Importar Tensorflow

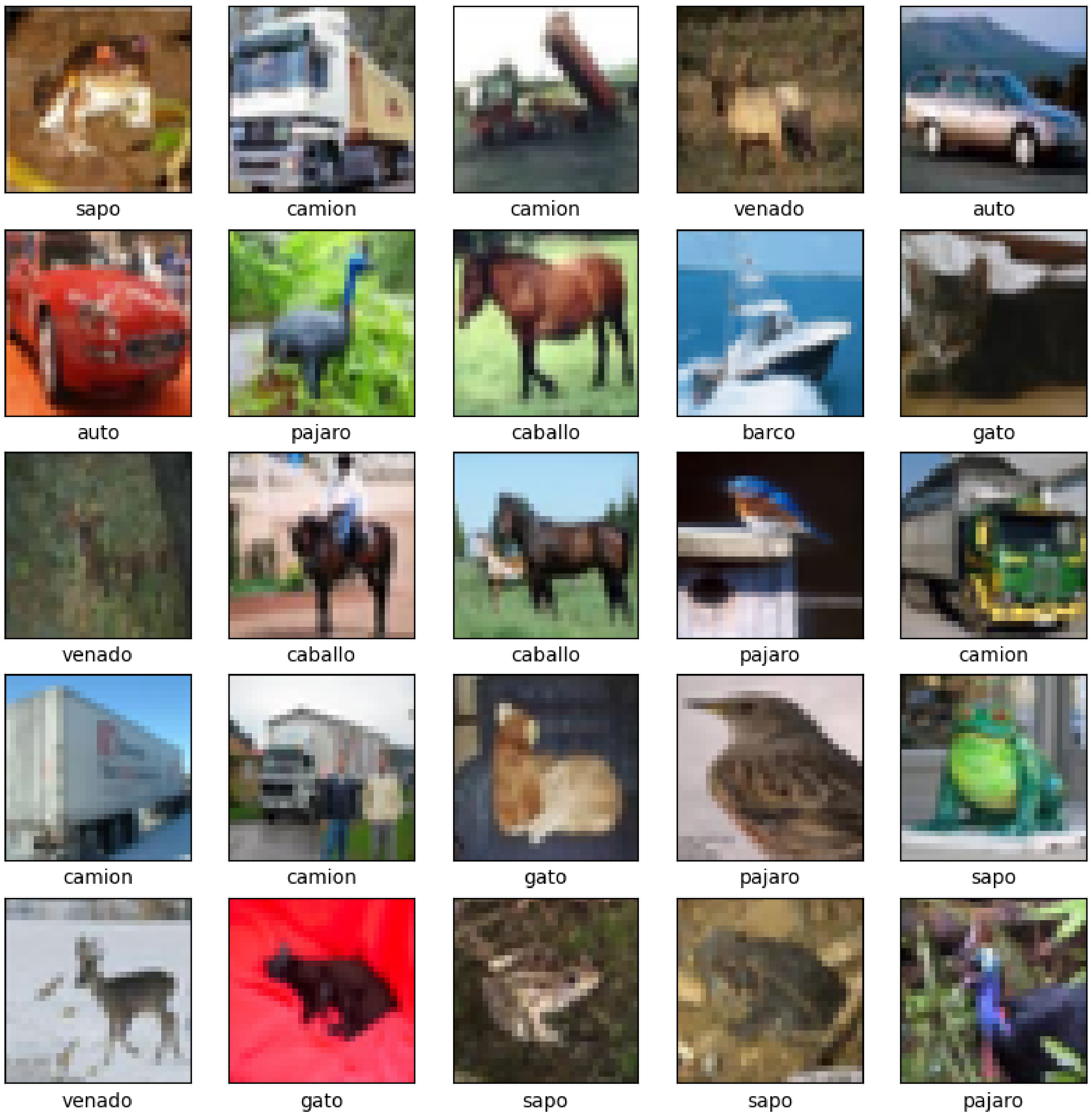
```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
```

Preparar dataset CIFAR-10

```
(train_images, train_labels), (test_images, test_labels) =
datasets.cifar10.load_data()

# Normalizar valores de los pixeles
train_images, test_images = train_images / 255.0, test_images / 255
```

Validación de datos

```
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

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation = "relu", input_shape =
(32, 32, 3)))
model.add(layers.MaxPool2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation = "relu"))
model.add(layers.MaxPool2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation = "relu"))
```

