("\nConteo de datos por categoría en el conjunto de prueba:")
for categoria, conteo in conteo_por_categoria_test.items():
    print(f"{label_names[categoria]}: {conteo} datos")
```

```
Conteo de datos por categoría en el conjunto de entrenamiento:
apple: 50 datos
aquarium_fish: 50 datos
baby: 50 datos
bear: 50 datos
beaver: 50 datos
bed: 50 datos
bee: 50 datos
beetle: 50 datos
bicycle: 50 datos
bottle: 50 datos
bowl: 50 datos
boy: 50 datos
bridge: 50 datos
bus: 50 datos
butterfly: 50 datos
camel: 50 datos
can: 50 datos
castle: 50 datos
caterpillar: 50 datos
cattle: 50 datos
chair: 50 datos
chimpanzee: 50 datos
clock: 50 datos
cloud: 50 datos
cockroach: 50 datos
couch: 50 datos
crab: 50 datos
crocodile: 50 datos
cup: 50 datos
dinosaur: 50 datos
dolphin: 50 datos
elephant: 50 datos
flatfish: 50 datos
forest: 50 datos
fox: 50 datos
girl: 50 datos
hamster: 50 datos
house: 50 datos
kangaroo: 50 datos
keyboard: 50 datos
lamp: 50 datos
lawn_mower: 50 datos
leopard: 50 datos
lion: 50 datos
lizard: 50 datos
lobster: 50 datos
man: 50 datos
maple_tree: 50 datos
motorcycle: 50 datos
mountain: 50 datos
mouse: 50 datos
mushroom: 50 datos
oak_tree: 50 datos
orange: 50 datos
orchid: 50 datos
otter: 50 datos
palm_tree: 50 datos
pear: 50 datos
pickup_truck: 50 datos
pine_tree: 50 datos
plain: 50 datos
plate: 50 datos
poppy: 50 datos
```

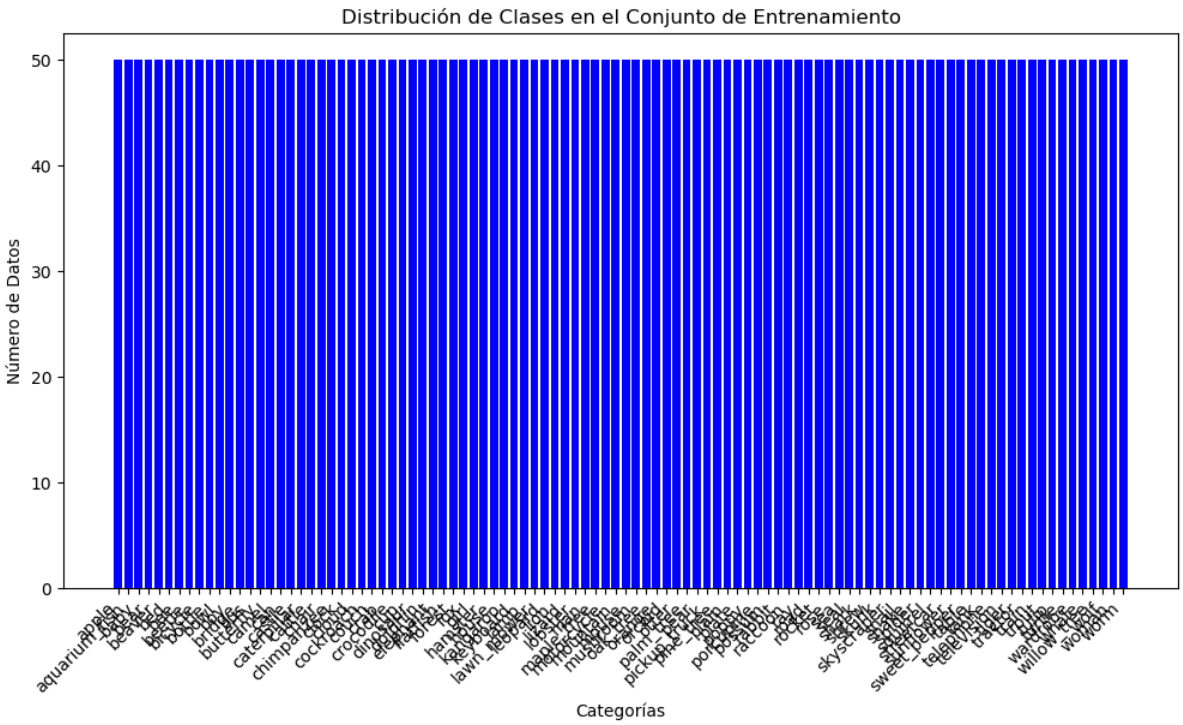
```
porcupine: 50 datos
possum: 50 datos
rabbit: 50 datos
raccoon: 50 datos
ray: 50 datos
road: 50 datos
rocket: 50 datos
rose: 50 datos
sea: 50 datos
seal: 50 datos
shark: 50 datos
shrew: 50 datos
skunk: 50 datos
skyscraper: 50 datos
snail: 50 datos
snake: 50 datos
spider: 50 datos
squirrel: 50 datos
streetcar: 50 datos
sunflower: 50 datos
sweet_pepper: 50 datos
table: 50 datos
tank: 50 datos
telephone: 50 datos
television: 50 datos
tiger: 50 datos
tractor: 50 datos
train: 50 datos
trout: 50 datos
tulip: 50 datos
turtle: 50 datos
wardrobe: 50 datos
whale: 50 datos
willow_tree: 50 datos
wolf: 50 datos
woman: 50 datos
worm: 50 datos

