Seamless Control: Employing Machine Learning to Predict Patient Movements for Assistance During Daily Living Activities

Jarren Bachiller Berdal
School of Engineering, Mechanical,
San Francisco State University,
Elk Grove, USA
Email: jarren.berdal@gmail.com

Ryan J. Hartnett
School of Engineering, Mechanical,
San Francisco State University,
Petaluma, USA
Email: rye.eng.makes@gmail.com

***Abstract*— This paper introduces Seamless Control, a machine learning application designed to classify upper limb movements during activities of daily living for exoskeleton control. The focused s on a specific actvities of daily living is drinking a Water bottle. Seamless Control utilizes surface electromyography sensors and an inertial measurement unit to capture user movement data. The data is then processed and classified by a Convolutional Neural Network (CNN) model.**

The paper details the hardware and software components used for data acquisition and classification. The process of collecting and labeling movement data is documented, along with the development and training of the CNN model. Discussed are the challenges encountered during model selection and implementation, including initial attempts with an Emformer model and subsequent success with a CNN architecture.

The results show significant improvement in movement classification accuracy with the CNN model, achieving up to 100% accuracy after hyperparameter tuning. Finally, the paper demonstrates the integration of the trained model into a live demonstration and outlines future work directions, including latency reduction, model optimization, and integration with a physical exoskeleton for real-world rehabilitation applications.

This project establishes a foundation for further research in real-time movement classification for upper limb exoskeleton control, aiming to improve patient mobility and rehabilitation outcomes.

I. Introduction

Annually, 795,000 people experience stroke in the United States, many of whom experience mobility impairment post-stroke [1]. Recent studies reveal 70% of stroke survivors will develop arm muscle weakness prohibiting them from articulating muscles under their own power [2]. Six months post-stroke, 60% of the survivors lose dexterity in their impaired arm. Stroke rehabilitation treatment is critical for stroke survivors to regain their motor functionality, which may include using robot-assisted devices. Exoskeletons have developed over the past two decades [3], [4]. Another study demonstrates patients are more engaged with robot-assisted treatment versus traditional treatments [5]. More importantly, there is evidence that exoskeleton devices improve a patient's impairment long-term with repetitive treatment [5], [6]. Various approaches in medical engineering research have been devised to create a portable upper limb exoskeleton. A previously developed proposed portable exoskeleton backpack allows patients to perform rehabilitation and assistive activities at home.

Seamless Control was developed to allow user input to actuate with the proposed exoskeleton. Seamless control combines various sensors and a machine-learning model to classify various movements during activities of daily living (ADL). This paper specifically classifies the concise movements during the ADL of drinking a water bottle (WBADL). The following paper will describe the software and hardware used for the project and dive into the process of obtaining, training and testing data of the principal ADL of the paper. Afterward, the paper will depict the creation of the machine learning model and the results of model tuning and training. Finally, the paper will describe the integration of this application deployed in a live demonstration, future works, and a conclusion.

II. Methodology

Seamless Control is a machine learning application that combines surface electromyography (sEMG) sensors and an inertial measurement unit (IMU) to classify various movements during activities of daily living (ADL). Seamless Control has two subsystems: data acquisition (DAS), and data classification (DCS). Each subsystem possesses its unique hardware, software, and integration.

A. Hardware & Software

The DAS comprises an Arduino Mega 2560, three DFRobot sEMG analog sensors, and Sparkfuns's ICM-20948 nine-degree freedom IMU sensor. The software for the DAS is the Arduino IDE and each sensor's Arduino Library. Data collecting of the (WBADL) uses the Coolterm.exe application to record sensor data from the IMU and sEMG from the serial terminal.

The DCS contains a Raspberry Pi 3B+, a keyboard, and a monitor. The Raspberry Pi has Arduino IDE, Thonny, and Pytorch. The Arduino IDE on the Pi allows communication between DAS and the DCS. Thonny allows Python files and the developed Pytorch classification model to be used on the Raspberry Pi. Experiment setup for Seamless Control can be seen in Figure 1.

B. Data Collecting & Movement Categorization

Initially, plans included using the dataset provided by Khan and their team [7], where they performed data acquisition on sEMG sensor readings for various activities.