Arquitectura

```
model.summary()

Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_13 (Conv2D)          (None, 30, 30, 32)        896       
                                                                 
 max_pooling2d_9 (MaxPoolin  (None, 15, 15, 32)        0         
 g2D)                                                            
                                                                 
 conv2d_14 (Conv2D)          (None, 13, 13, 64)        18496     
                                                                 
 max_pooling2d_10 (MaxPooli  (None, 6, 6, 64)          0         
 ng2D)                                                           
                                                                 
 conv2d_15 (Conv2D)          (None, 4, 4, 64)          36928     
                                                                 
=================================================================
Total params: 56320 (220.00 KB)
Trainable params: 56320 (220.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Capas Densas

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = "relu"))
model.add(layers.Dense(128, activation = "sigmoid"))

print(model.summary())

Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_10 (Conv2D)          (None, 30, 30, 32)        896       
                                                                 
 max_pooling2d_7 (MaxPoolin  (None, 15, 15, 32)        0         
 g2D)                                                            
                                                                 
 conv2d_11 (Conv2D)          (None, 13, 13, 128)       36992     
                                                                 
 max_pooling2d_8 (MaxPoolin  (None, 6, 6, 128)         0         
 g2D)                                                            
                                                                 
 conv2d_12 (Conv2D)          (None, 4, 4, 64)          73792     
                                                                 
 flatten_2 (Flatten)         (None, 1024)              0         
 dense_4 (Dense)             (None, 64)                65600

 dense_5 (Dense)             (None, 128)               8320

=================================================================
Total params: 185600 (725.00 KB)
Trainable params: 185600 (725.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

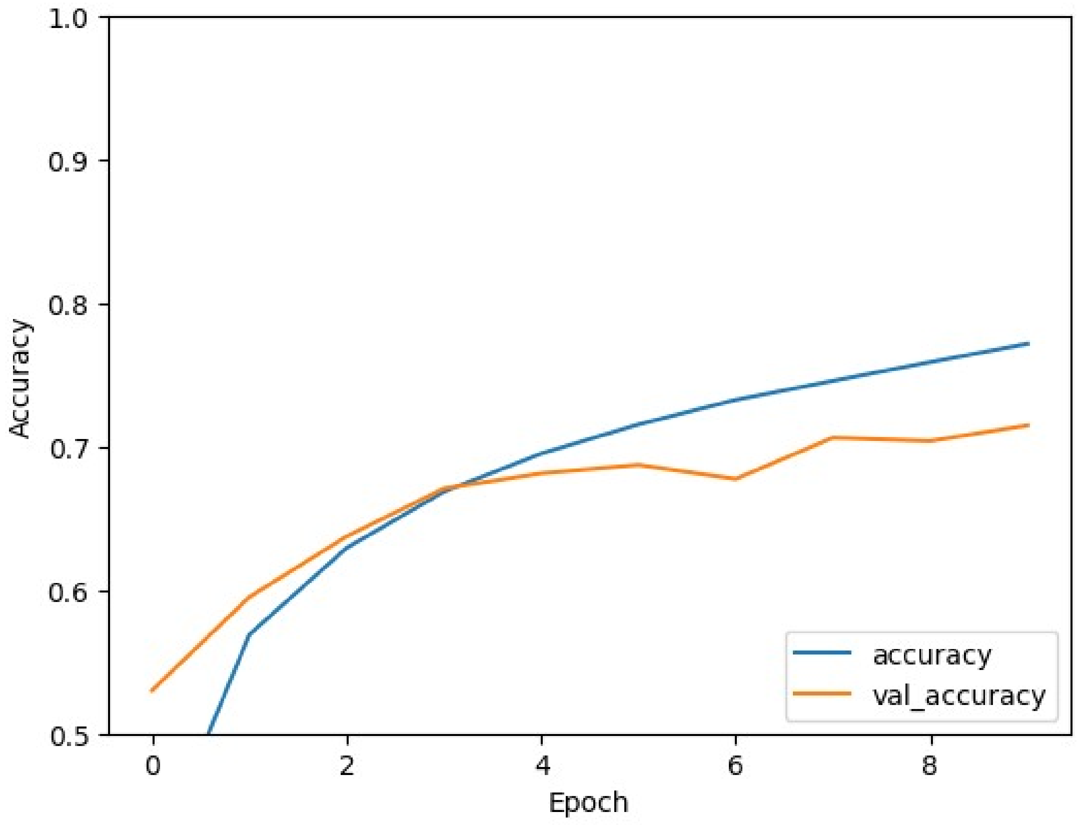
Entrenamiento

```
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_labels))

Epoch 1/10
1563/1563 [==============================] - 43s 27ms/step - loss:
1.6378 - accuracy: 0.4049 - val_loss: 1.2883 - val_accuracy: 0.5302
Epoch 2/10
1563/1563 [==============================] - 43s 27ms/step - loss:
1.2121 - accuracy: 0.5691 - val_loss: 1.1467 - val_accuracy: 0.5952
Epoch 3/10
1563/1563 [==============================] - 41s 26ms/step - loss:
1.0551 - accuracy: 0.6294 - val_loss: 1.0447 - val_accuracy: 0.6375
Epoch 4/10
1563/1563 [==============================] - 42s 27ms/step - loss:
0.9520 - accuracy: 0.6686 - val_loss: 0.9482 - val_accuracy: 0.6708
Epoch 5/10
1563/1563 [==============================] - 41s 26ms/step - loss:
0.8689 - accuracy: 0.6952 - val_loss: 0.9210 - val_accuracy: 0.6814
Epoch 6/10
1563/1563 [==============================] - 42s 27ms/step - loss:
0.8132 - accuracy: 0.7155 - val_loss: 0.9113 - val_accuracy: 0.6873
Epoch 7/10
1563/1563 [==============================] - 41s 26ms/step - loss:
0.7698 - accuracy: 0.7326 - val_loss: 0.9348 - val_accuracy: 0.6775
Epoch 8/10
1563/1563 [==============================] - 42s 27ms/step - loss:
0.7259 - accuracy: 0.7459 - val_loss: 0.8451 - val_accuracy: 0.7063
Epoch 9/10
1563/1563 [==============================] - 41s 26ms/step - loss:
0.6903 - accuracy: 0.7589 - val_loss: 0.8788 - val_accuracy: 0.7040
Epoch 10/10
```

