365 DataScience Tensorboard - Tuning hyperparameters

```
# In the following example, we will use the MNIST datase to show you ho
w to tune hyperparameters using Tensorboard

# Importing the relevant packages
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorboard.plugins.hparams import api as hp
```

Downloading and preprocessing the data

```
# Defining some constants/hyperparameters
BUFFER_SIZE = 70_000 # for reshuffling
BATCH_SIZE = 128
NUM_EPOCHS = 20

# Downloading the MNIST dataset
mnist_dataset, mnist_info = tfds.load(name='mnist', with_info=True, as_
supervised=True)

mnist_train, mnist_test = mnist_dataset['train'], mnist_dataset['test']

# Creating a function to scale our data
def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255.

    return image, label

# Scaling the data
train_and_validation_data = mnist_train.map(scale)
test_data = mnist_test.map(scale)

# Defining the size of the validation set
num_validation_samples = 0.1 * mnist_info.splits['train'].num_examples
num_validation_samples = tf.cast(num_validation_samples, tf.int64)

# Defining the size of the test set
num_test_samples = mnist_info.splits['test'].num_examples
num_test_samples = tf.cast(num_test_samples, tf.int64)

# Reshuffling the dataset
train_and_validation_data = train_and_validation_data.shuffle(BUFFER_SI
ZE)

# Splitting the dataset into trainig + validation
train_data = train_and_validation_data.skip(num_validation_samples)
validation_data = train_and_validation_data.take(num_validation_samples)
```

```
# Batching the data
train_data = train_data.batch(BATCH_SIZE)
validation_data = validation_data.batch(num_validation_samples)
test_data = mnist_test.map(scale).batch(num_test_samples)
```

Defining hyperparameters

```
# Defining the hypermatarest we would test and their range
HP_FILTER_SIZE = hp.HParam('filter_size', hp.Discrete([3,5,7]))
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd']))

METRIC_ACCURACY = 'accuracy'

# Logging setup info
with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
    hp.hparams_config(
        hparams=[HP_FILTER_SIZE, HP_OPTIMIZER],
        metrics=[hp.Metric(METRIC_ACCURACY, display_name='Accuracy')],
    )
```

Cerating functions for training our model and for logging purposes

```
# Wrapping our model and training in a function
def train_test_model(hparams):

    # Outlining the model/architecture of our CNN
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(50, hparams[HP_FILTER_SIZE], activation=
'relu', input_shape=(28, 28, 1)),
        tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
        tf.keras.layers.Conv2D(50, hparams[HP_FILTER_SIZE], activation=
'relu'),
        tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(10)
    ])

    # Defining the loss function
    loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits
=True)

    # Compiling the model with parameter value for the optimizer
