Activity Recognition on Arduino Nano 33 BLE Sense

Faaris Khilji

This document will go over the methods used to do activity recognition on the Arduino Nano 33 BLE Sense.

A major source for completing this task comes from the TinyML workshop which will be linked at the end of this document. Utilizing their code and slightly modifying it, instead of being able to recognize gestures, I am able to achieve recognition of 5 different activities.

The steps to recognize activities include:
Data Capture -> Training Model -> Converting Model -> Live Activity Recognition

Setup

The arduino is fitted inside a small cardboard box to avoid it being loose and pins getting caught when it is put inside the strap.

Data Capture

Data was captured utilizing the IMU present on the Arduino Nano 33 BLE sense. It has a 119 hz sample rate and data was captured in an interval of 2 seconds. For example, to record walking, the user would begin walking and simply press ‘s’ in the serial terminal window to begin the recording of data. It will record 2 seconds worth of data which will end up being 238 samples with 6 features being acceleration and gyroscope in the x,y,z axes. The data is saved into an array and then printed to the serial monitor. This process is repeated until you have a sufficient amount of samples for that activity. For example, I recorded around 50 samples for each activity. Save the data to a csv file.

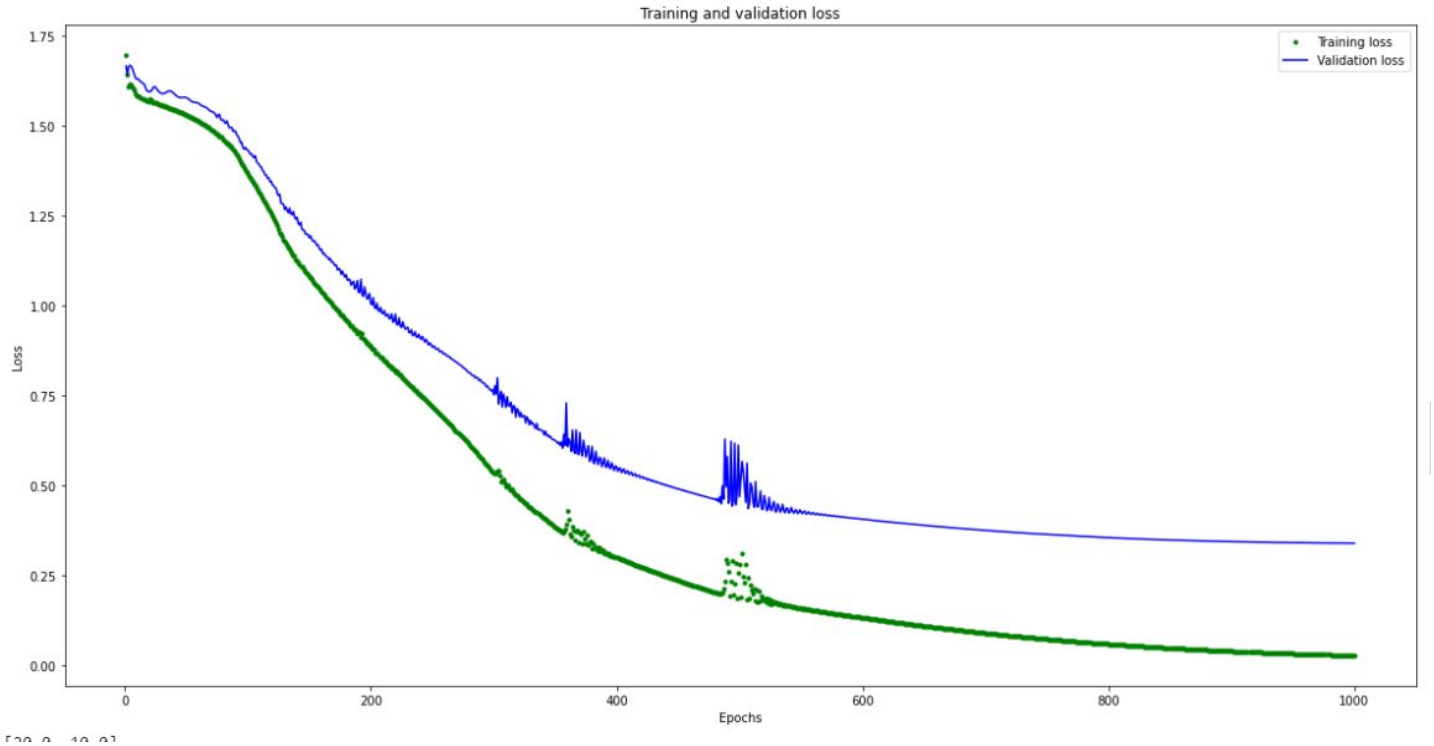
Training

The data is saved to a csv file and is ready to be used with Python to train the model. I utilized the code from TinyML workshop process the data and split the data only changes the SAMPLES_PER_GESTURE variable to 119*2 since we are recording 2 seconds or data instead of 1. Below is how the model was created.

The original model had 3 layers whereas mine contains 4. I found accuracy was increased with this added layer and loss also decreased. I changed the optimizer to ‘adam’ and loss to be ‘categorical_crossentropy’ and metrics to ‘accuracy’. Changing these from the original resulted in no overfitting and loss decreasing versus not decreasing (without the changes).

Another factor that helped loss significantly decrease was increasing the batch size. I found 512 was a good number.

```
