En la práctica la carga de modelos no sucede de manera atómica (no se carga solo los pesos o solo la arquitectura), lo hacemos todo a la vez, tampoco almacenamos todos los modelos, únicamente guardamos en disco los mejores desempeños. Vamos al código para entender cómo lograrlo.

**Mejorando el guardado de pesos**

Definiremos un path para guardar nuestro nuevo modelo. En el ModelCheckpoint haremos una serie de modificaciones que cambiarán el cómo se guardará el modelo. El primero de ellos será la opción de save\_weight\_only, esta irá a falso, también agregaremos val\_accuracy como monitor de desempeño y solo guardaremos la mejor versión del modelo.

```python checkpoint\_path = "model\_checkpoints\_complete"

checkpoint\_weight = ModelCheckpoint( filepath = checkpoint\_path, frecuency = "epoch", save\_weight\_only = False, monitor = "val\_accuracy", save\_best\_only = True, verbose = 1 ) ```

Crearemos un modelo nuevo para comprender cómo se guardan los datos.

```python model\_complete = get\_model() model\_complete.compile(optimizer = "adam", loss = "categorical\_crossentropy", metrics = ["accuracy"])

history\_complete = model\_complete.fit( train\_generator, epochs = 20, callbacks = [checkpoint\_weight], validation\_data = validation\_generator ) ```

Si durante el entrenamiento la red mejora en su val\_accuracy, entonces se guardará en disco, si por el contrario no detecta una mejora, entonces ignorará esa iteración.

python Epoch 1: val\_accuracy improved from -inf to 0.77614, saving model to model\_checkpoints\_complete Epoch 8: val\_accuracy did not improve from 0.86175

Al final obtendremos de output un directorio con varios archivos, asegúrate de guardarlos todos de manera local, dado que si falta alguno la carga del modelo fallará.

Si deseas guardar la configuración entera manualmente, puedes hacer uso del método save de los modelos.

python model\_complete.save("saved\_model\_complete/mymodel")

**Cargando modelos completos**

Para cargar un modelo completo desde disco sin necesidad de crear una arquitectura vacía podemos usar la función load\_model del módulo models de Keras indicando la ubicación del directorio.

python model\_complete3 = tf.keras.models.load\_model("saved\_model\_complete/mymodel")

Si comparamos el rendimiento de test entre ambos modelos encontraremos que son exactamente el mismo, reflejando que la carga ha funcionado.

```python model\_complete3.evaluate(test\_generator) 57/57 [==============================] - 2s 41ms/step - loss: 0.6733 - accuracy: 0.8826 [0.6733485460281372, 0.8825989961624146]

model\_complete.evaluate(test\_generator) 57/57 [==============================] - 2s 41ms/step - loss: 0.6733 - accuracy: 0.8826 [0.673348605632782, 0.8825989961624146] ```

**Carga y descarga desde archivos h5**

Podemos guardar configuraciones enteras desde un archivo h5 que sintetizará toda la estructura de directorios vista anteriormente, para poder usarla debemos instalar pyyaml y h5py.

python !pip install pyyaml h5py

Para guardar archivos como h5 bastará con modificar la extensión al final de la dirección a guardar. Para cargarlo bastará con indicar el archivo con extensión.

python model\_complete3.save("my\_model.h5") model\_complete4 = tf.keras.models.load\_model("my\_model.h5")

Con esto ya tienes un amplio abanico para guardar y cargar modelos, desde la sola arquitectura hasta un historial completo o seleccionado de generaciones de entrenamiento.

**Contribución creada por** Sebastián Franco Gómez.