Tiny ML on Arduino

Gesture recognition tutorial

CSCE 5612

Setup Python Environment

The next cell sets up the dependencies in required for the notebook, run it.

```
# Setup environment
!apt-get -qq install xxd
!pip install pandas numpy matplotlib
!pip install tensorflow==2.0.0-rc1
```

```
Requirement already satisfied: pandas in
/usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (1.26.4)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (4.55.8)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2-
>pandas) (1.17.0)
ERROR: Could not find a version that satisfies the requirement
```

```
tensorflow==2.0.0-rc1 (from versions: 2.12.0rc0, 2.12.0rc1, 2.12.0,
2.12.1, 2.13.0rc0, 2.13.0rc1, 2.13.0rc2, 2.13.0, 2.13.1, 2.14.0rc0,
2.14.0rc1, 2.14.0, 2.14.1, 2.15.0rc0, 2.15.0rc1, 2.15.0, 2.15.0.post1,
2.15.1, 2.16.0rc0, 2.16.1, 2.16.2, 2.17.0rc0, 2.17.0rc1, 2.17.0,
2.17.1, 2.18.0rc0, 2.18.0rc1, 2.18.0rc2, 2.18.0)
ERROR: No matching distribution found for tensorflow==2.0.0-rc1

from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

Upload Data

1. Open the panel on the left side of Colab by clicking on the **>**
2. Select the files tab
3. Drag `punch.csv` and `flex.csv` files from your computer to the tab to upload them into colab.

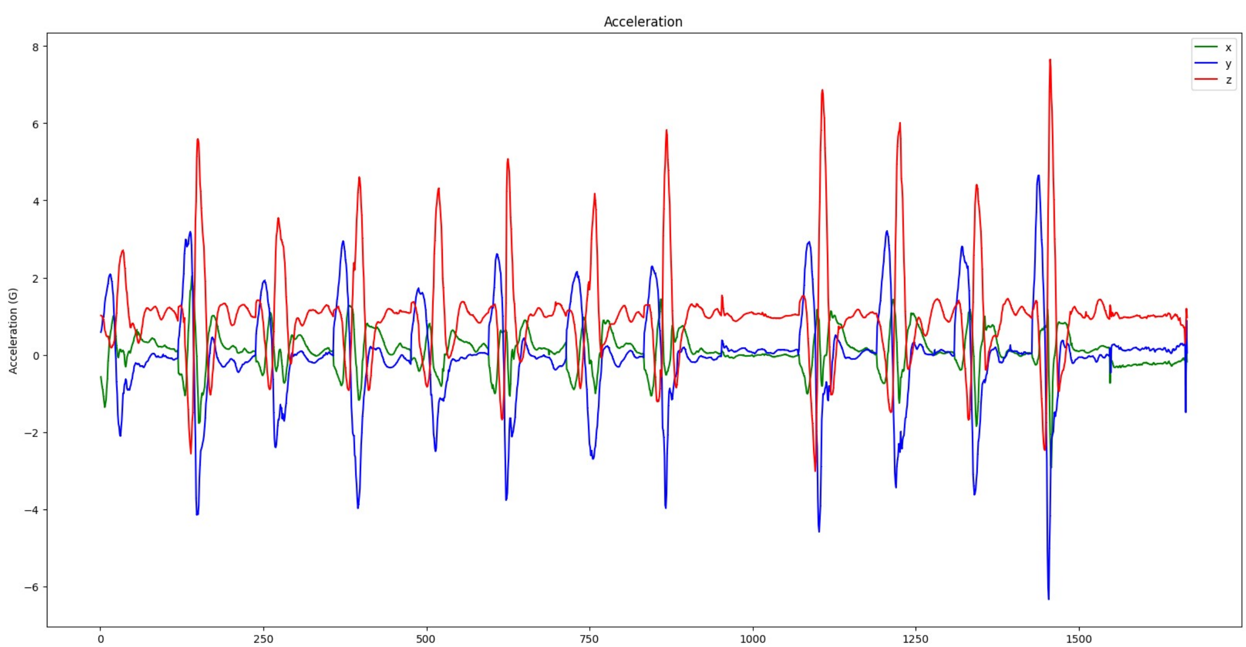
Graph Data (optional)

We'll graph the input files on two separate graphs, acceleration and gyroscope, as each data set has different units and scale.

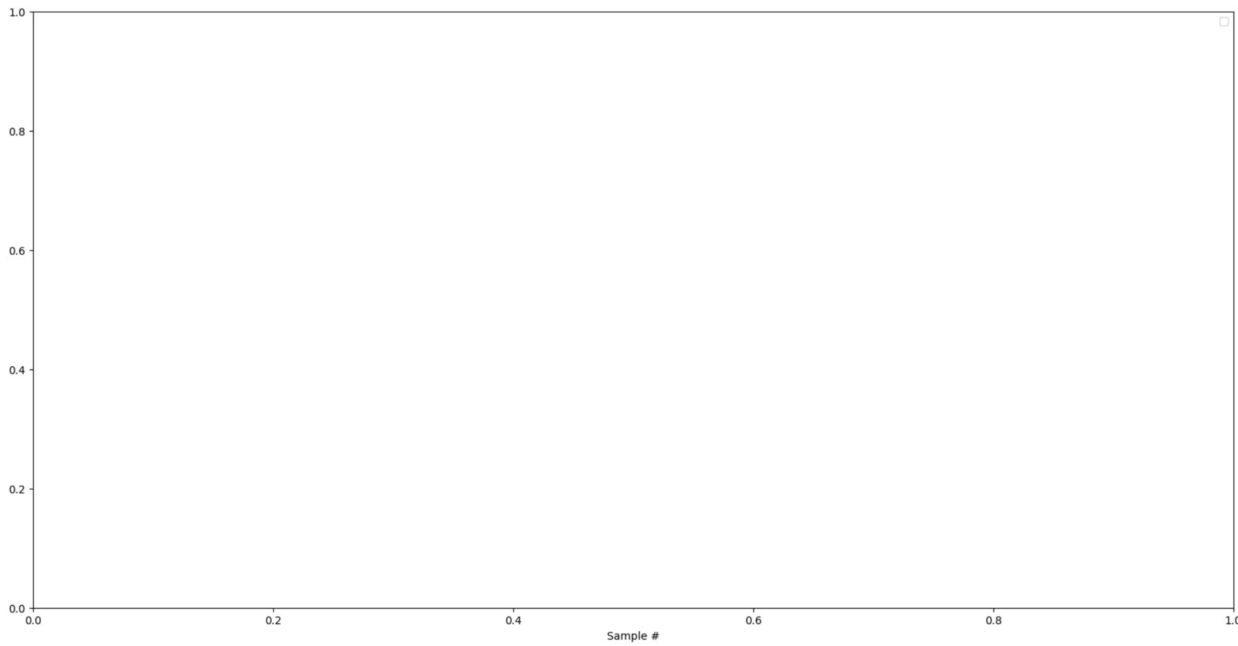
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

filename = "punch.csv"

df = pd.read_csv("/content/" + filename)

index = range(1, len(df['aX']) + 1)

plt.rcParams["figure.figsize"] = (20,10)

plt.plot(index, df['aX'], 'g.', label='x', linestyle='solid',
marker=',')
