Actividad Redes Neuronales Profundas Ejercicio 1

Mario Alberto Castañeda Martínez - A01640152

En este ejercicio, se tomarán diferentes acciones para mejorar el accuracy del modelo generado en clase (70%). Se explicarán las modificaciones que se hagan en el código para mejorar el accuracy.

Se importan las librerías necesarias para generar el modelo, además para generar gráficas y probar el modelo.

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

Se prepara el dataset Cifar10 y se normaliza:

```
#prepara dataset cifar10
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
#normalizar
train_images, test_images = train_images / 255.0, test_images/255
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
```

Se realiza una validación de los datos:

```
#validacion de datos
class_names = ['avion', 'auto', 'pajaro', 'gato', 'venado', 'perro', 'sapo', 'Sneaker', 'barco', 'camion']
plt.figure(figsize=(10,10))
for i in range (10):
  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()
```

Se comienza por agregar tres capas de convolución, utilizando Conv2D para realizar convoluciones con un conjunto de filtros con tamaño 3x3, además de utilizar la función de activación Relu. También se utiliza MaxPooling2D para tomar un valor máximo en una región 2x2, reduciendo la imagen.

En este caso la modificación que realizo al código realizado en clase, es que aquí agrego 128 flitros en la tercera capa de convolución.

```
#Capas de convolución
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(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu')) #se agregan 128 filtros
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 128)         73856

=================================================================
Total params: 93248 (364.25 KB)
Trainable params: 93248 (364.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

En esta etapa de la construcción del modelo, se empieza por aplanar la salida de las capas convolucionales, para después agregar dos capas densas, en este caso para mejorar el accuracy del modelo, se agregaron 512 neuronas con activación Relu, al final se tiene una capa de salida de 10 neuronas con activación sigmoide.

```
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu')) #se agregan 512 neuronas
model.add(layers.Dense(10, activation ='sigmoid'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 128)         73856

 flatten_2 (Flatten)         (None, 2048)              0

 dense_4 (Dense)             (None, 512)               1049088

 dense_5 (Dense)             (None, 10)                5130

=================================================================
Total params: 1147466 (4.38 MB)
Trainable params: 1147466 (4.38 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Proceso de entrenamiento del modelo.

Se compila el modelo, añadiendo adam para ajustar pesos de la red, se establece una función de pérdida. Finalmente se entrena el modelo con 10 épocas y el conjunto de prueba.

