ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

Real-Time Human Fall Detection and Alert System Using Autonomous Embedded Neural Network

Jangam Dhanush[1]; Dr. M. Asha Rani[2]

[1,2]Electronics and Communication Engineering, Jawaharlal Nehru Technological, University Hyderabad, India.

Publication Date: 2025/09/15

Abstract: Falls are a major safety risk for older adults and individuals with reduced mobility, making prompt detection essential to reduce the likelihood of serious outcomes. This paper presents a real-time fall detection system built around two ESP32-WROOM-32 microcontroller units (MCUs) arranged in a sender–receiver configuration. An MPU6050 inertial measurement unit (IMU) is connected to the sender via the I²C protocol to obtain motion data, which is subsequently transmitted using the ESP-NOW protocol. The receiver processes this data to perform activity inference using a pre-deployed Multilayer Perceptron (MLP) model trained and tested in Edge Impulse. Detection of a fall triggers the automatic dispatch of an SMTP email notification to caregivers. A testing accuracy of 82.53% demonstrates the system's viability for autonomous, cloud-independent, and resource-efficient wearable health monitoring.

Keywords: *Fall Detection, Wearable Health Monitoring, ESP32-WROOM-32, MPU6050, ESP-NOW, Multilayer Perceptron (MLP).*

How to Cite: Jangam Dhanush; Dr. M. Asha Rani (2025) Real-Time Human Fall Detection and Alert System Using Autonomous Embedded Neural Network. *International Journal of Innovative Science and Research Technology*, 10(9), 574-585 https://doi.org/10.38124/ijisrt/25sep394

I. INTRODUCTION

The convergence of embedded systems and machine learning has enabled significant advances in autonomous health monitoring technologies. Wearable fall detection systems have become essential for safeguarding elderly individuals and patients with motor or neurological impairments. Falls represent a major health risk due to the potential for severe injury, prolonged hospitalization, or fatal outcomes if left unassisted. Conventional detection methods, including camera-based monitoring and manual alert mechanisms, face limitations such as privacy concerns, dependency on fixed infrastructure, and delayed intervention. Wearable sensor-based systems offer an effective alternative by providing portability, real-time responsiveness, and enhanced user autonomy. Among available sensors, the MPU6050 inertial measurement unit (IMU), which integrates a 3-axis accelerometer and a 3-axis gyroscope, is widely adopted due to its compact size, cost-effectiveness, and reliable motion sensing capabilities.

This paper presents a real-time fall detection system utilizing two ESP32-WROOM-32 microcontrollers (MCUs). One MCU acts as a sender node interfaced with the MPU6050 via the I²C protocol, continuously capturing raw tri-axial acceleration and angular velocity data. The other functions as a receiver node that wirelessly acquires this data using the low-latency ESP-NOW protocol. On-device inference is performed by the receiver through a pre-deployed, regularized deep multilayer perceptron (MLP) model trained and validated via Edge Impulse. Upon detection of a fall event, the system promptly transmits an email alert via the SMTP protocol to notify caregivers. The proposed system operates fully autonomously from startup, executing sensing, data transmission, inference, and alerting entirely on-device. This design ensures minimal latency, preserves user privacy, and eliminates reliance on cloud infrastructure, thereby enhancing system robustness and user acceptance. The primary contributions of this work include: integration of ESP-NOW communication with Edge Impulse for embedded activity inference using a regularized deep MLP; development of a cloud-independent, low-latency fall detection prototype deployable on resource-constrained hardware; and establishment of a modular architecture conducive to scalable wearable health monitoring applications.

The remainder of this paper is organized as follows: Section II reviews relevant literature; Section III details the design and infrastructure; Section IV describes methodology and implementation; Section V presents results and discussion; and Section VI concludes with future work.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

II. LITERATURE REVIEW

Wearable sensors have been increasingly studied for automatic fall detection, yet many existing systems are limited in accuracy, responsiveness, or suitability for embedded deployment. LSTM-based approaches with MPU6050 sensors achieved accurate detection in controlled settings but were restricted to fall-versus-non-fall classification and were unsuitable for embedded deployment [1]. Back-propagation networks processing accelerometer and gyroscope data provided only modest accuracy, required substantial computation, and lacked class-imbalance handling [3]. Wireless alert systems via Wi-Fi or GSM/SMS enabled real-time notification but depended on external networks and could not perform on-device classification [4] [7]. Microcontroller-based TinyML implementations allowed falls to be identified in real time, yet they were limited to simple fall detection and did not capture fall direction or support continuous monitoring [2]. Low-cost tri-axial accelerometer systems offered simplicity but suffered from low sampling rates and minimal activity differentiation [5] [6] [8].

