Redes neuronales profundas

Ernesto Reynoso Lizárraga A01639915

Importar Tensorflow

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

Preparar dataset

Exportamos los datos desde keras y los normalizamos

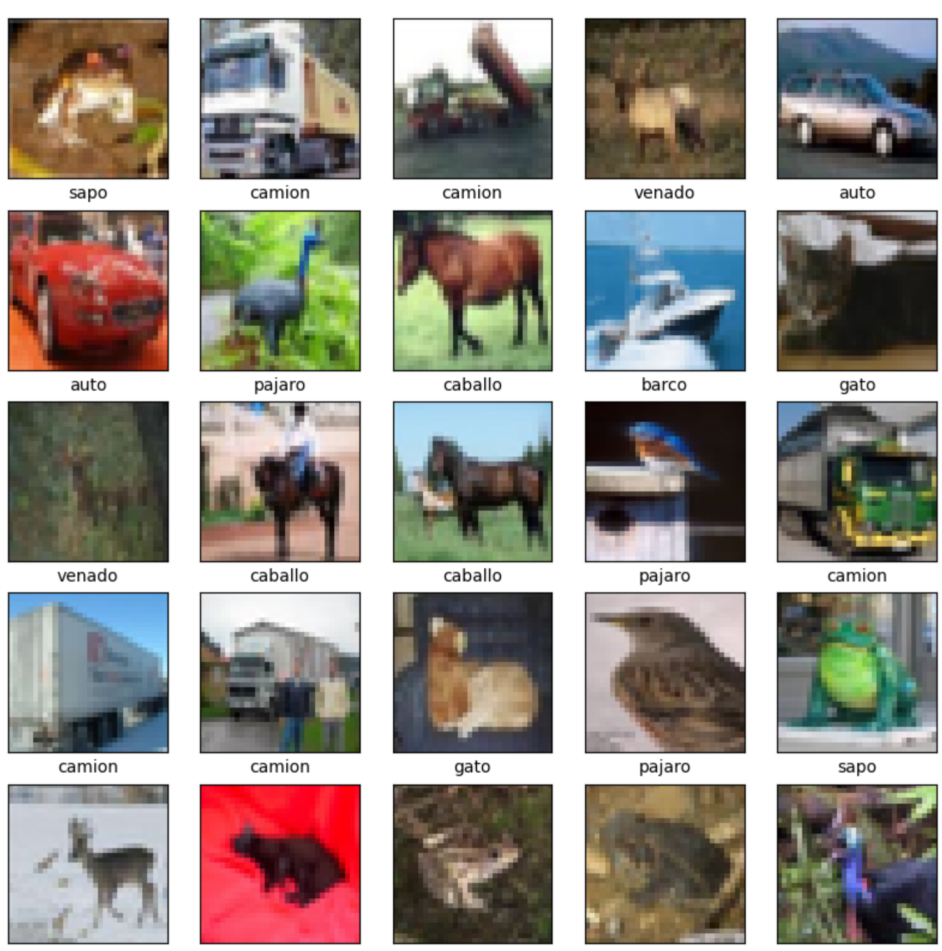
```
(train_images, train_labels),(test_images, test_labels) = datasets.cifar10.load_data()

#Normalizar
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
```

Validación de datos

Establecemos los nombres de las clases y los relacionamos con las imagenes de la base 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 Convolucion

```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu', input_shape=(32,32,3)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 128)       36992

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 128)         0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 128)         147584

=================================================================
Total params: 185472 (724.50 KB)
Trainable params: 185472 (724.50 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(10, activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 128)       36992

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 128)         0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 128)         147584

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 64)                131136

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 317258 (1.21 MB)
Trainable params: 317258 (1.21 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Entrenamiento

entrenamos el modelo midiendo la precision del mismo y establecemos 10 epocas