plt.plot(index, df['aY'], 'b.', label='y', linestyle='solid',
marker=',')
plt.plot(index, df['aZ'], 'r.', label='z', linestyle='solid',
marker=',')
plt.title("Acceleration")
plt.xlabel("Sample #")
plt.ylabel("Acceleration (G)")
plt.legend()
```

```
plt.show()

# # plt.plot(index, df[' Gyr_x'], 'g.', label='x', linestyle='solid', marker=',')
# # plt.plot(index, df[' Gyr_y'], 'b.', label='y', linestyle='solid', marker=',')
# # plt.plot(index, df[' Gyr_z'], 'r.', label='z', linestyle='solid', marker=',')
# plt.title("Gyroscope")
plt.xlabel("Sample #")
# plt.ylabel("Gyroscope (deg/sec)")
plt.legend()
plt.show()
```

```
<ipython-input-6-d3850897d0a9>:13: UserWarning: marker is redundantly defined by the 'marker' keyword argument and the fmt string "g." (-> marker='.'). The keyword argument will take precedence.
  plt.plot(index, df['aX'], 'g.', label='x', linestyle='solid', marker=',')
<ipython-input-6-d3850897d0a9>:14: UserWarning: marker is redundantly defined by the 'marker' keyword argument and the fmt string "b." (-> marker='.'). The keyword argument will take precedence.
  plt.plot(index, df['aY'], 'b.', label='y', linestyle='solid', marker=',')
<ipython-input-6-d3850897d0a9>:15: UserWarning: marker is redundantly defined by the 'marker' keyword argument and the fmt string "r." (-> marker='.'). The keyword argument will take precedence.
  plt.plot(index, df['aZ'], 'r.', label='z', linestyle='solid', marker=',')
```



```
<ipython-input-6-d3850897d0a9>:28: UserWarning: No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
  plt.legend()
```



Train Neural Network

Parse and prepare the data

The next cell parses the csv files and transforms them to a format that will be used to train the fully connected neural network.