TABLE I: Data Collecting Documentation and Classification Labeling

Movement Name	Rest	Reach	Drinking	Set Down	Reset
Classification #	0	1	2	3	4
Elbow Rotation	0° (straightened out)	0° → 90° → 0°	0° → 90°	90° → 0°	0° → 90° → 0°
Shoulder Rotation	0° (perpendicular to the floor)	0° → 90°	90°	90°	90° → 0
Checkpoints Pathway	1	1→ 2	2 → 3	3 → 2	2 → 1

The activities recorded were typing, push-up exercises, and lifting a heavy object. However, it was decided to adopt a similar data format to the paper they completed, yet use raw data without any pre-processing to eliminate latency in communication from the Arduino to Raspberry Pi.

The three sEMG analog sensors were placed on the *flexor carpi*, *brachii*, and *deltoid*. Each sensor measured the muscle activity during hand, bicep, and shoulder flexion and extension respectively. The Sparkfun was placed on the deltoid to track raw shoulder acceleration (m/s^2) and gyration (°/s) in the X, Y, and Z planes. Each sensors' reading and timestamps would formulate each movements' features.

WBADL was broken down into five movements with three main checkpoints for ease of simplicity. Checkpoint 1 is defined as the user's arm resting parallel to their body. Checkpoint 2 is defined as the user's hand grasping or releasing the water bottle on the table. Checkpoint 3 is defined as the water bottle near the user's head as if they are drinking water. Each movement would consist of a fluid motion from one checkpoint to another in numeric order. Each movement had ten trials done back to back with a ten-second break in between each trial to prevent muscle fatigue. WBADL movement, its definition, and model classification output label can be seen in Table I.

Each trial lasted 4.5 seconds and produced a 10 x 749 matrix and saved as a CSV file. For model training, the data was separated into training and testing folders. Testing folders consisted of trials 5 and 10 of each movement. The rest were placed in the training folder. Before getting fed into the model, data were collated into a 40 x 10 x 749 tensor then passed to the first layer of the model.

C. Movement Classification Model

In a literature review to support our work we examined multiple papers to understand how sEMG and IMU data were read into machine learning models and adapted for control [8]–[10]. Based on this, an attempt to use the Emformer model [11] for classification was chosen, since it contained a Long-Short-Term-Memory based neural network similar to [9]. This model was also readily available and maintained on PyTorch. Unfortunately, after much effort, the Emformer model yielded poor results. Therefore, a simple convolutional neural network (CNN) model was utilized [12]. We employed the PyTorch API NN modules' layered structure to build a CNN model with fully connected layers and a SoftMax classifier. Figure 8 labels the layers within the CNN model. The code for the implementation of these models is provided in Section 7.

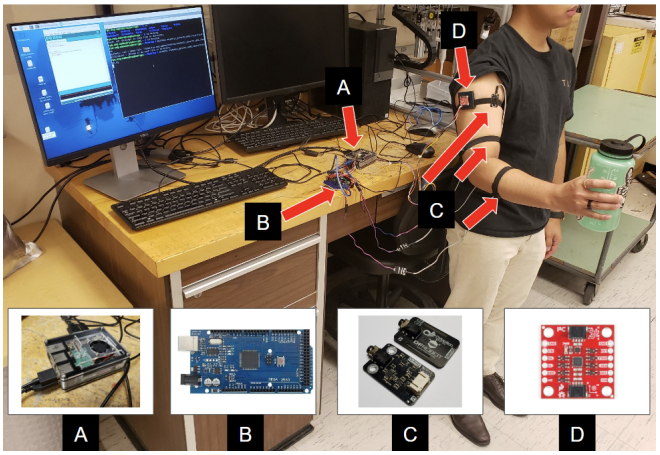

Fig. 1: Seamless Control Experiment Setup: A) Raspberry Pi 3B+ B) Arduino Mega 2560 C) DF Robot sEMG Sensor D) Sparkfun ICM-20948.

