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*.

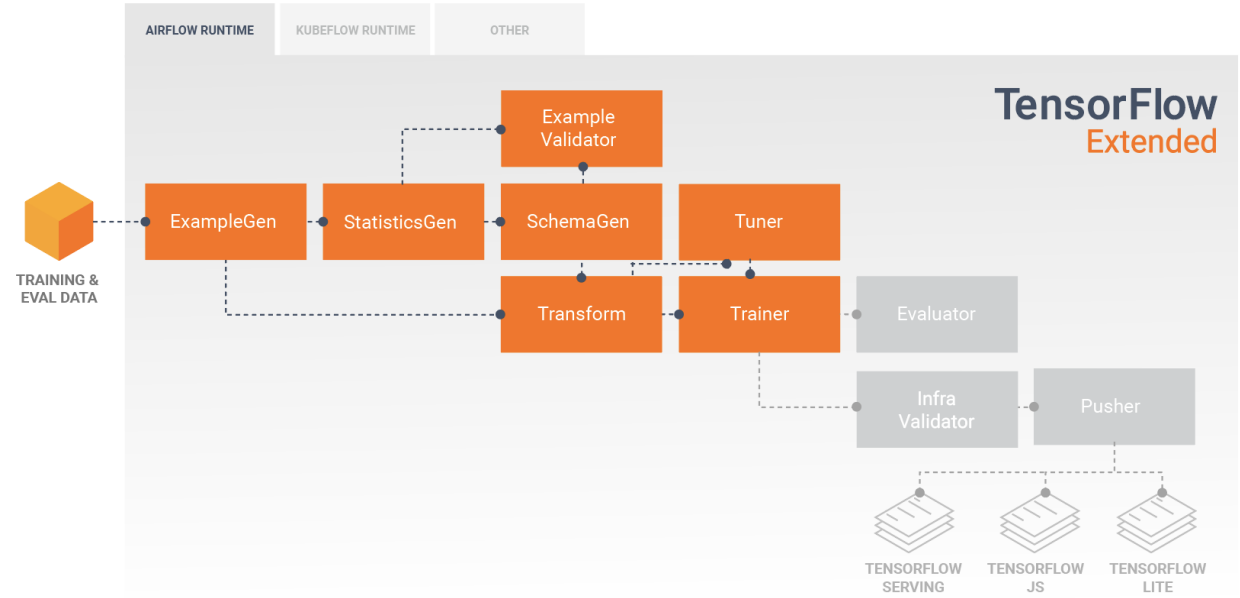

image source: https://www.tensorflow.org/tfx/guide

- 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
```

```
Requirement already satisfied: tensorflow-estimator<2.5.0,>=2.4.0 in /usr/local/lib/python
Requirement already satisfied: flatbuffers~=1.12.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: google-pasta~=0.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: tensorboard~=2.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: opt-einsum~=3.3.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: termcolor~=1.1.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: keras-preprocessing~=1.1.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: h5py~=2.10.0 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: gast==0.3.3 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: astunparse~=1.6.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/pyt
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: joblib<0.15,>=0.12 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: tensorflow-metadata<0.31,>=0.30 in /usr/local/lib/python3.7
Requirement already satisfied: pandas<2,>=1.0 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: ipywidgets<8,>=7 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: ipython<8,>=7 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/
Requirement already satisfied: backcall in /usr/local/lib/python3.7/dist-packages (from ip

Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-packages (from ip
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-packages (from i
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: widgetsnbextension~=3.5.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: nbformat>=4.2.0 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: tornado>=4.0 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages (from prc
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: Send2Trash in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: nbconvert in /usr/local/lib/python3.7/dist-packages (from n
Requirement already satisfied: terminado>=0.8.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.7/dist-packages (from j
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from i
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: testpath in /usr/local/lib/python3.7/dist-packages (from nb
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: defusedxml in /usr/local/lib/python3.7/dist-packages (from
```

```
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: bleach in /usr/local/lib/python3.7/dist-packages (from nbcc
Requirement already satisfied: webencodings in /usr/local/lib/python3.7/dist-packages (fro
```

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
Dl Completed...: 100%  4/4 [00:03<00:00, 1.16 url/s]
Dl Size...: 100%  29/29 [00:03<00:00, 8.53 MiB/s]
Extraction completed...: 100%  4/4 [00:03<00:00, 1.19 file/s]
Shuffling and writing examples to ./tempdir/fashion_mnist/3.0.1.incompleteSZIER7/fashion_mnis
75%  44950/60000 [00:00<00:00, 109168.54 examples/s]
Shuffling and writing examples to ./tempdir/fashion_mnist/3.0.1.incompleteSZIER7/fashion_mnis
0%  0/10000 [00:00<?, ? examples/s]
Dataset fashion_mnist downloaded and prepared to ./tempdir/fashion_mnist/3.0.1. Subsequent c
```

```
# 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 trai
    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)
```