Conteo de datos por categoría en el conjunto de prueba:
apple: 10 datos
aquarium_fish: 10 datos
baby: 10 datos
bear: 10 datos
beaver: 10 datos
bed: 10 datos
bee: 10 datos
beetle: 10 datos
bicycle: 10 datos
bottle: 10 datos
bowl: 10 datos
boy: 10 datos
bridge: 10 datos
bus: 10 datos
butterfly: 10 datos
camel: 10 datos
can: 10 datos
castle: 10 datos
caterpillar: 10 datos
cattle: 10 datos
chair: 10 datos
chimpanzee: 10 datos
clock: 10 datos
cloud: 10 datos
cockroach: 10 datos
```

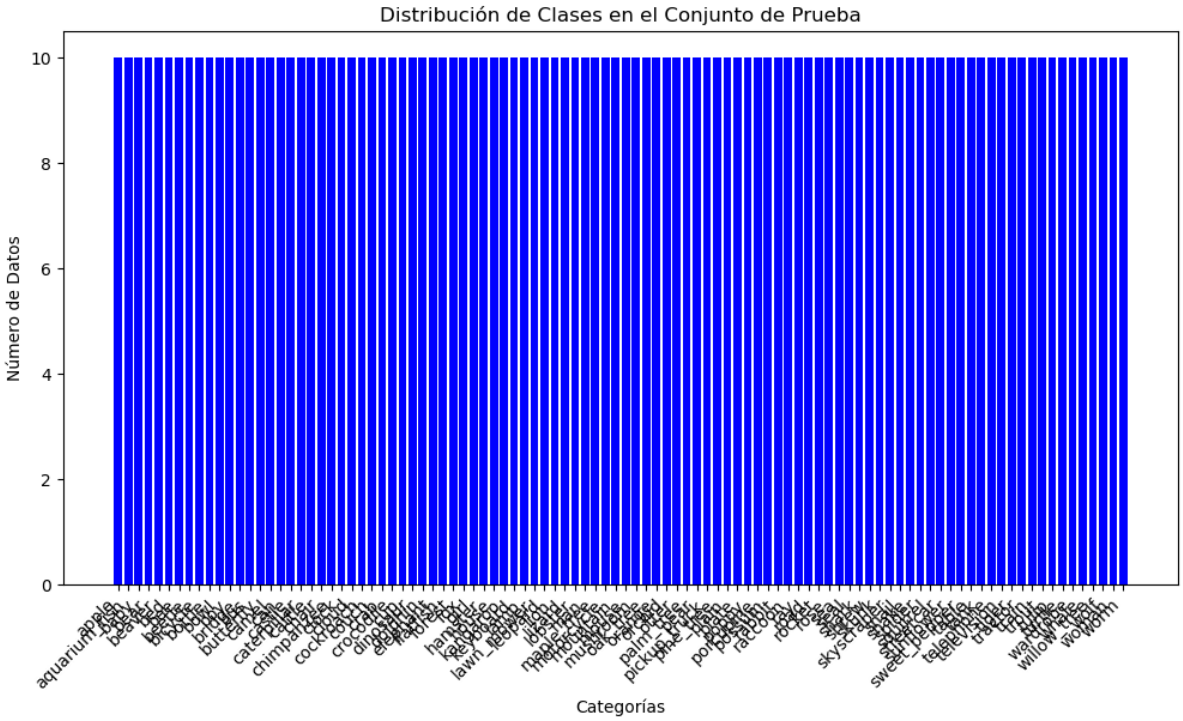
```
couch: 10 datos
crab: 10 datos
crocodile: 10 datos
cup: 10 datos
dinosaur: 10 datos
dolphin: 10 datos
elephant: 10 datos
flatfish: 10 datos
forest: 10 datos
fox: 10 datos
girl: 10 datos
hamster: 10 datos
house: 10 datos
kangaroo: 10 datos
keyboard: 10 datos
lamp: 10 datos
lawn_mower: 10 datos
leopard: 10 datos
lion: 10 datos
lizard: 10 datos
lobster: 10 datos
man: 10 datos
maple_tree: 10 datos
motorcycle: 10 datos
mountain: 10 datos
mouse: 10 datos
mushroom: 10 datos
oak_tree: 10 datos
orange: 10 datos
orchid: 10 datos
otter: 10 datos
palm_tree: 10 datos
pear: 10 datos
pickup_truck: 10 datos
pine_tree: 10 datos
plain: 10 datos
plate: 10 datos
poppy: 10 datos
porcupine: 10 datos
possum: 10 datos
rabbit: 10 datos
raccoon: 10 datos
ray: 10 datos
road: 10 datos
rocket: 10 datos
rose: 10 datos
sea: 10 datos
seal: 10 datos
shark: 10 datos
shrew: 10 datos
skunk: 10 datos
skyscraper: 10 datos
snail: 10 datos
snake: 10 datos
spider: 10 datos
squirrel: 10 datos
streetcar: 10 datos
sunflower: 10 datos
sweet_pepper: 10 datos
table: 10 datos
tank: 10 datos
telephone: 10 datos
television: 10 datos
tiger: 10 datos
```

