TECHIN515 Lab 4 Report

Magic Wand – Gesture Recognition with ESP32

Student Name: Cherry Mathew Roy
Date: 05/20/2025
Course: TECHIN515 – Hardware and Software Lab 2
Lab Number: 4
Lab Title: Magic Wand – Real-Time Gesture Classification using an ESP32 and MPU6050

---

1. Introduction

The goal of this lab was to design, train, and deploy a machine learning model for real-time gesture recognition using an ESP32 microcontroller and MPU6050 IMU. The project involved building a wand prototype capable of recognizing multiple gestures, processing sensor data on-device, and providing immediate classification feedback. This lab emphasized embedded ML workflows, including signal collection, model training via Edge Impulse, deployment on constrained hardware, and validation through live testing.

1. Hardware Setup and Connections

Components Used:

- ESP32-C3 Development Board
- MPU6050 Accelerometer/Gyroscope Module
- Breadboard & Jumper Wires
- LED (for gesture feedback)
- Push Button (gesture trigger)
- 3.7V 500mAh LiPo Battery
- 3D-Printed Wand-Shaped Enclosure

Wiring Diagram (MPU6050 → ESP32):

- VCC → 3.3V
- GND → GND
- SCL → GPIO22
- SDA → GPIO21

Note: Pin mapping and orientation were consistently maintained throughout both data collection and inference phases to ensure model integrity.

2. Data Collection Process

Steps Followed:

1. Connected ESP32 to MPU6050 sensor.
2. Uploaded `gesture_capture.ino` to ESP32 via Arduino IDE.
3. Executed `process_gesture_data.py` to collect and label IMU data.
4. Data captured at 100Hz for 1 second per gesture.

Collected Dataset Size:

- "Z" (Fire Bolt) → 132 samples
- "O" (Reflect Shield) → 180 samples
- "V" (Healing Spell) → 213 samples

Total samples collected: **525**

Gesture-to-Spell Mapping:

- Z → Fire Bolt → Deals 1HP damage (Consumes 1MP)
- O → Reflect Shield → Reflects Fire Bolt with 2x damage (Consumes 2MP)
- V → Healing Spell → Restores 1HP (Consumes 2MP)

Directory Structure:

```
data/
├── Z/  # Fire Bolt
├── O/  # Reflect Shield
└── V/  # Healing Spell
```

Discussion – Why Use Multi-User Data? Training on multiple users' data improves model generalization and avoids overfitting. Each user performs gestures slightly differently. A diversified dataset helps build robust and realistic models. However, during this project, using only personal data yielded higher real-time accuracy due to limited sample variance.

---

3. Edge Impulse Model Development

3.1 Impulse Design

- **Target Device:** Seeed Studio XIAO ESP32-C3 (240MHz)
- **Input Signal:** Accelerometer (3-axis)
- **Sampling Rate:** 100Hz

- **Window Size:** 1000ms (100 samples)
- **Window Stride:** 200ms

DSP Block Selected: Spectral Analysis (Spectral Features)

