Para esta ocasión importaremos modelos desde TensorFlow Hub, una plataforma dedicada a la divulgación de configuraciones por parte de la comunidad.

**Tensorflow Hub con Keras**

Para usar el hub, debemos importarlo.

**import** tensorflow\_hub **as** hub

Antes de todo, debemos detectar qué configuración usaremos, para esta ocasión usaremos la arquitectura [MobileNetV1](https://platzi.com/clases/2565-redes-neuronales-tensorflow/42860-uso-sistemas-pre-entrenados-de-tensorflow-hub/%5Bhttps:/tfhub.dev/google/imagenet/mobilenet_v1_050_160/classification/4%5D(https:/tfhub.dev/google/imagenet/mobilenet_v1_050_160/classification/4)).

Este modelo es secuencial, por lo que stackearemos sus capas en un modelo de este tipo.

Para los modelos del hub, basta con agregar una capa de entrada, añadir la capa de KerasLayer con la URL del sitio (no olvides configurarlo como no entrenable) y a partir de este punto podrás inyectar tu propia arquitectura.

module\_url = "https://tfhub.dev/google/imagenet/mobilenet\_v1\_050\_160/classification/4"

model\_hub = tf.keras.Sequential([

tf.keras.layers.InputLayer(input\_shape = (150, 150, 3)),

hub.KerasLayer(module\_url, trainable = False),

tf.keras.layers.Flatten(),

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

tf.keras.layers.Dropout(rate = 0.2),

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

])

Antes de compilar la red debemos hacer build al modelo, siendo explícitos en las dimensiones de su tensor, en este caso MobileNet soporta frames de video, por lo que para usarlo como detector de imágenes bastará con anular esta dimensión.

**model\_hub**.build((**None**, 150, 150, 3))

**model\_hub**.summary()

Podemos notar en la arquitectura que se añade toda la capa cargada y posteriormente nuestra arquitectura.

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Layer (type) Output Shape Param #**

**=================================================================**

keras\_layer\_2 (KerasLayer) (None, 1001) 1343049

flatten\_2 (Flatten) (None, 1001) 0

dense (Dense) (None, 128) 128256

dropout (Dropout) (None, 128) 0

dense\_1 (Dense) (None, 24) 3096

**=================================================================**

Total params: 1,474,401

Trainable params: 131,352

Non-trainable params: 1,343,049

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

El proceso de compilación y entrenamiento será el tradicional.

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

history\_hub = model\_hub.fit(

train\_generator\_resize,

epochs = 5,

validation\_data = validation\_generator\_resize

)

Podemos notar que el rendimiento es menor al de Inception, pero en contraparte su tiempo de procesamiento es menor.

57/57 [==============================] - 5s 91ms/step - loss: 0.4389 - accuracy: 0.8461

Este tipo de modelos son útiles a la hora de procesar imágenes en vivo (como en cámaras de drones). Pero traen en consecuencias una pérdida de precisión, dependerá de tu proyecto aceptar estas alternativas o sacrificar tiempo de cómputo por mejores resultados.

**Contribución creada por** Sebastián Franco Gómez.