    model.compile(optimizer=hparams[HP_OPTIMIZER], loss=loss_fn, metric
s=['accuracy'])

    # Defining early stopping to prevent overfitting
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor = 'val_loss',
        mode = 'auto',
        min_delta = 0,
        patience = 2,
```

```
        verbose = 0,
        restore_best_weights = True
    )

    # Training the model
    model.fit(
        train_data,
        epochs = NUM_EPOCHS,
        callbacks = [early_stopping],
        validation_data = validation_data,
        verbose = 2
    )

    _, accuracy = model.evaluate(test_data)

    return accuracy

# Creating a function to log the resuls
def run(log_dir, hparams):

    with tf.summary.create_file_writer(log_dir).as_default():
        hp.hparams(hparams)  # record the values used in this trial
        accuracy = train_test_model(hparams)
        tf.summary.scalar(METRIC_ACCURACY, accuracy, step=1)
```

Training the model with the different hyperparameters

```
# Performing a grid search on the hyperparameters we need to test
session_num = 0

for filter_size in HP_FILTER_SIZE.domain.values:
    for optimizer in HP_OPTIMIZER.domain.values:

        hparams = {
            HP_FILTER_SIZE: filter_size,
            HP_OPTIMIZER: optimizer
        }
        run_name = "run-%d" % session_num
        print('--- Starting trial: %s' % run_name)
        print({h.name: hparams[h] for h in hparams})
        run('logs/hparam_tuning/' + run_name, hparams)

        session_num += 1
```

```
--- Starting trial: run-0
{'filter_size': 3, 'optimizer': 'adam'}
Epoch 1/20
422/422 - 21s - loss: 0.2906 - accuracy: 0.9168 - val_loss: 0.0817 - va
l_accuracy: 0.9762
Epoch 2/20
422/422 - 20s - loss: 0.0801 - accuracy: 0.9754 - val_loss: 0.0600 - va
```

```
l_accuracy: 0.9830
Epoch 3/20
422/422 - 21s - loss: 0.0606 - accuracy: 0.9820 - val_loss: 0.0520 - va
l_accuracy: 0.9848
Epoch 4/20
422/422 - 21s - loss: 0.0496 - accuracy: 0.9850 - val_loss: 0.0389 - va
l_accuracy: 0.9882
Epoch 5/20
422/422 - 21s - loss: 0.0423 - accuracy: 0.9873 - val_loss: 0.0293 - va
l_accuracy: 0.9907
Epoch 6/20
422/422 - 20s - loss: 0.0370 - accuracy: 0.9887 - val_loss: 0.0266 - va
l_accuracy: 0.9935
Epoch 7/20
422/422 - 20s - loss: 0.0310 - accuracy: 0.9904 - val_loss: 0.0251 - va
l_accuracy: 0.9930
Epoch 8/20
422/422 - 21s - loss: 0.0276 - accuracy: 0.9915 - val_loss: 0.0220 - va
l_accuracy: 0.9932
Epoch 9/20
422/422 - 20s - loss: 0.0243 - accuracy: 0.9925 - val_loss: 0.0154 - va
l_accuracy: 0.9955
Epoch 10/20
422/422 - 21s - loss: 0.0215 - accuracy: 0.9935 - val_loss: 0.0176 - va
l_accuracy: 0.9945
Epoch 11/20
422/422 - 21s - loss: 0.0197 - accuracy: 0.9936 - val_loss: 0.0165 - va
l_accuracy: 0.9948
      1/Unknown - 1s 1s/step - loss: 0.0329 - accuracy: 0.99 - 1s 1s/st
ep - loss: 0.0329 - accuracy: 0.9903--- Starting trial: run-1
{'filter_size': 3, 'optimizer': 'sgd'}
Epoch 1/20
