Actividad: Redes Neuronales Profundas

Tania Sayuri Guizado Hernández A01640092

Importar TensorFlow

In [2]:
```
# Se cargan las librerias necesarias
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models
```

Descargamos el CIFAR-10 dataset y se prepara para su uso

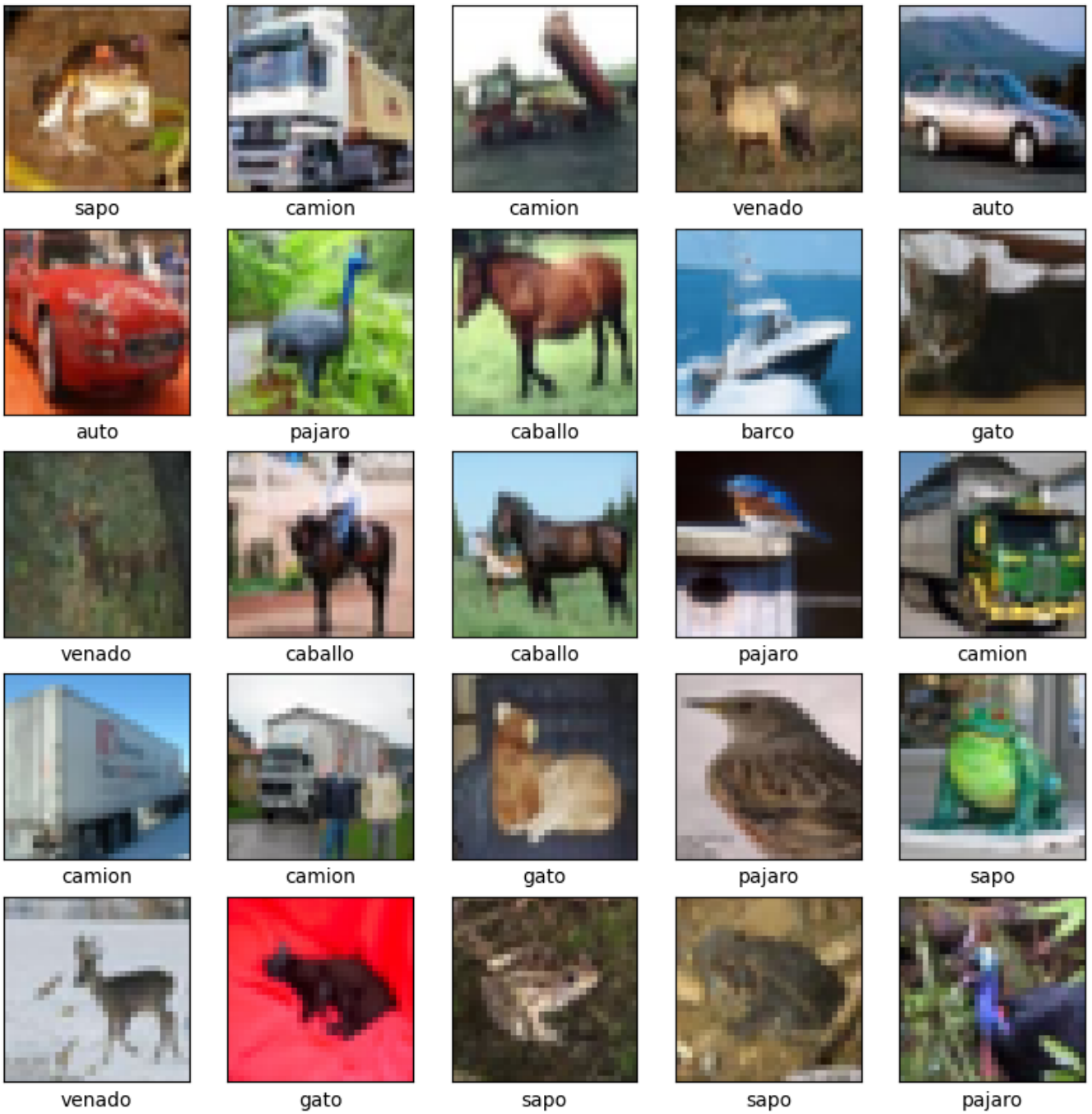
In [3]:
```
# Se cargan los datos de CIFAR10
(train_images, train_labels),(test_images, test_labels) = datasets.cifar10.load_data()

# Normalizamos los valores de los pixeles entre 0 y 1
train_images, test_images = train_images/255.0, test_images/255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

Se crea una función para visualizar las imágenes

In [5]:
```
# Se guarda el nombre de las clases
class_names = ['avion','auto','pajaro','gato','venado','perro','sapo',
               'caballo','barco','camion']
plt.figure(figsize=(10,10))
for i in range(25):
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(train_images[i])
  plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```