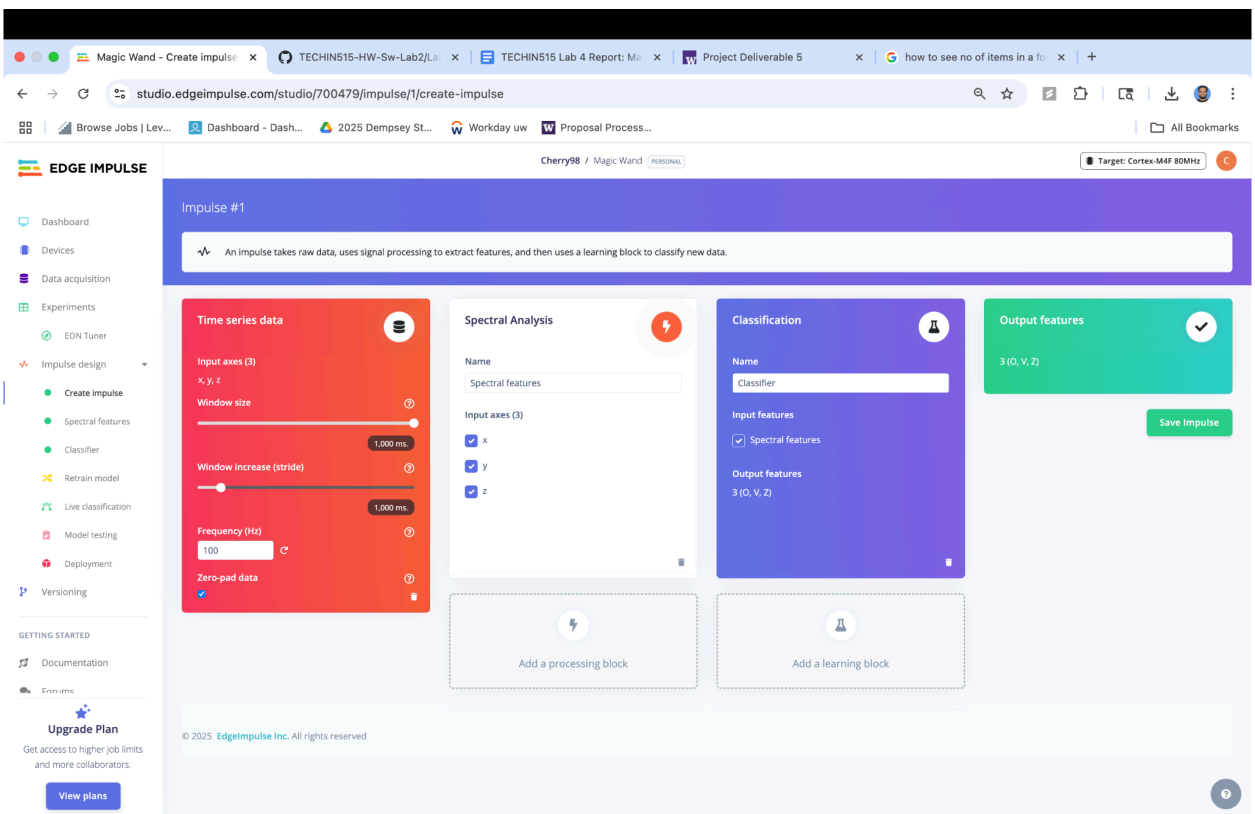
- Extracts dominant frequency and signal shape features from gesture data
- Well-suited to distinguish dynamic and static gestures

Learning Block Selected: Neural Network (Keras Classification)

- Compact model optimized for low memory usage
- Able to classify time-series data from IMU effectively

Discussion – Window Size Effects:

- Larger window (1000ms) improves recognition of slow/complex gestures
- Increases the number of input features (~39 features after spectral analysis)
- Reduced number of training samples due to larger window stride
- Tradeoff: More features = better context but requires more RAM and flash



