Open in Colab

Ungraded Lab: Hyperparameter tuning and model training with TFX

In this lab, you will be again doing hyperparameter tuning but this time, it will be within a Tensorflow Extended (TFX) pipeline.

We have already introduced some TFX components in Course 2 of this specialization related to data ingestion, validation, and transformation. In this notebook, you will get to work with two more which are related to model development and training: *Tuner* and *Trainer*.

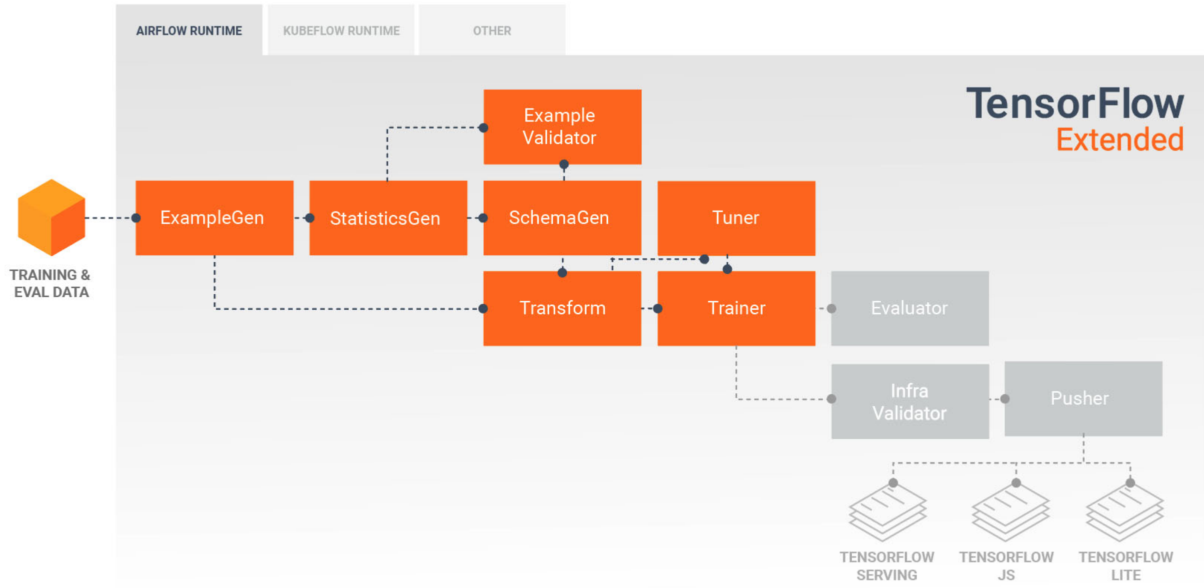

image source: https://www.tensorflow.org/tfx/guide

- The *Tuner* utilizes the Keras Tuner API under the hood to tune your model's hyperparameters.
- You can get the best set of hyperparameters from the Tuner component and feed it into the *Trainer* component to optimize your model for training.

You will again be working with the FashionMNIST dataset and will feed it though the TFX pipeline up to the Trainer component.You will quickly review the earlier components from Course 2, then focus on the two new components introduced.

Let's begin!

Setup

Install TFX

You will first install TFX, a framework for developing end-to-end machine learning pipelines.

```
!pip install tfx==0.30
```

Note: In Google Colab, you need to restart the runtime at this point to finalize updating the packages you just installed. You can do so by clicking the `Restart Runtime` at the end of the output cell above (after installation), or by selecting `Runtime > Restart Runtime` in the Menu bar. ***Please do not proceed to the next section without restarting.*** *You can also ignore the errors about version incompatibility of some of the bundled packages because we won't be using those in this notebook.*

Imports

You will then import the packages you will need for this exercise.

```
import tensorflow as tf
from tensorflow import keras
import tensorflow_datasets as tfds

import os
import pprint

from tfx.components import ImportExampleGen
from tfx.components import ExampleValidator
from tfx.components import SchemaGen
from tfx.components import StatisticsGen
from tfx.components import Transform
from tfx.components import Tuner
from tfx.components import Trainer

from tfx.proto import example_gen_pb2
from tfx.orchestration.experimental.interactive.interactive_context import InteractiveContext
```

Download and prepare the dataset

As mentioned earlier, you will be using the Fashion MNIST dataset just like in the previous lab. This will allow you to compare the similarities and differences when using Keras Tuner as a standalone library and within an ML pipeline.

You will first need to setup the directories that you will use to store the dataset, as well as the pipeline artifacts and metadata store.

```
# Location of the pipeline metadata store
_pipeline_root = './pipeline/'

# Directory of the raw data files
_data_root = './data/fmnist'

# Temporary directory
tempdir = './tempdir'
```

```
# Create the dataset directory
!mkdir -p {_data_root}

# Create the TFX pipeline files directory
!mkdir {_pipeline_root}
```

You will now download FashionMNIST from Tensorflow Datasets. The `with_info` flag will be set to `True` so you can display information about the dataset in the next cell (i.e. using `ds_info`).

```
# Download the dataset
ds, ds_info = tfds.load('fashion_mnist', data_dir=tempdir, with_info=True)
```

```
Downloading and preparing dataset fashion_mnist/3.0.1 (download: 29.45 MiB, generated: 36.42 MiB, total: 65.87 Mi
B) to ./tempdir/fashion_mnist/3.0.1...
```

```
Shuffling and writing examples to ./tempdir/fashion_mnist/3.0.1.incomplete5BWNUN/fashion_mnist-train.tfrecord
Shuffling and writing examples to ./tempdir/fashion_mnist/3.0.1.incomplete5BWNUN/fashion_mnist-test.tfrecord
Dataset fashion_mnist downloaded and prepared to ./tempdir/fashion_mnist/3.0.1. Subsequent calls will reuse this d
ata.
```

```
# Display info about the dataset
print(ds_info)
```

```
tfds.core.DatasetInfo(
    name='fashion_mnist',
    version=3.0.1,
    description='Fashion-MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 exa
mples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10
classes.',
    homepage='https://github.com/zalandoresearch/fashion-mnist',
    features=FeaturesDict({
        'image': Image(shape=(28, 28, 1), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=10),
    }),
    total_num_examples=70000,
    splits={
        'test': 10000,
        'train': 60000,
```

```
    },
    supervised_keys=('image', 'label'),
    citation="""@article{DBLP:journals/corr/abs-1708-07747,
      author    = {Han Xiao and
                   Kashif Rasul and
                   Roland Vollgraf},
      title     = {Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning
                   Algorithms},
      journal   = {CoRR},
      volume    = {abs/1708.07747},
      year      = {2017},
      url       = {http://arxiv.org/abs/1708.07747},
      archivePrefix = {arXiv},
      eprint    = {1708.07747},
      timestamp = {Mon, 13 Aug 2018 16:47:27 +0200},
      biburl    = {https://dblp.org/rec/bib/journals/corr/abs-1708-07747},
      bibsource = {dblp computer science bibliography, https://dblp.org}
    }""",
    redistribution_info=,
)
```

You can review the downloaded files with the code below. For this lab, you will be using the *train* TFRecord so you will need to take note of its filename. You will not use the *test* TFRecord in this lab.

```
# Define the location of the train tfrecord downloaded via TFDS
tfds_data_path = f'{tempdir}/{ds_info.name}/{ds_info.version}'

# Display contents of the TFDS data directory
os.listdir(tfds_data_path)
```

```
['fashion_mnist-train.tfrecord-00000-of-00001',
 'features.json',
 'fashion_mnist-test.tfrecord-00000-of-00001',
 'dataset_info.json',
 'label.labels.txt']
```

You will then copy the train split from the downloaded data so it can be consumed by the ExampleGen component in the next step. This component requires that your files are in a directory without extra files (e.g. JSONs and TXT files).

```
# Define the train tfrecord filename
train_filename = 'fashion_mnist-train.tfrecord-00000-of-00001'

# Copy the train tfrecord into the data root folder
!cp {tfds_data_path}/{train_filename} {_data_root}
```

TFX Pipeline

With the setup complete, you can now proceed to creating the pipeline.