Capas de convolución

In [6]:
```
# Se crea la red neuronal
model = models.Sequential()
model.add(layers.Conv2D(64, (3,3), activation='relu', input_shape=(32,32,3)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(256, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(256, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
```

Arquitectura

In [7]:
```
# Visualizamos la arquitectura de la red neuronal
model.summary()
```

```
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, 256)       147712

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 256)         0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 256)         590080

 max_pooling2d_2 (MaxPoolin  (None, 2, 2, 256)         0
 g2D)

=================================================================
Total params: 739584 (2.82 MB)
Trainable params: 739584 (2.82 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Capas densas

In [8]:
```
# Se agregan las capas densas
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(128, activation='sigmoid'))
```

In [9]:
```
# Visualizamos la arquitectura del modelo
model.summary()
```

```
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, 256)       147712

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 256)         0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 256)         590080

 max_pooling2d_2 (MaxPoolin  (None, 2, 2, 256)         0
 g2D)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 128)               131200

 dense_1 (Dense)             (None, 128)               16512

=================================================================
Total params: 887296 (3.38 MB)
Trainable params: 887296 (3.38 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compilación y entrenamiento

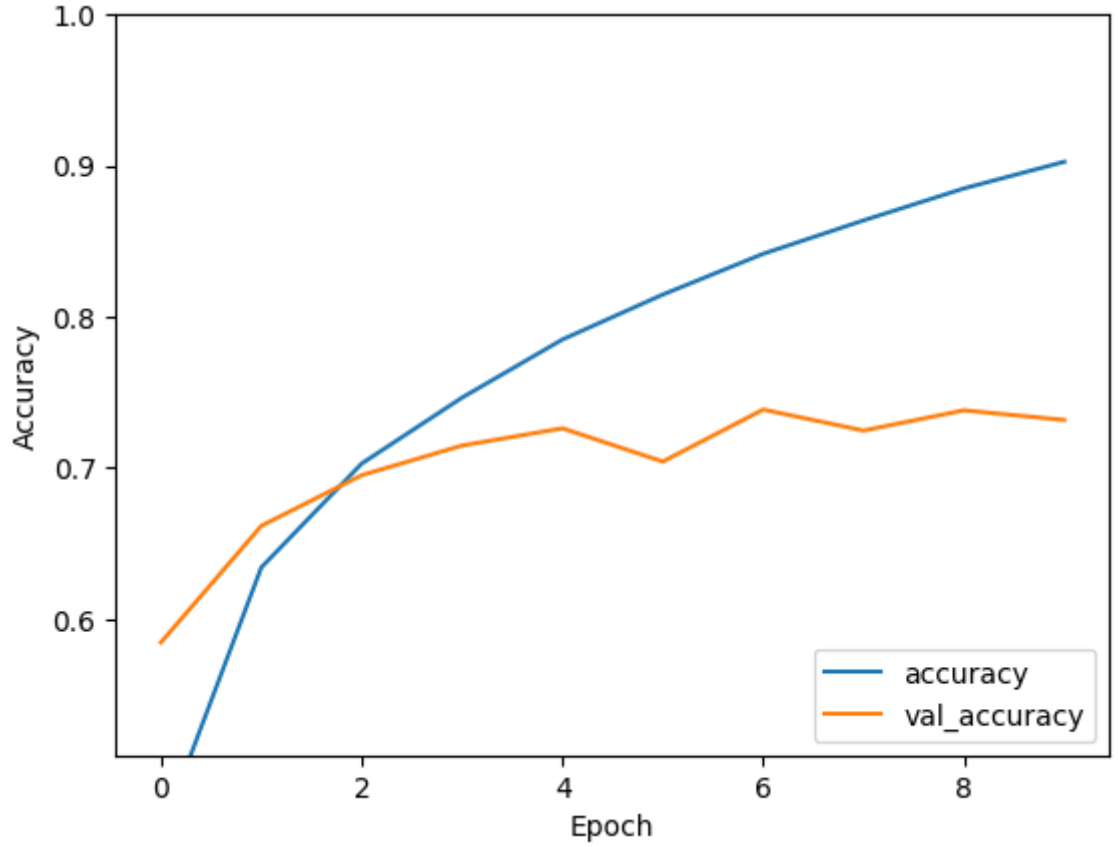
In [10]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images,test_
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/backend.py:5714: UserWarning: "`sparse_cate
gorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by