III. Model Training, Tuning & Live Demo Results

The Emformer model attempted to implement classification delivered low accuracy for predictions and consistent losses, demonstrating a clear absence of learning based on the epoch v. loss plot and confusion matrix, see Figure 2 & 3. Switching to the Simple CNN model produced reliable results from our training efforts. An overview of tuning hyperparameters to improve CNN implementation is summarized according to Table II. At a minimum, this performed better than the Emformer model's attempts, reducing losses and gaining in accuracy from 20% with Emformer to 30% with the CNN. With the next two training rounds and adjusted hyperparameters, accuracies of 50, 70, and 100% were achieved, which can be seen in Figures 4 to 7 respectively. In Figure 8, the final model's total parameters and file size can be seen.

TABLE II: Data Collecting Documentation and Classification Labeling.

Model Name	Train & Test Batch Size	Learning Rate	Epochs
Emformer	10	0.01	20
CNN	749	0.001	50
CNN	10	0.002	500
CNN	500	0.003	1000

After tuning our model and obtaining acceptable results for motion classification, The model was exported to our desktop using the torch.save() function. This saved a state dictionary file of the model which was then saved and loaded

to the Raspberry Pi. A Python script on the Raspberry Pi with Arduino serial communication, data parsing, logging, and shaping instructions passed data to our model, which was able to infer by loading the state dictionary file. Then the classification results were printed in real time. This can be seen in Figure 9 & 10. All code used to implement this functionality is provided in Section 7.

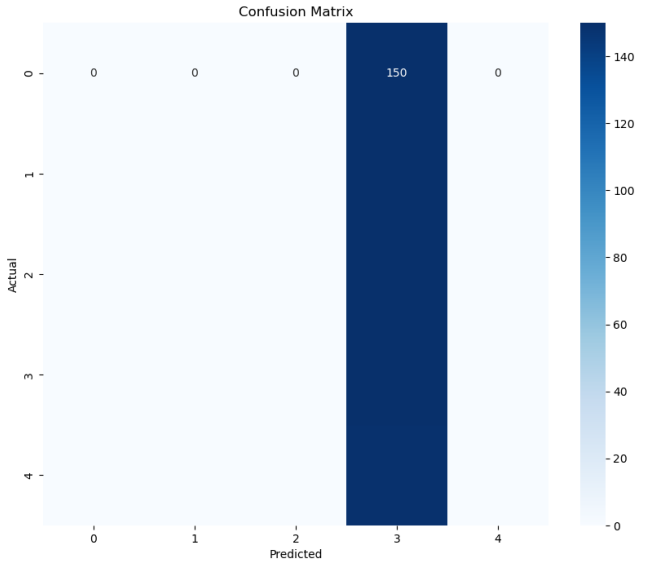

Fig. 2: Emformer Confusion Matrix: 20% accuracy.

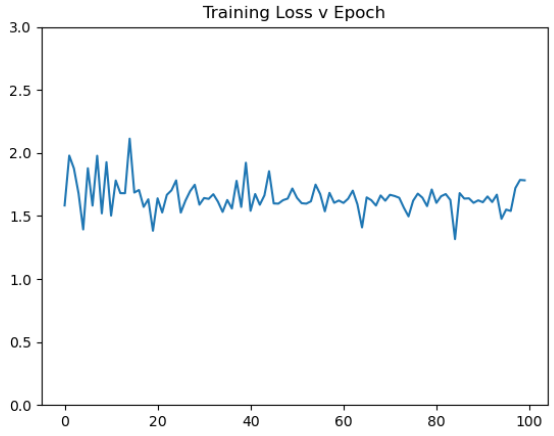

Fig. 3: Emformer Training Loss Vs Epoch Plot: Training loss fluctuated over the course of 100 epoch, training loss did not converge to a steady training loss.

