Hasta ahora solo hemos usado explícitamente modelos secuenciales (donde las capas conectan directa y linealmente), para esta ocasión estamos manejando un modelo funcional, por lo que la conexión de capas debe ser explícita.

**Conectando las nuevas capas**

Anteriormente cortamos la configuración de InceptionV3 en el mixed7 para considerarlo nuestra última capa no entrenable.

last\_layers = pre\_trained\_model.get\_layer("mixed7")

last\_output = last\_layers.output

Ahora la conectaremos con nuestras capas personalizadas. Recuerda que como manejamos un modelo funcional, debemos expresar explícitamente desde cuál capa viene conectada, esto lo haremos con la notación de paréntesis al final de la definición de cada objeto.

Aplanaremos todos los datos desde una capa flatten, agregaremos una capa densa de 128 neuronas con activación relu y un dropout de 20%, finalmente definiremos la nueva salida con 24 neuronas de activación softmax (la configuración original trae 1000 posibles outputs pero podemos personalizarla a voluntad).

Con todas las capas conectadas, definiremos el modelo final con la clase Model de Keras, esta recibirá el inicio y el final del modelo.

x = tf.keras.layers.Flatten()(last\_output)

x = tf.keras.layers.Dense(128, activation = "relu")(x)

x = tf.keras.layers.Dropout(0.2)(x)

x = tf.keras.layers.Dense(len(classes), activation = "softmax")(x)

model\_keras = tf.keras.Model(pre\_trained\_model.input, x)

model\_keras.compile(optimizer = "adam", loss = "categorical\_crossentropy", metrics = ["accuracy"])

model\_keras.summary()

Al compilar y resumir el modelo podemos notar que todas las capas después de mixed7 no existen, además de que la cantidad de parámetros bajó de casi 22 millones a casi 14, además solo 5 son entrenables (lo que reducirá la complejidad del modelo).

Total **params**: 13,795,384

Trainable **params**: 4,820,120

Non-trainable **params**: 8,975,264

**Entrenando el nuevo modelo**

Entrenaremos el nuevo modelo con los nuevos generadores durante 5 épocas. Notarás que el entrenamiento durará mucho más tiempo pero no tendremos que crear las abstracciones del modelo desde 0.

history\_keras = model\_keras.fit(

train\_generator\_resize,

epochs = 5,

validation\_data = validation\_generator\_resize

)

Podemos notar que incluso después de la primer iteración el accuracy y val\_accuracy mantuvieron un tiempo de entrenamiento excelente, donde en la épica final tuvimos una aproximación total en entrenamiento, una pérdida mínima y una precisión en validación del 98.95%.

Epoch 1/5

215/215 [==============================] - 37s 118ms/step - loss: 0.1648 - accuracy: 0.9545 - val\_loss: 0.0509 - val\_accuracy: 0.9832

Epoch 5/5

215/215 [==============================] - 24s 111ms/step - loss: 9.9922e-04 - accuracy: 0.9999 - val\_loss: 0.0385 - val\_accuracy: 0.9895

Para corroborar los resultados de validación evaluaremos el modelo en el subset de pruebas.

model\_keras.evaluate(test\_generator\_resize)

57/57 [==============================] - 7s 119ms/step - loss: 0.0463 - accuracy: 0.9865

[0.04633770138025284, 0.9864751696586609]

Obtuvimos una precisión de más del 98% en 8597 imágenes, un rendimiento más que aceptable para nuestro modelo.

**Contribución creada por** Sebastián Franco Gómez.