```
tractor: 10 datos
train: 10 datos
trout: 10 datos
tulip: 10 datos
turtle: 10 datos
wardrobe: 10 datos
whale: 10 datos
willow_tree: 10 datos
wolf: 10 datos
woman: 10 datos
worm: 10 datos
```

In [7]:

```python
import matplotlib.pyplot as plt

# Función para visualizar la distribución de clases
def plot_class_distribution(y, dataset_name):
    unique_classes, counts = np.unique(y, return_counts=True)
    class_labels = [label_names[c] for c in unique_classes]

    plt.figure(figsize=(12, 6))
    plt.bar(class_labels, counts, color='blue')
    plt.title(f'Distribución de Clases en el Conjunto de {dataset_name}')
    plt.xlabel('Categorías')
    plt.ylabel('Número de Datos')
    plt.xticks(rotation=45, ha="right")
    plt.show()

# Visualizar la distribución de clases en el conjunto de entrenamiento
plot_class_distribution(y_train_sample.flatten(), 'Entrenamiento')

# Visualizar la distribución de clases en el conjunto de prueba
plot_class_distribution(y_test_sample.flatten(), 'Prueba')
```





PARA CREAR EL MODELO CNN

VAMOS A NORMALIZAR LOS DATOS EN PRIMER LUGAR PARA PODER TRABAJAR DE UNA MEJOR MANERA

In [8]:
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical

from keras.layers import Dropout
```

In [9]:
```
# Normalizar X_train y X_test
X_train_normalized = x_train_sample.astype('float32') / 255.0
X_test_normalized = x_test_sample.astype('float32') / 255.0

# Verificar las dimensiones antes y después de la normalización
print("Dimensiones de X_train antes de la normalización:", x_train_sample.shape)
print("Dimensiones de X_test antes de la normalización:", x_test_sample.shape)

print("\nNormalizando...")

print("Dimensiones de X_train después de la normalización:", X_train_normalized.sha
print("Dimensiones de X_test después de la normalización:", X_test_normalized.shape
```

```
Dimensiones de X_train antes de la normalización: (5000, 32, 32, 3)
Dimensiones de X_test antes de la normalización: (1000, 32, 32, 3)

Normalizando...
Dimensiones de X_train después de la normalización: (5000, 32, 32, 3)
Dimensiones de X_test después de la normalización: (1000, 32, 32, 3)
```