Initialize the Interactive Context

You will start by initializing the InteractiveContext so you can run the components within this Colab environment. You can safely ignore the warning because you will just be using a local SQLite file for the metadata store.

```
# Initialize the InteractiveContext
context = InteractiveContext(pipeline_root=_pipeline_root)
```

```
WARNING:absl:InteractiveContext metadata_connection_config not provided: using SQLite ML Metadata database at ./pi
peline/metadata.sqlite.
```

ExampleGen

You will start the pipeline by ingesting the TFRecord you set aside. The ImportExampleGen consumes TFRecords and you can specify splits as shown below. For this exercise, you will split the train tfrecord to use 80% for the train set, and the remaining 20% as eval/validation set.

```
# Specify 80/20 split for the train and eval set
output = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=8),
        example_gen_pb2.SplitConfig.Split(name='eval', hash_buckets=2),
    ]))

# Ingest the data through ExampleGen
example_gen = ImportExampleGen(input_base=_data_root, output_config=output)

# Run the component
context.run(example_gen)
```

```
WARNING:apache_beam.runners.interactive.interactive_environment:Dependencies required for Interactive Beam PCollec
tion visualization are not available, please use: `pip install apache-beam[interactive]` to install necessary depe
ndencies to enable all data visualization features.
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.7 interpreter.
WARNING:apache_beam.io.tfrecordio:Couldn't find python-snappy so the implementation of _TFRecordUtil._masked_crc32
c is not as fast as it could be.
```

▼**ExecutionResult** at 0x7fd67ef02cd0

.execution_id	1
.component	**ImportExampleGen** at 0x7fd670438cd0
.component.inputs	{}
.component.outputs	**['examples']** **Channel** of type **'Examples'** (1 artifact) at 0x7fd670438d50

```
# Print split names and URI
artifact = example_gen.outputs['examples'].get()[0]
print(artifact.split_names, artifact.uri)
```

```
["train", "eval"] ./pipeline/ImportExampleGen/examples/1
```

StatisticsGen

Next, you will compute the statistics of the dataset with the StatisticsGen component.

```
# Run StatisticsGen
statistics_gen = StatisticsGen(
    examples=example_gen.outputs['examples'])

context.run(statistics_gen)
```

```
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.7 interpreter.
```

▼**ExecutionResult** at 0x7fd62f444390

.execution_id	2
.component	**StatisticsGen** at 0x7fd62f51ec90
.component.inputs	**['examples']** **Channel** of type **'Examples'** (1 artifact) at 0x7fd670438d50
.component.outputs	**['statistics']** **Channel** of type **'ExampleStatistics'** (1 artifact) at 0x7fd62f51ecd0

SchemaGen

You can then infer the dataset schema with SchemaGen. This will be used to validate incoming data to ensure that it is formatted correctly.

```
# Run SchemaGen
schema_gen = SchemaGen(
      statistics=statistics_gen.outputs['statistics'], infer_feature_shape=True)
context.run(schema_gen)
```

▼**ExecutionResult** at 0x7fd6704389d0

.execution_id	3
.component	▶ **SchemaGen** at 0x7fd631143450
.component.inputs	**['statistics']** ▶ **Channel** of type **'ExampleStatistics'** (1 artifact) at 0x7fd62f51ecd0
.component.outputs	**['schema']** ▶ **Channel** of type **'Schema'** (1 artifact) at 0x7fd631143310

```
# Visualize the results
context.show(schema_gen.outputs['schema'])
```

Artifact at ./pipeline/SchemaGen/schema/3

Feature name	**Type**	**Presence**	**Valency**	**Domain**
'image'	BYTES	required		-
'label'	INT	required		-

ExampleValidator

You can assume that the dataset is clean since we downloaded it from TFDS. But just to review, let's run it through ExampleValidator to detect if there are anomalies within the dataset.

```
# Run ExampleValidator
example_validator = ExampleValidator(
    statistics=statistics_gen.outputs['statistics'],
    schema=schema_gen.outputs['schema'])
context.run(example_validator)
```

▼**ExecutionResult** at 0x7fd631159cd0

.execution_id	4
.component	▶ **ExampleValidator** at 0x7fd63115bf90
.component.inputs	**['statistics']** ▶ **Channel** of type **'ExampleStatistics'** (1 artifact) at 0x7fd62f51ecd0 **['schema']** ▶ **Channel** of type **'Schema'** (1 artifact) at 0x7fd631143310
.component.outputs	**['anomalies']** ▶ **Channel** of type **'ExampleAnomalies'** (1 artifact) at 0x7fd63115b510

```
# Visualize the results. There should be no anomalies.
context.show(example_validator.outputs['anomalies'])
```

Artifact at ./pipeline/ExampleValidator/anomalies/4

'train' split:

```
/usr/local/lib/python3.7/dist-packages/tensorflow_data_validation/utils/display_util.py:217: FutureWarning: Passin
g a negative integer is deprecated in version 1.0 and will not be supported in future version. Instead, use None t
o not limit the column width.
  pd.set_option('max_colwidth', -1)
```

No anomalies found.

'eval' split:

No anomalies found.

Transform

Let's now use the Transform component to scale the image pixels and convert the data types to float. You will first define the transform module containing these operations before you run the component.

```
_transform_module_file = 'fmnist_transform.py'
```

```
%%writefile {_transform_module_file}

import tensorflow as tf
import tensorflow_transform as tft

# Keys
_LABEL_KEY = 'label'
_IMAGE_KEY = 'image'


def _transformed_name(key):
    return key + '_xf'

def _image_parser(image_str):
    '''converts the images to a float tensor'''
    image = tf.image.decode_image(image_str, channels=1)
    image = tf.reshape(image, (28, 28, 1))
    image = tf.cast(image, tf.float32)
    return image


def _label_parser(label_id):
    '''converts the labels to a float tensor'''
    label = tf.cast(label_id, tf.float32)
    return label


def preprocessing_fn(inputs):
    """tf.transform's callback function for preprocessing inputs.
    Args:
        inputs: map from feature keys to raw not-yet-transformed features.
    Returns:
        Map from string feature key to transformed feature operations.
    """

    # Convert the raw image and labels to a float array
    with tf.device("/cpu:0"):
        outputs = {
            _transformed_name(_IMAGE_KEY):
                tf.map_fn(
                    _image_parser,
                    tf.squeeze(inputs[_IMAGE_KEY], axis=1),
                    dtype=tf.float32),
            _transformed_name(_LABEL_KEY):
                tf.map_fn(
                    _label_parser,
                    inputs[_LABEL_KEY],
                    dtype=tf.float32)
        }

    # scale the pixels from 0 to 1
    outputs[_transformed_name(_IMAGE_KEY)] = tft.scale_to_0_1(outputs[_transformed_name(_IMAGE_KEY)])

    return outputs
```

```
Writing fmnist_transform.py
```

You will run the component by passing in the examples, schema, and transform module file.

Note: You can safely ignore the warnings and `udf_utils` related errors.

```
# Ignore TF warning messages
tf.get_logger().setLevel('ERROR')

# Setup the Transform component
transform = Transform(
    examples=example_gen.outputs['examples'],
    schema=schema_gen.outputs['schema'],
    module_file=os.path.abspath(_transform_module_file))

# Run the component
context.run(transform)
```

```
ERROR:absl:udf_utils.get_fn {'module_file': None, 'module_path': 'fmnist_transform@./pipeline/_wheels/tfx_user_code_Transform-0.0+2164e6d603bd93fd1392518ddba41d518c4d2ae0f4b327f2f6995f9cd89db491-py3-none-any.whl', 'preprocessing_fn': None} 'preprocessing_fn'
WARNING:root:This output type hint will be ignored and not used for type-checking purposes. Typically, output type hints for a PTransform are single (or nested) types wrapped by a PCollection, PDone, or None. Got: Tuple[Dict[str, Union[NoneType, _Dataset]], Union[Dict[str, Dict[str, PCollection]], NoneType]] instead.
WARNING:root:This output type hint will be ignored and not used for type-checking purposes. Typically, output type hints for a PTransform are single (or nested) types wrapped by a PCollection, PDone, or None. Got: Tuple[Dict[str, Union[NoneType, _Dataset]], Union[Dict[str, Dict[str, PCollection]], NoneType]] instead.
WARNING:apache_beam.typehints.typehints:Ignoring send_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring return_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring send_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring return_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring send_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring return_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring send_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring return_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring send_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring return_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring send_type hint: <class 'NoneType'>
WARNING:apache_beam.typehints.typehints:Ignoring return_type hint: <class 'NoneType'>
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.7 interpreter.
```

▼**ExecutionResult** at 0x7fd63115cfd0

.execution_id	5
.component	**Transform** at 0x7fd62f6b85d0
.component.inputs	**['examples']** **Channel** of type **'Examples'** (1 artifact) at 0x7fd670438d50 **['schema']** **Channel** of type **'Schema'** (1 artifact) at 0x7fd631143310
.component.outputs	**['transform_graph']** **Channel** of type **'TransformGraph'** (1 artifact) at 0x7fd62f6b8310 **['transformed_examples']** **Channel** of type **'Examples'** (1 artifact) at 0x7fd62f6b8e10 **['updated_analyzer_cache']** **Channel** of type **'TransformCache'** (1 artifact) at 0x7fd62f6b8550

Tuner

As the name suggests, the Tuner component tunes the hyperparameters of your model. To use this, you will need to provide a *tuner module file* which contains a `tuner_fn()` function. In this function, you will mostly do the same steps as you did in the previous ungraded lab but with some key differences in handling the dataset.

