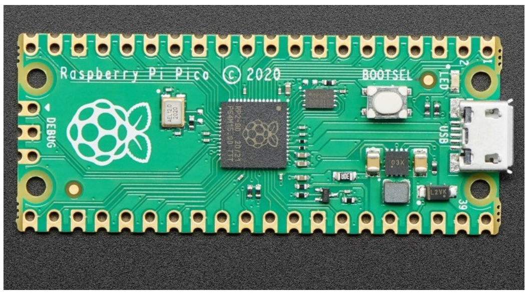
**********************Default***********************
Programming done on a Raspberry Pi4
Raspberry Pi Pico RP2040 with TensorFLow Lite
08/19/22

**********************Default***********************

Goal:

Step 1.

To program a Raspberry Pi Pico RP2040 with TensorFLow Lite.



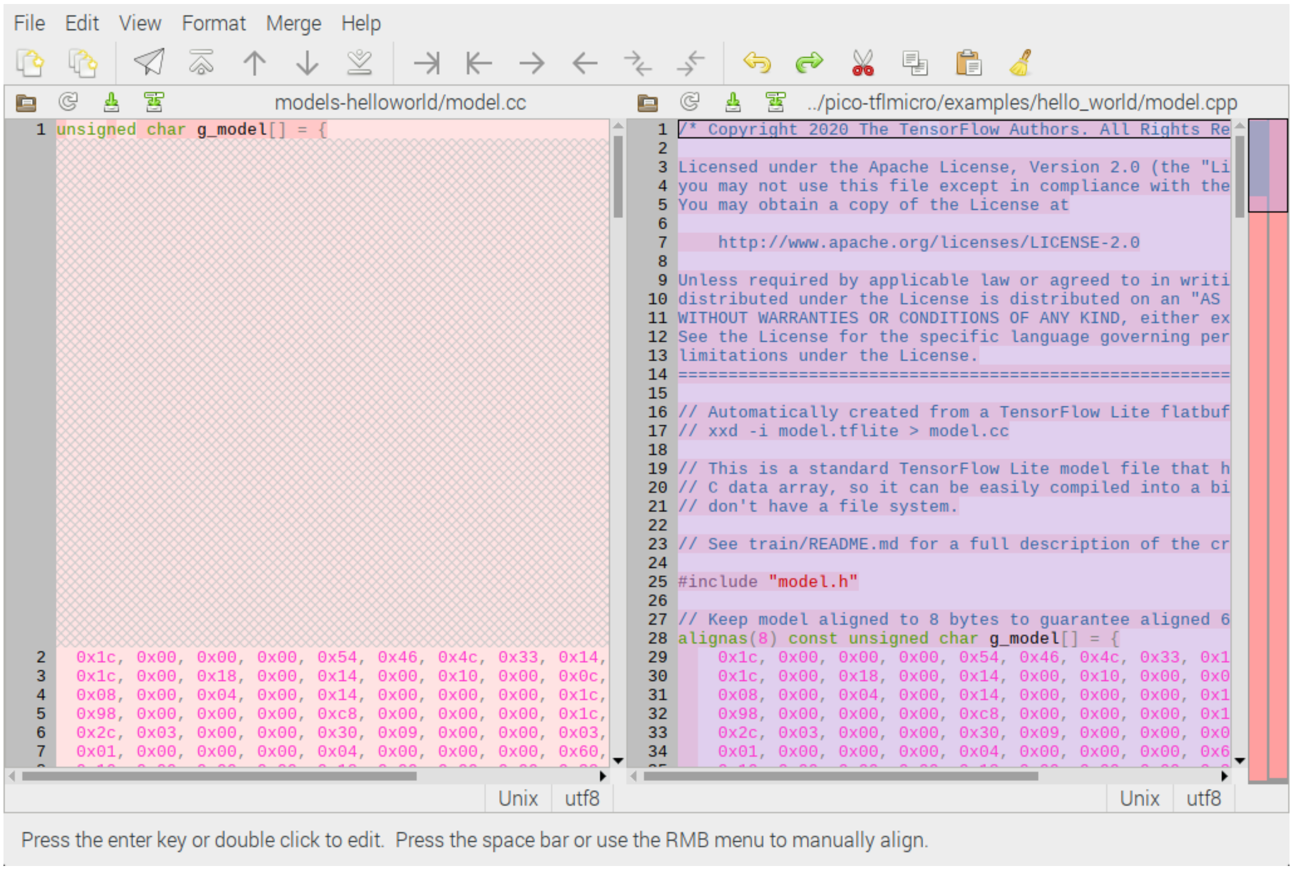
Additional information on process of compiling pico-tflmicro.

https://github.com/develone/my-projects-docs/blob/master/pico/tensorflow.txt

Steps to get the pico executables
hello_world.elf, hello_world_test.elf & output_handler_test.elf

```
git clone git@github.com:develone/pico-tflmicro.git

cd pico-tflmicro

git clone git@github.com:develone/pico-sdk.git

cd pico-sdk/

git submodule update --init

cd ../

mkdir build

cd build

export PICO_SDK_PATH=../pico-sdk/

cmake -DPICO_BOARD=pico ..

make
```

Step 2. To convert a TensorFLow model to a TensorFlow Lite model.



models-helloworld/model.cc

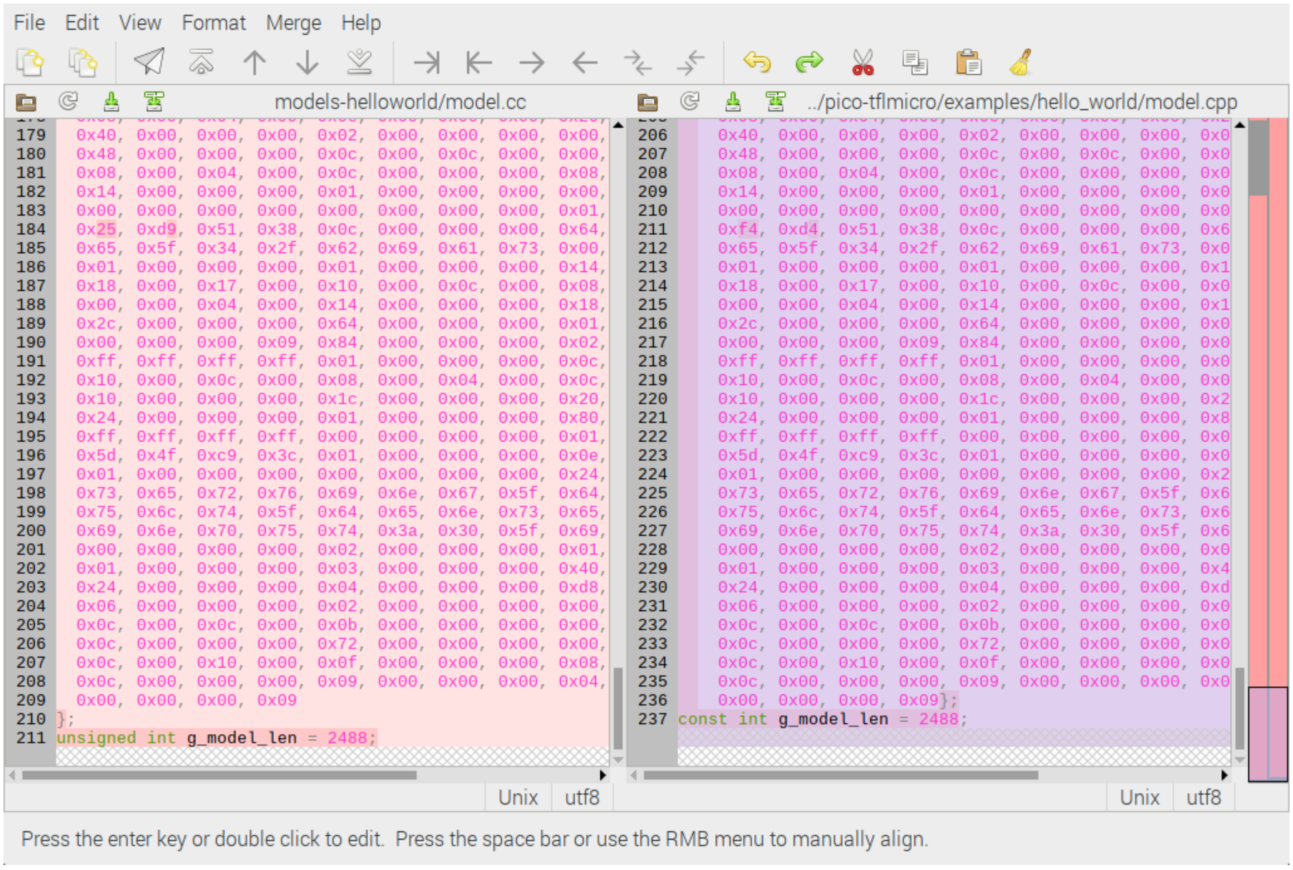
```
unsigned char g_model[] = {
  0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x14,
  0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0c,
  0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1c,
  0x98, 0x00, 0x00, 0x00, 0xc8, 0x00, 0x00, 0x00, 0x1c,
  0x2c, 0x03, 0x00, 0x00, 0x30, 0x09, 0x00, 0x00, 0x03,
  0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x60,
```