The present work addresses these gaps by enabling fully embedded, continuous fall detection with fine-grained classification—including walking, stair ascent/descent, and forward, backward, leftward, and rightward falls—while delivering real-time alerts on resource-constrained platforms.

III. DESIGN AND INFRASTRUCTURE

The detailed system design, including the architecture and data flow, is shown in Fig. 1. The infrastructure consists of hardware components, communication protocols, and software tools and development environments, as described below.

The hardware includes the ESP32-WROOM-32: A low-power SoC with dual-core processing, Wi-Fi, Bluetooth, and peripheral interfaces, suitable for IoT and edge computing; and MPU6050: A 6-axis IMU with an accelerometer and gyroscope for motion sensing, used in fall detection and gesture recognition.

The communication protocols include ESP-NOW: A low-power, peer-to-peer protocol enabling direct communication between ESP32 and ESP8266 devices without requiring a Wi-Fi network, ideal for sensor networks and real-time data transmission; I²C (Inter-Integrated Circuit): A serial protocol that facilitates short-distance communication between multiple devices using two wires, commonly used in embedded systems for connecting sensors and microcontrollers; and SMTP (Simple Mail Transfer Protocol): A protocol for automating the sending of emails and reports, typically used in IoT systems to facilitate alerts and remote monitoring.

The software tools and development environments include Arduino IDE (v2.3.5): An integrated development environment used for writing, compiling, and uploading code to Arduino boards. It includes an updated interface, built-in debugging tools, and support for multiple platforms, making it efficient for rapid embedded system prototyping; Edge Impulse Studio: A cloud-based solution for developing and deploying machine learning models, providing a comprehensive pipeline from data acquisition and preprocessing to deployment on edge devices with minimal latency and optimized power efficiency; Python with TensorFlow/Keras: A framework for developing machine learning models, enabling tasks such as image classification and time-series analysis in both cloud and embedded environments; and Tera Term (v5.3): A terminal emulator used for communication over serial ports, SSH, and Telnet, offering advanced capabilities for real-time data monitoring, debugging, and diagnostics in embedded system applications.