Update the `GESTURES` list with the gesture data you've collected in `.csv` format.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

print(f"TensorFlow version = {tf.__version__}\n")

# Set a fixed random seed value, for reproducibility, this will allow us to get
# the same random numbers each time the notebook is run
SEED = 1337
np.random.seed(SEED)
tf.random.set_seed(SEED)
```

```python
# the list of gestures that data is available for
GESTURES = [
    "punch",
    "flex",
]

SAMPLES_PER_GESTURE = 119

NUM_GESTURES = len(GESTURES)

# create a one-hot encoded matrix that is used in the output
ONE_HOT_ENCODED_GESTURES = np.eye(NUM_GESTURES)

inputs = []
outputs = []

# read each csv file and push an input and output
for gesture_index in range(NUM_GESTURES):
  gesture = GESTURES[gesture_index]
  print(f"Processing index {gesture_index} for gesture '{gesture}'.")

  output = ONE_HOT_ENCODED_GESTURES[gesture_index]

  df = pd.read_csv("/content/" + gesture + ".csv")

  # calculate the number of gesture recordings in the file
  num_recordings = int(df.shape[0] / SAMPLES_PER_GESTURE)

  print(f"\tThere are {num_recordings} recordings of the {gesture}
gesture.")

  for i in range(num_recordings):
    tensor = []
    for j in range(SAMPLES_PER_GESTURE):
      index = i * SAMPLES_PER_GESTURE + j
      # normalize the input data, between 0 to 1:
      # - acceleration is between: -4 to +4
      # - gyroscope is between: -2000 to +2000
      tensor += [
          (df['aX'][index] + 4) / 8,
          (df['aY'][index] + 4) / 8,
          (df['aZ'][index] + 4) / 8,
          # (df['gX'][index] + 2000) / 4000,
          # (df['gY'][index] + 2000) / 4000,
          # (df['gZ'][index] + 2000) / 4000
      ]

    inputs.append(tensor)
    outputs.append(output)
```

```
# convert the list to numpy array
inputs = np.array(inputs)
outputs = np.array(outputs)

print("Data set parsing and preparation complete.")
```

```
TensorFlow version = 2.18.0

Processing index 0 for gesture 'punch'.
	There are 14 recordings of the punch gesture.
Processing index 1 for gesture 'flex'.
	There are 25 recordings of the flex gesture.
Data set parsing and preparation complete.
```

Randomize and split the input and output pairs for training

Randomly split input and output pairs into sets of data: 60% for training, 20% for validation, and 20% for testing.

- the training set is used to train the model
- the validation set is used to measure how well the model is performing during training
- the testing set is used to test the model after training

```
# Randomize the order of the inputs, so they can be evenly distributed
for training, testing, and validation
# https://stackoverflow.com/a/37710486/2020087
num_inputs = len(inputs)
randomize = np.arange(num_inputs)
np.random.shuffle(randomize)

# Swap the consecutive indexes (0, 1, 2, etc) with the randomized
indexes
inputs = inputs[randomize]
outputs = outputs[randomize]

# Split the recordings (group of samples) into three sets: training,
testing and validation
TRAIN_SPLIT = int(0.6 * num_inputs)
TEST_SPLIT = int(0.2 * num_inputs + TRAIN_SPLIT)

inputs_train, inputs_test, inputs_validate = np.split(inputs,
[TRAIN_SPLIT, TEST_SPLIT])
outputs_train, outputs_test, outputs_validate = np.split(outputs,
[TRAIN_SPLIT, TEST_SPLIT])

print("Data set randomization and splitting complete.")
```

```
Data set randomization and splitting complete.
```

Build & Train the Model

Build and train a TensorFlow model using the high-level Keras API.