../pico-tflmicro/examples/hello_world/model.cpp

```
/* Copyright 2020 The TensorFlow Authors. All Rights Re

Licensed under the Apache License, Version 2.0 (the "Li
you may not use this file except in compliance with the
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writi
distributed under the License is distributed on an "AS 
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either ex
See the License for the specific language governing per
limitations under the License.
=======================================================

// Automatically created from a TensorFlow Lite flatbuf
// xxd -i model.tflite > model.cc

// This is a standard TensorFlow Lite model file that h
// C data array, so it can be easily compiled into a bi
// don't have a file system.

// See train/README.md for a full description of the cr

#include "model.h"

// Keep model aligned to 8 bytes to guarantee aligned 6
alignas(8) const unsigned char g_model[] = {
    0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x1
    0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0
    0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1
    0x98, 0x00, 0x00, 0x00, 0xc8, 0x00, 0x00, 0x00, 0x1
    0x2c, 0x03, 0x00, 0x00, 0x30, 0x09, 0x00, 0x00, 0x0
    0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x6
```

Press the enter key or double click to edit. Press the space bar or use the RMB menu to manually align.

Last of difference



models-helloworld/model.cc

```
  0x40, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00,
  0x48, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x0c, 0x00, 0x00,
  0x08, 0x00, 0x04, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x08,
  0x14, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
  0x25, 0xd9, 0x51, 0x38, 0x0c, 0x00, 0x00, 0x00, 0x64,
  0x65, 0x5f, 0x34, 0x2f, 0x62, 0x69, 0x61, 0x73, 0x00,
  0x01, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x14,
  0x18, 0x00, 0x17, 0x00, 0x10, 0x00, 0x0c, 0x00, 0x08,
  0x00, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18,
  0x2c, 0x00, 0x00, 0x00, 0x64, 0x00, 0x00, 0x00, 0x01,
  0x00, 0x00, 0x00, 0x09, 0x84, 0x00, 0x00, 0x00, 0x02,
  0xff, 0xff, 0xff, 0xff, 0x01, 0x00, 0x00, 0x00, 0x0c,
  0x10, 0x00, 0x0c, 0x00, 0x08, 0x00, 0x04, 0x00, 0x0c,
  0x10, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00, 0x20,
  0x24, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x80,
  0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x01,
  0x5d, 0x4f, 0xc9, 0x3c, 0x01, 0x00, 0x00, 0x00, 0x0e,
  0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24,
  0x73, 0x65, 0x72, 0x76, 0x69, 0x6e, 0x67, 0x5f, 0x64,
  0x75, 0x6c, 0x74, 0x5f, 0x64, 0x65, 0x6e, 0x73, 0x65,
  0x69, 0x6e, 0x70, 0x75, 0x74, 0x3a, 0x30, 0x5f, 0x69,
  0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x01,
  0x01, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x40,
  0x24, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0xd8,
  0x06, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00,
  0x0c, 0x00, 0x0c, 0x00, 0x0b, 0x00, 0x00, 0x00, 0x00,
  0x0c, 0x00, 0x00, 0x00, 0x72, 0x00, 0x00, 0x00, 0x00,
  0x0c, 0x00, 0x10, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x08,
  0x0c, 0x00, 0x00, 0x00, 0x09, 0x00, 0x00, 0x00, 0x04,
  0x00, 0x00, 0x00, 0x09
};
unsigned int g_model_len = 2488;
```