The Transform component earlier saved the transformed examples as TFRecords compressed in `.gz` format and you will need to load that into memory. Once loaded, you will need to create batches of features and labels so you can finally use it for hypertuning. This process is modularized in the `_input_fn()` below.

Going back, the `tuner_fn()` function will return a `TunerFnResult` namedtuple containing your `tuner` object and a set of arguments to pass to `tuner.search()` method. You will see these in action in the following cells. When reviewing the module file, we recommend viewing the `tuner_fn()` first before looking at the other auxiliary functions.

```
# Declare name of module file
_tuner_module_file = 'tuner.py'
```

```
%%writefile {_tuner_module_file}

# Define imports
from kerastuner.engine import base_tuner
import kerastuner as kt
from tensorflow import keras
from typing import NamedTuple, Dict, Text, Any, List
from tfx.components.trainer.fn_args_utils import FnArgs, DataAccessor
import tensorflow as tf
import tensorflow_transform as tft

# Declare namedtuple field names
TunerFnResult = NamedTuple('TunerFnResult', [('tuner', base_tuner.BaseTuner),
                                             ('fit_kwargs', Dict[Text, Any])])

# Label key
LABEL_KEY = 'label_xf'

# Callback for the search strategy
stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)


def _gzip_reader_fn(filenames):
  '''Load compressed dataset

  Args:
    filenames - filenames of TFRecords to load

  Returns:
    TFRecordDataset loaded from the filenames
  '''

  # Load the dataset. Specify the compression type since it is saved as `.gz`
  return tf.data.TFRecordDataset(filenames, compression_type='GZIP')


def _input_fn(file_pattern,
              tf_transform_output,
              num_epochs=None,
              batch_size=32) -> tf.data.Dataset:
  '''Create batches of features and labels from TF Records

  Args:
    file_pattern - List of files or patterns of file paths containing Example records.
    tf_transform_output - transform output graph
    num_epochs - Integer specifying the number of times to read through the dataset.
            If None, cycles through the dataset forever.
    batch_size - An int representing the number of records to combine in a single batch.

  Returns:
    A dataset of dict elements, (or a tuple of dict elements and label).
    Each dict maps feature keys to Tensor or SparseTensor objects.
  '''

  # Get feature specification based on transform output
  transformed_feature_spec = (
      tf_transform_output.transformed_feature_spec().copy())

  # Create batches of features and labels
  dataset = tf.data.experimental.make_batched_features_dataset(
      file_pattern=file_pattern,
      batch_size=batch_size,
      features=transformed_feature_spec,
      reader=_gzip_reader_fn,
      num_epochs=num_epochs,
      label_key=LABEL_KEY)

  return dataset


def model_builder(hp):
  '''
  Builds the model and sets up the hyperparameters to tune.

  Args:
    hp - Keras tuner object

  Returns:
    model with hyperparameters to tune
  '''

  # Initialize the Sequential API and start stacking the layers
  model = keras.Sequential()
```

```
  model.add(keras.layers.Flatten(input_shape=(28, 28, 1)))

  # 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', name='dense_1'))

  # Add next layers
  model.add(keras.layers.Dropout(0.2))
  model.add(keras.layers.Dense(10, activation='softmax'))

  # 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(),
                metrics=['accuracy'])

  return model

def tuner_fn(fn_args: FnArgs) -> TunerFnResult:
  """Build the tuner using the KerasTuner API.
  Args:
    fn_args: Holds args as name/value pairs.

      - working_dir: working dir for tuning.
      - train_files: List of file paths containing training tf.Example data.
      - eval_files: List of file paths containing eval tf.Example data.
      - train_steps: number of train steps.
      - eval_steps: number of eval steps.
      - schema_path: optional schema of the input data.
      - transform_graph_path: optional transform graph produced by TFT.

  Returns:
    A namedtuple contains the following:
      - tuner: A BaseTuner that will be used for tuning.
      - fit_kwargs: Args to pass to tuner's run_trial function for fitting the
                    model , e.g., the training and validation dataset. Required
                    args depend on the above tuner's implementation.
  """

  # Define tuner search strategy
  tuner = kt.Hyperband(model_builder,
                   objective='val_accuracy',
                   max_epochs=10,
                   factor=3,
                   directory=fn_args.working_dir,
                   project_name='kt_hyperband')

  # Load transform output
  tf_transform_output = tft.TFTransformOutput(fn_args.transform_graph_path)

  # Use _input_fn() to extract input features and labels from the train and val set
  train_set = _input_fn(fn_args.train_files[0], tf_transform_output)
  val_set = _input_fn(fn_args.eval_files[0], tf_transform_output)


  return TunerFnResult(
      tuner=tuner,
      fit_kwargs={
          "callbacks":[stop_early],
          'x': train_set,
          'validation_data': val_set,
          'steps_per_epoch': fn_args.train_steps,
          'validation_steps': fn_args.eval_steps
      }
  )
```

Since you passed `500` in the `num_steps` of the train args, this means that some examples will be skipped. This will likely result in lower accuracy readings but will save time in doing the hypertuning. Try modifying this value later and see if you arrive at the same set of hyperparameters.

```
from tfx.proto import trainer_pb2

# Setup the Tuner component
tuner = Tuner(
    module_file=_tuner_module_file,
    examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    schema=schema_gen.outputs['schema'],
    train_args=trainer_pb2.TrainArgs(splits=['train'], num_steps=500),
    eval_args=trainer_pb2.EvalArgs(splits=['eval'], num_steps=100)
    )
```

```
# Run the component. This will take around 5 minutes to run.
context.run(tuner, enable_cache=False)
```

```
ERROR:absl:udf_utils.get_fn {'module_file': None, 'tuner_fn': None, 'train_args': '{\n  "num_steps": 500,\n  "spli
ts": [\n    "train"\n  ]\n}', 'eval_args': '{\n  "num_steps": 100,\n  "splits": [\n    "eval"\n  ]\n}', 'tune_args
': None, 'custom_config': 'null', 'module_path': 'tuner@./pipeline/_wheels/tfx_user_code_Tuner-0.0+fb669ea9388d791
678ab2cac445c2422ea4e34ba71e98b6d6d2d0dcc5c0d6b53-py3-none-any.whl'} 'tuner_fn'
WARNING:absl:Examples artifact does not have payload_format custom property. Falling back to FORMAT_TF_EXAMPLE
WARNING:absl:Examples artifact does not have payload_format custom property. Falling back to FORMAT_TF_EXAMPLE
WARNING:absl:Examples artifact does not have payload_format custom property. Falling back to FORMAT_TF_EXAMPLE
```

Search space summary

|-Default search space size: 2

units (Int)

|-default: None

|-max_value: 512

|-min_value: 32

|-sampling: None

|-step: 32

learning_rate (Choice)

|-default: 0.01

|-ordered: True

|-values: [0.01, 0.001, 0.0001]

```
Epoch 1/2
500/500 [==============================] - 6s 6ms/step - loss: 0.9779 - accuracy: 0.6593 - val_loss: 0.6228 - val_
accuracy: 0.7622
Epoch 2/2
500/500 [==============================] - 2s 5ms/step - loss: 0.6568 - accuracy: 0.7615 - val_loss: 0.5327 - val_
accuracy: 0.8012
```