In [10]:
```
x_train_sample.shape[1:]
```

Out[10]:
```
(32, 32, 3)
```

In [11]:
```
# Convertir etiquetas a formato one-hot encoding
y_train_one_hot = to_categorical(y_train_sample, num_classes=100)
y_test_one_hot = to_categorical(y_test_sample, num_classes=100)

def create_compile_model_v2(filters, conv_layers, dense_neurons, epochs, batch_size
    model = Sequential()

    # Capa de convolución y pooling
    for _ in range(conv_layers):
        model.add(Conv2D(filters, kernel_size=(3, 3), activation='relu', input_shap
        model.add(MaxPooling2D(pool_size=(2, 2)))  # Ajusta el tamaño de la capa de

    # Aplanar antes de las capas densas
    model.add(Flatten())

    # Capas densas
    for _ in range(2):
        model.add(Dense(dense_neurons, activation='relu'))


    # Capa de salida
    model.add(Dense(100, activation='softmax'))

    # Compilar el modelo
    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accu

    # Imprimir la arquitectura del modelo
    model.summary()

    # Entrenar el modelo
    history = model.fit(X_train_normalized, y_train_one_hot, epochs=epochs, batch_s
    return model, history
#model1, history1 = create_compile_model_v2(filters=64, conv_layers=2, dense_neuron
```

In [12]:
```
#model2, history2 = create_compile_model_v2(filters=64, conv_layers=2, dense_neuron
```

In [13]:
```
#model3, history3 = create_compile_model_v2(filters=64, conv_layers=2, dense_neuron
```