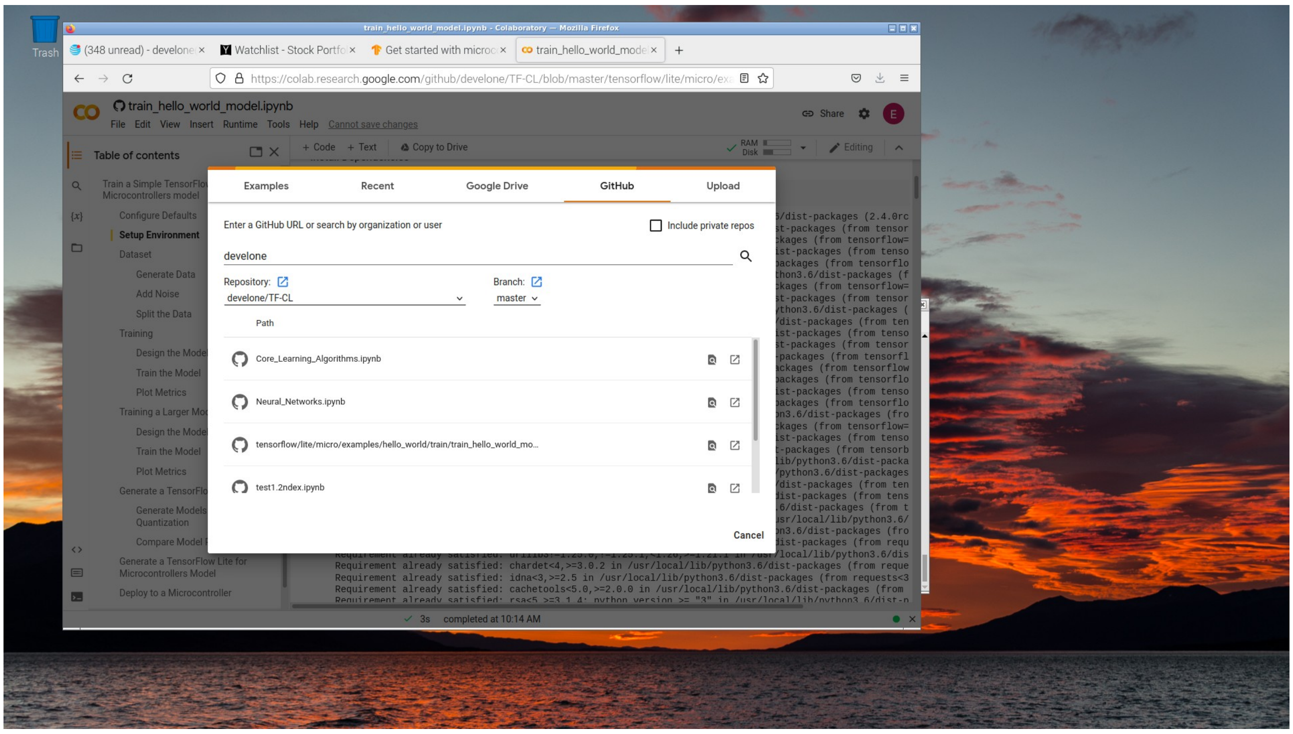
../pico-tflmicro/examples/hello_world/model.cpp

```
    0x40, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x0
    0x48, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x0c, 0x00, 0x0
    0x08, 0x00, 0x04, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x0
    0x14, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x0
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0
    0xf4, 0xd4, 0x51, 0x38, 0x0c, 0x00, 0x00, 0x00, 0x6
    0x65, 0x5f, 0x34, 0x2f, 0x62, 0x69, 0x61, 0x73, 0x0
    0x01, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x1
    0x18, 0x00, 0x17, 0x00, 0x10, 0x00, 0x0c, 0x00, 0x0
    0x00, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1
    0x2c, 0x00, 0x00, 0x00, 0x64, 0x00, 0x00, 0x00, 0x0
    0x00, 0x00, 0x00, 0x09, 0x84, 0x00, 0x00, 0x00, 0x0
    0xff, 0xff, 0xff, 0xff, 0x01, 0x00, 0x00, 0x00, 0x0
    0x10, 0x00, 0x0c, 0x00, 0x08, 0x00, 0x04, 0x00, 0x0
    0x10, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00, 0x2
    0x24, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x8
    0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x0
    0x5d, 0x4f, 0xc9, 0x3c, 0x01, 0x00, 0x00, 0x00, 0x0
    0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x2
    0x73, 0x65, 0x72, 0x76, 0x69, 0x6e, 0x67, 0x5f, 0x6
    0x75, 0x6c, 0x74, 0x5f, 0x64, 0x65, 0x6e, 0x73, 0x6
    0x69, 0x6e, 0x70, 0x75, 0x74, 0x3a, 0x30, 0x5f, 0x6
    0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x0
    0x01, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x4
    0x24, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0xd
    0x06, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x0
    0x0c, 0x00, 0x0c, 0x00, 0x0b, 0x00, 0x00, 0x00, 0x0
    0x0c, 0x00, 0x00, 0x00, 0x72, 0x00, 0x00, 0x00, 0x0
    0x0c, 0x00, 0x10, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x0
    0x0c, 0x00, 0x00, 0x00, 0x09, 0x00, 0x00, 0x00, 0x0
    0x00, 0x00, 0x00, 0x09};
const int g_model_len = 2488;
```

Press the enter key or double click to edit. Press the space bar or use the RMB menu to manually align.

The model was saved to my github from

https://www.tensorflow.org/lite/microcontrollers/get_started_low_level



Enter a GitHub URL or search by organization or user

develone

Repository: develone/TF-CL

Branch: master

Path

Core_Learning_Algorithms.ipynb

Neural_Networks.ipynb

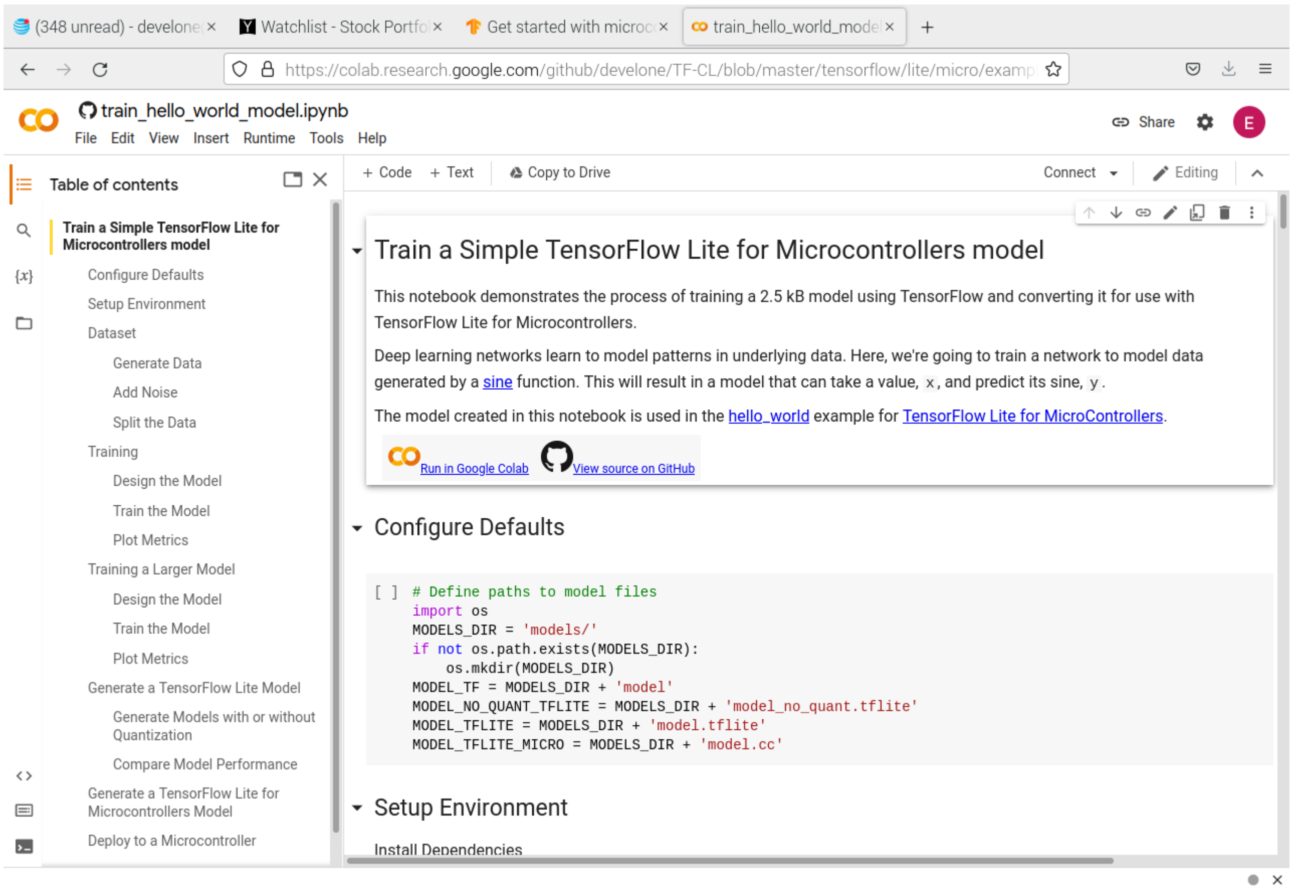
tensorflow/lite/micro/examples/hello_world/train/train_hello_world_mo...

test1.2ndex.ipynb

```
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from reque
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests<3
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.6/dist-packages (from
```

3s completed at 10:14 AM

Loading the TensorFlow hello_world


train_hello_world_model.ipynb
File Edit View Insert Runtime Tools Help
Share
Table of contents
+ Code + Text
Copy to Drive
Connect
Editing

Train a Simple TensorFlow Lite for Microcontrollers model
Configure Defaults
Setup Environment
Dataset
Generate Data
Add Noise
Split the Data
Training
Design the Model
Train the Model
Plot Metrics
Training a Larger Model
Design the Model
Train the Model
Plot Metrics
Generate a TensorFlow Lite Model
Generate Models with or without Quantization
Compare Model Performance
Generate a TensorFlow Lite for Microcontrollers Model
Deploy to a Microcontroller

Train a Simple TensorFlow Lite for Microcontrollers model

This notebook demonstrates the process of training a 2.5 kB model using TensorFlow and converting it for use with TensorFlow Lite for Microcontrollers.