Trial complete

Trial summary

|-Trial ID: e714457907e9fa266ba7219ea4c7724d

|-Score: 0.8012499809265137

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 64

```
Epoch 1/2
500/500 [==============================] - 3s 5ms/step - loss: 0.9707 - accuracy: 0.6909 - val_loss: 0.5689 - val_
accuracy: 0.7997
Epoch 2/2
500/500 [==============================] - 3s 5ms/step - loss: 0.6231 - accuracy: 0.7806 - val_loss: 0.5186 - val_
accuracy: 0.8219
```

Trial complete

Trial summary

|-Trial ID: 73077bd26b7a50722c7d30178aa20be3

|-Score: 0.8218749761581421

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 320

```
Epoch 1/2
500/500 [==============================] - 3s 5ms/step - loss: 1.0319 - accuracy: 0.6816 - val_loss: 0.5458 - val_
accuracy: 0.8000
Epoch 2/2
500/500 [==============================] - 3s 5ms/step - loss: 0.6223 - accuracy: 0.7696 - val_loss: 0.4695 - val_
accuracy: 0.8281
```

Trial complete

Trial summary

|-Trial ID: 86747f17f6aee4d75062647ea75ce59b

|-Score: 0.828125

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 224

```
Epoch 1/2
500/500 [==============================] - 3s 5ms/step - loss: 1.4505 - accuracy: 0.5328 - val_loss: 0.7439 - val_
accuracy: 0.7575
Epoch 2/2
500/500 [==============================] - 2s 5ms/step - loss: 0.7505 - accuracy: 0.7460 - val_loss: 0.6137 - val_
accuracy: 0.7981
```

Trial complete

Trial summary

|-Trial ID: d08388f45327b2404c204bc216ce2521

|-Score: 0.7981250286102295

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 128

```
Epoch 1/2
500/500 [==============================] - 3s 5ms/step - loss: 1.3135 - accuracy: 0.5635 - val_loss: 0.6694 - val_
accuracy: 0.7887
Epoch 2/2
500/500 [==============================] - 3s 5ms/step - loss: 0.6522 - accuracy: 0.7864 - val_loss: 0.5384 - val_
accuracy: 0.8194
```

Trial complete

Trial summary

|-Trial ID: 62c398b9253be23c50b5224c49066dd3

|-Score: 0.8193749785423279

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 288

```
Epoch 1/2
500/500 [==============================] - 3s 5ms/step - loss: 1.0286 - accuracy: 0.6730 - val_loss: 0.5429 - val_
accuracy: 0.7962
Epoch 2/2
500/500 [==============================] - 3s 5ms/step - loss: 0.6151 - accuracy: 0.7740 - val_loss: 0.5089 - val_
accuracy: 0.8106
```

Trial complete

Trial summary

|-Trial ID: c41308efa7f430734b68585da976b73c

|-Score: 0.8106250166893005

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 192

```
Epoch 1/2
500/500 [==============================] - 3s 5ms/step - loss: 1.1118 - accuracy: 0.5753 - val_loss: 0.6304 - val_
accuracy: 0.8034
Epoch 2/2
500/500 [==============================] - 3s 5ms/step - loss: 0.8032 - accuracy: 0.6904 - val_loss: 0.5479 - val_
accuracy: 0.8128
```

Trial complete

Trial summary

|-Trial ID: 58d429975e985ed424db30a19a20df80

|-Score: 0.8128125071525574

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 32

```
Epoch 1/2
500/500 [==============================] - 3s 6ms/step - loss: 1.0424 - accuracy: 0.6844 - val_loss: 0.5361 - val_
accuracy: 0.7972
Epoch 2/2
500/500 [==============================] - 3s 6ms/step - loss: 0.6105 - accuracy: 0.7779 - val_loss: 0.5374 - val_
accuracy: 0.8109
```

Trial complete

Trial summary

|-Trial ID: 3d31e28e23205ab4747ccf075f3cbb84

|-Score: 0.8109375238418579

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 448

```
Epoch 1/2
500/500 [==============================] - 3s 6ms/step - loss: 1.3903 - accuracy: 0.5530 - val_loss: 0.6567 - val_
accuracy: 0.7975
Epoch 2/2
500/500 [==============================] - 3s 5ms/step - loss: 0.6966 - accuracy: 0.7699 - val_loss: 0.5640 - val_
accuracy: 0.8150
```

Trial complete

Trial summary

|-Trial ID: db01831f1e655b819d28e3b4701ddc19

|-Score: 0.8149999976158142

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 224

```
Epoch 1/2
500/500 [==============================] - 3s 6ms/step - loss: 1.8317 - accuracy: 0.3399 - val_loss: 1.0659 - val_
accuracy: 0.6616
Epoch 2/2
500/500 [==============================] - 3s 6ms/step - loss: 1.0955 - accuracy: 0.6366 - val_loss: 0.8285 - val_
accuracy: 0.7319
```

Trial complete

Trial summary

|-Trial ID: da812b4feff04d15a00d5497f405e3a9

|-Score: 0.7318750023841858

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 32

```
Epoch 1/2
500/500 [==============================] - 4s 6ms/step - loss: 1.2364 - accuracy: 0.5949 - val_loss: 0.6093 - val_
accuracy: 0.8044
Epoch 2/2
500/500 [==============================] - 3s 6ms/step - loss: 0.6369 - accuracy: 0.7893 - val_loss: 0.5238 - val_
accuracy: 0.8294
```

Trial complete

Trial summary

|-Trial ID: a4bdf1619f090599ceccd789be8e6bbe

|-Score: 0.8293750286102295

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 448

```
Epoch 1/2
500/500 [==============================] - 3s 6ms/step - loss: 1.0261 - accuracy: 0.6499 - val_loss: 0.5293 - val_
accuracy: 0.8112
Epoch 2/2
500/500 [==============================] - 3s 6ms/step - loss: 0.5651 - accuracy: 0.7982 - val_loss: 0.4218 - val_
accuracy: 0.8450
```

Trial complete

Trial summary

|-Trial ID: 8ab377f462d64ad6ace35d2afdab02dc

|-Score: 0.8450000286102295

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 2

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 64

```
Epoch 3/4
500/500 [==============================] - 3s 6ms/step - loss: 1.0715 - accuracy: 0.6264 - val_loss: 0.5322 - val_
accuracy: 0.8144
Epoch 4/4
500/500 [==============================] - 3s 6ms/step - loss: 0.5506 - accuracy: 0.8078 - val_loss: 0.4379 - val_
accuracy: 0.8516
```

Trial complete

Trial summary

|-Trial ID: 9e52616340338ab3403ad0a8ca72ec87

|-Score: 0.8515625

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 4

|-tuner/initial_epoch: 2

|-tuner/round: 1

|-tuner/trial_id: 8ab377f462d64ad6ace35d2afdab02dc

|-units: 64

```
Epoch 3/4
500/500 [==============================] - 4s 6ms/step - loss: 1.2317 - accuracy: 0.6001 - val_loss: 0.6275 - val_
accuracy: 0.7978
Epoch 4/4
500/500 [==============================] - 3s 6ms/step - loss: 0.6313 - accuracy: 0.7841 - val_loss: 0.5378 - val_
accuracy: 0.8184
```

Trial complete

Trial summary

|-Trial ID: 4479e0a4a09ecb2d44f75ba5ff4cce47

|-Score: 0.8184375166893005

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 2

|-tuner/epochs: 4

|-tuner/initial_epoch: 2

|-tuner/round: 1

|-tuner/trial_id: a4bdf1619f090599ceccd789be8e6bbe

|-units: 448

```
Epoch 3/4
500/500 [==============================] - 3s 6ms/step - loss: 0.9817 - accuracy: 0.6749 - val_loss: 0.5755 - val_
accuracy: 0.7797
Epoch 4/4
500/500 [==============================] - 3s 6ms/step - loss: 0.6437 - accuracy: 0.7638 - val_loss: 0.4969 - val_
accuracy: 0.8244
```

Trial complete

Trial summary

|-Trial ID: 7a6de832694cfa1bf2ec46b361e8fff9

|-Score: 0.8243749737739563

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 2

|-tuner/epochs: 4

|-tuner/initial_epoch: 2

|-tuner/round: 1

|-tuner/trial_id: 86747f17f6aee4d75062647ea75ce59b

|-units: 224

```
Epoch 3/4
500/500 [==============================] - 4s 6ms/step - loss: 1.0233 - accuracy: 0.6841 - val_loss: 0.5497 - val_
accuracy: 0.7919
Epoch 4/4
500/500 [==============================] - 3s 6ms/step - loss: 0.6244 - accuracy: 0.7628 - val_loss: 0.5614 - val_
accuracy: 0.8163
```