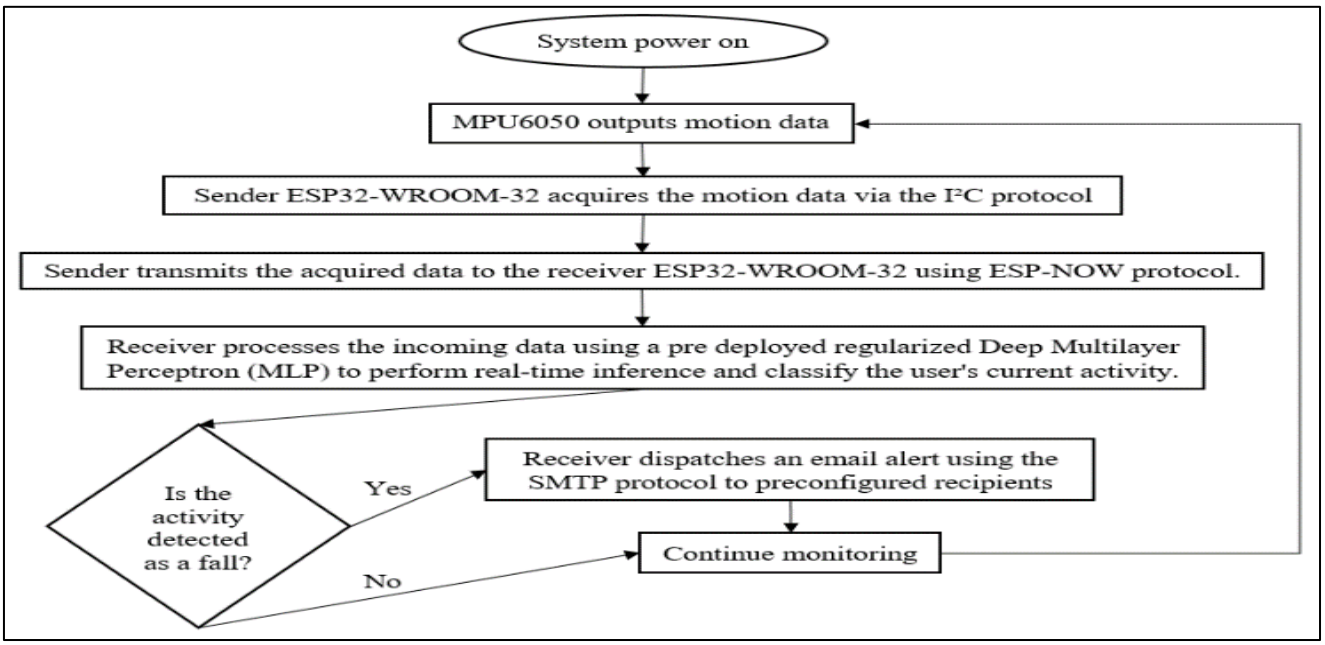


Fig 1 Architecture and Data Flow

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

IV. METHODOLOGY AND IMPLEMENTATION

The experimental setup involved interfacing the MPU6050 sensor with the sender ESP32 via I²C, using GPIO 21 (SDA) and GPIO 22 (SCL) for communication, and powering the sensor at 3.3V to match the ESP32's logic levels. The sender captured inertial data and transmitted it to the receiver ESP32 using ESP-NOW. The receiver was connected to a laptop through USB, supplying power and facilitating serial communication. Data packets were monitored using either the Arduino IDE Serial Monitor or Tera Term, with Tera Term being primarily used for logging and data collection due to its larger buffer and data export capabilities. This configuration ensured reliable transmission and real-time monitoring of sensor data. The entire setup is illustrated in Fig. 2 for the sender and Fig. 3 for the receiver.

The MPU6050 integrates three sensing units: a three-axis accelerometer, a three-axis gyroscope, and an internal temperature sensor. The accelerometer measures linear acceleration along the X, Y, and Z directions finally in terms of g (where 1g = 9.81 m/s²), while the gyroscope measures angular velocity about the same axes finally in degrees per second (°/s). Each unit provides 16-bit signed outputs ranging from −32,768 to +32,767, which are converted into physical values through sensitivity scaling.