Deep learning networks learn to model patterns in underlying data. Here, we're going to train a network to model data generated by a sine function. This will result in a model that can take a value, x, and predict its sine, y.

The model created in this notebook is used in the hello_world example for TensorFlow Lite for MicroControllers.

Run in Google Colab   View source on GitHub

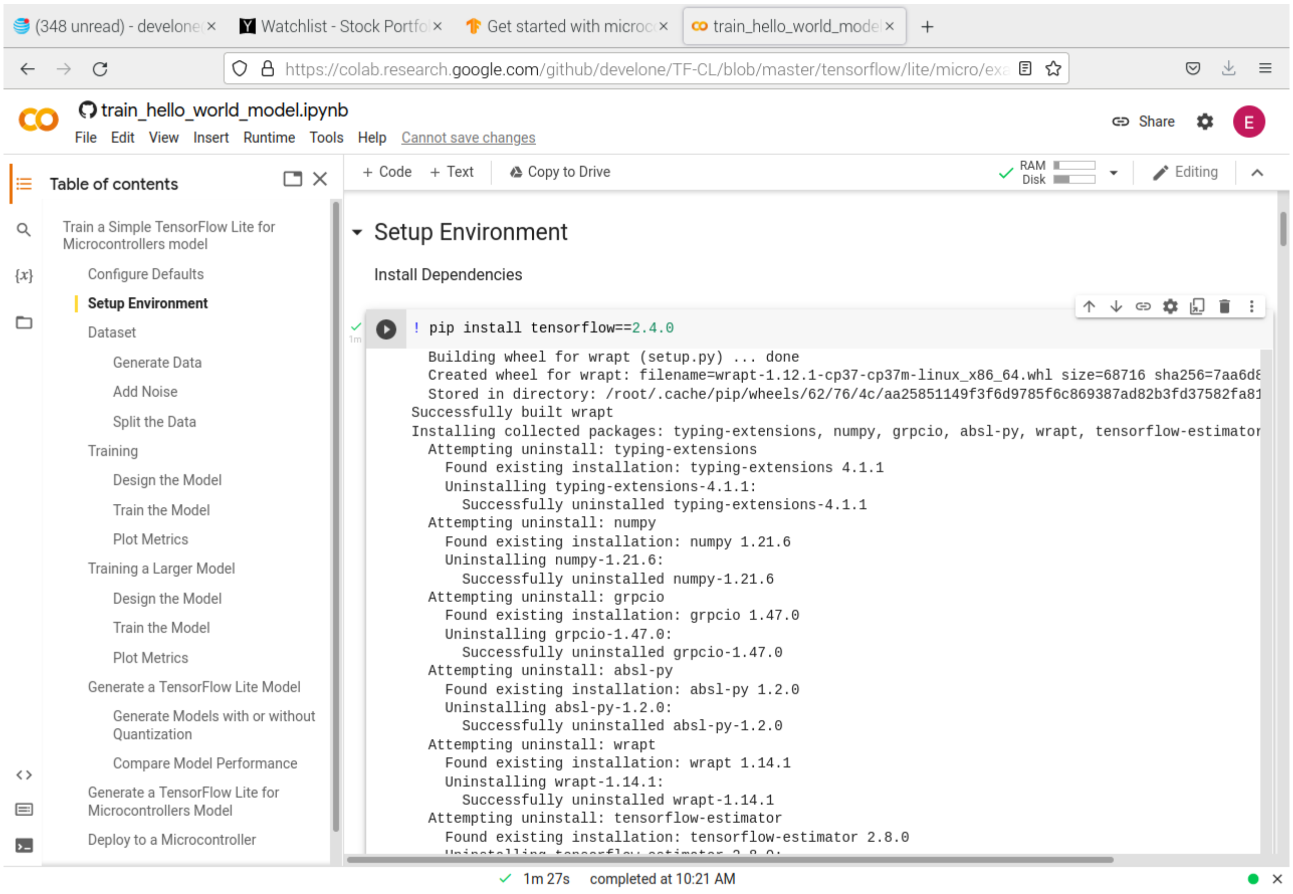
Configure Defaults

```
# Define paths to model files
import os
MODELS_DIR = 'models/'
if not os.path.exists(MODELS_DIR):
    os.mkdir(MODELS_DIR)
MODEL_TF = MODELS_DIR + 'model'
MODEL_NO_QUANT_TFLITE = MODELS_DIR + 'model_no_quant.tflite'
MODEL_TFLITE = MODELS_DIR + 'model.tflite'
MODEL_TFLITE_MICRO = MODELS_DIR + 'model.cc'
```

Setup Environment

Install Dependencies

Setup Environment.


train_hello_world_model.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
Table of contents

Setup Environment

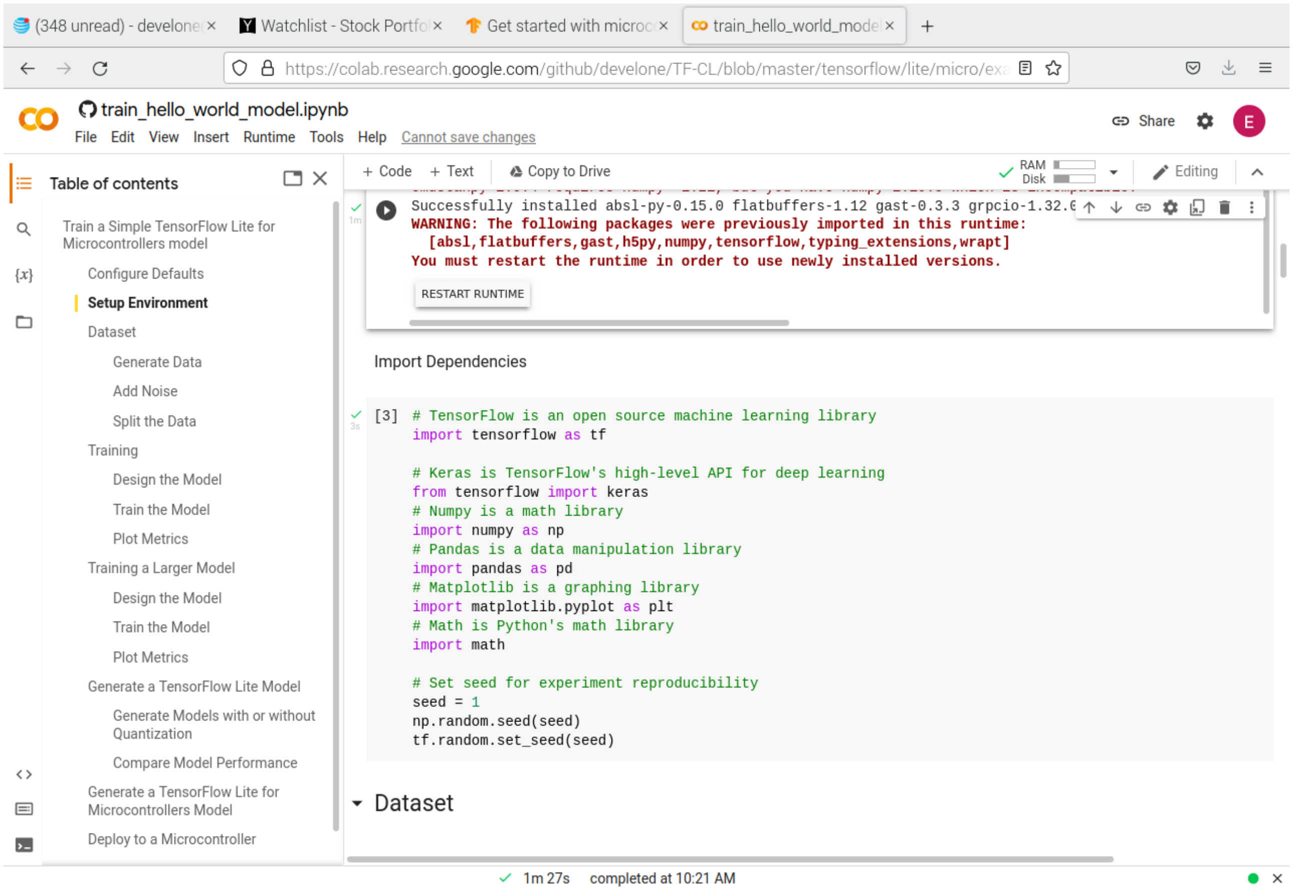
Install Dependencies

```
! pip install tensorflow==2.4.0
```

```
    Building wheel for wrapt (setup.py) ... done
    Created wheel for wrapt: filename=wrapt-1.12.1-cp37-cp37m-linux_x86_64.whl size=68716 sha256=7aa6d8
    Stored in directory: /root/.cache/pip/wheels/62/76/4c/aa25851149f3f6d9785f6c869387ad82b3fd37582fa81
  Successfully built wrapt
  Installing collected packages: typing-extensions, numpy, grpcio, absl-py, wrapt, tensorflow-estimator
    Attempting uninstall: typing-extensions
      Found existing installation: typing-extensions 4.1.1
      Uninstalling typing-extensions-4.1.1:
        Successfully uninstalled typing-extensions-4.1.1
    Attempting uninstall: numpy
      Found existing installation: numpy 1.21.6
      Uninstalling numpy-1.21.6:
        Successfully uninstalled numpy-1.21.6
    Attempting uninstall: grpcio
      Found existing installation: grpcio 1.47.0
      Uninstalling grpcio-1.47.0:
        Successfully uninstalled grpcio-1.47.0
    Attempting uninstall: absl-py
      Found existing installation: absl-py 1.2.0
      Uninstalling absl-py-1.2.0:
        Successfully uninstalled absl-py-1.2.0
    Attempting uninstall: wrapt
      Found existing installation: wrapt 1.14.1
      Uninstalling wrapt-1.14.1:
        Successfully uninstalled wrapt-1.14.1
    Attempting uninstall: tensorflow-estimator
      Found existing installation: tensorflow-estimator 2.8.0
```