Trial complete

Trial summary

|-Trial ID: 081edce46ed46559a672ac248ca070ac

|-Score: 0.8162500262260437

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 2

|-tuner/epochs: 4

|-tuner/initial_epoch: 2

|-tuner/round: 1

|-tuner/trial_id: 73077bd26b7a50722c7d30178aa20be3

|-units: 320

```
Epoch 5/10
500/500 [==============================] - 4s 6ms/step - loss: 1.0456 - accuracy: 0.6464 - val_loss: 0.5465 - val_
accuracy: 0.8203
Epoch 6/10
500/500 [==============================] - 3s 6ms/step - loss: 0.5567 - accuracy: 0.8041 - val_loss: 0.4907 - val_
accuracy: 0.8263
Epoch 7/10
500/500 [==============================] - 3s 6ms/step - loss: 0.4955 - accuracy: 0.8225 - val_loss: 0.4186 - val_
accuracy: 0.8525
Epoch 8/10
500/500 [==============================] - 3s 6ms/step - loss: 0.4514 - accuracy: 0.8394 - val_loss: 0.4012 - val_
accuracy: 0.8544
Epoch 9/10
500/500 [==============================] - 3s 6ms/step - loss: 0.4547 - accuracy: 0.8374 - val_loss: 0.4142 - val_
accuracy: 0.8512
Epoch 10/10
500/500 [==============================] - 3s 6ms/step - loss: 0.4185 - accuracy: 0.8488 - val_loss: 0.3985 - val_
accuracy: 0.8575
```

Trial complete

Trial summary

|-Trial ID: f07d549ad19d270d9fff5de9aa6e0c43

|-Score: 0.8575000166893005

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 10

|-tuner/initial_epoch: 4

|-tuner/round: 2

|-tuner/trial_id: 9e52616340338ab3403ad0a8ca72ec87

|-units: 64

```
Epoch 5/10
500/500 [==============================] - 4s 6ms/step - loss: 0.9898 - accuracy: 0.6775 - val_loss: 0.5458 - val_
accuracy: 0.8094
Epoch 6/10
500/500 [==============================] - 3s 6ms/step - loss: 0.6291 - accuracy: 0.7695 - val_loss: 0.4654 - val_
accuracy: 0.8359
Epoch 7/10
500/500 [==============================] - 3s 6ms/step - loss: 0.5783 - accuracy: 0.7945 - val_loss: 0.4406 - val_
accuracy: 0.8425
Epoch 8/10
500/500 [==============================] - 3s 6ms/step - loss: 0.5576 - accuracy: 0.7976 - val_loss: 0.4853 - val_
accuracy: 0.8306
Epoch 9/10
500/500 [==============================] - 3s 6ms/step - loss: 0.5669 - accuracy: 0.7934 - val_loss: 0.4593 - val_
```

```
accuracy: 0.8353
Epoch 10/10
500/500 [==============================] - 3s 6ms/step - loss: 0.5611 - accuracy: 0.7954 - val_loss: 0.4564 - val_
accuracy: 0.8419
```

Trial complete

Trial summary

|-Trial ID: acb5b6b7858951e871d6d62463828747

|-Score: 0.8424999713897705

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 2

|-tuner/epochs: 10

|-tuner/initial_epoch: 4

|-tuner/round: 2

|-tuner/trial_id: 7a6de832694cfa1bf2ec46b361e8fff9

|-units: 224

```
Epoch 1/4
500/500 [==============================] - 4s 6ms/step - loss: 1.2691 - accuracy: 0.5831 - val_loss: 0.6466 - val_
accuracy: 0.7853
Epoch 2/4
500/500 [==============================] - 3s 6ms/step - loss: 0.6384 - accuracy: 0.7873 - val_loss: 0.5378 - val_
accuracy: 0.8175
Epoch 3/4
500/500 [==============================] - 3s 6ms/step - loss: 0.5502 - accuracy: 0.8100 - val_loss: 0.4888 - val_
accuracy: 0.8356
Epoch 4/4
500/500 [==============================] - 3s 6ms/step - loss: 0.4956 - accuracy: 0.8321 - val_loss: 0.4622 - val_
accuracy: 0.8469
```

Trial complete

Trial summary

|-Trial ID: bf036cf61be8282ecdd1175bc8e8df86

|-Score: 0.846875011920929

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 1

|-tuner/epochs: 4

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 384

```
Epoch 1/4
500/500 [==============================] - 4s 6ms/step - loss: 1.0224 - accuracy: 0.6821 - val_loss: 0.6252 - val_
accuracy: 0.7850
Epoch 2/4
500/500 [==============================] - 3s 6ms/step - loss: 0.6596 - accuracy: 0.7700 - val_loss: 0.5618 - val_
accuracy: 0.7944
Epoch 3/4
500/500 [==============================] - 3s 6ms/step - loss: 0.5835 - accuracy: 0.7854 - val_loss: 0.4557 - val_
accuracy: 0.8325
Epoch 4/4
500/500 [==============================] - 3s 6ms/step - loss: 0.5774 - accuracy: 0.7946 - val_loss: 0.5137 - val_
accuracy: 0.8156
```

Trial complete

Trial summary

|-Trial ID: 93de35c852dcb6e1546551e3dc46654f

|-Score: 0.8324999809265137

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 1

|-tuner/epochs: 4

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 480

```
Epoch 1/4
500/500 [==============================] - 4s 6ms/step - loss: 0.8147 - accuracy: 0.7087 - val_loss: 0.5507 - val_
accuracy: 0.7966
Epoch 2/4
500/500 [==============================] - 3s 6ms/step - loss: 0.5228 - accuracy: 0.8139 - val_loss: 0.4387 - val_
accuracy: 0.8409
Epoch 3/4
500/500 [==============================] - 3s 6ms/step - loss: 0.4359 - accuracy: 0.8402 - val_loss: 0.4182 - val_
accuracy: 0.8459
Epoch 4/4
500/500 [==============================] - 3s 6ms/step - loss: 0.4170 - accuracy: 0.8470 - val_loss: 0.4119 - val_
accuracy: 0.8444
```

Trial complete

Trial summary

|-Trial ID: d3a841dcdee49cc9fc16cb7056ba9e1c

|-Score: 0.8459374904632568

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 1

|-tuner/epochs: 4

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 320

```
Epoch 1/4
500/500 [==============================] - 4s 6ms/step - loss: 1.4879 - accuracy: 0.5021 - val_loss: 0.7311 - val_
accuracy: 0.7644
Epoch 2/4
500/500 [==============================] - 3s 6ms/step - loss: 0.7316 - accuracy: 0.7625 - val_loss: 0.5886 - val_
accuracy: 0.8116
Epoch 3/4
500/500 [==============================] - 3s 6ms/step - loss: 0.6351 - accuracy: 0.7881 - val_loss: 0.5631 - val_
accuracy: 0.8138
Epoch 4/4
500/500 [==============================] - 3s 6ms/step - loss: 0.5679 - accuracy: 0.8110 - val_loss: 0.5567 - val_
accuracy: 0.8141
```

Trial complete

Trial summary

|-Trial ID: f234a62021bfb7ca4a776e072fa3a585

|-Score: 0.8140624761581421

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 1

|-tuner/epochs: 4

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 160

```
Epoch 1/4
500/500 [==============================] - 4s 6ms/step - loss: 0.9226 - accuracy: 0.6786 - val_loss: 0.6187 - val_
accuracy: 0.7897
Epoch 2/4
500/500 [==============================] - 3s 6ms/step - loss: 0.6363 - accuracy: 0.7689 - val_loss: 0.5164 - val_
accuracy: 0.8075
Epoch 3/4
500/500 [==============================] - 3s 6ms/step - loss: 0.5911 - accuracy: 0.7875 - val_loss: 0.5359 - val_
accuracy: 0.7959
Epoch 4/4
500/500 [==============================] - 3s 6ms/step - loss: 0.5619 - accuracy: 0.8046 - val_loss: 0.4730 - val_
accuracy: 0.8313
```