a Softmax activation and thus does not represent logits. Was this intended?
  output, from_logits = _get_logits(
1563/1563 [==============================] - 317s 200ms/step - loss: 1.5088 - accuracy: 0.457
8 - val_loss: 1.1615 - val_accuracy: 0.5846
Epoch 2/10
1563/1563 [==============================] - 307s 197ms/step - loss: 1.0474 - accuracy: 0.634
2 - val_loss: 0.9670 - val_accuracy: 0.6617
Epoch 3/10
1563/1563 [==============================] - 302s 193ms/step - loss: 0.8600 - accuracy: 0.702
9 - val_loss: 0.8685 - val_accuracy: 0.6952
Epoch 4/10
1563/1563 [==============================] - 308s 197ms/step - loss: 0.7320 - accuracy: 0.746
5 - val_loss: 0.8203 - val_accuracy: 0.7148
Epoch 5/10
1563/1563 [==============================] - 305s 195ms/step - loss: 0.6194 - accuracy: 0.784
9 - val_loss: 0.8275 - val_accuracy: 0.7262
Epoch 6/10
1563/1563 [==============================] - 311s 199ms/step - loss: 0.5318 - accuracy: 0.814
6 - val_loss: 0.9106 - val_accuracy: 0.7042
Epoch 7/10
1563/1563 [==============================] - 307s 196ms/step - loss: 0.4547 - accuracy: 0.841
6 - val_loss: 0.8082 - val_accuracy: 0.7386
Epoch 8/10
1563/1563 [==============================] - 307s 197ms/step - loss: 0.3877 - accuracy: 0.863
7 - val_loss: 0.9435 - val_accuracy: 0.7247
Epoch 9/10
1563/1563 [==============================] - 309s 198ms/step - loss: 0.3298 - accuracy: 0.884
8 - val_loss: 0.8870 - val_accuracy: 0.7381
Epoch 10/10
1563/1563 [==============================] - 310s 199ms/step - loss: 0.2744 - accuracy: 0.902
5 - val_loss: 1.0280 - val_accuracy: 0.7317
```

Evaluacion

In [11]:
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.510,1])
plt.legend(loc='lower right')
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

```
313/313 - 15s - loss: 1.0280 - accuracy: 0.7317 - 15s/epoch - 48ms/step
```



Se imprime el accuracy obteniendo un 0.7317 como resultado

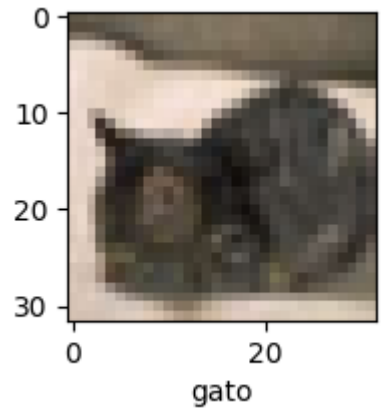
In [12]:
```
print(test_acc)
```