```
# build the model and train it
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(4, activation='relu')) # relu is used for performance
model.add(tf.keras.layers.Dense(1, activation='relu'))
model.add(tf.keras.layers.Dense(NUM_GESTURES, activation='softmax')) # softmax is used, because we only expect one gesture to occur per input
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
history = model.fit(inputs_train, outputs_train, epochs=10, batch_size=1, validation_data=(inputs_validate, outputs_validate))
```

```
Epoch 1/10
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.2503 - mae: 0.5003 - val_loss: 0.2499 - val_mae: 0.4999
Epoch 2/10
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2501 - mae: 0.5001 - val_loss: 0.2494 - val_mae: 0.4994
Epoch 3/10
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.2503 - mae: 0.5003 - val_loss: 0.2490 - val_mae: 0.4990
Epoch 4/10
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2505 - mae: 0.5005 - val_loss: 0.2485 - val_mae: 0.4985
Epoch 5/10
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.2507 - mae: 0.5006 - val_loss: 0.2480 - val_mae: 0.4980
Epoch 6/10
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.2508 - mae: 0.5008 - val_loss: 0.2476 - val_mae: 0.4976
Epoch 7/10
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2510 - mae: 0.5010 - val_loss: 0.2472 - val_mae: 0.4971
Epoch 8/10
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.2512 - mae: 0.5011 - val_loss: 0.2468 - val_mae: 0.4967
Epoch 9/10
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.2514 - mae: 0.5013 - val_loss: 0.2464 - val_mae: 0.4962
Epoch 10/10
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.2516 - mae: 0.5014 - val_loss: 0.2460 - val_mae: 0.4958
```

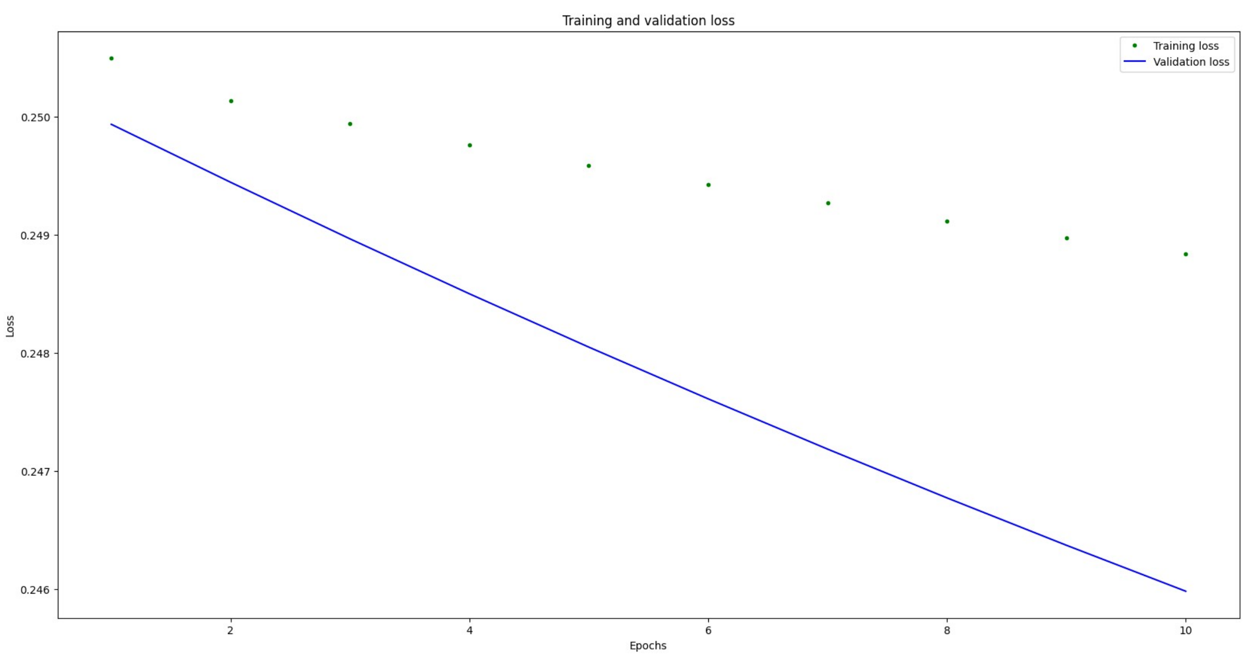
Verify

Graph the models performance vs validation.

Graph the loss

Graph the loss to see when the model stops improving.