# build the model and train it
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(50, activation='relu')) # relu is used for performance
model.add(tf.keras.layers.Dense(15, activation='relu'))
model.add(tf.keras.layers.Dense(10,activation='relu'))
# the final layer is softmax because we only expect one gesture to occur per input
model.add(tf.keras.layers.Dense(NUM_GESTURES, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(inputs_train, outputs_train, epochs=1000, batch_size=512, validation_data=(inputs_validate, outputs_validate))
```



Initially I recorded 30 samples of each activity and found that the model's validation loss would not decrease, but once I recorded 50+ I found the curves began to match each other more. I suspect having more data will result in better results during training.

Converting Model

I utilized the given code to convert the model and these were the sizes of the 2 files:
gesture_model.tflite: 291,480 bytes.
model.h: 1,797,522 bytes.

Activity Recognition

I performed each activity 10 times and these were the results put into a confusion matrix:

```
In [7]: from sklearn.metrics import confusion_matrix
        from sklearn.metrics import classification_report

In [8]: predictions = [0,0,0,0,0,0,0,0,0,0, 1,1,1,1,1,1,1,1,1,1, 2,2,2,2,2,2,2,2,3,2,
                       3,3,2,3,2,3,3,3,3,2, 4,4,4,4,4,4,4,4,4,4]
        actual =      [0,0,0,0,0,0,0,0,0,0, 1,1,1,1,1,1,1,1,1,1, 2,2,2,2,2,2,2,2,2,2,
                       3,3,3,3,3,3,3,3,3,3, 4,4,4,4,4,4,4,4,4,4]

In [9]: print(sklearn.metrics.confusion_matrix(actual,predictions))
        print(sklearn.metrics.classification_report(actual,predictions))

[[10  0  0  0  0]
 [ 0 10  0  0  0]
 [ 0  0  9  1  0]
 [ 0  0  3  7  0]
 [ 0  0  0  0 10]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        10
           2       0.75      0.90      0.82        10
           3       0.88      0.70      0.78        10
           4       1.00      1.00      1.00        10

    accuracy                           0.92        50
   macro avg       0.93      0.92      0.92        50
weighted avg       0.93      0.92      0.92        50
```

I later modified the code to predict activity every 2 seconds instead of waiting for the user to enter ‘s’ to begin capture and prediction.

Sources:
https://github.com/don/tinyml-workshop