```
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))
```

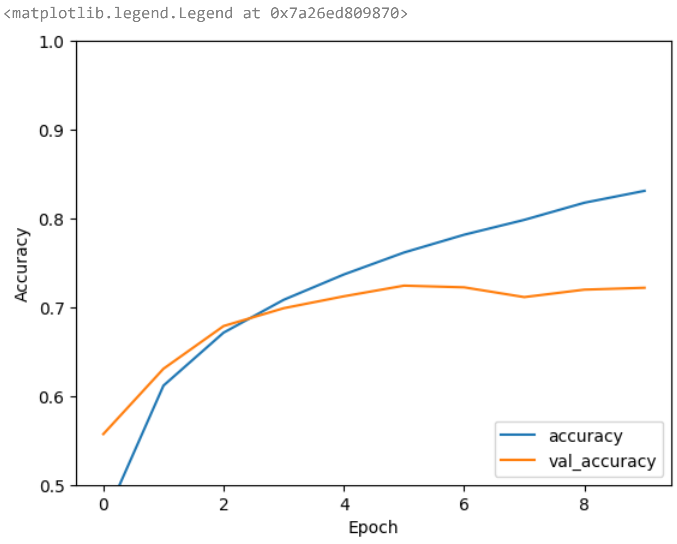
```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/backend.py:5714: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a Softmax activati
  output, from_logits = _get_logits(
1563/1563 [==============================] - 127s 80ms/step - loss: 1.4923 - accuracy: 0.4585 - val_loss: 1.2404 - val_accuracy: 0.5571
Epoch 2/10
1563/1563 [==============================] - 123s 78ms/step - loss: 1.0979 - accuracy: 0.6117 - val_loss: 1.0554 - val_accuracy: 0.6307
Epoch 3/10
1563/1563 [==============================] - 121s 77ms/step - loss: 0.9378 - accuracy: 0.6714 - val_loss: 0.9201 - val_accuracy: 0.6788
Epoch 4/10
1563/1563 [==============================] - 129s 83ms/step - loss: 0.8346 - accuracy: 0.7084 - val_loss: 0.8883 - val_accuracy: 0.6989
Epoch 5/10
1563/1563 [==============================] - 123s 79ms/step - loss: 0.7559 - accuracy: 0.7369 - val_loss: 0.8466 - val_accuracy: 0.7123
Epoch 6/10
1563/1563 [==============================] - 123s 79ms/step - loss: 0.6878 - accuracy: 0.7616 - val_loss: 0.8260 - val_accuracy: 0.7243
Epoch 7/10
1563/1563 [==============================] - 122s 78ms/step - loss: 0.6269 - accuracy: 0.7817 - val_loss: 0.8371 - val_accuracy: 0.7224
Epoch 8/10
1563/1563 [==============================] - 120s 77ms/step - loss: 0.5746 - accuracy: 0.7984 - val_loss: 0.8413 - val_accuracy: 0.7114
Epoch 9/10
1563/1563 [==============================] - 120s 77ms/step - loss: 0.5218 - accuracy: 0.8177 - val_loss: 0.8396 - val_accuracy: 0.7198
Epoch 10/10
1563/1563 [==============================] - 125s 80ms/step - loss: 0.4822 - accuracy: 0.8311 - val_loss: 0.8620 - val_accuracy: 0.7219
```

Evaluacion

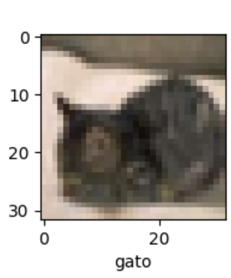
Visualizamos los resultados de las epocas y ponemos a prueba el modelo intentando predecir el nombre de una imagen en concreto

```
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 0x7a26ed809870>
```




```
n = 115

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])

import numpy as np

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

```
313/313 [==============================] - 7s 23ms/step
[2.2583282e-02 4.5385532e-05 7.7638221e-01 4.4223192e-01 5.9436804e-01
 5.5142683e-01 1.8418372e-02 6.2163311e-01 3.1931014e-04 1.6456782e-03]
La imagen pertenece al grupo pajaro con una probabilidad de 77.638221%
```