**BMI 598: Embedded Machine Learning**

**Mini Project 4 - Report**

**Aravind Siddharth Lakshmi Narasimhan**

**Praharsha Jandhyala**

# System Design

* Our system consists of an Arduino Nano 33 BLE with an embedded Inertial Measurement Unit (IMU) sensor.
* The IMU sensor is responsible for gathering orientation data, representing various postures such as supine, prone, right or left side, sitting, and unknown posture.
* The approach for posture detection was based on the sensor data collected when the board was held in orientations that mimic different postures. The sampling frequency used for the sensor data collection was 300 milliseconds.
* We have designed and evaluated a machine learning algorithm offline using the data collected from the IMU sensor.
* To assess the model's performance, we conducted tests using a specific test dataset constructed for this purpose.
* It's important to note that the data collection did not involve participants wearing the device; rather, the data was collected through simulated postures without user interaction.

# Experiment

* The experiment utilized the C programming language through the Arduino IDE for programming an Arduino board. The main goal was to utilize an IMU (Inertial Measurement Unit) sensor to capture acceleration, gyroscopic, and magnetometer data across the X, Y, and Z spatial axes.
* To collect data from the IMU sensor, the experiment involved incorporating the 'Arduino LSM9DS1 library into the source code. This library offered necessary functions to gather essential data. The collected data was crucial for developing algorithms focused on determining the user's posture concerning the Arduino chip.
* For this experiment, we considered the Arduino as a representation of the user, addressing the difficulty of finding an appropriate object to imitate a person's natural movements.
* We created our own Convolutional Neural Network (CNN) model from the ground up for this project. To prepare the data, we removed unnecessary header information and read the data in a format suitable for our needs. We made sure each type of posture ('supine,' 'prone,' 'unknown,' 'side,' and 'sitting') had a specific number assigned to it.
* We tilted the board to the right or left during supine and prone positions. Also, we moved it in various directions while keeping it upright, aiming to capture data variations along the X-axis of the sensor. This method was used to gather data for unknown positions.

# Algorithm

Gather data from the IMU sensor for various positions being taken into account.

The data was divided in training and test data. The training data was further divided into training and validation sets which was used for the actual training of the model. Train and test a CNN model.

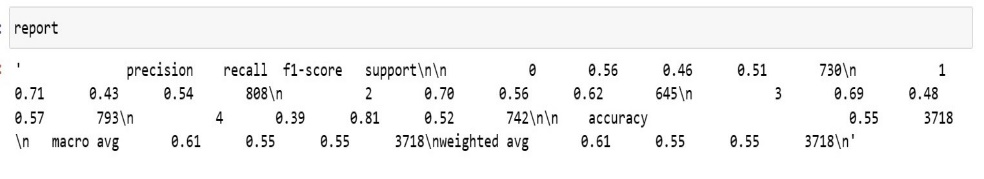
For the real-time deployment of the trained model on the Arduino Nano BLE Sense it was first converted to a Tensorflow Lite Model and then to a raw C++ file containing an array that nested the model.