IV. Discussion & Conclusions

Our original goal was to have a user wear IMU & sEMG sensors, perform various movements, then have a servo-powered robot arm track the user's movements. Unfortunately, we did not get to our end goal, but took significant strides toward achieving this aim. We developed a classifier

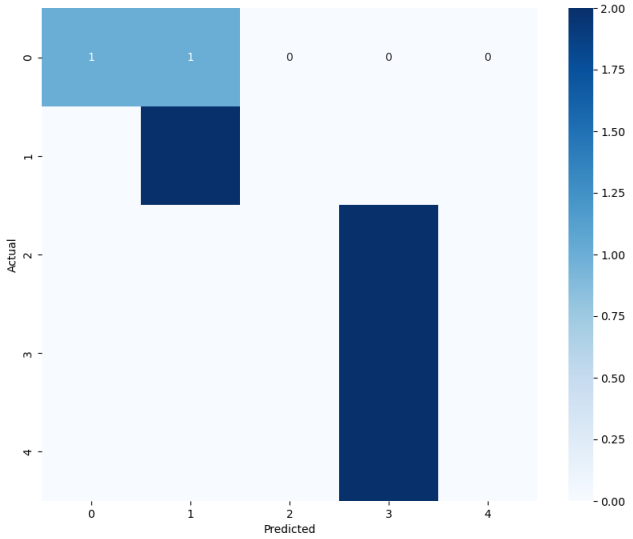

Fig. 4: CNN Test 1 Confusion Matrix: 50% accuracy.

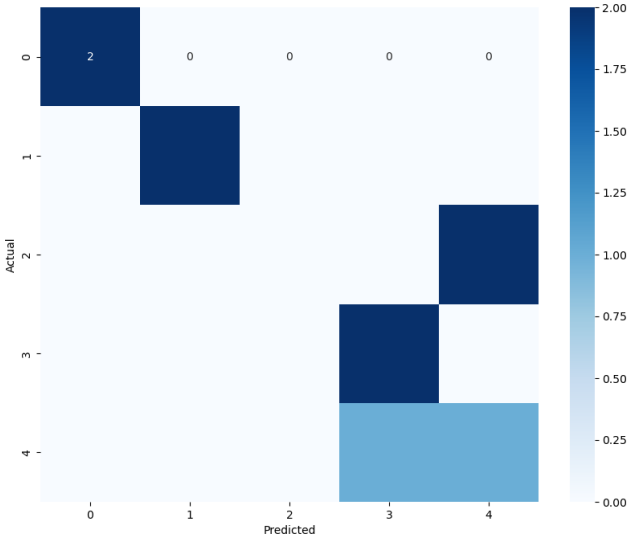

Fig. 5: CNN Test 2 Confusion Matrix: 75% accuracy.

that functioned well enough to track our movements in quasi-real time. We did our best, but the model has room for improvement. Pytorch has useful guides, including one on tuning models for performance [13]. Unfortunately, we were constrained for time so were not able to maximize this opportunity.

There were some latency issues in the tracking of the movements, yet, tracking was achieved. The issues of latency could be improved by quantizing the model so the pre-trained weights and parameters stored by the model could take up less space. A limitation of the Raspberry Pi (Version 3B+) hardware could have been an issue. We get a marked improvement in newer versions of the Raspberry Pi (current V5 series) which would increase the speed of inference since SRAM and core architecture are more efficiently designed.

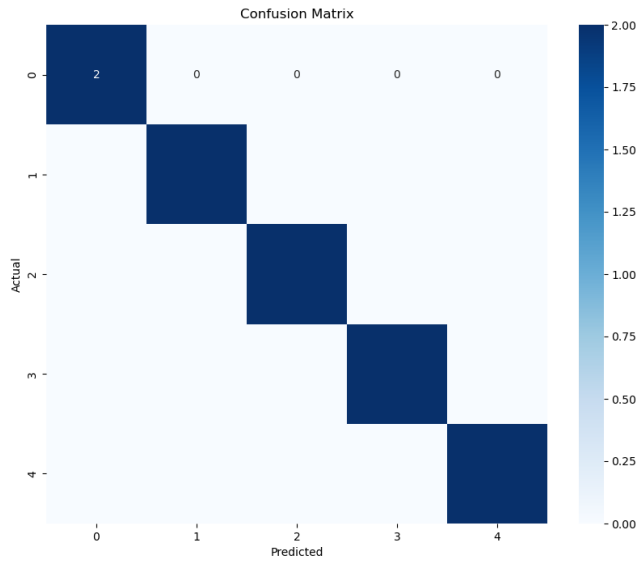

Fig. 6: CNN Test 3 Confusion Matrix: 100% accuracy.

