**https://www.tensorflow.org/tutorials/keras/keras\_tuner**

**Introduction to the Keras Tuner**

The Keras Tuner is a library that helps you pick the optimal set of hyperparameters for your TensorFlow program. The process of selecting the right set of hyperparameters for your machine learning (ML) application is called *hyperparameter tuning* or *hypertuning*.

Hyperparameters are the variables that govern the training process and the topology of an ML model. These variables remain constant over the training process and directly impact the performance of your ML program. Hyperparameters are of two types:

1. **Model hyperparameters** which influence model selection such as the number and width of hidden layers
2. **Algorithm hyperparameters** which influence the speed and quality of the learning algorithm such as the learning rate for Stochastic Gradient Descent (SGD) and the number of nearest neighbors for a k Nearest Neighbors (KNN) classifier

**Setup**

import tensorflow as tf  
from tensorflow import keras

Install and import the Keras Tuner.

pip install -q -U keras-tuner

import keras\_tuner as kt

**Download and prepare the dataset**

(img\_train, label\_train), (img\_test, label\_test) = keras.datasets.fashion\_mnist.load\_data()

# Normalize pixel values between 0 and 1  
img\_train = img\_train.astype('float32') / 255.0  
img\_test = img\_test.astype('float32') / 255.0

**Define the model**

When you build a model for hypertuning, you also define the hyperparameter search space in addition to the model architecture. The model you set up for hypertuning is called a *hypermodel*.

You can define a hypermodel through two approaches:

* By using a model builder function
* By subclassing the HyperModel class of the Keras Tuner API

You can also use two pre-defined HyperModel classes - [HyperXception](https://keras-team.github.io/keras-tuner/documentation/hypermodels/" \l "hyperxception-class) and [HyperResNet](https://keras-team.github.io/keras-tuner/documentation/hypermodels/" \l "hyperresnet-class) for computer vision applications.

In this tutorial, you use a model builder function to define the image classification model. The model builder function returns a compiled model and uses hyperparameters you define inline to hypertune the model.

def model\_builder(hp):  
  model = keras.Sequential()  
  model.add(keras.layers.Flatten(input\_shape=(28, 28)))  
  
**# Tune the number of units in the first Dense layer  
  # Choose an optimal value between 32-512**  
  hp\_units = hp.Int('units', min\_value=32, max\_value=512, step=32)  
  model.add(keras.layers.Dense(units=hp\_units, activation='relu'))  
  model.add(keras.layers.Dense(10))  
  
**# Tune the learning rate for the optimizer  
  # Choose an optimal value from 0.01, 0.001, or 0.0001**  
  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=keras.losses.SparseCategoricalCrossentropy(from\_logits=True),  
                metrics=['accuracy'])  
  
  return model

**Instantiate the tuner and perform hypertuning**

Instantiate the tuner to perform the hypertuning. The Keras Tuner has four tuners available - RandomSearch, Hyperband, BayesianOptimization, and Sklearn. In this tutorial, you use the [Hyperband](https://arxiv.org/pdf/1603.06560.pdf) tuner.

To instantiate the Hyperband tuner, you must specify the hypermodel, the objective to optimize and the maximum number of epochs to train (max\_epochs).

tuner = kt.Hyperband(model\_builder,  
                     objective='val\_accuracy',  
                     max\_epochs=10,  
                     factor=3,  
                     directory='my\_dir',  
                     project\_name='intro\_to\_kt')

The Hyperband tuning algorithm uses adaptive resource allocation and early-stopping to quickly converge on a high-performing model. This is done using a sports championship style bracket. The algorithm trains a large number of models for a few epochs and carries forward only the top-performing half of models to the next round. Hyperband determines the number of models to train in a bracket by computing 1 + logfactor(max\_epochs) and rounding it up to the nearest integer.

Create a callback to stop training early after reaching a certain value for the validation loss.

stop\_early = tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=5)

Run the hyperparameter search. The arguments for the search method are the same as those used for tf.keras.model.fit in addition to the callback above.

tuner.search(img\_train, label\_train, epochs=50, validation\_split=0.2, callbacks=[stop\_early])  
  
**# Get the optimal hyperparameters**  
best\_hps=tuner.get\_best\_hyperparameters(num\_trials=1)[0]  
  
print(f"""  
The hyperparameter search is complete. The optimal number of units in the first densely-connected  
layer is {best\_hps.get('units')} and the optimal learning rate for the optimizer  
is {best\_hps.get('learning\_rate')}.  
""")

Trial 30 Complete [00h 00m 35s]

val\_accuracy: 0.8925833106040955

Best val\_accuracy So Far: 0.8925833106040955

Total elapsed time: 00h 07m 26s

INFO:tensorflow:Oracle triggered exit

The hyperparameter search is complete. The optimal number of units in the first densely-connected

layer is 320 and the optimal learning rate for the optimizer

is 0.001.