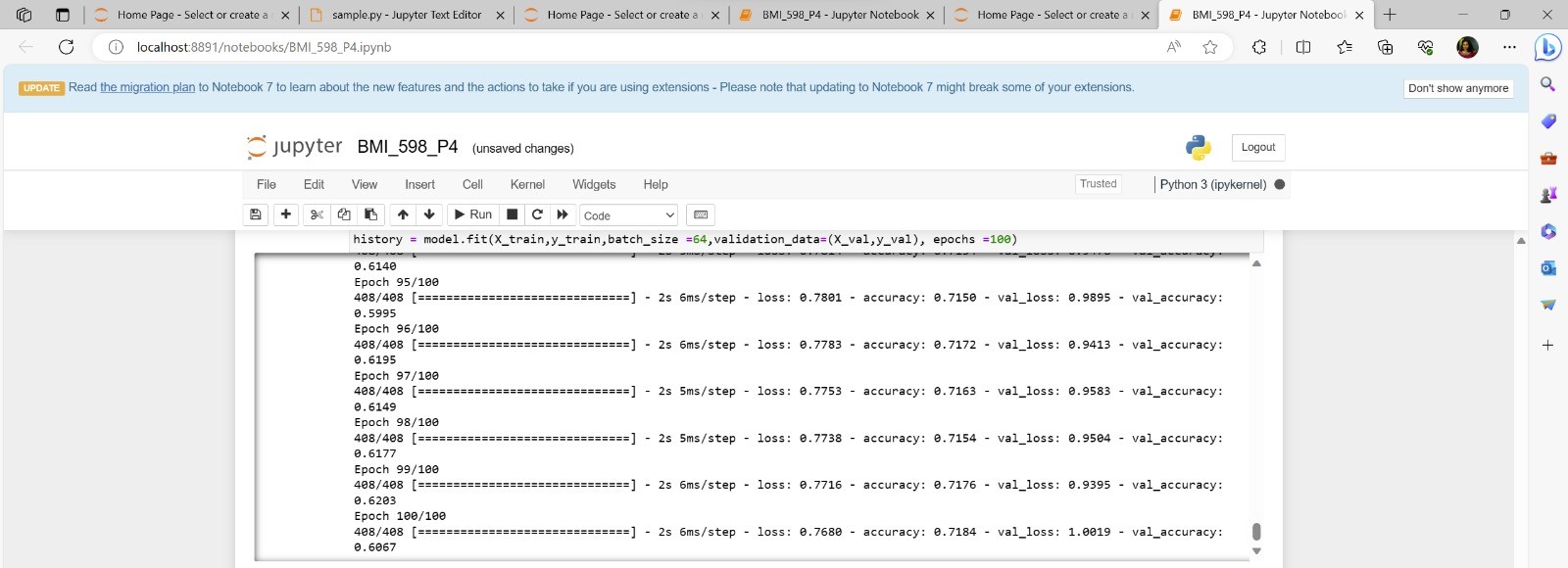
The model was then loaded into the run-time environment using the TensorFlow lite library for Arduino and invoked to make predictions on the sampled sensor data.

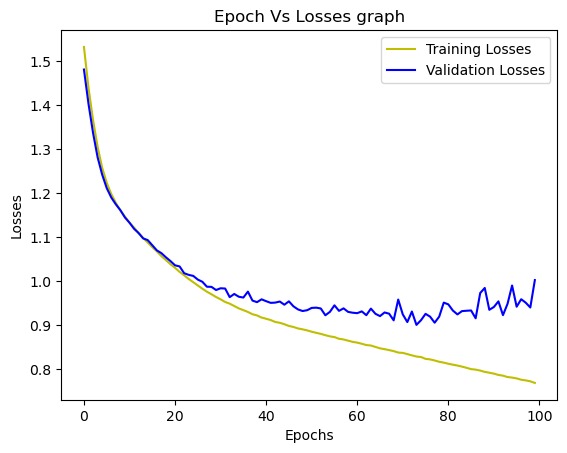
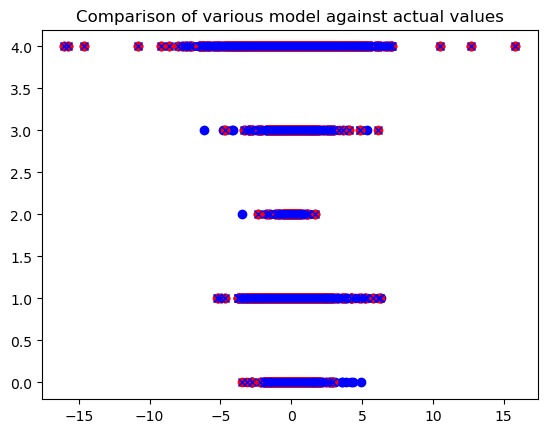
Regarding data partitioning, we divided the dataset into three portions: Training (70%), validation (20%), and testing (10%). The validation sets were utilized to assess the model's performance and calculate the loss, ensuring effective performance on new, unseen data.

# Result

* It is worth observing our accuracy is low. This is indicative that the model was making more large errors than smaller ones, which makes intuitive sense in terms that the model was making large errors when predicting data from the gyroscope or the magnetometer, for which in theory the model should not perform better than a random classifier.
* This was further supported by the results obtained when making predictions in real-time on the device. It was observed that when the sensor input to the model was isolated to the accelerometer the model performed as good as our model in project 3, achieving an accuracy of about 97%, whereas on the other two sensors, it failed on most of its predictions and its behavior appeared to be almost random.





# Discussions

* We effectively trained a basic neural network and evaluated its performance.
* Developing a model, selecting parameters, and tuning it to ensure robustness. Future improvements can involve collecting data with actual sensor wearers and refining the model to handle different orientations.
* Furthermore, delving into advanced architectures and utilizing techniques to augment the data can lead to improved performance.
* The toughest aspect involved creating a model, choosing appropriate parameters and fine-tuning it to ensure resilience across different activation functions.