In [14]:
```
model4, history4 = create_compile_model_v2(filters=64, conv_layers=2, dense_neurons
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 64)        1792

 max_pooling2d (MaxPooling2  (None, 15, 15, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 128)               295040

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 100)               12900

=================================================================
Total params: 363172 (1.39 MB)
Trainable params: 363172 (1.39 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/40
200/200 [==============================] - 9s 33ms/step - loss: 4.5427 - accuracy:
0.0160 - val_loss: 4.4039 - val_accuracy: 0.0240
Epoch 2/40
200/200 [==============================] - 6s 31ms/step - loss: 4.2934 - accuracy:
0.0416 - val_loss: 4.1864 - val_accuracy: 0.0550
Epoch 3/40
200/200 [==============================] - 6s 31ms/step - loss: 4.0254 - accuracy:
0.0752 - val_loss: 4.0148 - val_accuracy: 0.0800
Epoch 4/40
200/200 [==============================] - 6s 32ms/step - loss: 3.7658 - accuracy:
0.1144 - val_loss: 3.8370 - val_accuracy: 0.1160
Epoch 5/40
200/200 [==============================] - 6s 32ms/step - loss: 3.5436 - accuracy:
0.1476 - val_loss: 3.7466 - val_accuracy: 0.1190
Epoch 6/40
200/200 [==============================] - 6s 31ms/step - loss: 3.3401 - accuracy:
0.1794 - val_loss: 3.6722 - val_accuracy: 0.1360
Epoch 7/40
200/200 [==============================] - 6s 32ms/step - loss: 3.1527 - accuracy:
0.2128 - val_loss: 3.5802 - val_accuracy: 0.1700
Epoch 8/40
200/200 [==============================] - 6s 32ms/step - loss: 2.9474 - accuracy:
0.2566 - val_loss: 3.6546 - val_accuracy: 0.1660
Epoch 9/40
200/200 [==============================] - 7s 33ms/step - loss: 2.7459 - accuracy:
0.2872 - val_loss: 3.6948 - val_accuracy: 0.1630
Epoch 10/40
200/200 [==============================] - 7s 34ms/step - loss: 2.5222 - accuracy:
0.3350 - val_loss: 3.7381 - val_accuracy: 0.1710
Epoch 11/40
200/200 [==============================] - 7s 33ms/step - loss: 2.3174 - accuracy:
0.3840 - val_loss: 3.8659 - val_accuracy: 0.1860
Epoch 12/40
200/200 [==============================] - 6s 32ms/step - loss: 2.0868 - accuracy:
0.4372 - val_loss: 4.1394 - val_accuracy: 0.1960
Epoch 13/40
```

```
200/200 [==============================] - 7s 33ms/step - loss: 1.8948 - accuracy:
0.4726 - val_loss: 4.2146 - val_accuracy: 0.1870
Epoch 14/40
200/200 [==============================] - 6s 32ms/step - loss: 1.6585 - accuracy:
0.5406 - val_loss: 4.3996 - val_accuracy: 0.1900
Epoch 15/40
200/200 [==============================] - 7s 33ms/step - loss: 1.4658 - accuracy:
0.5902 - val_loss: 4.7765 - val_accuracy: 0.1840
Epoch 16/40
200/200 [==============================] - 6s 32ms/step - loss: 1.2965 - accuracy:
0.6334 - val_loss: 5.0729 - val_accuracy: 0.1710
Epoch 17/40
200/200 [==============================] - 6s 32ms/step - loss: 1.1381 - accuracy:
0.6768 - val_loss: 5.2019 - val_accuracy: 0.1840
Epoch 18/40
200/200 [==============================] - 7s 33ms/step - loss: 0.9675 - accuracy:
0.7262 - val_loss: 5.8597 - val_accuracy: 0.1760
Epoch 19/40
200/200 [==============================] - 7s 33ms/step - loss: 0.8303 - accuracy:
0.7546 - val_loss: 6.1481 - val_accuracy: 0.1770
Epoch 20/40
200/200 [==============================] - 7s 33ms/step - loss: 0.6956 - accuracy:
0.7982 - val_loss: 6.7427 - val_accuracy: 0.1800
Epoch 21/40
200/200 [==============================] - 7s 33ms/step - loss: 0.5999 - accuracy:
0.8310 - val_loss: 6.8125 - val_accuracy: 0.1840
Epoch 22/40
200/200 [==============================] - 7s 34ms/step - loss: 0.5027 - accuracy:
0.8578 - val_loss: 7.3337 - val_accuracy: 0.1680
Epoch 23/40
200/200 [==============================] - 7s 33ms/step - loss: 0.4428 - accuracy:
0.8726 - val_loss: 7.8818 - val_accuracy: 0.1920
Epoch 24/40
200/200 [==============================] - 7s 34ms/step - loss: 0.4295 - accuracy:
0.8734 - val_loss: 8.0782 - val_accuracy: 0.1870
Epoch 25/40
200/200 [==============================] - 7s 33ms/step - loss: 0.3304 - accuracy:
0.9064 - val_loss: 8.7867 - val_accuracy: 0.1840
Epoch 26/40
200/200 [==============================] - 7s 33ms/step - loss: 0.2964 - accuracy:
0.9108 - val_loss: 8.6380 - val_accuracy: 0.1890
Epoch 27/40
200/200 [==============================] - 6s 31ms/step - loss: 0.2553 - accuracy:
0.9290 - val_loss: 9.6197 - val_accuracy: 0.1860
Epoch 28/40
200/200 [==============================] - 6s 32ms/step - loss: 0.2831 - accuracy:
0.9190 - val_loss: 9.5011 - val_accuracy: 0.1780
Epoch 29/40
200/200 [==============================] - 6s 32ms/step - loss: 0.2361 - accuracy:
0.9358 - val_loss: 9.9952 - val_accuracy: 0.1840
Epoch 30/40
200/200 [==============================] - 7s 35ms/step - loss: 0.2170 - accuracy:
0.9388 - val_loss: 10.1335 - val_accuracy: 0.1740
Epoch 31/40
200/200 [==============================] - 7s 35ms/step - loss: 0.1535 - accuracy:
0.9584 - val_loss: 10.8689 - val_accuracy: 0.1790
Epoch 32/40
200/200 [==============================] - 7s 34ms/step - loss: 0.1257 - accuracy:
0.9674 - val_loss: 11.0865 - val_accuracy: 0.1810
Epoch 33/40
200/200 [==============================] - 6s 32ms/step - loss: 0.1825 - accuracy:
0.9500 - val_loss: 11.4739 - val_accuracy: 0.1760
Epoch 34/40
200/200 [==============================] - 7s 33ms/step - loss: 0.2578 - accuracy:
```