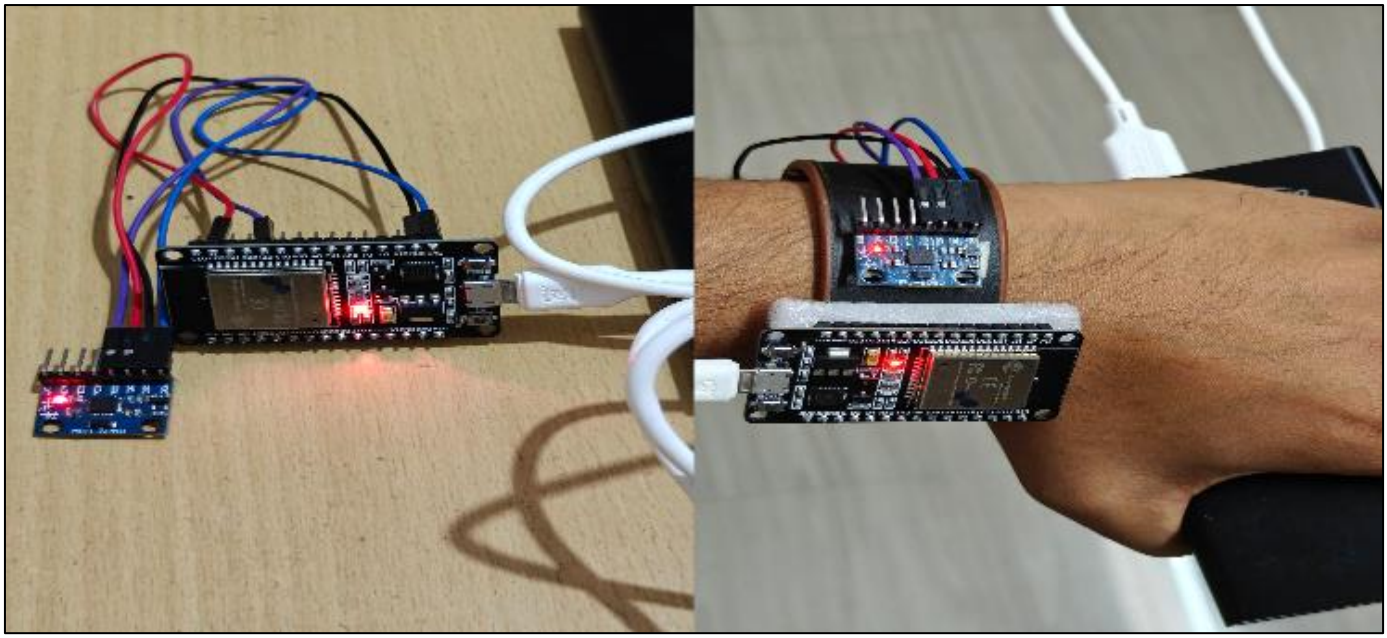


Fig 2 Sender Unit

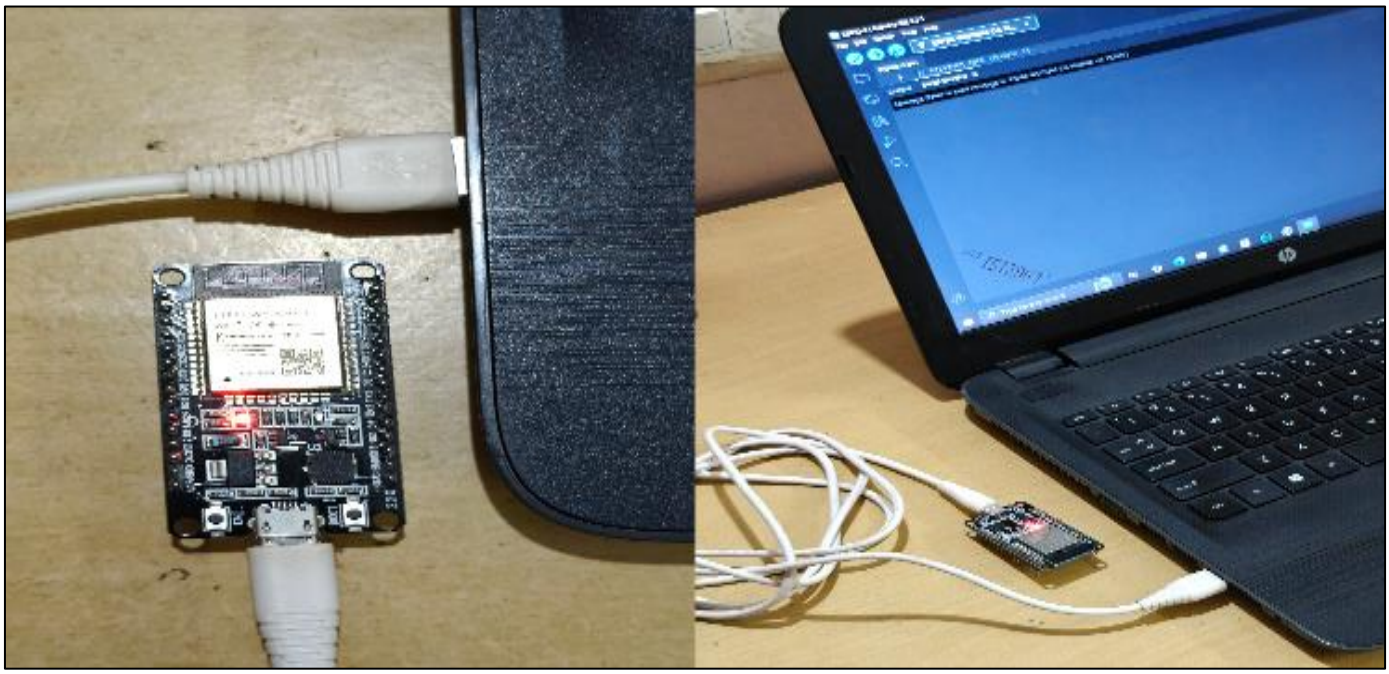


Fig 3 Receiver Unit

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

Sensitivity defines how many digital steps, or Least Significant Bits (LSB), represent one unit of measurement. For a 16-bit resolution (65,536 levels), this range is linearly mapped to the chosen physical span. For example, at ±2g (total range 4g) the sensitivity equals 65,536/4 = 16,384 LSB/g, and at ±500°/s it equals 65.5 LSB/°/s. More generally, higher ranges reduce sensitivity, while lower ranges increase resolution. In this study, the accelerometer was configured at ±4g to capture both routine movements and high-impact events such as falls. The gyroscope was set to ±500°/s, which effectively records angular velocities typical of daily human motions without signal saturation. The temperature sensor was not used, as it does not directly contribute to motion analysis. To improve accuracy, the sensor is calibrated to remove axis-specific biases introduced by manufacturing tolerances, electrical noise, or slight orientation errors. Calibration is performed with the device placed flat and motionless. In this state, the accelerometer should read 0 g on the X and Y axes and +1 g on the Z-axis, while the gyroscope should register 0 °/s on all axes. Any deviation from these values is determined by averaging a large set of stationary readings (about 1000 samples per axis) and treating the result as the offset. These offsets are then subtracted from all subsequent measurements, producing corrected outputs. The procedure is executed automatically during startup, ensuring that all data collected during operation are bias-compensated from the beginning. Mathematically, this is given for accelerometer as:

$$AX_{calibrated} = \frac{1}{S_A} \cdot \left(AX_{raw} - \sum_{k=1}^{N} AX_k \right) \cdot g \quad (1)$$

Similarly for AY. And for AZ, it's shown as:

$$AZ_{calibrated} = \left(\frac{1}{S_A} \cdot \left(AZ_{raw} - \frac{1}{N} \sum_{k=1}^{N} AZ_k \right) + 1 \right) \cdot g \quad (2)$$

And for gyroscope, it's given as:

$$GX_{calibrated} = \frac{1}{S_G} \cdot \left(GX_{raw} - \sum_{k=1}^{N} GX_k \right) \quad (3)$$

Similarly, for GY and GZ, where

$AX_{calibrated}$: Calibrated acceleration value along the X axis, in meters per second squared (m/s²).

AX_{raw}: Raw accelerometer reading along the X axis, in least significant bits (LSB) or counts.

AX_k: Accelerometer calibration samples for the k^{th} reading (k = 1 to N), used to calculate sensor biases.