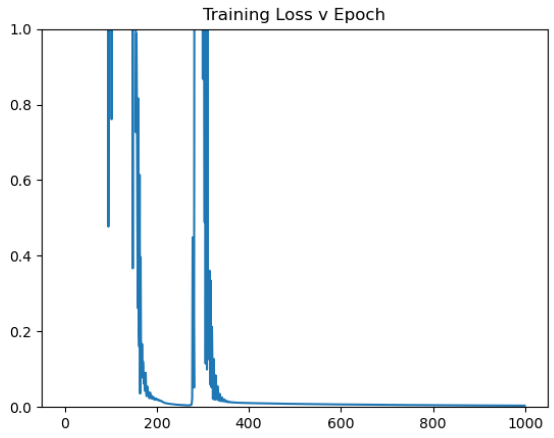

Fig. 7: CNN Training Loss Vs Epoch Plot: Training loss fluctuates from 100 epoch to around 350 epoch, afterwards, training loss converges to around 0.

Our model learned quite well, in terms of accuracy. Perhaps some pruning and retraining would also improve our inference time. To make this project work we developed our own training data set. As we continue to work up this technology, we could counter the loss of motions by improving how the weights are calculated and eventually broaden the model's library of motions it can track. Overall, we are satisfied with how the model performed and excited to see where it can be taken with further research. We hope this provides a pipeline for other students to employ ML in medical devices and support the aim to provide affordable exoskeletons to patients with upper-limb mobility impairments.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
SimpleCNN                                [10, 5]                   --
├─Conv1d: 1-1                            [10, 32, 749]             992
├─MaxPool1d: 1-2                         [10, 32, 374]             --
├─Conv1d: 1-3                            [10, 64, 374]             6,208
├─MaxPool1d: 1-4                         [10, 64, 187]             --
├─Conv1d: 1-5                            [10, 128, 187]            24,704
├─MaxPool1d: 1-6                         [10, 128, 93]             --
├─Linear: 1-7                            [10, 512]                 6,095,360
├─Linear: 1-8                            [10, 5]                   2,565
==========================================================================
Total params: 6,129,829
Trainable params: 6,129,829
Non-trainable params: 0
Total mult-adds (M): 137.82
==========================================================================
Input size (MB): 0.30
Forward/backward pass size (MB): 5.79
Params size (MB): 24.52
Estimated Total Size (MB): 30.61
==========================================================================
```

Fig. 8: Final CNN Model Parameters: 6.1 million trainable parameters and a total size of 30.61 MB.

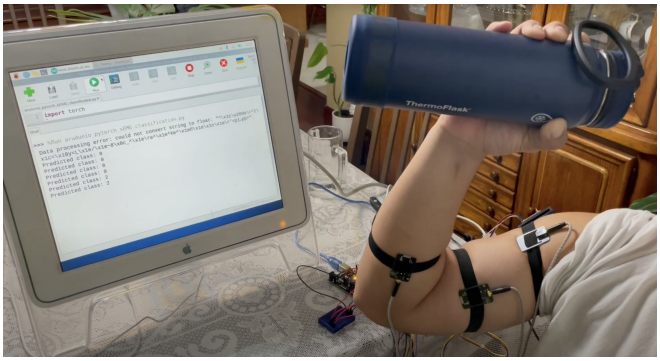

Fig. 9: Seamless Control Demo: Image of real-time classification of WBADL.

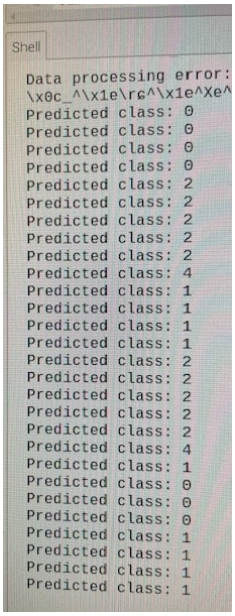

Fig. 10: Classification results from Seamless Control Demo. See Table 1 for classification numbers. Results above show that the model inference was consistent.

V. References

[1] CDC, "Stroke Facts — cdc.gov," Centers for Disease Control and Prevention. Accessed: Feb. 14, 2024. [Online]. Available: https://www.cdc.gov/stroke/facts.htm

[2] R. Allison, L. Shenton, K. Bamforth, C. Kilbride, and D. Richards, "Incidence, Time Course and Predictors of Impairments Relating to Caring for the Profoundly Affected arm After Stroke: A Systematic Review," Physiother. Res. Int., vol. 21, no. 4, pp. 210–227, Dec. 2016, doi: 10.1002/pri.1634.

[3] J. S. Khan, M. Mohammadi, J. Rasmussen, S. Bai, and N. S. Lotte Andreasen Struijk, "A review on the design of assistive cable-driven upper-limb exoskeletons and their experimental evaluation," in 2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Prague, Czech Republic: IEEE, Oct. 2022, pp. 59–64. doi: 10.1109/SMC53654.2022.9945523.

