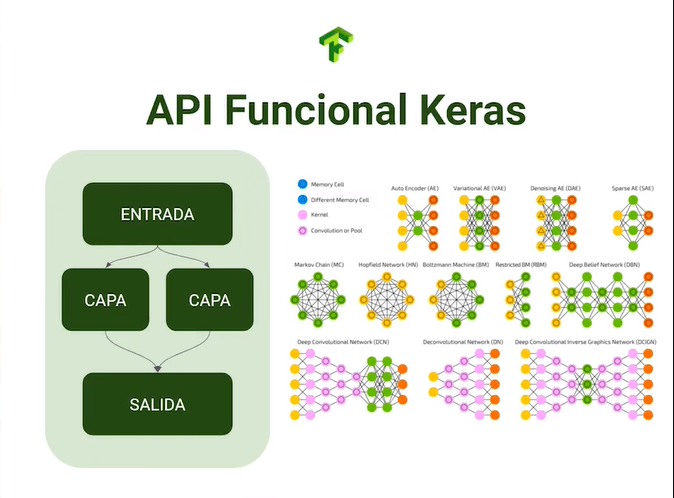
Existen múltiples métodos para cargar sistemas preentrenados, para esta ocasión interiorizaremos en el de Keras.

Hasta ahora hemos trabajado con la API secuencial de Keras que nos permite apilar capas de manera lineal, su recorrido es unidireccional y siempre el mismo. La API Funcional de Keras rompe este paradigma y nos permite configurar redes a gusto, donde los saltos de capas pueden ser alterados y se pueden implementar configuraciones infinitamente más complejas.



Estos modelos funcionales pueden dar mejores resultados que sus homólogos secuenciales, por lo que indagaremos como cargarlos y usarlos desde Keras.

**Usando la API funcional de Keras**

Antes de cargar la configuración es de vital importancia reconocer las dimensiones de entrada del modelo a usar, esto es especialmente necesario en contextos de detección de objetos y transfer learning.

No importa si nuestro dataset consta de imágenes en 4k, si el modelo espera un input de 300x300 debemos redimensionarlo, inclusive si perdemos features.

Para esta ocasión usaremos la arquitectura inception V3 que podrás encontrar [en la documentación de Keras](https://platzi.com/clases/2565-redes-neuronales-tensorflow/43032-carga-de-sistemas-pre-entrenados-en-keras/%5Bhttps:/keras.io/api/applications/%5D(https:/keras.io/api/applications/)), puedes notar que sus dimensiones de entrada son (150, 150, 3) y nuestras imágenes son (28, 28, 1), por lo que debemos cambiar su tamaño y escala de colores (escalar imágenes hacia arriba no es recomendable dada la pérdida de información, por lo que siempre deberías buscar un modelo que se adapte a las dimensiones de tu dataset).

Crearemos nuevos data generators, donde modificaremos el target size y el color mode. Esto será aplicado a los 3 generadores.

train\_generator\_resize = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size = (150,150),

batch\_size = 128,

class\_mode = "categorical",

color\_mode = "rgb",

subset = "training"

)

validation\_generator\_resize = test\_datagen.flow\_from\_directory(

test\_dir,

target\_size = (150,150),

batch\_size = 128,

class\_mode = "categorical",

color\_mode = "rgb",

subset = "validation"

)

test\_generator\_resize = train\_datagen.flow\_from\_directory(

test\_dir,

target\_size = (150,150),

batch\_size = 128,

class\_mode = "categorical",

color\_mode = "rgb",

)

Ahora que los datos de entrada son compatibles, podremos importar la configuración desde Keras.

from tensorflow.keras.applications.inception\_v3 import InceptionV3

Cargaremos nuestro modelo en la variable pre\_trained\_model que será una instancia de la clase InceptionV3, esta tendrá como parámetros un include\_top a falso y el tensor de entrada que será de 150x150x3.

Posteriormente, desactivaremos el entrenamiento de cada capa dado que poseemos la configuración óptima.

pre\_trained\_model = InceptionV3(

include\_top = False,

input\_tensor = tf.keras.layers.Input(shape = (150, 150, 3))

)

**for** layer **in** pre\_trained\_model.layers:

layer.trainable = False

pre\_trained\_model.summary()

Si vemos la configuración del modelo obtendremos una lista gigantesca de capas, pero si nos concentramos en el final, nos encontraremos con casi 22 millones de parámetros.

Total **params**: 21,802,784

Trainable **params**: 0

Non-trainable **params**: 21,802,784

Para la configuración de InceptionV3 encontramos capas con configuraciones no secuenciales, estas serán sincronizadas en la capa mixed, donde se aplica todo el proceso de pooling y batch normalization.

Elegiremos una de estas capas mixed para cortar la arquitectura y agregar nuestros output, en esta ocasión elegiremos mixed7.

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

last\_output = last\_layers.output

Este será nuestro punto de partida para agregar nuestro modelo.

**Contribución creada por** Sebastián Franco Gómez.