1 Problem: BLE Connection

1. Specify your recognition model (if not speech recognition) and the categories/classes your model includes.

   This neural network is a speech recognition model trained to distinguish between "yes", "no" and "ok" commands.

2. Explain your data collection process by answering the below questions:

   a. How do you trigger a sample window for collecting data?

   It is based on time, every 0.64 s a new window is triggered

   b. How long is your sample window?

   The sample window is 0.64 s

   c. Mention how many samples you collected for each category.

   For each category, fifty samples were collected

   d. What are the features you used in training your model? Did visualizing the data using the "Serial Plotter" help you selecting your features?

   Using the serial plotter, it was quickly determined that adequate distance should be kept to the arduino to avoid cutting off data. A blink was used to ensure that you are collecting the samples inside the window.

   e. Given your data type, specify any other important consideration your data collecting procedure involved

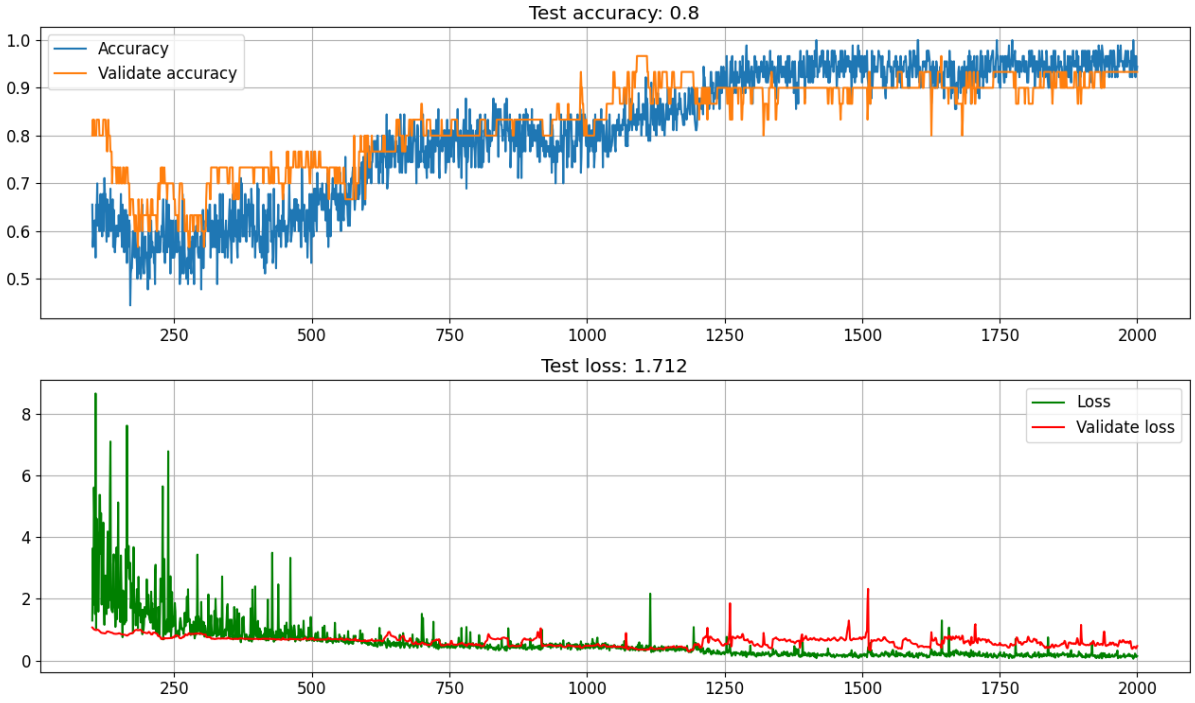
   The data was collected with the same positioning of the arduino and the same distance to the arduino. Baud rate to 115200. The mic records 256 readings throughout the window, the values are then read as 128 PDM data. A threshold is used to determine if something loud enough has been said. The LED light would turn on to start, then the arduino is ready when the onboard LED blinks again.

3. Follow the tinyML tutorial explained in Discussion, build your model (in Neural Network). Split your data into training (60%), validation(20%), and testing(20%) sets.

   a. In less than 4 sentences analyze your model architecture.