✓ 1m 27s completed at 10:21 AM

Import Dependencies


train_hello_world_model.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
Table of contents
RESTART RUNTIME

```
Successfully installed absl-py-0.15.0 flatbuffers-1.12 gast-0.3.3 grpcio-1.32.0
WARNING: The following packages were previously imported in this runtime:
  [absl,flatbuffers,gast,h5py,numpy,tensorflow,typing_extensions,wrapt]
You must restart the runtime in order to use newly installed versions.
```

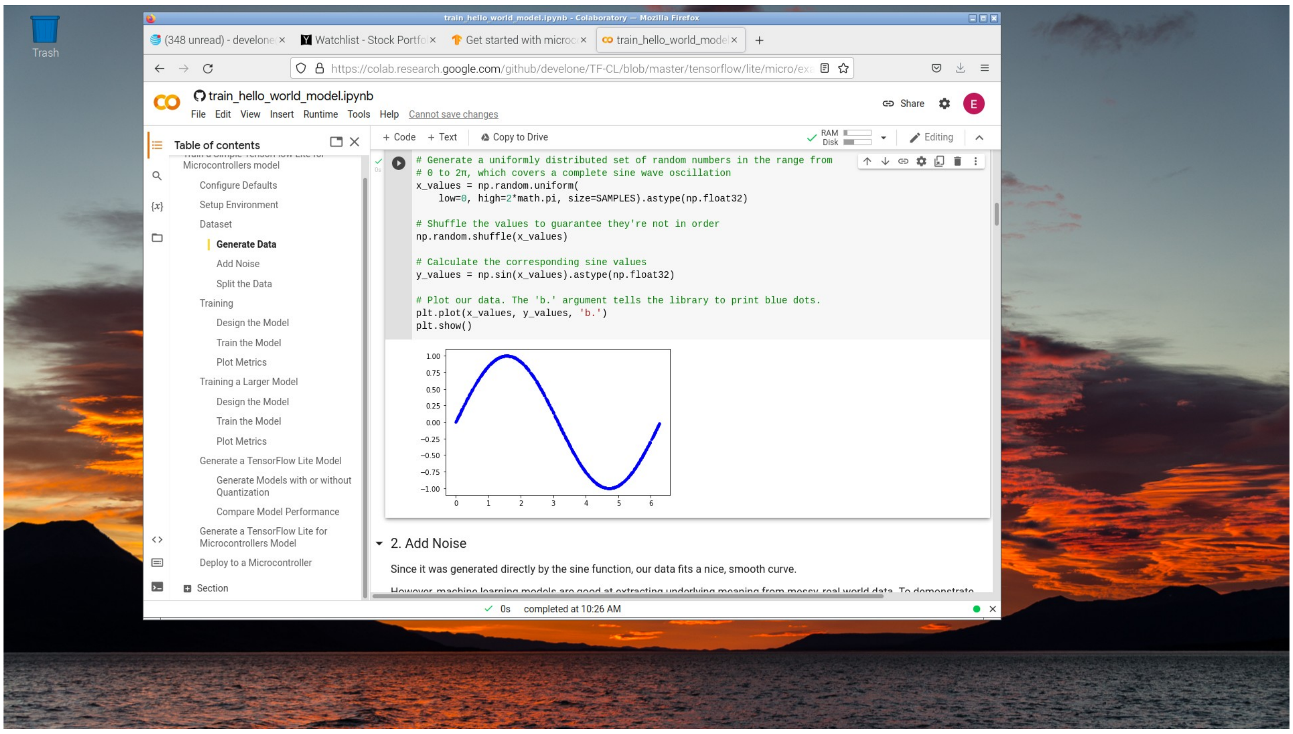
Import Dependencies

```
[3] # TensorFlow is an open source machine learning library
    import tensorflow as tf

    # Keras is TensorFlow's high-level API for deep learning
    from tensorflow import keras
    # Numpy is a math library
    import numpy as np
    # Pandas is a data manipulation library
    import pandas as pd
    # Matplotlib is a graphing library
    import matplotlib.pyplot as plt
    # Math is Python's math library
    import math

    # Set seed for experiment reproducibility
    seed = 1
    np.random.seed(seed)
    tf.random.set_seed(seed)
```