Trial complete

Trial summary

|-Trial ID: c31065327415aa1e33446cdbc6b6c8e6

|-Score: 0.831250011920929

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 1

|-tuner/epochs: 4

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 128

```
Epoch 1/4
500/500 [==============================] - 4s 6ms/step - loss: 0.8498 - accuracy: 0.7038 - val_loss: 0.4559 - val_
accuracy: 0.8459
Epoch 2/4
500/500 [==============================] - 3s 6ms/step - loss: 0.5010 - accuracy: 0.8242 - val_loss: 0.4200 - val_
accuracy: 0.8434
Epoch 3/4
500/500 [==============================] - 3s 6ms/step - loss: 0.4506 - accuracy: 0.8423 - val_loss: 0.3813 - val_
accuracy: 0.8656
Epoch 4/4
500/500 [==============================] - 3s 6ms/step - loss: 0.4188 - accuracy: 0.8519 - val_loss: 0.3864 - val_
accuracy: 0.8619
```

Trial complete

Trial summary

|-Trial ID: 61e182d5fb0001e9db0042c990bbab29

|-Score: 0.8656250238418579

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 1

|-tuner/epochs: 4

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 192

```
Epoch 5/10
500/500 [==============================] - 4s 7ms/step - loss: 0.8558 - accuracy: 0.7071 - val_loss: 0.4902 - val_
accuracy: 0.8147
Epoch 6/10
500/500 [==============================] - 3s 6ms/step - loss: 0.5139 - accuracy: 0.8160 - val_loss: 0.4250 - val_
accuracy: 0.8484
Epoch 7/10
500/500 [==============================] - 3s 6ms/step - loss: 0.4509 - accuracy: 0.8346 - val_loss: 0.4218 - val_
accuracy: 0.8441
Epoch 8/10
500/500 [==============================] - 3s 6ms/step - loss: 0.4217 - accuracy: 0.8437 - val_loss: 0.3954 - val_
accuracy: 0.8584
Epoch 9/10
500/500 [==============================] - 3s 6ms/step - loss: 0.4011 - accuracy: 0.8544 - val_loss: 0.3898 - val_
accuracy: 0.8512
Epoch 10/10
500/500 [==============================] - 3s 6ms/step - loss: 0.3879 - accuracy: 0.8572 - val_loss: 0.3692 - val_
accuracy: 0.8697
```

Trial complete

Trial summary

|-Trial ID: 280a1c197eebecdb7613e6734e12e1c7

|-Score: 0.8696874976158142

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 1

|-tuner/epochs: 10

|-tuner/initial_epoch: 4

|-tuner/round: 1

|-tuner/trial_id: 61e182d5fb0001e9db0042c990bbab29

|-units: 192

```
Epoch 5/10
500/500 [==============================] - 4s 7ms/step - loss: 1.2366 - accuracy: 0.6034 - val_loss: 0.6091 - val_
accuracy: 0.8047
Epoch 6/10
500/500 [==============================] - 3s 6ms/step - loss: 0.6296 - accuracy: 0.7915 - val_loss: 0.5181 - val_
```

```
accuracy: 0.8291
Epoch 7/10
500/500 [==============================] - 3s 6ms/step - loss: 0.5651 - accuracy: 0.8123 - val_loss: 0.4933 - val_
accuracy: 0.8353
Epoch 8/10
500/500 [==============================] - 3s 7ms/step - loss: 0.4995 - accuracy: 0.8269 - val_loss: 0.4697 - val_
accuracy: 0.8406
Epoch 9/10
500/500 [==============================] - 3s 6ms/step - loss: 0.5081 - accuracy: 0.8235 - val_loss: 0.4581 - val_
accuracy: 0.8478
Epoch 10/10
500/500 [==============================] - 3s 7ms/step - loss: 0.4507 - accuracy: 0.8445 - val_loss: 0.4235 - val_
accuracy: 0.8553
```

Trial complete

Trial summary

|-Trial ID: bb641589a538f419c42badf9c2716f40

|-Score: 0.8553125262260437

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 1

|-tuner/epochs: 10

|-tuner/initial_epoch: 4

|-tuner/round: 1

|-tuner/trial_id: bf036cf61be8282ecdd1175bc8e8df86

|-units: 384

```
Epoch 1/10
500/500 [==============================] - 4s 7ms/step - loss: 1.0820 - accuracy: 0.6782 - val_loss: 0.6831 - val_
accuracy: 0.7716
Epoch 2/10
500/500 [==============================] - 3s 7ms/step - loss: 0.6382 - accuracy: 0.7690 - val_loss: 0.5528 - val_
accuracy: 0.8163
Epoch 3/10
500/500 [==============================] - 3s 7ms/step - loss: 0.5816 - accuracy: 0.7896 - val_loss: 0.4776 - val_
accuracy: 0.8300
Epoch 4/10
500/500 [==============================] - 3s 6ms/step - loss: 0.5694 - accuracy: 0.7951 - val_loss: 0.5110 - val_
accuracy: 0.8219
Epoch 5/10
500/500 [==============================] - 3s 7ms/step - loss: 0.5875 - accuracy: 0.7902 - val_loss: 0.4890 - val_
accuracy: 0.8150
Epoch 6/10
500/500 [==============================] - 3s 7ms/step - loss: 0.5475 - accuracy: 0.8048 - val_loss: 0.4417 - val_
accuracy: 0.8441
Epoch 7/10
500/500 [==============================] - 3s 7ms/step - loss: 0.5391 - accuracy: 0.8092 - val_loss: 0.4288 - val_
accuracy: 0.8478
Epoch 8/10
500/500 [==============================] - 3s 7ms/step - loss: 0.5456 - accuracy: 0.8076 - val_loss: 0.4987 - val_
accuracy: 0.8431
Epoch 9/10
500/500 [==============================] - 3s 7ms/step - loss: 0.5426 - accuracy: 0.8066 - val_loss: 0.4572 - val_
accuracy: 0.8459
Epoch 10/10
500/500 [==============================] - 3s 7ms/step - loss: 0.5448 - accuracy: 0.8082 - val_loss: 0.4400 - val_
accuracy: 0.8500
```

Trial complete

Trial summary

|-Trial ID: 135b0a15a05edaf19cff657ac99d39bd

|-Score: 0.8500000238418579

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01

|-tuner/bracket: 0

|-tuner/epochs: 10

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 384

```
Epoch 1/10
500/500 [==============================] - 4s 7ms/step - loss: 0.7890 - accuracy: 0.7267 - val_loss: 0.5001 - val_
```

```
accuracy: 0.8216
Epoch 2/10
500/500 [==============================] - 4s 7ms/step - loss: 0.4913 - accuracy: 0.8256 - val_loss: 0.4505 - val_
accuracy: 0.8366
Epoch 3/10
500/500 [==============================] - 4s 7ms/step - loss: 0.4456 - accuracy: 0.8409 - val_loss: 0.4160 - val_
accuracy: 0.8453
Epoch 4/10
500/500 [==============================] - 4s 7ms/step - loss: 0.4138 - accuracy: 0.8476 - val_loss: 0.4220 - val_
accuracy: 0.8500
Epoch 5/10
500/500 [==============================] - 4s 7ms/step - loss: 0.3945 - accuracy: 0.8556 - val_loss: 0.3700 - val_
accuracy: 0.8631
Epoch 6/10
500/500 [==============================] - 4s 7ms/step - loss: 0.3862 - accuracy: 0.8537 - val_loss: 0.3569 - val_
accuracy: 0.8653
Epoch 7/10
500/500 [==============================] - 4s 7ms/step - loss: 0.3698 - accuracy: 0.8639 - val_loss: 0.3786 - val_
accuracy: 0.8650
Epoch 8/10
500/500 [==============================] - 4s 7ms/step - loss: 0.3651 - accuracy: 0.8673 - val_loss: 0.3440 - val_
accuracy: 0.8759
Epoch 9/10
500/500 [==============================] - 3s 7ms/step - loss: 0.3487 - accuracy: 0.8730 - val_loss: 0.3435 - val_
accuracy: 0.8691
Epoch 10/10
500/500 [==============================] - 3s 7ms/step - loss: 0.3427 - accuracy: 0.8801 - val_loss: 0.3950 - val_
accuracy: 0.8612
```