422/422 - 21s - loss: 1.3202 - accuracy: 0.6425 - val_loss: 0.4644 - va
l_accuracy: 0.8770
Epoch 2/20
422/422 - 21s - loss: 0.3828 - accuracy: 0.8887 - val_loss: 0.3248 - va
l_accuracy: 0.9080
Epoch 3/20
422/422 - 21s - loss: 0.2924 - accuracy: 0.9145 - val_loss: 0.2717 - va
l_accuracy: 0.9192
Epoch 4/20
422/422 - 21s - loss: 0.2408 - accuracy: 0.9288 - val_loss: 0.2170 - va
l_accuracy: 0.9350
Epoch 5/20
422/422 - 21s - loss: 0.2026 - accuracy: 0.9405 - val_loss: 0.1897 - va
l_accuracy: 0.9432
Epoch 6/20
422/422 - 21s - loss: 0.1750 - accuracy: 0.9485 - val_loss: 0.1671 - va
l_accuracy: 0.9510
Epoch 7/20
```

```
422/422 - 21s - loss: 0.1549 - accuracy: 0.9547 - val_loss: 0.1530 - va
l_accuracy: 0.9563
Epoch 8/20
422/422 - 21s - loss: 0.1413 - accuracy: 0.9589 - val_loss: 0.1266 - va
l_accuracy: 0.9632
Epoch 9/20
422/422 - 21s - loss: 0.1273 - accuracy: 0.9634 - val_loss: 0.1187 - va
l_accuracy: 0.9647
Epoch 10/20
422/422 - 21s - loss: 0.1199 - accuracy: 0.9650 - val_loss: 0.1067 - va
l_accuracy: 0.9692
Epoch 11/20
422/422 - 21s - loss: 0.1113 - accuracy: 0.9667 - val_loss: 0.1164 - va
l_accuracy: 0.9627
Epoch 12/20
422/422 - 21s - loss: 0.1053 - accuracy: 0.9685 - val_loss: 0.1024 - va
l_accuracy: 0.9685
Epoch 13/20
422/422 - 21s - loss: 0.0996 - accuracy: 0.9704 - val_loss: 0.1017 - va
l_accuracy: 0.9722
Epoch 14/20
422/422 - 21s - loss: 0.0943 - accuracy: 0.9714 - val_loss: 0.0822 - va
l_accuracy: 0.9745
Epoch 15/20
422/422 - 21s - loss: 0.0909 - accuracy: 0.9731 - val_loss: 0.0908 - va
l_accuracy: 0.9708
Epoch 16/20
422/422 - 21s - loss: 0.0879 - accuracy: 0.9735 - val_loss: 0.0918 - va
l_accuracy: 0.9702
      1/Unknown - 1s 1s/step - loss: 0.0812 - accuracy: 0.97 - 1s 1s/st
ep - loss: 0.0812 - accuracy: 0.9765--- Starting trial: run-2
{'filter_size': 5, 'optimizer': 'adam'}
Epoch 1/20
422/422 - 24s - loss: 0.2376 - accuracy: 0.9320 - val_loss: 0.0793 - va
l_accuracy: 0.9767
Epoch 2/20
422/422 - 23s - loss: 0.0644 - accuracy: 0.9804 - val_loss: 0.0504 - va
l_accuracy: 0.9858
Epoch 3/20
422/422 - 23s - loss: 0.0451 - accuracy: 0.9860 - val_loss: 0.0392 - va
l_accuracy: 0.9890
Epoch 4/20
422/422 - 23s - loss: 0.0361 - accuracy: 0.9888 - val_loss: 0.0334 - va
l_accuracy: 0.9902
Epoch 5/20
422/422 - 23s - loss: 0.0299 - accuracy: 0.9905 - val_loss: 0.0223 - va
l_accuracy: 0.9932
Epoch 6/20
422/422 - 23s - loss: 0.0241 - accuracy: 0.9923 - val_loss: 0.0216 - va
l_accuracy: 0.9928
```

```
Epoch 7/20
422/422 - 23s - loss: 0.0214 - accuracy: 0.9935 - val_loss: 0.0177 - va
l_accuracy: 0.9945
Epoch 8/20
422/422 - 23s - loss: 0.0181 - accuracy: 0.9942 - val_loss: 0.0121 - va
l_accuracy: 0.9968
Epoch 9/20
422/422 - 23s - loss: 0.0153 - accuracy: 0.9952 - val_loss: 0.0123 - va
l_accuracy: 0.9963
Epoch 10/20
422/422 - 23s - loss: 0.0137 - accuracy: 0.9956 - val_loss: 0.0127 - va
l_accuracy: 0.9958
      1/Unknown - 1s 1s/step - loss: 0.0298 - accuracy: 0.99 - 1s 1s/st