S_A: Accelerometer sensitivity in LSB per g (gravitational acceleration), specific to the full-scale range used.

$GX_{calibrated}$: Calibrated angular velocity value along the X axis, in degrees per second (°/s).

GX_{raw}: Raw gyroscope reading along the X axis, in least significant bits (LSB) or counts.

GX_k: Gyroscope calibration samples for the k^{th} reading (k = 1 to N), used to calculate sensor biases.

S_G: Gyroscope sensitivity in LSB per degree per second, specific to the full-scale range used.

N: Number of calibration samples used to compute average sensor bias.

g: Standard gravitational acceleration constant (9.81 m/s²), used to convert acceleration from g to m/s².

Following automatic calibration at startup, motion data was collected from four participants: three males aged 24, 26, and 55, and one female aged 47. All participants were briefed on the study objectives and provided informed consent before participation. For consistency, the sensor was strapped to the left wrist using a wristband, with the Y-axis aligned along the forearm and the X-axis pointing outward. This orientation enabled reliable recording of arm movements across all sessions. Each participant performed one activity per session, including walking, stair ascent, stair descent, or simulated falls in the forward, backward, left, or right direction. Non-fall activities were performed in controlled environments free of obstacles. For falls, safety was ensured by using a mattress to cushion the impact, minimizing the risk of injury. Fig. 4 and Fig. 5 compare this process for non-fall and fall activities respectively. During each session, calibrated acceleration (m/s²) and angular velocity (°/s) values along the three axes were continuously recorded. After data collection, activity segments were labeled immediately to ensure accurate ground truth. For instance, data from a walking trial was tagged as "Walking," and similar labeling was applied for all other activities. Data sampling rates were chosen based on activity type: 100 Hz for rapid fall events and 50 Hz for routine movements. Timestamps, initially included, were later removed during preprocessing to simplify the dataset. The cleaned dataset was structured into an Excel sheet containing acceleration, angular velocity, and activity labels. After preprocessing and consolidation, the final dataset comprised 43,523 rows across seven classes: Walking (19,802 rows), Upstairs (5,095), Downstairs (4,908), Front Fall (3,056), Back fall (3,219), Left Fall (5,801), and Right Fall (1,642). The finalized dataset was exported in CSV format for analysis.