[4] P. Maciejasz, J. Eschweiler, K. Gerlach-Hahn, A. Jansen-Troy, and S. Leonhardt, "A survey on robotic devices for upper limb rehabilitation," J. NeuroEngineering Rehabil., vol. 11, no. 1, p. 3, Dec. 2014, doi: 10.1186/1743-0003-11-3.

[5] N. Rehmat, J. Zuo, W. Meng, Q. Liu, S. Q. Xie, and H. Liang, "Upper limb rehabilitation using robotic exoskeleton systems: a systematic review," Int. J. Intell. Robot. Appl., vol. 2, no. 3, pp. 283–295, Sep. 2018, doi: 10.1007/s41315-018-0064-8.

[6] W. H. Chang and Y.-H. Kim, "Robot-assisted Therapy in Stroke Rehabilitation," J. Stroke, vol. 15, no. 3, p. 174, 2013, doi: 10.5853/jos.2013.15.3.174.

[7] A. M. Khan, S. G. Khawaja, M. U. Akram, and A. S. Khan, "sEMG dataset of routine activities," Data Brief, vol. 33, p. 106543, Dec. 2020, doi: 10.1016/j.dib.2020.106543.

[8] Q. Wu, Z. Wang, and Y. Chen, "sEMG-Based Adaptive Cooperative Multi-Mode Control of a Soft Elbow Exoskeleton Using Neural Network Compensation," IEEE Trans. Neural Syst. Rehabil. Eng., vol. 31, pp. 3384–3396, 2023, doi: 10.1109/TNSRE.2023.3306201.

[9] J.-L. Ren, Y.-H. Chien, E.-Y. Chia, L.-C. Fu, and J.-S. Lai, "Deep Learning based Motion Prediction for Exoskeleton Robot Control in Upper Limb Rehabilitation," in 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada: IEEE, May 2019, pp. 5076–5082. doi: 10.1109/ICRA.2019.8794187.

[10] C.-H. Lin et al., "NTUH-II robot arm with dynamic torque gain adjustment method for frozen shoulder rehabilitation," in 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA: IEEE, Sep. 2014, pp. 3555–3560. doi: 10.1109/IROS.2014.6943059.

[11] Y. Shi et al., "Emformer: Efficient Memory Transformer Based Acoustic Model for Low Latency Streaming Speech Recognition," in ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Jun. 2021, pp. 6783–6787. doi: 10.1109/ICASSP39728.2021.9414560.

[12] "CNN Explainer." Accessed: May 22, 2024. [Online]. Available: https://poloclub.github.io/cnn-explainer/

[13] "Performance Tuning Guide — PyTorch Tutorials 2.3.0+cu121 documentation." Accessed: May 21, 2024. [Online]. Available: https://pytorch.org/tutorials/recipes/recipes/tuning_guide.html

VI. Resources

A. Pytorch Emformer Model:

https://pytorch.org/audio/stable/generated/torchaudio.models.Emformer.html

https://pytorch.org/audio/stable/models.html

B. Pytorch Tutorial series by Daniel Bourke:

https://github.com/mrdbourke/pytorch-deep-learning

https://www.learnpytorch.io/04_pytorch_custom_datasets

https://www.learnpytorch.io/05_pytorch_going_modular

https://www.learnpytorch.io/06_pytorch_transfer_learning

C. Model Deployment:

https://pytorch.org/tutorials/beginner/saving_loading_models.html

https://github.com/MIC-Laboratory/CNN-HD-sEMG-Classifier

1) Model Summary & Shape: https://pypi.org/project/torchinfo

2) Raspberry Pi & Python Deployment: https://pytorch.org/tutorials/intermediate/realtime_rpi.html

3) Raspberry Pi General: Set time from command line:
$sudo date -s '2024-05-25 12:34:56'
Allows for internet functionality

https://raspberrypi.stackexchange.com/questions/76182/set-manually-datetime-raspberry

https://roboticsbackend.com/raspberry-pi-arduino-serial-communication

https://roboticsbackend.com/enable-ssh-on-raspberry-pi-raspbian

https://roboticsbackend.com/when-to-use-arduino-vs-raspberry-pi

https://www.instructables.com/Installing-Using-Arduino-IDE-on-a-Raspberry-Pi-3

4) Raspberry Pi 3B+ Specifications: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus

VII. Code

A. Project Repository