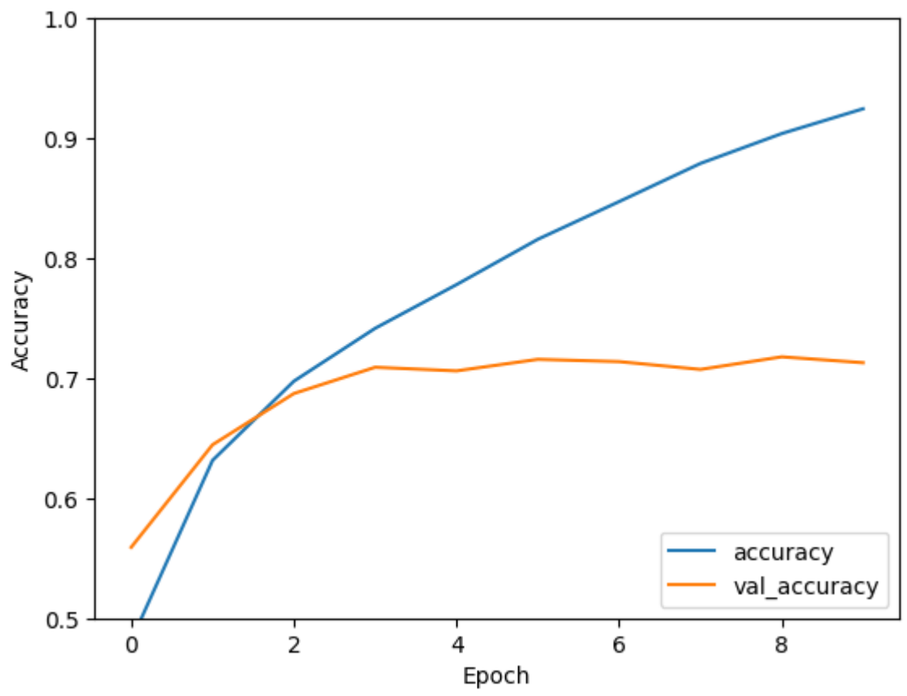
```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 105s 64ms/step - loss: 1.4356 - accuracy: 0.4811 - val_loss: 1.2242 - val_accuracy: 0.5591
Epoch 2/10
1563/1563 [==============================] - 95s 61ms/step - loss: 1.0393 - accuracy: 0.6315 - val_loss: 1.0049 - val_accuracy: 0.6445
Epoch 3/10
1563/1563 [==============================] - 96s 61ms/step - loss: 0.8608 - accuracy: 0.6975 - val_loss: 0.8967 - val_accuracy: 0.6872
Epoch 4/10
1563/1563 [==============================] - 99s 63ms/step - loss: 0.7338 - accuracy: 0.7416 - val_loss: 0.8330 - val_accuracy: 0.7091
Epoch 5/10
1563/1563 [==============================] - 94s 60ms/step - loss: 0.6288 - accuracy: 0.7780 - val_loss: 0.8781 - val_accuracy: 0.7061
Epoch 6/10
1563/1563 [==============================] - 95s 61ms/step - loss: 0.5264 - accuracy: 0.8157 - val_loss: 0.8690 - val_accuracy: 0.7157
Epoch 7/10
1563/1563 [==============================] - 97s 62ms/step - loss: 0.4329 - accuracy: 0.8472 - val_loss: 0.9090 - val_accuracy: 0.7138
Epoch 8/10
1563/1563 [==============================] - 94s 60ms/step - loss: 0.3436 - accuracy: 0.8790 - val_loss: 1.0125 - val_accuracy: 0.7073
Epoch 9/10
1563/1563 [==============================] - 100s 64ms/step - loss: 0.2732 - accuracy: 0.9039 - val_loss: 1.0795 - val_accuracy: 0.7178
Epoch 10/10
1563/1563 [==============================] - 95s 61ms/step - loss: 0.2122 - accuracy: 0.9245 - val_loss: 1.1968 - val_accuracy: 0.7129
```

Se evalúa el modelo generado, donde se obtiene una gran exactitud en el entrenamiento y una exactitud un poco mejor para la prueba con nuevos datos:

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

```
313/313 - 4s - loss: 1.1968 - accuracy: 0.7129 - 4s/epoch - 13ms/step
```



Se obtiene la siguiente exactitud que fue un poco superior a la generada en clase:

```
print(test_acc)
```

```
0.7128999829292297
```

El accuracy del modelo, fue del 71%

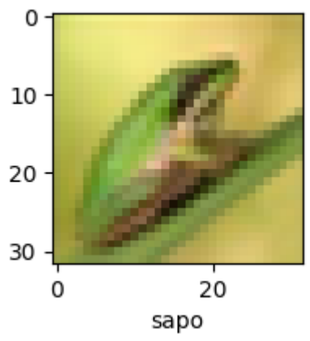
A pesar de que se realizaron muchas y diferentes modificaciones al modelo (añadir más filtros, agregar más, o menos neuronas y cambiar el número de épocas), la configuración actual del modelo fue la que mejor exactitud (val accuracy) arrojó para la base de datos cifar10.

+ Código + Texto

Se realiza ahora una predicción para probar el modelo con la imagen de un sapo:

```
n=112 ##Number of image
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])], 100 * np.max(predictions[n])))
```

```
313/313 [==============================] - 6s 18ms/step
[5.4938137e-03 1.3937216e-05 9.9341846e-01 9.5878798e-01 1.3188359e-01
 1.3502583e-01 9.9887335e-01 4.6028399e-05 1.0647287e-06 1.5411276e-06]
This image most likely belongs to sapo with a 99.89 percent confidence.
```

El modelo pudo decir correctamente la imagen mostrada.