Once data collection was complete, the next phase involved training a deep neural network to recognize activity patterns in real time. This was carried out on Edge Impulse, a platform designed for embedded machine learning development. The deployment target was the Espressif ESP-EYE (ESP32, 240 MHz), ensuring compatibility with the intended hardware. The CSV dataset consisting of 43,523 labeled samples was uploaded to the platform. An automatic split created two sets: 80% (34,929 samples) for training and 20% (8,594 samples) for testing. The impulse, representing the end-to-end data processing pipeline, was configured with three blocks:

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

Fig 4 Walking, Upstairs, Downstairs

Fig 5 Front Fall, Back Fall, Left Fall, Right Fall

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

- Input Block: Configured to accept six sensor-derived features: linear accelerations (AX, AY, AZ) and angular velocities (GX, GY, GZ). Each sample was thus represented as a six-dimensional vector:

$$X^i = [AX^i, AY^i, AZ^i, GX^i, GY^i, GZ^i] \in \mathbb{R}^6 \quad (4)$$

Where i is the sample index.

- Processing Block: Set to Raw Data, passing the input vectors directly to the learning block without applying filtering, windowing, or normalization. This preserved the calibrated physical units (m/s² for acceleration, °/s for angular velocity). On-device performance analysis for the target confirmed negligible overhead, with an estimated processing time of 1ms and RAM usage of 24 B.

- Learning Block: Implemented as a Classification module, mapping the six input features onto seven activity classes:

$$f : \mathbb{R}^6 \rightarrow Y \quad (5)$$

Where Y is a 7-dimensional one-hot encoded vector corresponding to the class set {Back Fall, Downstairs, Front Fall, Left Fall, Right Fall, Upstairs, Walking}. For example, if the activity is Walking, the output is represented as {0, 0, 0, 0, 0, 0, 1}, while Front Fall is encoded as {0, 0, 1, 0, 0, 0, 0}.

A Deep Multi-Layer Perceptron (MLP) model was trained to learn complex decision boundaries among seven activity classes using six input features (AX, AY, AZ, GX, GY, GZ). The network followed a pyramidal structure with three hidden layers of 168, 84, and 42 neurons, designed to capture rich representations in the initial layer and progressively compress them to reduce over fitting. ReLU or Rectified Linear Unit activations (an activation introduces non-linearity to a neuron's output) were applied to all hidden layers to maintain computational efficiency and stable training. This is given by:

$$ReLU\ (x) = max\ (0, x) \quad (6)$$

Where x is the pre-activation value of a neuron. The output layer consisted of seven neurons with Softmax activation to generate normalized class probabilities, defined as

$$P_{i,j} = softmax_j\ (Z_i) = \frac{e^{Z_{i,j}}}{\sum_{k=1}^{C} e^{Z_{i,k}}} \forall\ i \in \{1, \ldots, B\}, j \in \{1, \ldots, C\} \quad (7)$$

Where

$P_{i,j}$ is the predicted probability of sample i belonging to class j.

$Z_{i,j}$ is the logit (pre-activation score) for class j of sample i.

C is the total number of output classes.

B is the batch size.

