Actividad Redes Neuronales Profundas - Problema 1

Andrés Alejandro Guzmáz González - A01633819

Llamado a librerías

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

Carga de datos

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

Proceso de normalización de las imágenes

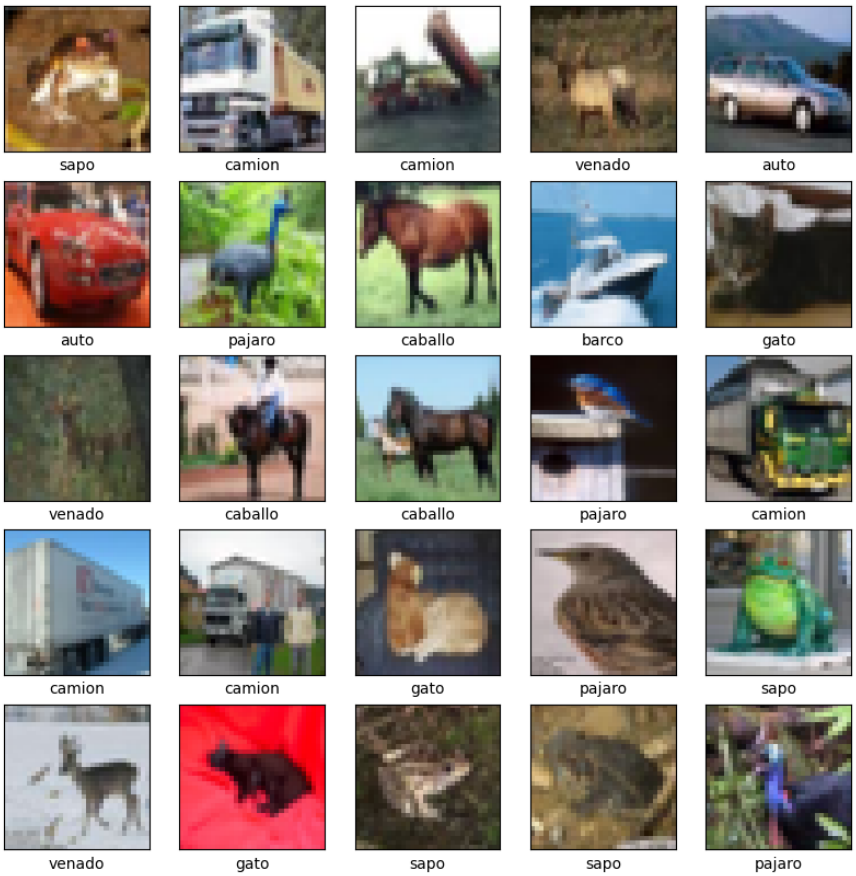
```
train_images, test_images = train_images/255.0, test_images/255.0
```

```
# Validación de respuestas "Labels"
print(train_labels)
```

```
[[6]
 [9]
 [9]
 ...
 [9]
 [1]
 [1]]
```

Mostramos un grid con el ejemplo de las imágenes

```
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

```
# Agrego las primeras 3 capas de convolución
model = models.Sequential()
# En la primer capa es necesario definir el tamaño de entrada de las imágenes
model.add(layers.Conv2D(64, (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'))
```

Arquitectura

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 30, 30, 64)        1792

 max_pooling2d_4 (MaxPoolin  (None, 15, 15, 64)        0
 g2D)

 conv2d_7 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_5 (MaxPoolin  (None, 6, 6, 128)         0
 g2D)

 conv2d_8 (Conv2D)           (None, 4, 4, 128)         147584

=================================================================
Total params: 223232 (872.00 KB)
Trainable params: 223232 (872.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Capas densas