Dataset

✓ 1m 27s completed at 10:21 AM

Generate Data

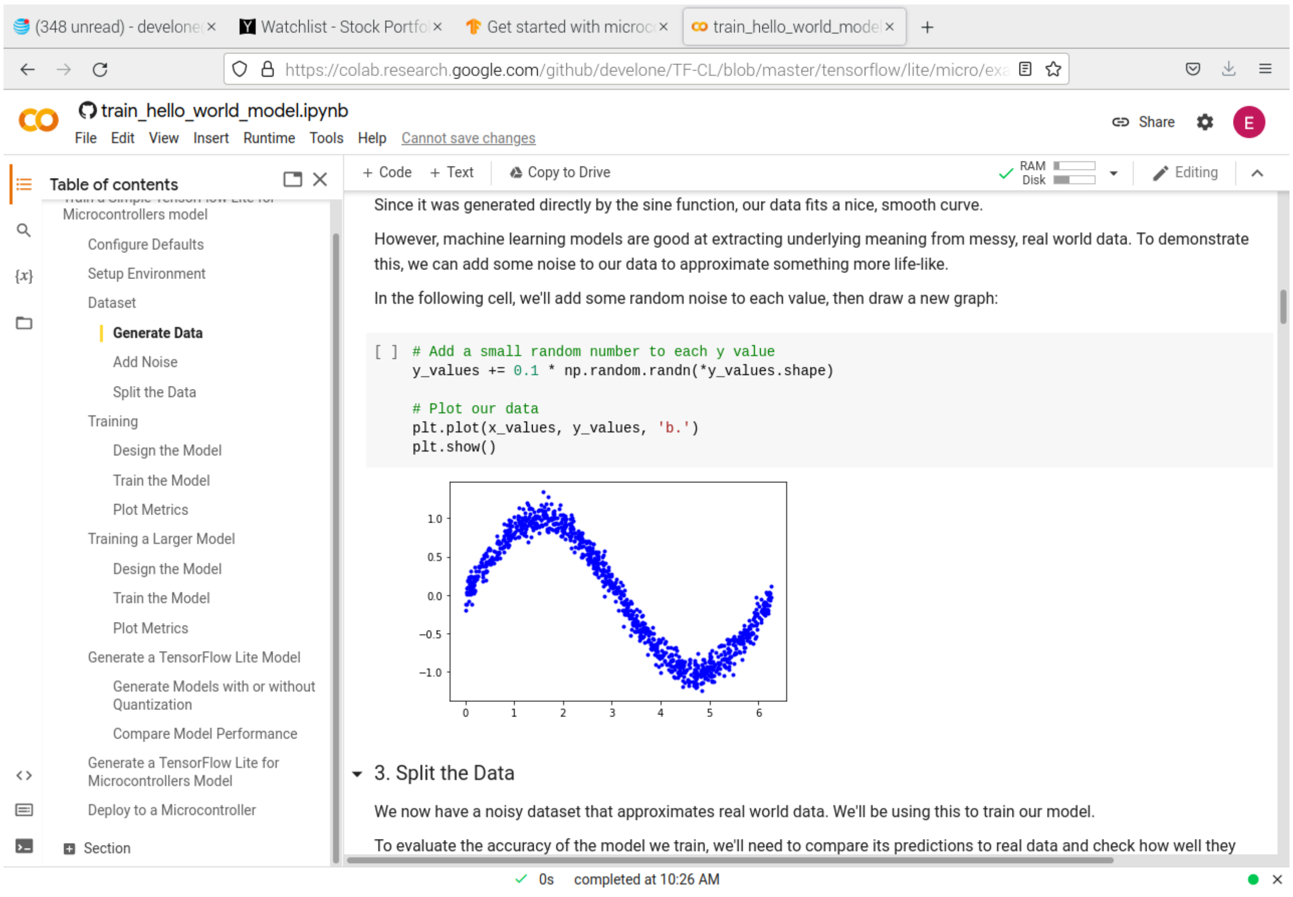

train_hello_world_model.ipynb - Colaboratory — Mozilla Firefox
Table of contents
Configure Defaults
Setup Environment
Dataset
Generate Data
Add Noise
Split the Data
Training
Design the Model
Train the Model
Plot Metrics
Training a Larger Model
Generate a TensorFlow Lite Model
Generate Models with or without Quantization
Compare Model Performance
Generate a TensorFlow Lite for Microcontrollers Model
Deploy to a Microcontroller
Section
completed at 10:26 AM

```
# Generate a uniformly distributed set of random numbers in the range from
# 0 to 2π, which covers a complete sine wave oscillation
x_values = np.random.uniform(
    low=0, high=2*math.pi, size=SAMPLES).astype(np.float32)

# Shuffle the values to guarantee they're not in order
np.random.shuffle(x_values)

# Calculate the corresponding sine values
y_values = np.sin(x_values).astype(np.float32)

# Plot our data. The 'b.' argument tells the library to print blue dots.
plt.plot(x_values, y_values, 'b.')
plt.show()
```

2. Add Noise

Since it was generated directly by the sine function, our data fits a nice, smooth curve.

Add Noise



Since it was generated directly by the sine function, our data fits a nice, smooth curve.

However, machine learning models are good at extracting underlying meaning from messy, real world data. To demonstrate this, we can add some noise to our data to approximate something more life-like.

In the following cell, we'll add some random noise to each value, then draw a new graph:

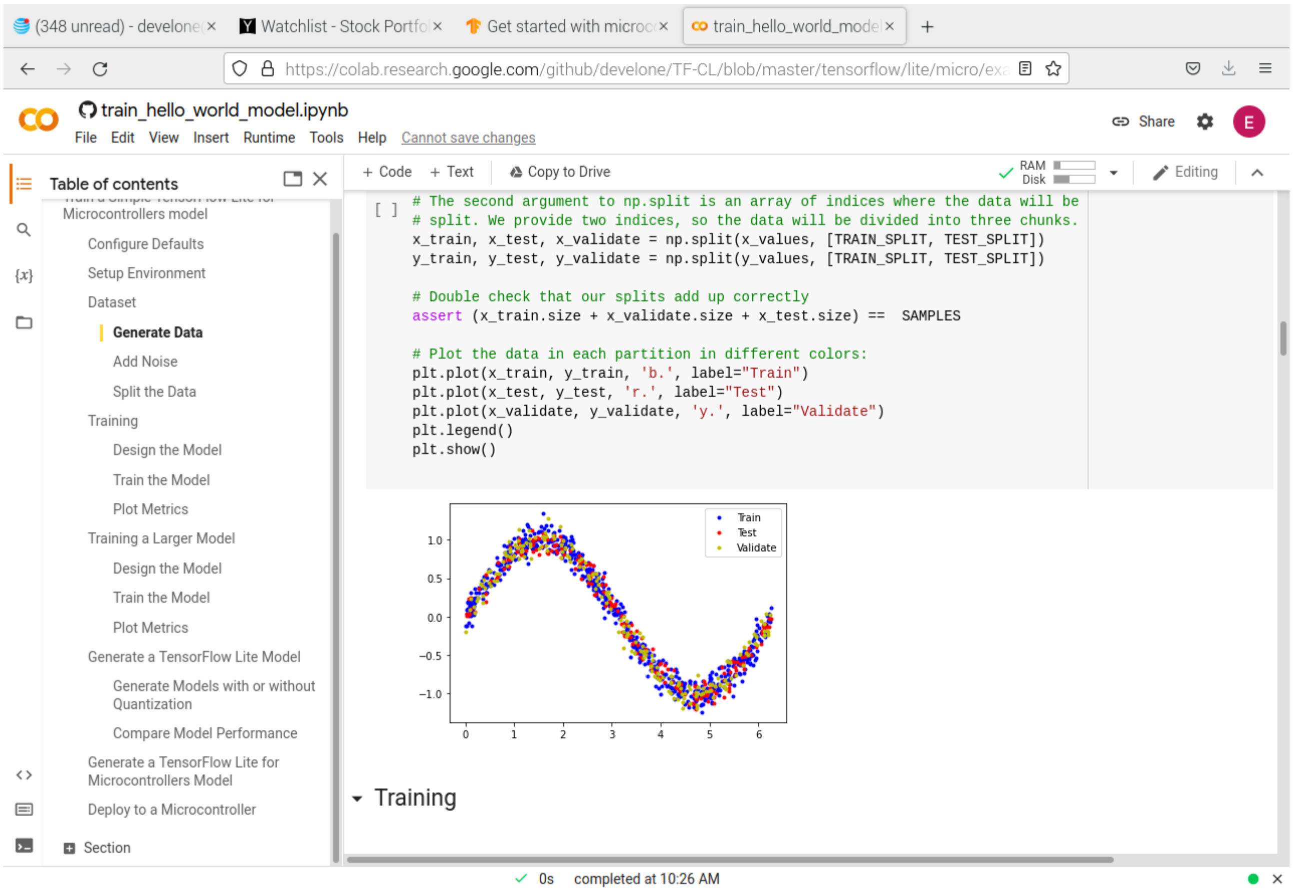
```
# Add a small random number to each y value
y_values += 0.1 * np.random.randn(*y_values.shape)

# Plot our data
plt.plot(x_values, y_values, 'b.')
plt.show()
```

3. Split the Data

We now have a noisy dataset that approximates real world data. We'll be using this to train our model.