   Due to the reading already being converted to PDM signals by the Arduino's mic, the model did not use Fast Fourier Transform. A neural network with three fully-connected layers and two dropout layers was defined. Softmax and sparse categorical crossentropy functions were utilized for classification. As for the final layer, three floating numbers between zero and one are generated representing probabilities for each word.

   b. Report your validation and testing results. Feel free to add plots for showing the loss and the mean absolute error as explained in Discussion.




```
Prediction Accuracy: 0.8
Test accuracy: 0.8
Test loss: 1.712

              precision    recall  f1-score   support

           0       0.73      0.80      0.76        10
           1       0.88      0.78      0.82         9
           2       0.82      0.82      0.82        11

    accuracy                           0.80        30
   macro avg       0.81      0.80      0.80        30
weighted avg       0.80      0.80      0.80        30
```

4. Convert your model to Tensor Flow Lite. Encode the model in an Arduino header file. Now using your Arduino run your trained model and report its accuracy.

    a. You can try each category for 10 times and report the classification accuracy.

$$\begin{aligned} \text{Ok} &= \frac{5}{10} \approx 0.5 \\ \text{No} &= \frac{7}{10} \approx 0.7 \\ \text{Yes} &= \frac{7}{10} \approx 0.7 \end{aligned} \tag{1}$$

    b. Record a short video (less than 20 seconds) showing your trained model in action.

```
1 import os
2 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # only print out fatal log
3
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

import tensorflow as tf
from tensorflow.keras import layers, Sequential


# force computer to use CPU if there are no GPUs present

tf.config.list_physical_devices('GPU')
tf.test.is_gpu_available()


# set random seed

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)


# import dataset and transform target (label data) to categorical arrays

from voice_dataset import data, target


# create training data (60%), validation data (20%) and testing data (20%)

data_train, data_test, target_train, target_test = train_test_split(
    data, target, test_size=0.2, random_state=RANDOM_SEED)
data_train, data_validate, target_train, target_validate = train_test_split(
    data_train, target_train, test_size=0.25, random_state=RANDOM_SEED)


# create a TF model

model = Sequential()
model.add(layers.Dense(data.shape[1], activation='relu', input_shape=(data.shape[1],)))
model.add(layers.Dropout(0.25))
model.add(layers.Dense(np.unique(target).size * 4, activation='relu'))
model.add(layers.Dropout(0.25))
model.add(layers.Dense(np.unique(target).size, activation='softmax'))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()


# training TF model

ITERATION = 2000
BATCH_SIZE = 16

history = model.fit(data_train, target_train, epochs=ITERATION, batch_size=BATCH_SIZE,
                    validation_data=(data_validate, target_validate))
predictions = model.predict(data_test)
test_score = model.evaluate(data_test, target_test)


# get the predicted label based on probability

predictions_categorical = np.argmax(predictions, axis=1)


# display prediction performance on validation data and test data

print('Prediction Accuracy:', accuracy_score(target_test, predictions_categorical).round(3))
print('Test accuracy:', round(test_score[1], 3))
print('Test loss:', round(test_score[0], 3))
print('')
print(classification_report(target_test, predictions_categorical))


# convert TF model to TF Lite model as a C header file (for the classifier)

from tinymlgen import port
```

```
with open('tf_lite_model.h', 'w') as f:  # change path if needed
    f.write(port(model, optimize=False))


# visualize prediction performance

DISPLAY_SKIP = 100

import matplotlib.pyplot as plt


accuracy = history.history['accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
val_accuracy = history.history['val_accuracy']
epochs = np.arange(len(accuracy)) + 1

plt.rcParams['font.size'] = 12
plt.figure(figsize=(14, 8))

plt.subplot(211)
plt.title(f'Test accuracy: {round(test_score[1], 3)}')
plt.plot(epochs[DISPLAY_SKIP:], accuracy[DISPLAY_SKIP:], label='Accuracy')
plt.plot(epochs[DISPLAY_SKIP:], val_accuracy[DISPLAY_SKIP:], label='Validate accuracy')
plt.grid(True)
plt.legend()

plt.subplot(212)
plt.title(f'Test loss: {round(test_score[0], 3)}')
plt.plot(epochs[DISPLAY_SKIP:], loss[DISPLAY_SKIP:], label='Loss', color='green')
plt.plot(epochs[DISPLAY_SKIP:], val_loss[DISPLAY_SKIP:], label='Validate loss', color='red')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```

The following model implementation served as inspiration. https://create.arduino.cc/projecthub/alankrantas/eloquenttinyml-easier-voice-classifier-on-nano-33-ble-sense-ebb81e?fbclid=IwAR0ncvauxwhzd_rjjCsd1pcr68itaEkVegg22upEmMx7MK-zz4

2 Problem

Due to time constraints, this problem was not attempted.