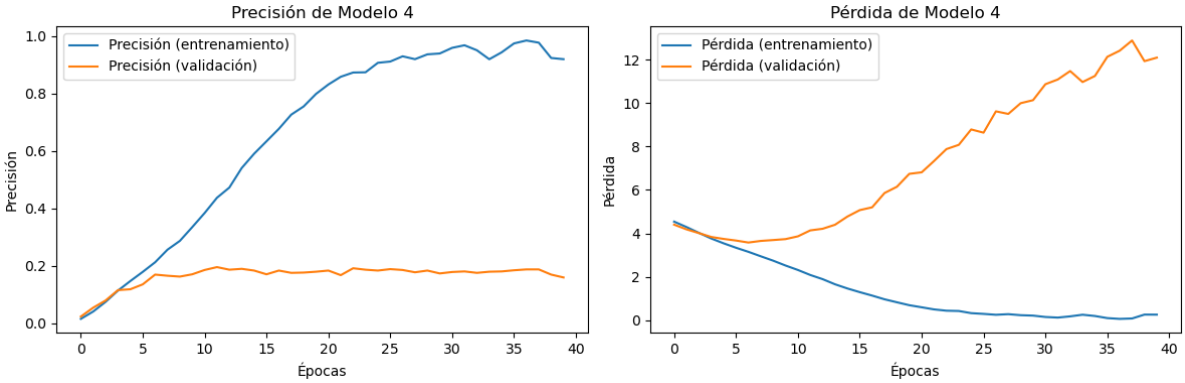
```
0.9188 - val_loss: 10.9683 - val_accuracy: 0.1800
Epoch 35/40
200/200 [==============================] - 7s 33ms/step - loss: 0.1991 - accuracy:
0.9428 - val_loss: 11.2483 - val_accuracy: 0.1810
Epoch 36/40
200/200 [==============================] - 7s 33ms/step - loss: 0.1020 - accuracy:
0.9738 - val_loss: 12.1298 - val_accuracy: 0.1850
Epoch 37/40
200/200 [==============================] - 6s 32ms/step - loss: 0.0665 - accuracy:
0.9840 - val_loss: 12.4119 - val_accuracy: 0.1880
Epoch 38/40
200/200 [==============================] - 7s 33ms/step - loss: 0.0845 - accuracy:
0.9764 - val_loss: 12.8814 - val_accuracy: 0.1880
Epoch 39/40
200/200 [==============================] - 7s 33ms/step - loss: 0.2636 - accuracy:
0.9232 - val_loss: 11.9293 - val_accuracy: 0.1700
Epoch 40/40
200/200 [==============================] - 7s 33ms/step - loss: 0.2616 - accuracy:
0.9190 - val_loss: 12.0918 - val_accuracy: 0.1600
```

In [15]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report

def print_results(model, history, model_name):
    # Evaluar el modelo en el conjunto de prueba
    test_loss, test_accuracy = model.evaluate(X_test_normalized, y_test_one_hot, ve
    print(f'\n{model_name} - Loss en conjunto de prueba: {test_loss:.4f}')
    print(f'{model_name} - Precisión en conjunto de prueba: {test_accuracy * 100:.2

    # Predicciones en el conjunto de prueba
    y_probs = model.predict(X_test_normalized)
    y_pred = np.argmax(y_probs, axis=1)
    y_true = np.argmax(y_test_one_hot, axis=1)

    # Calcular recall y F1-score
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')

    # Mostrar métricas de rendimiento generales
    print(f'{model_name} - Recall en conjunto de prueba: {recall:.4f}')
    print(f'{model_name} - F1-Score en conjunto de prueba: {f1:.4f}')

    # Mostrar el reporte de clasificación general (precision, recall, f1-score)
    classification_rep = classification_report(y_true, y_pred, output_dict=True)
    print(f'\nReporte de Clasificación para {model_name}:')
    print(f"   Acc: {classification_rep['accuracy']:.4f}")
    print(f"   Precision: {classification_rep['weighted avg']['precision']:.4f}")
    print(f"   Recall: {classification_rep['weighted avg']['recall']:.4f}")
    print(f"   F1-Score: {classification_rep['weighted avg']['f1-score']:.4f}")