```
1563/1563 [==============================] - 42s 27ms/step - loss:
0.6539 - accuracy: 0.7717 - val_loss: 0.8416 - val_accuracy: 0.7148
```

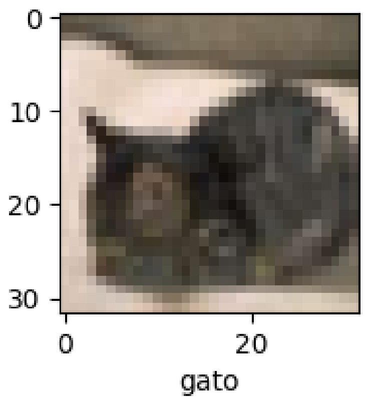
Evaluación

```
plt.plot(history.history["accuracy"], label = "accuracy")
plt.plot(history.history["val_accuracy"], label = "val_accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.ylim([0.5, 1])
plt.legend(loc = "lower right")

<matplotlib.legend.Legend at 0x7e0971f83a90>
```



```
test_loss, test_acc = model.evaluate(test_images, test_labels,
verbose=2)

313/313 - 2s - loss: 0.8416 - accuracy: 0.7148 - 2s/epoch - 8ms/step

print(test_acc)

0.7148000001907349
```

```
n = 115 # Numero de imagen

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



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

print(
    "This image most likely belongs to {} with a {:.2f} percent
confidence."
    .format(class_names[np.argmax(predictions[n])], 10 *
np.max(predictions[n]))
)
```

```
313/313 [==============================] - 4s 12ms/step
[2.35993877e-01 4.03696150e-02 6.47892714e-01 8.80811334e-01
 9.16369438e-01 9.04740453e-01 3.88731778e-01 9.67739999e-01
 2.57964153e-02 5.03297634e-02 6.42513820e-11 1.60369412e-10
 7.93619684e-11 2.36430875e-10 4.42155257e-10 9.37285232e-12
 1.49424001e-10 6.12355860e-11 6.15812262e-11 3.90750834e-11
 1.07563680e-10 1.78046133e-10 2.08712811e-10 4.85814555e-10
 2.02039205e-10 1.65766165e-10 2.87396010e-11 5.18968687e-11
 6.91261423e-11 3.90180353e-11 4.26629079e-11 2.42743187e-10
 1.17524740e-10 1.19827620e-10 2.53165978e-11 1.81729964e-10
 4.84904505e-10 2.22193861e-11 5.03300145e-11 3.65257789e-11
 6.37208619e-11 5.55731294e-11 2.45913342e-11 1.40635906e-10
 8.26304927e-11 1.03493646e-11 7.59270216e-11 8.13908663e-11
 2.93943453e-10 4.90594891e-11 2.08593299e-11 4.56740146e-10
 5.74575248e-11 5.95839766e-11 7.16437049e-11 2.71092621e-10
 1.90286162e-10 2.76456830e-10 1.01022059e-10 2.92624779e-11
 3.80261794e-11 4.24963363e-11 1.76798770e-10 3.07346822e-11
 7.86984228e-11 1.47206192e-10 9.49405485e-11 2.62003308e-10
 2.02719355e-10 7.20294435e-10 7.37907166e-11 7.24278207e-11
 6.09614137e-10 1.03658061e-10 5.56960103e-11 1.23977550e-10
 5.11889454e-11 8.28600077e-10 9.25389348e-11 7.85763329e-11
 1.03500863e-09 5.88725402e-10 9.40158888e-12 4.30220443e-11
 3.34787892e-10 1.28990041e-10 4.11818704e-11 8.28446339e-11
 1.62752631e-10 6.18283036e-10 6.73360326e-11 6.83970727e-11
 3.69676130e-11 2.30476385e-11 4.62877167e-11 2.04910297e-10
 2.84014312e-10 2.12665250e-11 2.16456075e-10 7.41783579e-11
 1.51519366e-10 9.96934549e-11 1.35021411e-10 6.24896940e-11
 5.80519348e-11 3.77433571e-11 3.14681753e-10 9.32733224e-10
 1.08987194e-10 1.77576759e-10 6.18983920e-10 1.81736556e-10
 2.09145860e-11 2.07335205e-10 2.61161474e-11 4.73099025e-11
 4.70318021e-11 1.07029947e-10 6.70272171e-11 2.91384639e-10
 4.12149516e-11 1.17654816e-10 2.74740564e-10 3.17497660e-11
 1.12745119e-10 5.79436360e-11 1.19193239e-10 6.66533009e-11]
This image most likely belongs to caballo with a 9.68 percent
confidence.
```