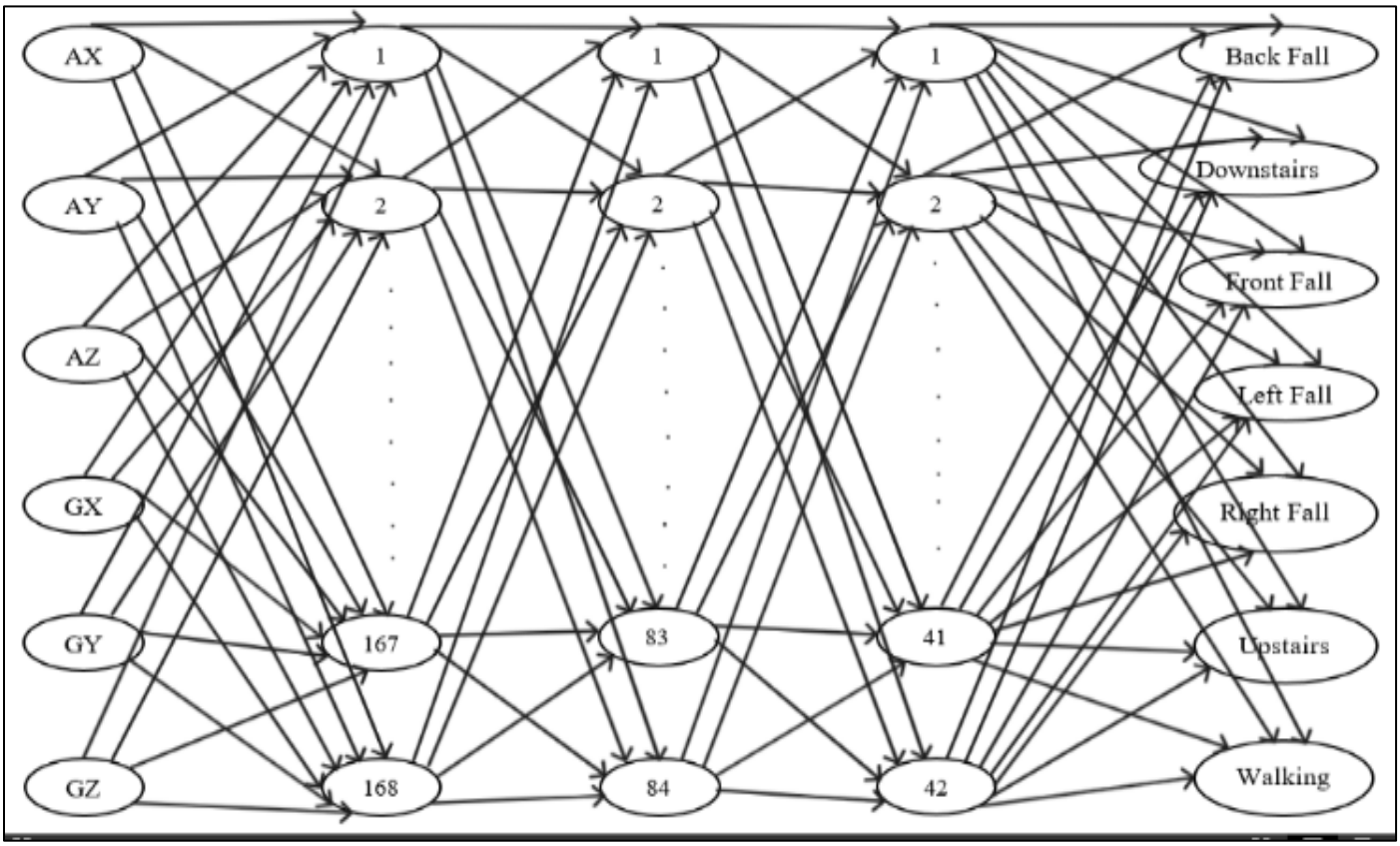


Fig 6 MLP Model Architecture

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

Weights (control the strength of input signals) were initialized using HeNormal initializer for hidden layers (suited to ReLU) and GlorotUniform initializer for the output layer, and biases (provide a constant offset to the activation) were set to zero, ensuring balanced signal variance. These initializations are defined for HeNormal as:

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{fan_{in}}}\right) \quad (8)$$

And for GlorotUniform as:

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{fan_{in} + fan_{out}}}, \sqrt{\frac{6}{fan_{in} + fan_{out}}}\right) \quad (9)$$

Where

W denotes the weight matrix connecting two layers in a neural network.

fan_{in} refers to the number of input units to a neuron or layer (i.e., the number of incoming connections).

fan_{out} refers to the number of output units from a neuron or layer (i.e., the number of outgoing connections).

To further control overfitting, empirically tuned L2 regularization penalties (0.0005, 0.0003, 0.0002, and 0.0001 for successive layers) were applied. Training used the Adam optimizer with a learning rate of 0.00075, chosen for its robustness to sparse gradients, along with a class-weighted categorical cross-entropy loss to account for the imbalance between frequent activities and rarer fall events. This can be shown as:

$$w_c = \frac{N_t}{C \,.\, N_c} \quad (10)$$

Where

w_c is the class weight for class c.

N_t is the total number of samples in the training set.

C is the number of output classes.

N_c is the number of samples of class c.