```
# increase the size of the graphs. The default size is (6,4).
plt.rcParams["figure.figsize"] = (20,10)

# graph the loss, the model above is configure to use "mean squared
error" as the loss function
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'g.', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

print(plt.rcParams["figure.figsize"])
```

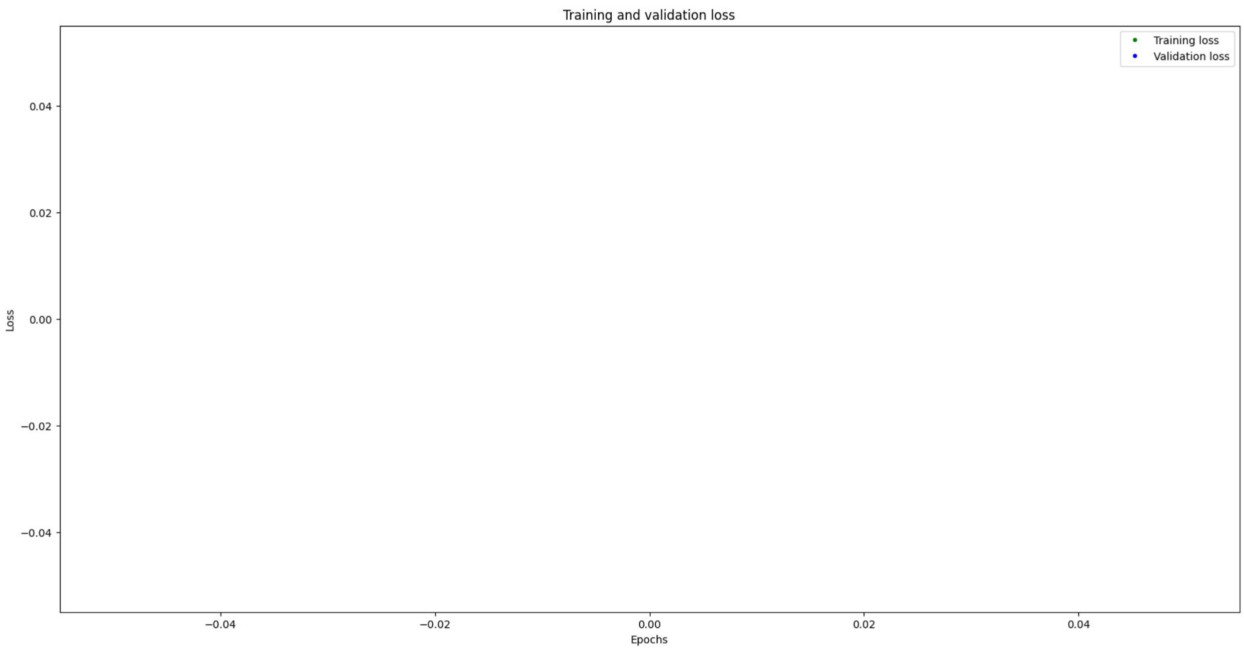



```
[20.0, 10.0]
```

Graph the loss again, skipping a bit of the start

We'll graph the same data as the previous code cell, but start at index 100 so we can further zoom in once the model starts to converge.

```
# graph the loss again skipping a bit of the start
SKIP = 10
plt.plot(epochs[SKIP:], loss[SKIP:], 'g.', label='Training loss')
plt.plot(epochs[SKIP:], val_loss[SKIP:], 'b.', label='Validation
loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