To evaluate the accuracy of the model we train, we'll need to compare its predictions to real data and check how well they

Split Data


(348 unread) - develone
Watchlist - Stock Portfo
Get started with microc
train_hello_world_mode
https://colab.research.google.com/github/develone/TF-CL/blob/master/tensorflow/lite/micro/ex
train_hello_world_model.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
Table of contents
+ Code + Text Copy to Drive
RAM Disk
Editing

```
# The second argument to np.split is an array of indices where the data will be
# split. We provide two indices, so the data will be divided into three chunks.
x_train, x_test, x_validate = np.split(x_values, [TRAIN_SPLIT, TEST_SPLIT])
y_train, y_test, y_validate = np.split(y_values, [TRAIN_SPLIT, TEST_SPLIT])

# Double check that our splits add up correctly
assert (x_train.size + x_validate.size + x_test.size) ==  SAMPLES

# Plot the data in each partition in different colors:
plt.plot(x_train, y_train, 'b.', label="Train")
plt.plot(x_test, y_test, 'r.', label="Test")
plt.plot(x_validate, y_validate, 'y.', label="Validate")
plt.legend()
plt.show()
```

Training

0s completed at 10:26 AM

Design the model

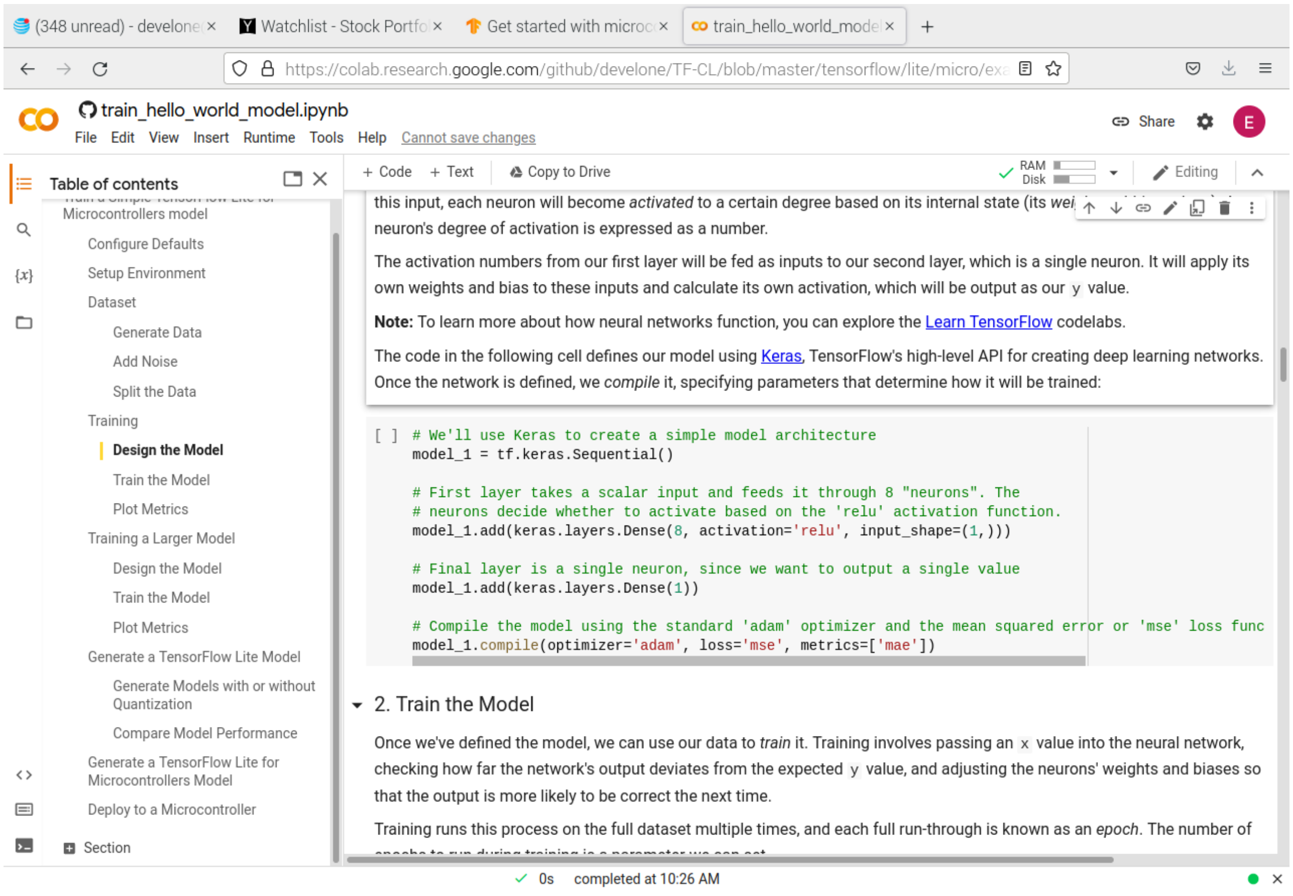
### 1. Design the Model
We're going to build a simple neural network model that will take an input value (in this case, `x`) and use it to predict a numeric output value (the sine of `x`). This type of problem is called a _regression_. It will use _layers_ of _neurons_ to attempt to learn any patterns underlying the training data, so it can make predictions.

To begin with, we'll define two layers. The first layer takes a single input (our `x` value) and runs it through 8 neurons. Based on this input, each neuron will become _activated_ to a certain degree based on its internal state (its _weight_ and _bias_ values). A neuron's degree of activation is expressed as a number.

The activation numbers from our first layer will be fed as inputs to our second layer, which is a single neuron. It will apply its own weights and bias to these inputs and calculate its own activation, which will be output as our `y` value.

Note: To learn more about how neural networks function, you can explore the [Learn TensorFlow](https://codelabs.developers.google.com/codelabs/tensorflow-lab1-helloworld) codelabs.

The code in the following cell defines our model using [Keras](https://www.tensorflow.org/guide/keras), TensorFlow's high-level API for creating deep learning networks. Once the network is defined, we _compile_ it, specifying parameters that determine how it will be trained:


train_hello_world_model.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
Table of contents
+ Code + Text Copy to Drive
RAM Disk
Editing
Microcontrollers model
Configure Defaults
Setup Environment
Dataset
Generate Data
Add Noise
Split the Data
Training
Design the Model
Train the Model
Plot Metrics
Training a Larger Model
Design the Model
Train the Model
Plot Metrics
Generate a TensorFlow Lite Model
Generate Models with or without Quantization
Compare Model Performance
Generate a TensorFlow Lite for Microcontrollers Model
Deploy to a Microcontroller
Section
0s completed at 10:26 AM

this input, each neuron will become *activated* to a certain degree based on its internal state (its wei… neuron's degree of activation is expressed as a number.

The activation numbers from our first layer will be fed as inputs to our second layer, which is a single neuron. It will apply its own weights and bias to these inputs and calculate its own activation, which will be output as our `y` value.