```
0.7317000031471252
```

Predicción

In [13]:
```
n = 115 # Número de imagen

plt.figure(figsize=(2,2))
plt.imshow(test_images[n])
plt.xlabel(class_names[test_labels[n][0]])
plt.show()
```



In [14]:
```
predictions = model.predict(test_images)
print(predictions[n])

import numpy as np
print('La imagen pertenece al grupo {} con una probalididad de {:.2f}%'
      .format(class_names[np.argmax(predictions[n])], 100*np.max(predictions[n])))
```

```
313/313 [==============================] - 16s 49ms/step
[3.30988348e-01 1.16577685e-04 5.82631767e-01 9.69883442e-01
 9.99720991e-01 9.07494545e-01 4.40881878e-01 9.93510008e-01
 7.95677697e-05 9.70939845e-02 4.83510593e-11 6.93124308e-11
 1.51508302e-11 3.25139590e-12 9.65281024e-13 1.50641468e-12
 1.41729315e-12 1.74666520e-12 2.06333345e-12 3.12821609e-11
 5.11755189e-13 4.45048425e-12 2.99983482e-10 1.26959235e-10
 3.08201988e-12 5.44233278e-12 5.84768692e-12 2.09510686e-12
 1.17943308e-10 1.03099803e-11 1.96956302e-13 1.60939502e-12
 7.56235284e-14 5.15883891e-10 1.98884492e-10 1.24566189e-12
 1.69795449e-12 1.63234850e-13 5.50385995e-12 1.14981852e-12
 7.24131042e-13 1.16462439e-12 9.12997282e-12 4.56322317e-13
 1.31783924e-11 2.45881926e-11 1.09288559e-11 1.93267902e-11
 5.93407539e-13 8.38248637e-11 1.61588505e-12 5.33733578e-11
 2.80121629e-12 7.86221609e-12 1.48432551e-11 7.00218269e-12
 2.63367973e-11 3.77690657e-11 1.18020155e-12 9.39516585e-13
 3.56749461e-12 8.35964891e-12 4.67823891e-10 2.20528955e-12
 2.35063271e-12 1.71396189e-10 1.09321645e-12 1.83346556e-13
 3.19521102e-12 1.75570426e-11 1.93865150e-12 2.13940584e-12
 9.92696485e-13 2.28548387e-12 3.91090632e-12 2.52998837e-12
 1.29509884e-11 1.72147661e-11 6.66876346e-11 2.00748130e-14
 2.46465088e-11 7.45092703e-12 3.43555208e-13 1.43677110e-11
 7.85053272e-11 3.94431224e-11 5.82751417e-11 8.26109060e-12
 1.57250927e-12 4.66051586e-10 4.18428774e-13 2.49782318e-13
 1.18623462e-11 1.05715992e-11 1.40072502e-12 2.56888749e-13
 3.47688653e-12 1.18952443e-11 1.93775592e-13 2.98046332e-12
 1.39587245e-10 1.17309623e-12 3.08236076e-12 1.18831577e-11
 2.63732287e-12 1.67557569e-12 3.35080956e-11 4.29429305e-11
 7.90282510e-12 8.22577249e-13 9.61357036e-12 1.52765731e-10
 7.23971230e-12 2.87984653e-12 2.30806070e-13 4.73580593e-13
 6.22339040e-13 1.55178145e-11 1.43306658e-13 8.74147543e-12
 6.73509859e-13 5.66079301e-12 3.29799014e-12 3.70800873e-12
 3.79005800e-13 4.26858988e-13 6.33189638e-12 5.53454309e-13]
La imagen pertenece al grupo venado con una probalididad de 99.97%
```