    # Crear gráfico para analizar el rendimiento del modelo
    plt.figure(figsize=(12, 4))

    # Gráfico de precisión
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Precisión (entrenamiento)')
    plt.plot(history.history['val_accuracy'], label='Precisión (validación)')
    plt.xlabel('Épocas')
    plt.ylabel('Precisión')
    plt.title(f'Precisión de {model_name}')
    plt.legend()

    # Gráfico de pérdida
    plt.subplot(1, 2, 2)
```

```
    plt.plot(history.history['loss'], label='Pérdida (entrenamiento)')
    plt.plot(history.history['val_loss'], label='Pérdida (validación)')
    plt.xlabel('Épocas')
    plt.ylabel('Pérdida')
    plt.title(f'Pérdida de {model_name}')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Imprimir resultados y gráfico para el modelo1
print_results(model4, history4, 'Modelo 4')
```

```
Modelo 4 - Loss en conjunto de prueba: 12.0918
Modelo 4 - Precisión en conjunto de prueba: 16.00%
32/32 [==============================] - 0s 8ms/step
Modelo 4 - Recall en conjunto de prueba: 0.1600
Modelo 4 - F1-Score en conjunto de prueba: 0.1538

Reporte de Clasificación para Modelo 4:
   Acc: 0.1600
   Precision: 0.1734
   Recall: 0.1600
   F1-Score: 0.1538
```



In [20]:
```
guardarObjeto(model4,'CIFAR100model4CNN')
```

```
Guardando Objeto en Archivo
Objeto Guardado en Archivo
```

In [21]:
```
guardarNN(model4,'CIFAR100model4CNN')
```

```
Guardando Red Neuronal en Archivo
D:\anaconda3\Lib\site-packages\keras\src\engine\training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
Red Neuronal Guardada en Archivo
```

In [19]:
```
import numpy as np
from tensorflow.keras.preprocessing import image
import random

# Función para cargar y preprocesar una imagen
def load_and_preprocess_image(image_path):
    img = image.load_img(image_path, target_size=(32, 32))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    #img_array = preprocess_input(img_array)  # Asegúrate de aplicar el mismo prepr
    return img_array
```

```
# Ruta de la nueva imagen que deseas predecir
nueva_imagen_path = 'carro1.jpeg'

# Cargar y preprocesar la imagen
nueva_imagen = load_and_preprocess_image(nueva_imagen_path)

# Realizar la predicción
predicciones = model4.predict(nueva_imagen)

# Obtener la clase predicha
clase_predicha = np.argmax(predicciones, axis=1)

# Obtener el nombre de la clase predicha
nombre_clase_predicha = label_names[clase_predicha[0]]

# Mostrar el resultado
print(f'Clase predicha: {nombre_clase_predicha}')

# Mostrar la probabilidad de cada clase (opcional)
probabilidades = predicciones[0]
for i, probabilidad in enumerate(probabilidades):
    print(f'Probabilidad de la clase {label_names[i]}: {probabilidad:.4f}')