Trial complete

Trial summary

|-Trial ID: 50b0ed4b1a2363e15f00115ca2c72607

|-Score: 0.8759375214576721

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 0

|-tuner/epochs: 10

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 448

```
Epoch 1/10
500/500 [==============================] - 4s 7ms/step - loss: 0.7863 - accuracy: 0.7302 - val_loss: 0.4708 - val_
accuracy: 0.8350
Epoch 2/10
500/500 [==============================] - 3s 7ms/step - loss: 0.4853 - accuracy: 0.8295 - val_loss: 0.4240 - val_
accuracy: 0.8497
Epoch 3/10
500/500 [==============================] - 4s 7ms/step - loss: 0.4441 - accuracy: 0.8396 - val_loss: 0.4187 - val_
accuracy: 0.8500
Epoch 4/10
500/500 [==============================] - 3s 7ms/step - loss: 0.4150 - accuracy: 0.8507 - val_loss: 0.4279 - val_
accuracy: 0.8425
Epoch 5/10
500/500 [==============================] - 3s 7ms/step - loss: 0.4071 - accuracy: 0.8525 - val_loss: 0.4152 - val_
accuracy: 0.8491
Epoch 6/10
500/500 [==============================] - 3s 7ms/step - loss: 0.3737 - accuracy: 0.8625 - val_loss: 0.4021 - val_
accuracy: 0.8509
Epoch 7/10
500/500 [==============================] - 3s 7ms/step - loss: 0.3611 - accuracy: 0.8650 - val_loss: 0.3880 - val_
accuracy: 0.8581
Epoch 8/10
500/500 [==============================] - 3s 7ms/step - loss: 0.3645 - accuracy: 0.8656 - val_loss: 0.3791 - val_
accuracy: 0.8578
Epoch 9/10
500/500 [==============================] - 4s 7ms/step - loss: 0.3499 - accuracy: 0.8708 - val_loss: 0.3618 - val_
accuracy: 0.8672
Epoch 10/10
500/500 [==============================] - 3s 7ms/step - loss: 0.3361 - accuracy: 0.8811 - val_loss: 0.3512 - val_
accuracy: 0.8712
```

Trial complete

Trial summary

|-Trial ID: 141caf0aa307f2a8dfcaeee78f389a7c

|-Score: 0.8712499737739563

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 0

|-tuner/epochs: 10

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 512

```
Epoch 1/10
500/500 [==============================] - 4s 7ms/step - loss: 1.2195 - accuracy: 0.6130 - val_loss: 0.6227 - val_
accuracy: 0.7941
Epoch 2/10
500/500 [==============================] - 4s 7ms/step - loss: 0.6039 - accuracy: 0.8034 - val_loss: 0.5160 - val_
accuracy: 0.8278
Epoch 3/10
500/500 [==============================] - 4s 7ms/step - loss: 0.5284 - accuracy: 0.8242 - val_loss: 0.4612 - val_
accuracy: 0.8491
Epoch 4/10
500/500 [==============================] - 4s 7ms/step - loss: 0.4963 - accuracy: 0.8328 - val_loss: 0.4669 - val_
accuracy: 0.8369
Epoch 5/10
500/500 [==============================] - 4s 7ms/step - loss: 0.4505 - accuracy: 0.8474 - val_loss: 0.4098 - val_
accuracy: 0.8600
Epoch 6/10
500/500 [==============================] - 4s 7ms/step - loss: 0.4727 - accuracy: 0.8380 - val_loss: 0.4164 - val_
accuracy: 0.8634
Epoch 7/10
500/500 [==============================] - 4s 7ms/step - loss: 0.4234 - accuracy: 0.8563 - val_loss: 0.3945 - val_
accuracy: 0.8619
Epoch 8/10
500/500 [==============================] - 4s 7ms/step - loss: 0.4310 - accuracy: 0.8503 - val_loss: 0.3925 - val_
accuracy: 0.8634
Epoch 9/10
500/500 [==============================] - 3s 7ms/step - loss: 0.4099 - accuracy: 0.8597 - val_loss: 0.3950 - val_
accuracy: 0.8653
Epoch 10/10
500/500 [==============================] - 4s 7ms/step - loss: 0.3995 - accuracy: 0.8631 - val_loss: 0.4008 - val_
accuracy: 0.8603
```

Trial complete

Trial summary

|-Trial ID: c7f02a011920ac3dd6c6f8195846da2a

|-Score: 0.8653125166893005

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 0

|-tuner/epochs: 10

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 480

Results summary

|-Results in ./pipeline/.temp/6/kt_hyperband

|-Showing 10 best trials

|-Objective(name='val_accuracy', direction='max')

Trial summary

|-Trial ID: 50b0ed4b1a2363e15f00115ca2c72607

|-Score: 0.8759375214576721

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 0

|-tuner/epochs: 10

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 448

Trial summary

|-Trial ID: 141caf0aa307f2a8dfcaeee78f389a7c

|-Score: 0.8712499737739563

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 0

|-tuner/epochs: 10

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 512

Trial summary

|-Trial ID: 280a1c197eebecdb7613e6734e12e1c7

|-Score: 0.8696874976158142

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 1

|-tuner/epochs: 10

|-tuner/initial_epoch: 4

|-tuner/round: 1

|-tuner/trial_id: 61e182d5fb0001e9db0042c990bbab29

|-units: 192

Trial summary

|-Trial ID: 61e182d5fb0001e9db0042c990bbab29

|-Score: 0.8656250238418579

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 1

|-tuner/epochs: 4

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 192

Trial summary

|-Trial ID: c7f02a011920ac3dd6c6f8195846da2a

|-Score: 0.8653125166893005

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001

|-tuner/bracket: 0

|-tuner/epochs: 10

|-tuner/initial_epoch: 0

|-tuner/round: 0

|-units: 480

Trial summary

|-Trial ID: f07d549ad19d270d9fff5de9aa6e0c43

|-Score: 0.8575000166893005

|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001

|-tuner/bracket: 2

|-tuner/epochs: 10

|-tuner/initial_epoch: 4

|-tuner/round: 2

|-tuner/trial_id: 9e52616340338ab3403ad0a8ca72ec87
|-units: 64

Trial summary

|-Trial ID: bb641589a538f419c42badf9c2716f40
|-Score: 0.8553125262260437
|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001
|-tuner/bracket: 1
|-tuner/epochs: 10
|-tuner/initial_epoch: 4
|-tuner/round: 1
|-tuner/trial_id: bf036cf61be8282ecdd1175bc8e8df86
|-units: 384

Trial summary

|-Trial ID: 9e52616340338ab3403ad0a8ca72ec87
|-Score: 0.8515625
|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001
|-tuner/bracket: 2
|-tuner/epochs: 4
|-tuner/initial_epoch: 2
|-tuner/round: 1
|-tuner/trial_id: 8ab377f462d64ad6ace35d2afdab02dc
|-units: 64

Trial summary

|-Trial ID: 135b0a15a05edaf19cff657ac99d39bd
|-Score: 0.8500000238418579
|-Best step: 0

Hyperparameters:

|-learning_rate: 0.01
|-tuner/bracket: 0
|-tuner/epochs: 10
|-tuner/initial_epoch: 0
|-tuner/round: 0
|-units: 384

Trial summary

|-Trial ID: bf036cf61be8282ecdd1175bc8e8df86
|-Score: 0.846875011920929
|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001
|-tuner/bracket: 1
|-tuner/epochs: 4
|-tuner/initial_epoch: 0
|-tuner/round: 0
|-units: 384

▼**ExecutionResult** at 0x7fd62f51afd0

.execution_id	6	
.component	▶**Tuner** at 0x7fd679e8bad0	
.component.inputs	**['examples']**	▶**Channel** of type **'Examples'** (1 artifact) at 0x7fd62f6b8e10
	['schema']	▶**Channel** of type **'Schema'** (1 artifact) at 0x7fd631143310

Trainer

Like the Tuner component, the Trainer component also requires a module file to setup the training process. It will look for a `run_fn()` function that defines and trains the model. The steps will look similar to the tuner module file:

- Define the model - You can get the results of the Tuner component through the `fn_args.hyperparameters` argument. You will see it passed into the `model_builder()` function below. If you didn't run `Tuner`, then you can just explicitly define the number of hidden units and learning rate.
- Load the train and validation sets - You have done this in the Tuner component. For this module, you will pass in a `num_epochs` value (10) to indicate how many batches will be prepared. You can opt not to do this and pass a `num_steps` value as before.
- Setup and train the model - This will look very familiar if you're already used to the Keras Models Training API. You can pass in callbacks like the TensorBoard callback so you can visualize the results later.
- Save the model - This is needed so you can analyze and serve your model. You will get to do this in later parts of the course and specialization.

```
# Declare trainer module file
_trainer_module_file = 'trainer.py'
```

```
%%writefile {_trainer_module_file}

from tensorflow import keras
from typing import NamedTuple, Dict, Text, Any, List
from tfx.components.trainer.fn_args_utils import FnArgs, DataAccessor
import tensorflow as tf
import tensorflow_transform as tft

# Define the label key
LABEL_KEY = 'label_xf'

def _gzip_reader_fn(filenames):
  '''Load compressed dataset

  Args:
    filenames - filenames of TFRecords to load

  Returns:
    TFRecordDataset loaded from the filenames
  '''

  # Load the dataset. Specify the compression type since it is saved as `.gz`
  return tf.data.TFRecordDataset(filenames, compression_type='GZIP')


def _input_fn(file_pattern,
              tf_transform_output,
              num_epochs=None,
              batch_size=32) -> tf.data.Dataset:
  '''Create batches of features and labels from TF Records

  Args:
    file_pattern - List of files or patterns of file paths containing Example records.
    tf_transform_output - transform output graph
    num_epochs - Integer specifying the number of times to read through the dataset.
            If None, cycles through the dataset forever.
    batch_size - An int representing the number of records to combine in a single batch.

  Returns:
    A dataset of dict elements, (or a tuple of dict elements and label).
    Each dict maps feature keys to Tensor or SparseTensor objects.
  '''
  transformed_feature_spec = (
      tf_transform_output.transformed_feature_spec().copy())

  dataset = tf.data.experimental.make_batched_features_dataset(
      file_pattern=file_pattern,
      batch_size=batch_size,
      features=transformed_feature_spec,
      reader=_gzip_reader_fn,
      num_epochs=num_epochs,
      label_key=LABEL_KEY)

  return dataset


def model_builder(hp):
  '''
  Builds the model and sets up the hyperparameters to tune.

  Args:
    hp - Keras tuner object

  Returns:
    model with hyperparameters to tune
  '''

  # Initialize the Sequential API and start stacking the layers
  model = keras.Sequential()
  model.add(keras.layers.Flatten(input_shape=(28, 28, 1)))

  # Get the number of units from the Tuner results
  hp_units = hp.get('units')
  model.add(keras.layers.Dense(units=hp_units, activation='relu'))

  # Add next layers
  model.add(keras.layers.Dropout(0.2))
  model.add(keras.layers.Dense(10, activation='softmax'))

  # Get the learning rate from the Tuner results
  hp_learning_rate = hp.get('learning_rate')

  # Setup model for training
```

```
  model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate),
                loss=keras.losses.SparseCategoricalCrossentropy(),
                metrics=['accuracy'])

  # Print the model summary
  model.summary()

  return model


def run_fn(fn_args: FnArgs) -> None:
  """Defines and trains the model.
  Args:
    fn_args: Holds args as name/value pairs. Refer here for the complete attributes:
    https://www.tensorflow.org/tfx/api_docs/python/tfx/components/trainer/fn_args_utils/FnArgs#attributes
  """

  # Callback for TensorBoard
  tensorboard_callback = tf.keras.callbacks.TensorBoard(
      log_dir=fn_args.model_run_dir, update_freq='batch')

  # Load transform output
  tf_transform_output = tft.TFTransformOutput(fn_args.transform_graph_path)

  # Create batches of data good for 10 epochs
  train_set = _input_fn(fn_args.train_files[0], tf_transform_output, 10)
  val_set = _input_fn(fn_args.eval_files[0], tf_transform_output, 10)

  # Load best hyperparameters
  hp = fn_args.hyperparameters.get('values')

  # Build the model
  model = model_builder(hp)

  # Train the model
  model.fit(
```

```
# Setup the Trainer component
trainer = Trainer(
    module_file=_trainer_module_file,
    examples=transform.outputs['transformed_examples'],
    hyperparameters=tuner.outputs['best_hyperparameters'],
    transform_graph=transform.outputs['transform_graph'],
    schema=schema_gen.outputs['schema'],
    train_args=trainer_pb2.TrainArgs(splits=['train']),
    eval_args=trainer_pb2.EvalArgs(splits=['eval']))
```

Take note that when re-training your model, you don't always have to retune your hyperparameters. Once you have a set that you think performs well, you can just import it with the ImporterNode as shown in the official docs:

```
hparams_importer = ImporterNode(
    instance_name='import_hparams',
    # This can be Tuner's output file or manually edited file. The file contains
    # text format of hyperparameters (kerastuner.HyperParameters.get_config())
    source_uri='path/to/best_hyperparameters.txt',
    artifact_type=HyperParameters)

trainer = Trainer(
    ...
    # An alternative is directly use the tuned hyperparameters in Trainer's user
    # module code and set hyperparameters to None here.
    hyperparameters = hparams_importer.outputs['result'])
```

```
# Run the component
context.run(trainer, enable_cache=False)
```

```
WARNING:absl:Examples artifact does not have payload_format custom property. Falling back to FORMAT_TF_EXAMPLE
WARNING:absl:Examples artifact does not have payload_format custom property. Falling back to FORMAT_TF_EXAMPLE
```

```
WARNING:absl:Examples artifact does not have payload_format custom property. Falling back to FORMAT_TF_EXAMPLE
ERROR:absl:udf_utils.get_fn {'train_args': '{\n  "splits": [\n    "train"\n  ]\n}', 'eval_args': '{\n  "splits": [\n    "eval"\n  ]\n}', 'module_file': None, 'run_fn': None, 'trainer_fn': None, 'custom_config': 'null', 'module_path': 'trainer@./pipeline/_wheels/tfx_user_code_Trainer-0.0+af80eb329330e0fb546dd47287700a6d9d013b8948736ddb2a41e38e7235e74e-py3-none-any.whl'} 'run_fn'
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_1 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 448)               351680    
_________________________________________________________________
dropout_1 (Dropout)          (None, 448)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                4490      
=================================================================
Total params: 356,170
Trainable params: 356,170
Non-trainable params: 0
_________________________________________________________________
14993/14993 [==============================] - 95s 6ms/step - loss: 0.4170 - accuracy: 0.8489 - val_loss: 0.3102 - val_accuracy: 0.8876
```

▼**ExecutionResult** at 0x7fd62f2c02d0

.execution_id	7
.component	**Trainer** at 0x7fd62da1ac10
.component.inputs	**['examples']** **Channel** of type **'Examples'** (1 artifact) at 0x7fd62f6b8e10 **['transform_graph']** **Channel** of type **'TransformGraph'** (1 artifact) at 0x7fd62f6b8310 **['schema']** **Channel** of type **'Schema'** (1 artifact) at 0x7fd631143310 **['hyperparameters']** **Channel** of type **'HyperParameters'** (1 artifact) at 0x7fd679e8be90
.component.outputs	**['model']** **Channel** of type **'Model'** (1 artifact) at 0x7fd62da1ad10 **['model_run']** **Channel** of type **'ModelRun'** (1 artifact) at 0x7fd630cf7510

Your model should now be saved in your pipeline directory and you can navigate through it as shown below. The file is saved as `saved_model.pb`.

```python
# Get artifact uri of trainer model output
model_artifact_dir = trainer.outputs['model'].get()[0].uri

# List subdirectories artifact uri
print(f'contents of model artifact directory:{os.listdir(model_artifact_dir)}')

# Define the model directory
model_dir = os.path.join(model_artifact_dir, 'Format-Serving')

# List contents of model directory
print(f'contents of model directory: {os.listdir(model_dir)}')
```

```
contents of model artifact directory:['Format-Serving']
contents of model directory: ['saved_model.pb', 'assets', 'variables']
```

You can also visualize the training results by loading the logs saved by the Tensorboard callback.

```python
model_run_artifact_dir = trainer.outputs['model_run'].get()[0].uri

%load_ext tensorboard
%tensorboard --logdir {model_run_artifact_dir}
```

Congratulations! You have now created an ML pipeline that includes hyperparameter tuning and model training. You will know more about the next components in future lessons but in the next section, you will first learn about a framework for automatically building ML pipelines: AutoML. Enjoy the rest of the course!