```
# Se agregan las capas densas a la red
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(20, activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 30, 30, 64)        1792

 max_pooling2d_4 (MaxPoolin  (None, 15, 15, 64)        0
 g2D)

 conv2d_7 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_5 (MaxPoolin  (None, 6, 6, 128)         0
 g2D)

 conv2d_8 (Conv2D)           (None, 4, 4, 128)         147584

 flatten_5 (Flatten)         (None, 2048)              0

 dense_10 (Dense)            (None, 128)               262272

 dense_11 (Dense)            (None, 20)                2580

=================================================================
Total params: 488084 (1.86 MB)
Trainable params: 488084 (1.86 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
'''
Se procede a hacer el entrenaiento de la red y se definen la función de
optimización, la función de pérdida y las épocas de 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))
```

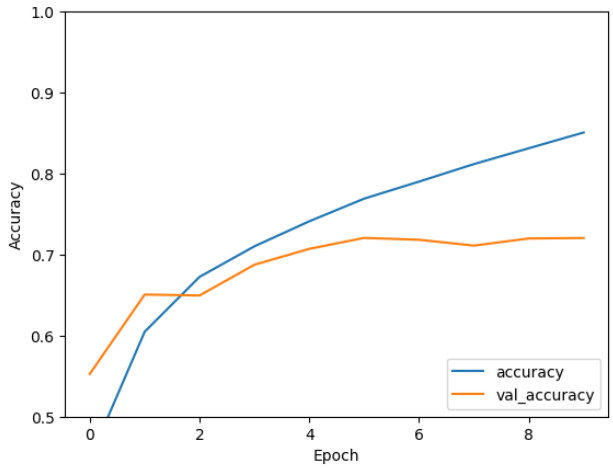
```
Epoch 1/10
1563/1563 [==============================] - 197s 125ms/step - loss: 1.5172 - accuracy: 0.4476 - val_loss: 1.2468 - val_accuracy: 0.5527
Epoch 2/10
1563/1563 [==============================] - 194s 124ms/step - loss: 1.1149 - accuracy: 0.6048 - val_loss: 1.0044 - val_accuracy: 0.6507
Epoch 3/10
1563/1563 [==============================] - 197s 126ms/step - loss: 0.9383 - accuracy: 0.6725 - val_loss: 1.0235 - val_accuracy: 0.6495
Epoch 4/10
1563/1563 [==============================] - 191s 122ms/step - loss: 0.8298 - accuracy: 0.7102 - val_loss: 0.9076 - val_accuracy: 0.6875
Epoch 5/10
1563/1563 [==============================] - 192s 123ms/step - loss: 0.7389 - accuracy: 0.7411 - val_loss: 0.8706 - val_accuracy: 0.7071
```

```
Epoch 6/10
1563/1563 [==============================] - 195s 125ms/step - loss: 0.6572 - accuracy: 0.7691 - val_loss: 0.8304 - val_accuracy: 0.7206
Epoch 7/10
1563/1563 [==============================] - 192s 123ms/step - loss: 0.5939 - accuracy: 0.7901 - val_loss: 0.8572 - val_accuracy: 0.7183
Epoch 8/10
1563/1563 [==============================] - 214s 137ms/step - loss: 0.5315 - accuracy: 0.8117 - val_loss: 0.9084 - val_accuracy: 0.7111
Epoch 9/10
1563/1563 [==============================] - 228s 146ms/step - loss: 0.4738 - accuracy: 0.8313 - val_loss: 0.9246 - val_accuracy: 0.7199
Epoch 10/10
1563/1563 [==============================] - 205s 131ms/step - loss: 0.4178 - accuracy: 0.8507 - val_loss: 0.9411 - val_accuracy: 0.7205
```

Evaluacion

```
# Gráfica para mostrar el accuracy de la red en cada época
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 0x7a13bed4fd60>
```

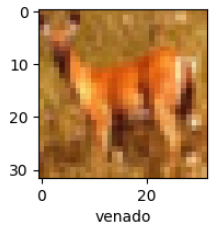


Predicción

Pruba con un dato random del conjunto de imágenes para validar si la red lo clasifica correctamente

```
n = 110 # Número 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])

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 [==============================] - 19s 61ms/step
[5.4086104e-02 1.0279529e-05 8.9884537e-01 9.9181402e-01 9.9862134e-01
 9.8938763e-01 6.4045197e-01 9.9489468e-01 2.1216829e-01 2.7005994e-03
 1.8875913e-07 2.5557863e-08 5.8199252e-08 4.9177586e-09 3.3470313e-09
 1.0095552e-08 1.4103871e-08 3.0464664e-09 1.3154530e-08 2.8931135e-09]
La imagen pertenece al grupo venado con una probalididad de 99.86%
```