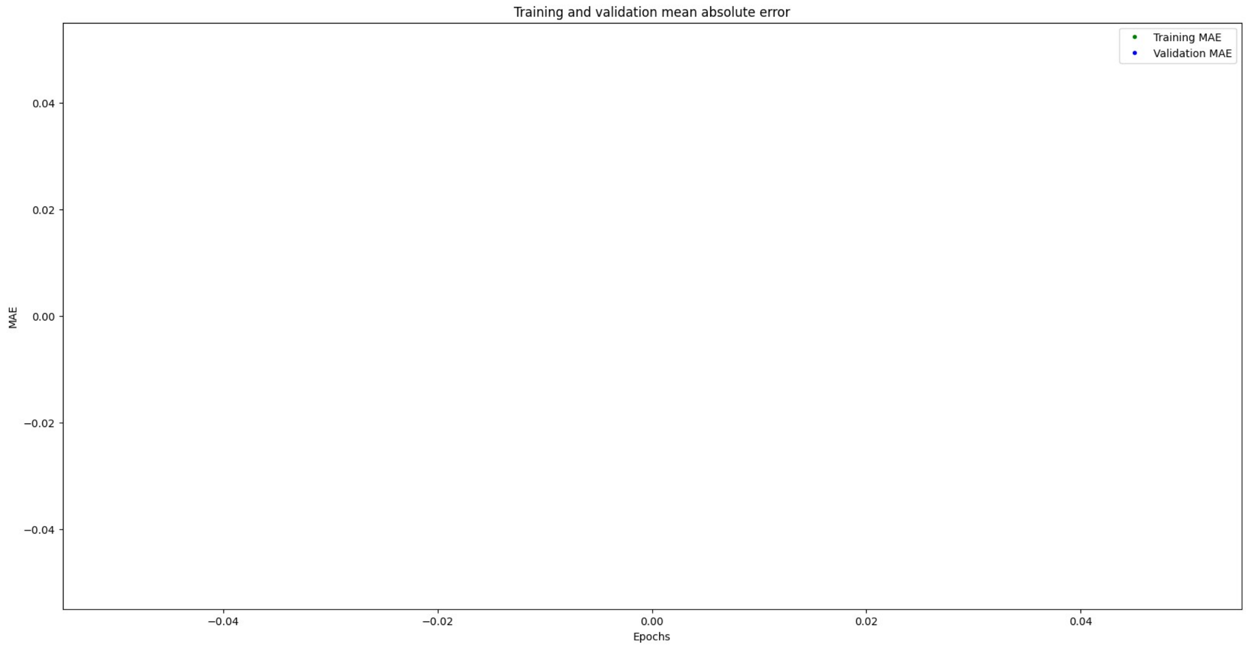


Graph the mean absolute error

Mean absolute error is another metric to judge the performance of the model.

```
# graph of mean absolute error
mae = history.history['mae']
val_mae = history.history['val_mae']
plt.plot(epochs[SKIP:], mae[SKIP:], 'g.', label='Training MAE')
plt.plot(epochs[SKIP:], val_mae[SKIP:], 'b.', label='Validation MAE')
plt.title('Training and validation mean absolute error')
plt.xlabel('Epochs')
plt.ylabel('MAE')
```

```
plt.legend()
plt.show()
```



Run with Test Data

Put our test data into the model and plot the predictions

```
# use the model to predict the test inputs
predictions = model.predict(inputs_test)

# print the predictions and the expected ouputs
print("predictions =\n", np.round(predictions, decimals=3))
print("actual =\n", outputs_test)

# Plot the predictions along with to the test data
# plt.clf()
# plt.title('Training data predicted vs actual values')
# plt.plot(inputs_test, outputs_test, 'b.', label='Actual')
# plt.plot(inputs_test, predictions, 'r.', label='Predicted')
# plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 146ms/step
predictions =
 [[0.487 0.513]
 [0.487 0.513]
 [0.487 0.513]
 [0.487 0.513]
 [0.487 0.513]
 [0.487 0.513]
```

```
 [0.487 0.513]]
actual =
 [[0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]]
```

Convert the Trained Model to Tensor Flow Lite

The next cell converts the model to TFlite format. The size in bytes of the model is also printed out.

```
# Convert the model to the TensorFlow Lite format without quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model to disk
open("gesture_model.tflite", "wb").write(tflite_model)

import os
basic_model_size = os.path.getsize("gesture_model.tflite")
print("Model is %d bytes" % basic_model_size)
```

```
Saved artifact at '/tmp/tmpf50_f1ku'. The following endpoints are
available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(1, 357),
dtype=tf.float32, name='keras_tensor')
Output Type:
  TensorSpec(shape=(1, 2), dtype=tf.float32, name=None)
Captures:
  133827576131472: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133827576132624: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133827576132816: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133827576133584: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133827576133008: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133827576134544: TensorSpec(shape=(), dtype=tf.resource, name=None)
Model is 7472 bytes
```

Encode the Model in an Arduino Header File

The next cell creates a constant byte array that contains the TFlite model. Import it as a tab with the sketch below.

```
!echo "const unsigned char model[] = {" > /content/model.h
!cat gesture_model.tflite | xxd -i      >> /content/model.h
!echo "};"                              >> /content/model.h

import os
model_h_size = os.path.getsize("model.h")
print(f"Header file, model.h, is {model_h_size:,} bytes.")
print("\nOpen the side panel (refresh if needed). Double click model.h
to download the file.")
```

```
Header file, model.h, is 46,112 bytes.

Open the side panel (refresh if needed). Double click model.h to
download the file.
```

Classifying IMU Data

Now it's time to switch back to the tutorial instructions and run our new model on the Arduino Nano 33 BLE or Seeed Xiao nrf52 Sense to classify the accelerometer and gyroscope data.