ep - loss: 0.0298 - accuracy: 0.9912--- Starting trial: run-3
{'filter_size': 5, 'optimizer': 'sgd'}
Epoch 1/20
422/422 - 23s - loss: 1.1076 - accuracy: 0.7357 - val_loss: 0.3821 - va
l_accuracy: 0.8963
Epoch 2/20
422/422 - 23s - loss: 0.3235 - accuracy: 0.9052 - val_loss: 0.2613 - va
l_accuracy: 0.9222
Epoch 3/20
422/422 - 23s - loss: 0.2427 - accuracy: 0.9287 - val_loss: 0.2046 - va
l_accuracy: 0.9432
Epoch 4/20
422/422 - 23s - loss: 0.2001 - accuracy: 0.9419 - val_loss: 0.1846 - va
l_accuracy: 0.9477
Epoch 5/20
422/422 - 23s - loss: 0.1694 - accuracy: 0.9506 - val_loss: 0.1487 - va
l_accuracy: 0.9588
Epoch 6/20
422/422 - 23s - loss: 0.1472 - accuracy: 0.9577 - val_loss: 0.1310 - va
l_accuracy: 0.9625
Epoch 7/20
422/422 - 24s - loss: 0.1324 - accuracy: 0.9616 - val_loss: 0.1300 - va
l_accuracy: 0.9632
Epoch 8/20
422/422 - 23s - loss: 0.1203 - accuracy: 0.9647 - val_loss: 0.1218 - va
l_accuracy: 0.9650
Epoch 9/20
422/422 - 23s - loss: 0.1105 - accuracy: 0.9680 - val_loss: 0.1057 - va
l_accuracy: 0.9683
Epoch 10/20
422/422 - 23s - loss: 0.1012 - accuracy: 0.9711 - val_loss: 0.0976 - va
l_accuracy: 0.9698
Epoch 11/20
422/422 - 23s - loss: 0.0957 - accuracy: 0.9721 - val_loss: 0.1097 - va
l_accuracy: 0.9702
Epoch 12/20
422/422 - 23s - loss: 0.0902 - accuracy: 0.9732 - val_loss: 0.0805 - va
```

```
l_accuracy: 0.9733
Epoch 13/20
422/422 - 23s - loss: 0.0852 - accuracy: 0.9753 - val_loss: 0.0810 - va
l_accuracy: 0.9743
Epoch 14/20
422/422 - 23s - loss: 0.0826 - accuracy: 0.9768 - val_loss: 0.0892 - va
l_accuracy: 0.9707
      1/Unknown - 1s 1s/step - loss: 0.0750 - accuracy: 0.97 - 1s 1s/st
ep - loss: 0.0750 - accuracy: 0.9761--- Starting trial: run-4
{'filter_size': 7, 'optimizer': 'adam'}
Epoch 1/20
422/422 - 20s - loss: 0.2375 - accuracy: 0.9318 - val_loss: 0.0883 - va
l_accuracy: 0.9747
Epoch 2/20
422/422 - 19s - loss: 0.0717 - accuracy: 0.9790 - val_loss: 0.0643 - va
l_accuracy: 0.9805
Epoch 3/20
422/422 - 20s - loss: 0.0497 - accuracy: 0.9853 - val_loss: 0.0388 - va
l_accuracy: 0.9883
Epoch 4/20
422/422 - 20s - loss: 0.0406 - accuracy: 0.9875 - val_loss: 0.0312 - va
l_accuracy: 0.9883
Epoch 5/20
422/422 - 19s - loss: 0.0313 - accuracy: 0.9904 - val_loss: 0.0334 - va
l_accuracy: 0.9897
Epoch 6/20
422/422 - 19s - loss: 0.0263 - accuracy: 0.9919 - val_loss: 0.0240 - va
l_accuracy: 0.9923
Epoch 7/20
422/422 - 19s - loss: 0.0225 - accuracy: 0.9928 - val_loss: 0.0167 - va
l_accuracy: 0.9948
Epoch 8/20
422/422 - 19s - loss: 0.0189 - accuracy: 0.9941 - val_loss: 0.0124 - va
l_accuracy: 0.9965
Epoch 9/20
422/422 - 19s - loss: 0.0156 - accuracy: 0.9952 - val_loss: 0.0086 - va
l_accuracy: 0.9983
Epoch 10/20
422/422 - 19s - loss: 0.0139 - accuracy: 0.9954 - val_loss: 0.0184 - va
l_accuracy: 0.9927
Epoch 11/20
422/422 - 19s - loss: 0.0126 - accuracy: 0.9957 - val_loss: 0.0129 - va
l_accuracy: 0.9952
      1/Unknown - 1s 882ms/step - loss: 0.0300 - accuracy: 0.991 - 1s 8