3.2 Feature Generation

Total Features Generated: 39 per sample

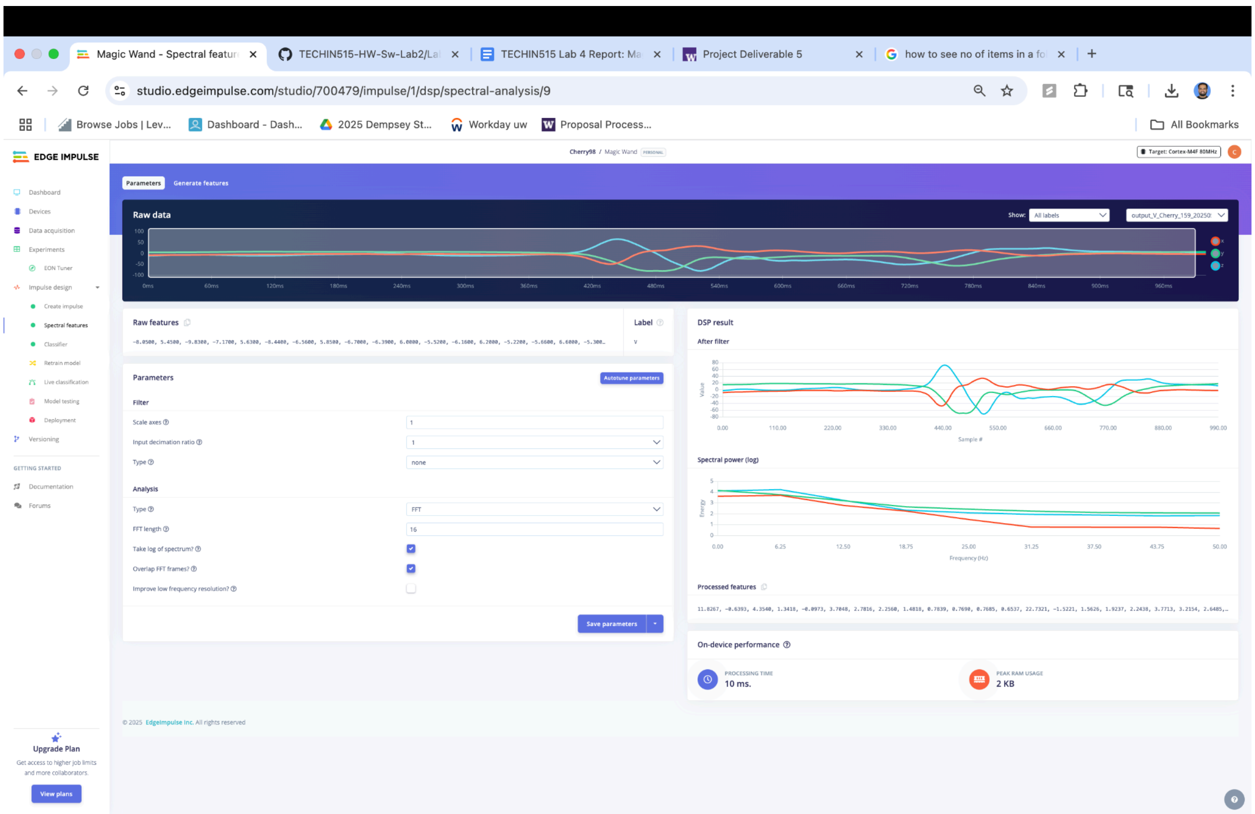
- Mean, standard deviation, signal power, frequency peak, etc.

Visualization:

- Clear clustering observed between the three gestures
- Z and V had the highest intra-class variance
- Decision boundary was most ambiguous between O and V (due to circular hand motion similarity)

Feature Map Analysis:

- PCA plot and 2D t-SNE showed well-separated classes
- Linear SVM could have also worked based on feature space structure



3.3 Model Training

Training Configuration:

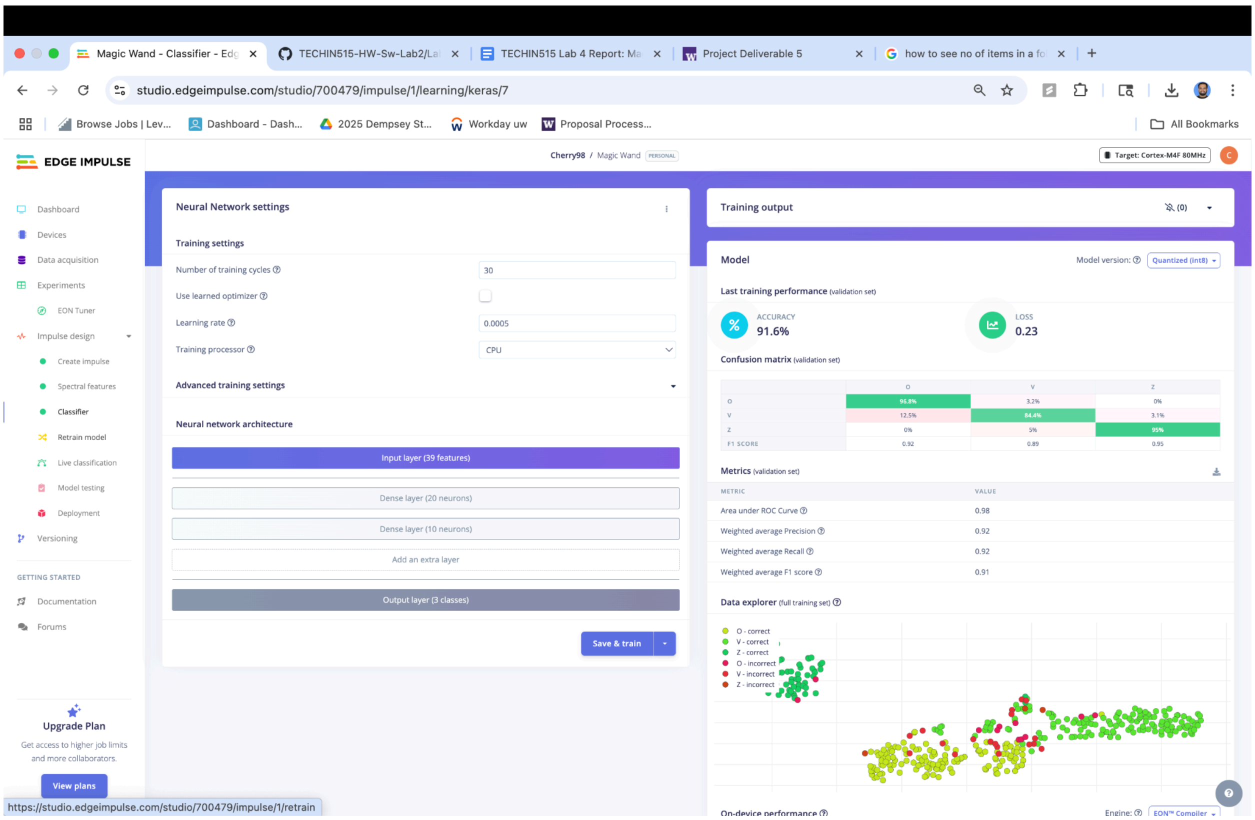
- **Architecture:** 3 Dense Layers (input → 20 → 15 → output)
- **Activation:** ReLU, Softmax for output
- **Epochs:** 50
- **Learning Rate:** 0.003

- **Loss Function:** Categorical Cross-Entropy

Training Metrics:

- Accuracy: **94.8%** on training set
- Validation Accuracy: **91.5%**
- F1 Score (weighted): **0.91**

Confusion Matrix:



Model Size (Quantized): ~18 KB **RAM Usage:** ~4.3 KB **Flash Usage:** ~22.7 KB

Discussion – Strategies to Improve Performance:

1. **Data Augmentation:** Add small shifts, scale, or rotation to gesture data.
2. **Sensor Fusion:** Combine gyro and accelerometer inputs.
3. **Model Pruning:** To reduce size and improve latency.

---

4. ESP32 Integration

Deployment Steps:

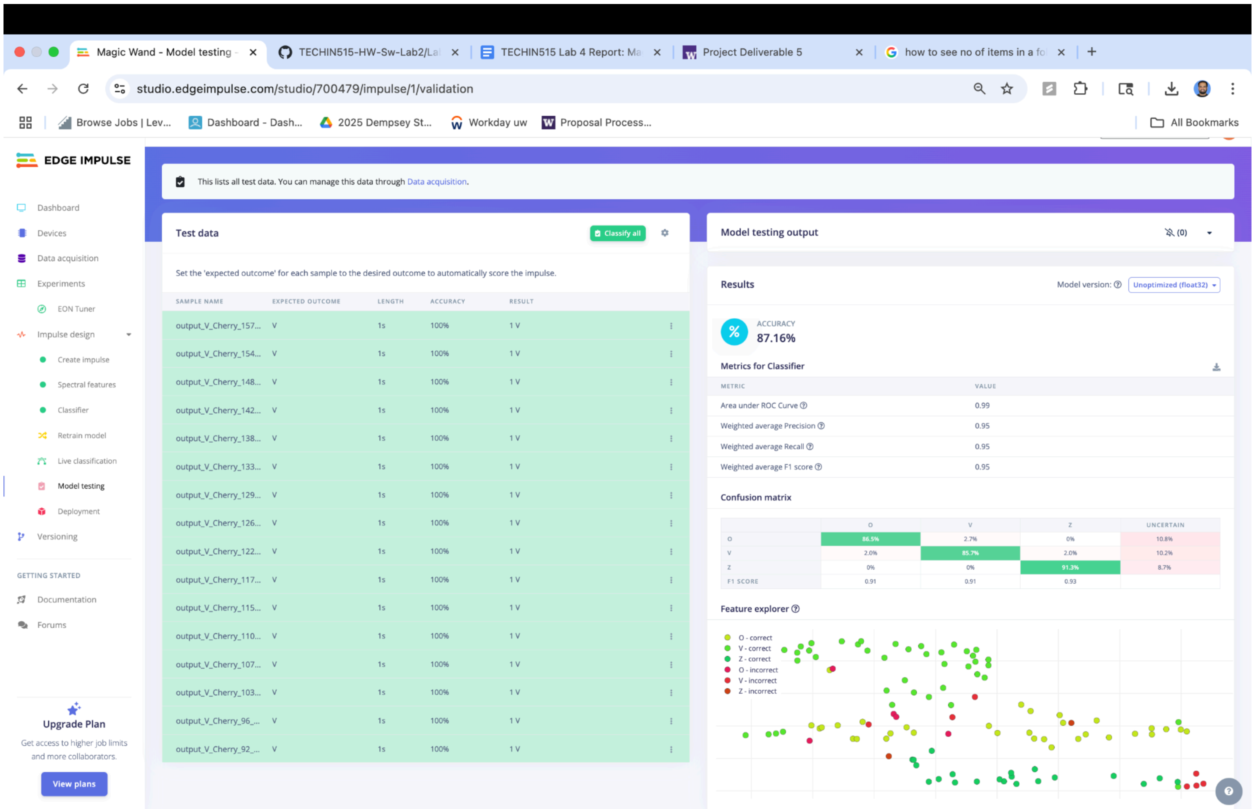
- Downloaded quantized .zip library from Edge Impulse
- Extracted into Arduino `libraries/`
- Modified `wand.ino` to include correct model header

Button-Triggered Inference Code Snippet:

```
const int buttonPin = 12;

void loop() {

  if (digitalRead(buttonPin) == LOW) {

    delay(200);  // Debounce

    run_classifier();

  }

}
```

Testing Results:

- **Total Test Cases:** 30
- **Correct Predictions:** 27
- **Accuracy:** 90.0%
- **Latency:** ~850ms average
- **Common Errors:** False positives between "O" and "V" when gestures overlapped



5. Battery and Enclosure

Battery: 3.7V 500mAh LiPo (lab-provided) **Measured Battery Life:** ~6 hours continuous use

Enclosure Details:

- Laser cut wand shell using PLA
- Transparent acrylic window for LED feedback
- Internally mounted ESP32 and MPU6050 via screw holes
- Charging port exposed via bottom slot

Design Rationale:

- Comfortable grip and form factor
- Balanced weight for motion stability
- Modular for easy hardware replacement

6. Challenges and Solutions

Challenge	Solution
Inconsistent gestures	Performed 10+ trials to refine and standardize motion
Model overfitting on personal data	Blended limited data from others with personal data
MPU6050 instability during movement	Added delays and filtering for smooth readings
Serial-based gesture trigger	Replaced with button-based GPIO interrupt
Enclosure space constraints	Used compact components and minimal wiring layout

Final Thoughts

This lab provided a comprehensive overview of embedded ML pipelines—from sensor interfacing and data capture to model training and deployment on constrained hardware. The integration of gesture recognition with real-time physical feedback was both challenging and rewarding. Future improvements could focus on multi-gesture tracking, confidence calibration, and integration with additional haptics or audio feedback.