**Train the model**

Find the optimal number of epochs to train the model with the hyperparameters obtained from the search.

**# Build the model with the optimal hyperparameters and train it on the data for 50 epochs**  
model = tuner.hypermodel.build(best\_hps)  
history = **model.fit**(img\_train, label\_train, epochs=50, validation\_split=0.2)  
  
val\_acc\_per\_epoch = history.history['val\_accuracy']  
best\_epoch = val\_acc\_per\_epoch.index(max(val\_acc\_per\_epoch)) + 1  
print('Best epoch: %d' % (best\_epoch,))

Epoch 1/50

1500/1500 [==============================] - 4s 2ms/step - loss: 0.4988 - accuracy: 0.8232 - val\_loss: 0.4142 - val\_accuracy: 0.8517

Epoch 2/50

1500/1500 [==============================] - 3s 2ms/step - loss: 0.3717 - accuracy: 0.8646 - val\_loss: 0.3437 - val\_accuracy: 0.8773

Epoch 3/50

1500/1500 [==============================] - 3s 2ms/step - loss: 0.3317 - accuracy: 0.8779 - val\_loss: 0.3806 - val\_accuracy: 0.8639

Epoch 49/50

1500/1500 [==============================] - 3s 2ms/step - loss: 0.0747 - accuracy: 0.9718 - val\_loss: 0.4982 - val\_accuracy: 0.8953

Epoch 50/50

1500/1500 [==============================] - 3s 2ms/step - loss: 0.0786 - accuracy: 0.9706 - val\_loss: 0.5198 - val\_accuracy: 0.8976

Best epoch: 50

Re-instantiate the hypermodel and train it with the optimal number of epochs from above.

hypermodel = tuner.hypermodel.build(best\_hps)  
  
**# Retrain the model**  
hypermodel.fit(img\_train, label\_train, epochs=best\_epoch, validation\_split=0.2)

Epoch 1/50

1500/1500 [==============================] - 4s 2ms/step - loss: 0.4987 - accuracy: 0.8236 - val\_loss: 0.4065 - val\_accuracy: 0.8488

Epoch 2/50

1500/1500 [==============================] - 3s 2ms/step - loss: 0.3738 - accuracy: 0.8652 - val\_loss: 0.3847 - val\_accuracy: 0.8613

Epoch 3/50

1500/1500 [==============================] - 4s 2ms/step - loss: 0.3344 - accuracy: 0.8775 - val\_loss: 0.3568 - val\_accuracy: 0.8750

Epoch 49/50

1500/1500 [==============================] - 3s 2ms/step - loss: 0.0751 - accuracy: 0.9714 - val\_loss: 0.5719 - val\_accuracy: 0.8924

Epoch 50/50

1500/1500 [==============================] - 3s 2ms/step - loss: 0.0785 - accuracy: 0.9707 - val\_loss: 0.5123 - val\_accuracy: 0.8985

<keras.callbacks.History at 0x7fb39810a150>

To finish this tutorial, evaluate the hypermodel on the test data.

eval\_result = **hypermodel.evaluate**(img\_test, label\_test)  
print("[test loss, test accuracy]:", eval\_result)

313/313 [==============================] - 1s 2ms/step - loss: 0.5632 - accuracy: 0.8908

[test loss, test accuracy]: [0.56, 0.89]

The my\_dir/intro\_to\_kt directory contains detailed logs and checkpoints for every trial (model configuration) run during the hyperparameter search. If you re-run the hyperparameter search, the Keras Tuner uses the existing state from these logs to resume the search. To disable this behavior, pass an additional overwrite=True argument while instantiating the tuner.

**Summary**

In this tutorial, you learned how to use the Keras Tuner to tune hyperparameters for a model. To learn more about the Keras Tuner, check out these additional resources:

* [Keras Tuner on the TensorFlow blog](https://blog.tensorflow.org/2020/01/hyperparameter-tuning-with-keras-tuner.html)
* [Keras Tuner website](https://keras-team.github.io/keras-tuner/)

Also check out the [HParams Dashboard](https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams) in TensorBoard to interactively tune your model hyperparameters.