Please see this Github Repository for all the code that produced this project:
https://github.com/rye-tech/engineering-on-device-ML

B. CNN Model Code

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import serial

# Define the model architecture
class SimpleCNN(nn.Module):
    def __init__(self, input_channels, num_classes):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=input_channels, out_channels=32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.pool = nn.MaxPool1d(kernel_size=2, stride=2)

        self.flattened_size = self._calculate_flattened_size(input_channels)
        self.fc1 = nn.Linear(self.flattened_size, 512)
        self.fc2 = nn.Linear(512, num_classes)

    def _calculate_flattened_size(self, input_channels):
        x = torch.zeros(1, input_channels, 749)  # OG 749 Adjust sequence length as needed
        x = self._forward_features(x)
        flattened_size = x.view(-1).shape[0]
        return flattened_size

    def _forward_features(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = self.pool(x)
        return x

    def forward(self, x):
        x = self._forward_features(x)
        x = x.view(x.size(0), -1)  # Flatten the tensor
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Load the model
model_path = '/home/rye-eng-makes/Desktop/model.pth'
input_channels = 10  # Adjust based on your data
num_classes = 5  # Adjust based on your classes
model = SimpleCNN(input_channels, num_classes)
model.load_state_dict(torch.load(model_path))
model.eval()

# Open serial port
ser = serial.Serial('/dev/ttyACM0', 115200)  # Adjust port and baud rate as per your setup

def process_data(data):
    data_lines = data.split('\n')
    data_list = []

    for line in data_lines:
        if line.strip():
            values = list(map(float, line.split(',')))
            data_list.append(values)

    data_array = np.array(data_list).T

    if data_array.shape[0] != input_channels:
        raise ValueError(f"Expected {input_channels} channels, but got {data_array.shape[0]}")

    data_tensor = torch.tensor(data_array, dtype=torch.float32).unsqueeze(0)
    return data_tensor

buffer = ""

while True:
    try:
    # Read data from serial port
        line = ser.readline().decode('utf-8', errors='ignore').strip()

        buffer += line + "\n"

        if buffer.count('\n') >= 749:
            try:

                data_tensor = process_data(buffer)

                with torch.no_grad():
                    output = model(data_tensor)
                    _, predicted = torch.max(output.data, 1)

                print(f'Predicted class: {predicted.item()}')

                buffer = ""

        # Print or handle the prediction

            except ValueError as e:
                print(f"Data processing error: {e}")
                buffer = ""

    except serial.SerialException as e:
        print(f"Serial communication error: {e}")
    except Exception as e:
        print(f"Unexpected error: {e}")
```

Listing 1: In real time classification code and CNN model

Acknowledgment

The authors would like to thank Dr. Zhuwei Qin for organizing the On-Device Machine Learning Course (ENGR 859) at San Francisco State University. We would like to thank Dr. David Quintero for general research guidance. We are grateful to Mr. Daniel Bourke for his open-source machine learning tutorials that were companion to the course.