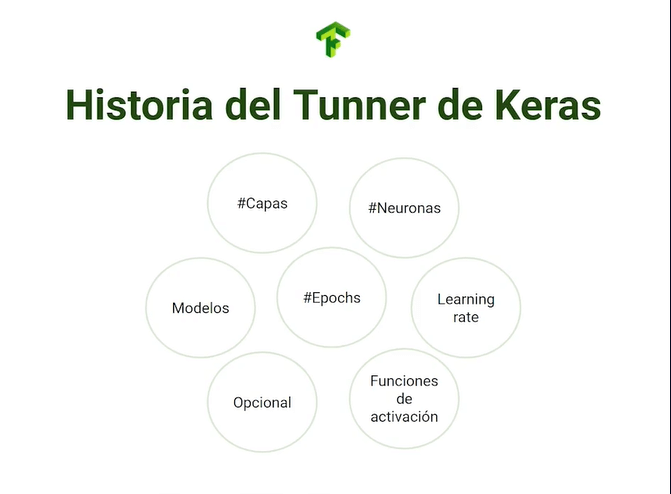
El autotuner de Keras nos permitirá automatizar el proceso de configuración de nuestra red neuronal.

Cuando deseemos iterar sobre múltiples configuraciones para nuestro modelo (probar diferentes capas, neuronas, épocas, learning rate y demás) no tendremos que hacerlo manualmente modelo a modelo, esta herramienta nos permitirá cargar y ejecutar diferentes fórmulas para comparar sus rendimientos.



**Implementando el autotuner**

El autotuner de Keras no viene cargado por defecto en Google Colab, por lo que debemos instalarlo por terminal de comandos.

!pip install -**q** -U keras-tuner

Para usarlo lo importaremos como kerastuner.

**import** kerastuner as kt

from tensorflow **import** keras

Para esta ocasión crearemos un nuevo constructor de modelos, este recibirá como parámetros un objeto tuner que determinará las variaciones de diferentes hiperparámetros.

Definiremos una arquitectura general, donde agregaremos una capa de convolución, Max Pooling y aplanamiento de manera fija, luego determinaremos la primer variable del constructor: La cantidad de neuronas en la siguiente capa oculta, se inicializará en 32 e incrementará hasta 512 dando saltos de 32 en 32.

La cantidad de neuronas de la siguiente capa será el objeto iterador. El resto de la red se mantendrá estable.

Finalmente, definiremos variaciones en el learning rate, donde empezaremos el modelo con 3 posibles learning rate: 0.01, 0.001 y 0.0001.

Al momento de compilar el modelo definiremos Adam como optimizador, sin embargo, llamaremos directamente a la clase y le entregaremos el objeto iterador. El resto de parámetros seguirán iguales.

def constructor\_modelos(hp):

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Conv2D(75, (3,3), activation = "relu", input\_shape = (28,28,1)))

model.add(tf.keras.layers.MaxPooling2D((2,2)))

model.add(tf.keras.layers.Flatten())

hp\_units = hp.Int("units", min\_value = 32, max\_value = 512, step = 32)

model.add(tf.keras.layers.Dense(units = hp\_units, activation = "relu", kernel\_regularizer = regularizers.l2(1e-5)))

model.add(tf.keras.layers.Dropout(0.2))

model.add(tf.keras.layers.Dense(128, activation = "relu", kernel\_regularizer = regularizers.l2(1e-5)))

model.add(tf.keras.layers.Dropout(0.2))

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

hp\_learning\_rate = hp.Choice("learning\_rate", values = [1e-2, 1e-3, 1e-4])

model.compile(optimizer = keras.optimizers.Adam(learning\_rate=hp\_learning\_rate), loss = "categorical\_crossentropy", metrics = ["accuracy"])

return model

Esta función será la materia prima del tuner, el cual hará pruebas con todas las combinatorias para encontrar el modelo más optimo.