93ms/step - loss: 0.0300 - accuracy: 0.9917--- Starting trial: run-5
{'filter_size': 7, 'optimizer': 'sgd'}
Epoch 1/20
422/422 - 21s - loss: 1.0211 - accuracy: 0.7491 - val_loss: 0.4012 - va
l_accuracy: 0.8838
Epoch 2/20
```

```
422/422 - 21s - loss: 0.3284 - accuracy: 0.9050 - val_loss: 0.2717 - va
l_accuracy: 0.9228
Epoch 3/20
422/422 - 21s - loss: 0.2476 - accuracy: 0.9284 - val_loss: 0.2274 - va
l_accuracy: 0.9327
Epoch 4/20
422/422 - 21s - loss: 0.2079 - accuracy: 0.9397 - val_loss: 0.1732 - va
l_accuracy: 0.9505
Epoch 5/20
422/422 - 21s - loss: 0.1795 - accuracy: 0.9475 - val_loss: 0.1708 - va
l_accuracy: 0.9473
Epoch 6/20
422/422 - 21s - loss: 0.1606 - accuracy: 0.9528 - val_loss: 0.1347 - va
l_accuracy: 0.9613
Epoch 7/20
422/422 - 21s - loss: 0.1458 - accuracy: 0.9574 - val_loss: 0.1331 - va
l_accuracy: 0.9592
Epoch 8/20
422/422 - 21s - loss: 0.1324 - accuracy: 0.9601 - val_loss: 0.1226 - va
l_accuracy: 0.9643
Epoch 9/20
422/422 - 20s - loss: 0.1224 - accuracy: 0.9645 - val_loss: 0.1288 - va
l_accuracy: 0.9605
Epoch 10/20
422/422 - 21s - loss: 0.1148 - accuracy: 0.9660 - val_loss: 0.1146 - va
l_accuracy: 0.9683
Epoch 11/20
422/422 - 20s - loss: 0.1082 - accuracy: 0.9683 - val_loss: 0.1077 - va
l_accuracy: 0.9710
Epoch 12/20
422/422 - 20s - loss: 0.1022 - accuracy: 0.9702 - val_loss: 0.0974 - va
l_accuracy: 0.9718
Epoch 13/20
422/422 - 20s - loss: 0.0969 - accuracy: 0.9713 - val_loss: 0.0961 - va
l_accuracy: 0.9710
Epoch 14/20
422/422 - 21s - loss: 0.0935 - accuracy: 0.9728 - val_loss: 0.0795 - va
l_accuracy: 0.9780
Epoch 15/20
422/422 - 21s - loss: 0.0893 - accuracy: 0.9740 - val_loss: 0.0865 - va
l_accuracy: 0.9742
Epoch 16/20
422/422 - 20s - loss: 0.0847 - accuracy: 0.9756 - val_loss: 0.0802 - va
l_accuracy: 0.9750
      1/Unknown - 1s 899ms/step - loss: 0.0812 - accuracy: 0.975 - 1s 9
10ms/step - loss: 0.0812 - accuracy: 0.9756
```

Visualizing the hyperparameter results with Tensorboard

```
# You can find the relevant data in the "HPARAMS" tab

# Loading the Tensorboard extension
%load_ext tensorboard
%tensorboard --logdir "logs/hparam_tuning"

# NOTE: On Windows, TensorBoard has trouble starting if the extension h
as been running. So, the first time you start it,
# it will run properly. But if you subsequently try to restart it, or o
pen a different directory,
# the extension will encounter an error. Luckily, there is a quick work
around. All you need to do
# is write 2 commands in the shell. First, open the command prompt, or
'cmd.exe'. In there,
# you need to paste the following 2 lines , one after another:
#
# taskkill /im TensorBoard.exe /f
# del /q %TMP%\.TensorBoard-info\*
#
# These will end already active TensorBoard processes and clean the tem
porary data associated with TensorBoard,
# so you can run it again. If either of those gives an error, that's ok
ay: you can ignore it.
```

Start your 365 Journey!