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

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 Met
```

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 In
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 _TFRe
```

▼**ExecutionResult** at 0x7f9b8ddffc50

.execution_id	1
.component	▶**ImportExampleGen** at 0x7f9b2a20c890
.component.inputs	{}
.component.outputs	**['examples']** ▶**Channel** of type **'Examples'** (1 artifact) at 0x7f9b2a20c910

```
# 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 0x7f9b28eb6950

.execution_id	2
.component	**StatisticsGen** at 0x7f9b802d4d50
.component.inputs	**['examples']** **Channel** of type **'Examples'** (1 artifact) at 0x7f9b2a20c910
.component.outputs	**['statistics']** **Channel** of type **'ExampleStatistics'** (1 artifact) at

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 0x7f9b2a0ea110

.execution_id	3
.component	**SchemaGen** at 0x7f9b3c1903d0
.component.inputs	**['statistics']** **Channel** of type **'ExampleStatistics'** (1 artifact) at 0x7f9b802d4f50
.component.outputs	**['schema']** **Channel** of type **'Schema'** (1 artifact) at

```
# 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 0x7f9b3c19bfd0
.execution_id        4
.component           ExampleValidator at 0x7f9b3c199750
.component.inputs    ['statistics'] Channel of type 'ExampleStatistics' (1 artifact) at 0x7f9b802d4f50
                     ['schema']     Channel of type 'Schema' (1 artifact) at 0x7f9b3c1904d0
.component.outputs
```

```
# 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:
  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@./pipelin
WARNING:root:This output type hint will be ignored and not used for type-checking purposes.
WARNING:root:This output type hint will be ignored and not used for type-checking purposes.
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 0x7f9b3c1854d0

.execution_id	5		
.component	▶**Transform** at 0x7f9b2a51c3d0		
.component.inputs	**['examples']**	▶**Channel** of type **'Examples'** (1 artifact) at 0x7f9b2a20c910	
	['schema']	▶**Channel** of type **'Schema'** (1 artifact) at 0x7f9b3c1904d0	
.component.outputs	**['transform_graph']**	▶**Channel** of type **'TransformGraph'** (1 artifact) at 0x7f9b3c1b0050	
	['transformed_examples']	▶**Channel** of type **'Examples'** (1 artifact) at 0x7f9b28ad4fd0	

Tuner

As the name suggests, the [Tuner](https://www.tensorflow.org/tfx/guide/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
        }
    )
```

```
Writing tuner.py
```

With the module defined, you can now setup the Tuner component. You can see the description of each argument here.

Notice that we passed a `num_steps` argument to the train and eval args and this was used in the `steps_per_epoch` and `validation_steps` arguments in the tuner module above. This can be useful if you don't want to go through the entire dataset when tuning. For example, if you have 10GB of training data, it would be incredibly time consuming if you will iterate through it entirely just for one epoch and one set of hyperparameters. You can set the number of steps so your program will only go through a fraction of the dataset.

You can compute for the total number of steps in one epoch by: `number of examples / batch size`. For this particular example, we have `48000 examples / 32 (default size)` which equals `1500` steps per epoch for the train set (compute val steps from 12000 examples). 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)
```

|-learning_rate: 0.001
|-tuner/bracket: 1
|-tuner/epochs: 4
|-tuner/initial_epoch: 0
|-tuner/round: 0
|-units: 96

Trial summary

|-Trial ID: c2be151f6821c8334eef1453a4b73d7e
|-Score: 0.848437488079071
|-Best step: 0

Hyperparameters:

|-learning_rate: 0.0001
|-tuner/bracket: 1
|-tuner/epochs: 10
|-tuner/initial_epoch: 4
|-tuner/round: 1
|-tuner/trial_id: 4a2cad6030ef433b6f6a05e8dc995d34
|-units: 384

Trial summary

|-Trial ID: f222421150f97193a50db94492202ceb
|-Score: 0.8475000262260437
|-Best step: 0

Hyperparameters:

|-learning_rate: 0.001
|-tuner/bracket: 2
|-tuner/epochs: 2
|-tuner/initial_epoch: 0
|-tuner/round: 0
|-units: 416

Trial summary

|-Trial ID: 4a2cad6030ef433b6f6a05e8dc995d34
|-Score: 0.8475000262260437
|-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 0x7f9b27eae410

.execution_id	6
.component	▶**Tuner** at 0x7f9b28d62290
.component.inputs	
['examples']	▶**Channel** of type **'Examples'** (1 artifact) at 0x7f9b28ad4fd0
['schema']	▶**Channel** of type **'Schema'** (1 artifact) at 0x7f9b3c1904d0
['transform_graph']	▶**Channel** of type **'TransformGraph'** (1 artifact) at 0x7f9b3c1b0050