Note: To learn more about how neural networks function, you can explore the Learn TensorFlow codelabs.

The code in the following cell defines our model using Keras, TensorFlow's high-level API for creating deep learning networks. Once the network is defined, we *compile* it, specifying parameters that determine how it will be trained:

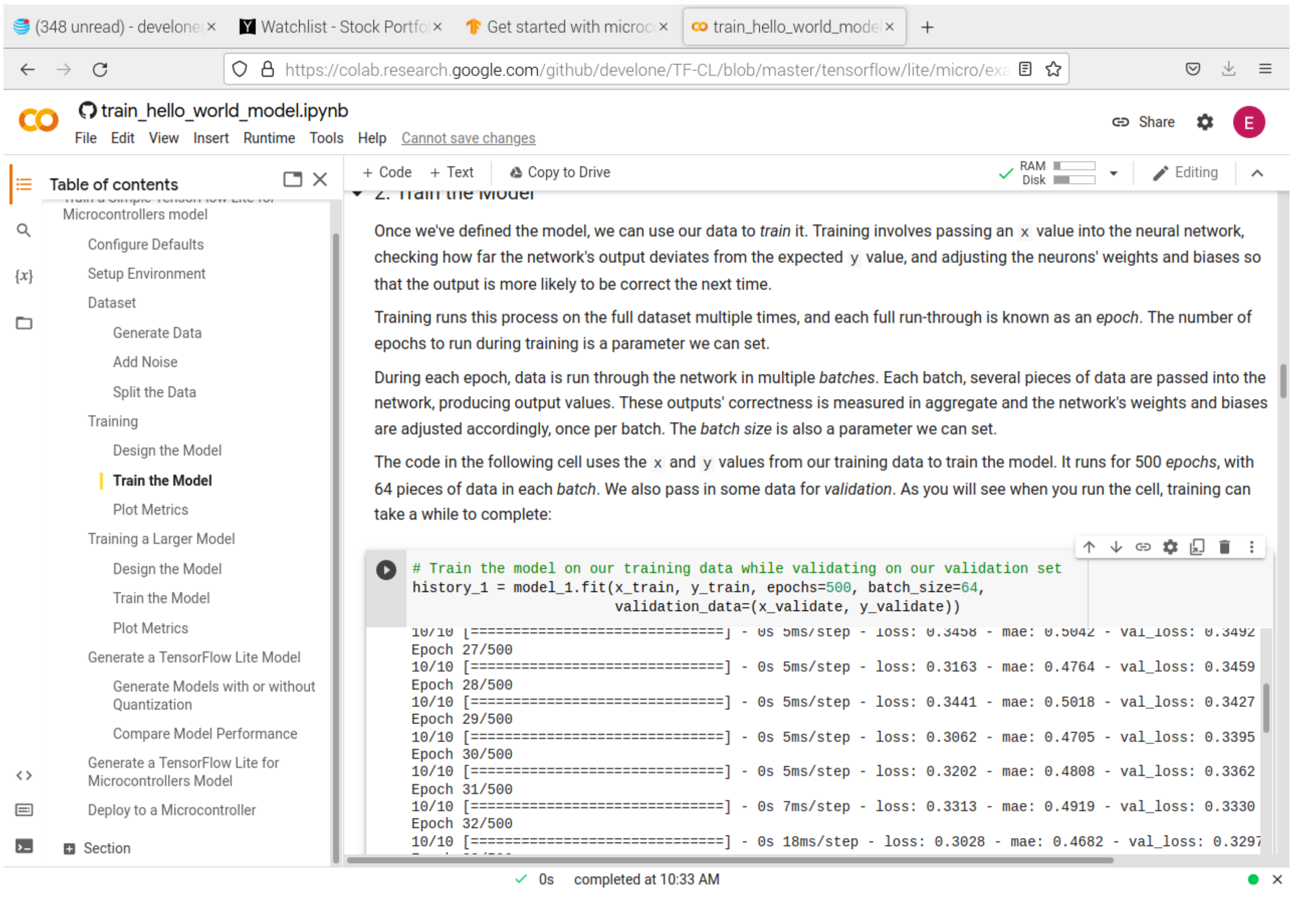
```
[ ]  # We'll use Keras to create a simple model architecture
     model_1 = tf.keras.Sequential()

     # First layer takes a scalar input and feeds it through 8 "neurons". The
     # neurons decide whether to activate based on the 'relu' activation function.
     model_1.add(keras.layers.Dense(8, activation='relu', input_shape=(1,)))

     # Final layer is a single neuron, since we want to output a single value
     model_1.add(keras.layers.Dense(1))

     # Compile the model using the standard 'adam' optimizer and the mean squared error or 'mse' loss func
     model_1.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

2. Train the Model

Once we've defined the model, we can use our data to *train* it. Training involves passing an `x` value into the neural network, checking how far the network's output deviates from the expected `y` value, and adjusting the neurons' weights and biases so that the output is more likely to be correct the next time.

Training runs this process on the full dataset multiple times, and each full run-through is known as an *epoch*. The number of

Train the model


train_hello_world_model.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
Table of contents
+ Code + Text Copy to Drive
RAM Disk
Editing
Microcontrollers model
Configure Defaults
Setup Environment
Dataset
Generate Data
Add Noise
Split the Data
Training
Design the Model
Train the Model
Plot Metrics
Training a Larger Model
Design the Model
Train the Model
Plot Metrics
Generate a TensorFlow Lite Model
Generate Models with or without Quantization
Compare Model Performance
Generate a TensorFlow Lite for Microcontrollers Model
Deploy to a Microcontroller
Section
0s completed at 10:33 AM

Once we've defined the model, we can use our data to *train* it. Training involves passing an `x` value into the neural network, checking how far the network's output deviates from the expected `y` value, and adjusting the neurons' weights and biases so that the output is more likely to be correct the next time.

Training runs this process on the full dataset multiple times, and each full run-through is known as an *epoch*. The number of epochs to run during training is a parameter we can set.

During each epoch, data is run through the network in multiple *batches*. Each batch, several pieces of data are passed into the network, producing output values. These outputs' correctness is measured in aggregate and the network's weights and biases are adjusted accordingly, once per batch. The *batch size* is also a parameter we can set.

The code in the following cell uses the `x` and `y` values from our training data to train the model. It runs for 500 *epochs*, with 64 pieces of data in each *batch*. We also pass in some data for *validation*. As you will see when you run the cell, training can take a while to complete:

```
# Train the model on our training data while validating on our validation set
history_1 = model_1.fit(x_train, y_train, epochs=500, batch_size=64,
                        validation_data=(x_validate, y_validate))
```

```
10/10 [==============================] - 0s 5ms/step - loss: 0.3458 - mae: 0.5042 - val_loss: 0.3492
Epoch 27/500
10/10 [==============================] - 0s 5ms/step - loss: 0.3163 - mae: 0.4764 - val_loss: 0.3459
Epoch 28/500
10/10 [==============================] - 0s 5ms/step - loss: 0.3441 - mae: 0.5018 - val_loss: 0.3427
Epoch 29/500
10/10 [==============================] - 0s 5ms/step - loss: 0.3062 - mae: 0.4705 - val_loss: 0.3395
Epoch 30/500
10/10 [==============================] - 0s 5ms/step - loss: 0.3202 - mae: 0.4808 - val_loss: 0.3362
Epoch 31/500
10/10 [==============================] - 0s 7ms/step - loss: 0.3313 - mae: 0.4919 - val_loss: 0.3330
Epoch 32/500
10/10 [==============================] - 0s 18ms/step - loss: 0.3028 - mae: 0.4682 - val_loss: 0.3297
```