# Mostrar la imagen
plt.imshow(image.load_img(nueva_imagen_path))
plt.title(f'Imagen - Clase predicha: {nombre_clase_predicha}')
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 0s 25ms/step
Clase predicha: pickup_truck
Probabilidad de la clase apple: 0.0000
Probabilidad de la clase aquarium_fish: 0.0000
Probabilidad de la clase baby: 0.0000
Probabilidad de la clase bear: 0.0000
Probabilidad de la clase beaver: 0.0000
Probabilidad de la clase bed: 0.0000
Probabilidad de la clase bee: 0.0000
Probabilidad de la clase beetle: 0.0000
Probabilidad de la clase bicycle: 0.0000
Probabilidad de la clase bottle: 0.0000
Probabilidad de la clase bowl: 0.0000
Probabilidad de la clase boy: 0.0000
Probabilidad de la clase bridge: 0.0000
Probabilidad de la clase bus: 0.0000
Probabilidad de la clase butterfly: 0.0000
Probabilidad de la clase camel: 0.0000
Probabilidad de la clase can: 0.0000
Probabilidad de la clase castle: 0.0000
Probabilidad de la clase caterpillar: 0.0000
Probabilidad de la clase cattle: 0.0000
Probabilidad de la clase chair: 0.0000
Probabilidad de la clase chimpanzee: 0.0000
Probabilidad de la clase clock: 0.0000
Probabilidad de la clase cloud: 0.0000
Probabilidad de la clase cockroach: 0.0000
Probabilidad de la clase couch: 0.0000
Probabilidad de la clase crab: 0.0000
Probabilidad de la clase crocodile: 0.0000
Probabilidad de la clase cup: 0.0000
Probabilidad de la clase dinosaur: 0.0000
Probabilidad de la clase dolphin: 0.0000
Probabilidad de la clase elephant: 0.0000
Probabilidad de la clase flatfish: 0.0000
Probabilidad de la clase forest: 0.0000
Probabilidad de la clase fox: 0.0000
Probabilidad de la clase girl: 0.0000
Probabilidad de la clase hamster: 0.0000
Probabilidad de la clase house: 0.0000
Probabilidad de la clase kangaroo: 0.0000
Probabilidad de la clase keyboard: 0.0000
Probabilidad de la clase lamp: 0.0000
Probabilidad de la clase lawn_mower: 0.0000
Probabilidad de la clase leopard: 0.0000
Probabilidad de la clase lion: 0.0000
Probabilidad de la clase lizard: 0.0000
Probabilidad de la clase lobster: 0.0000
Probabilidad de la clase man: 0.0000
Probabilidad de la clase maple_tree: 0.0000
Probabilidad de la clase motorcycle: 0.0000
Probabilidad de la clase mountain: 0.0000
Probabilidad de la clase mouse: 0.0000
Probabilidad de la clase mushroom: 0.0000
Probabilidad de la clase oak_tree: 0.0000
Probabilidad de la clase orange: 0.0000
Probabilidad de la clase orchid: 0.0000
Probabilidad de la clase otter: 0.0000
Probabilidad de la clase palm_tree: 0.0000
Probabilidad de la clase pear: 0.0000
Probabilidad de la clase pickup_truck: 1.0000
Probabilidad de la clase pine_tree: 0.0000
Probabilidad de la clase plain: 0.0000
Probabilidad de la clase plate: 0.0000
```

```
Probabilidad de la clase poppy: 0.0000
Probabilidad de la clase porcupine: 0.0000
Probabilidad de la clase possum: 0.0000
Probabilidad de la clase rabbit: 0.0000
Probabilidad de la clase raccoon: 0.0000
Probabilidad de la clase ray: 0.0000
Probabilidad de la clase road: 0.0000
Probabilidad de la clase rocket: 0.0000
Probabilidad de la clase rose: 0.0000
Probabilidad de la clase sea: 0.0000
Probabilidad de la clase seal: 0.0000
Probabilidad de la clase shark: 0.0000
Probabilidad de la clase shrew: 0.0000
Probabilidad de la clase skunk: 0.0000
Probabilidad de la clase skyscraper: 0.0000
Probabilidad de la clase snail: 0.0000
Probabilidad de la clase snake: 0.0000
Probabilidad de la clase spider: 0.0000
Probabilidad de la clase squirrel: 0.0000
Probabilidad de la clase streetcar: 0.0000
Probabilidad de la clase sunflower: 0.0000
Probabilidad de la clase sweet_pepper: 0.0000
Probabilidad de la clase table: 0.0000
Probabilidad de la clase tank: 0.0000
Probabilidad de la clase telephone: 0.0000
Probabilidad de la clase television: 0.0000
Probabilidad de la clase tiger: 0.0000
Probabilidad de la clase tractor: 0.0000
Probabilidad de la clase train: 0.0000
Probabilidad de la clase trout: 0.0000
Probabilidad de la clase tulip: 0.0000
Probabilidad de la clase turtle: 0.0000
Probabilidad de la clase wardrobe: 0.0000
Probabilidad de la clase whale: 0.0000
Probabilidad de la clase willow_tree: 0.0000
Probabilidad de la clase wolf: 0.0000
Probabilidad de la clase woman: 0.0000
Probabilidad de la clase worm: 0.0000
```