The training set of 34,929 samples was split into 80% (27,943) for training subset and 20% (6,986) for validation subset. Training proceeded in mini-batches of 256 samples, so that each epoch (a full pass through the training subset) consisted of about 27,943/256 = 109 batch updates. For the first batch of the first epoch, the model parameters (weights and biases) were initialized. For each batch, inputs were passed forward through the network to compute predictions, the batch loss and batch accuracy were calculated, and gradients were back propagated. The Adam optimizer then updated the parameters. After completing all batches in an epoch, the average batch loss and average batch accuracy were computed. The updated parameters from that epoch were then used to evaluate the untouched validation subset, yielding the validation loss and validation accuracy. This process was repeated for 1,500 epochs. The mathematical formulation of the generalized training flow for each batch is presented as follows:

During the forward pass, for the input layer, let

$$A^0 = X^{B\times D} \quad (11)$$

Where

X is the input matrix of order BxD.

A^0 is the input activation.

B is the batch size.

D is the input feature size.

And for hidden layers (h = 1 to H), it's

$$[Z^h]^{B\times U_h} = ([A^{h-1}]^{B\times U_{h-1}} \cdot [W^h]^{U_{h-1}\times U_h} + [b^h]^{1\times U_h})^{B\times U_h} \quad (12)$$

And

$$A^h = ReLU\,(Z^h) \quad (13)$$

Where

H is the number of hidden layers.

Z^h is the pre-activation values for hidden layer h.

A^h is the post-activation outputs for layer h.

W^h is the weight matrix for hidden layer h.

b^h is the bias vector for hidden layer h.

U_h is the number of neurons in hidden layer h.

At the output layer, it's

$$[Z^C]^{B\times C} = ([A^H]^{B\times U_h} \cdot [W^h]^{U_h\times C} + [b^h]^{1\times C})^{B\times C} \quad (14)$$

And

$$P = softmax\,(Z^C) \quad (15)$$

Where P is the predicted class probabilities (after Softmax). The categorical cross entropy loss function is expressed as:

$$\mathcal{L}_{CCE} = \frac{-1}{B}\sum_{i=1}^{B}\sum_{c=1}^{C} w_c \,.\, [Y_{i,c}]^{B\times C} \,.\, log\left([P_{i,c}]^{B\times C}\right) \quad (16)$$

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

L2 regularization is applied as:

$$\mathcal{L}_2{}^{h} = \lambda_h \,.\, \| W^h \|^2 = \lambda_h \,. \sum_{i=1}^{U_{h-1}} \sum_{j=1}^{U_h} \left(W_{i,j}{}^{h}\right)^2 \quad (17)$$

$$\mathcal{L}_{L2} = \sum_{h=1}^{H} \mathcal{L}_2{}^{h} + \lambda_{out} \,.\, \| W^C \|^2 \quad (18)$$

The total loss is then computed as:

$$\mathcal{L} = \mathcal{L}_{CCE} + \mathcal{L}_{L2} \quad (19)$$

Where

$L_2{}^h$ is the L2 regularization loss for the hidden layer h.

L_{L2} is sum of L2 losses over all layers (hidden + output).

λ_h is the L2 regularization coefficient for layer h.

λ_{out} is the L2 regularization coefficient for the output layer.

The notation || . || is the squared L2 norm of the weight matrix, i.e., the sum of the squares of all its elements.
Back propagation updates network parameters by propagating the error from the output layer back through the hidden layers. The gradient at the output is:

$$\frac{\partial \mathcal{L}}{\partial Z^C} = (P - Y)^{B \times C} \odot (W_{class})^{B \times C} \quad (20)$$

Where

⊙represents the element-wise (component-wise) multiplication of two matrices or vectors of the same dimensions.

Y is the one-hot encoded true class labels of the current batch.

W_{class} is a weight matrix with each element $W_{class}[i, c] = w_c$.

For hidden layer gradients from h = H down to 1:

$$\frac{\partial \mathcal{L}}{\partial A^h} = \frac{\partial \mathcal{L}}{\partial Z^{h+1}} \cdot (W^{h+1})^T \quad (21)$$

$$\frac{\partial \mathcal{L}}{\partial Z^h} = \frac{\partial \mathcal{L}}{\partial A^h} \odot ReLU'(Z^h) \quad (22)$$

Regularization gradients are incorporated as:

$$\frac{\partial \mathcal{L}}{\partial W^h} \mathrel{+}= 2 \cdot \lambda_h \,.\, W^h \quad (23)$$

Where ReLU' (Z) = 1, if Z > 0, otherwise 0.

An optimizer is an algorithm that updates a model's parameters during training to minimize the loss function and improve performance. The Adam optimizer (Adaptive Moment Estimation) is an advanced optimization algorithm that integrates the benefits of momentum and adaptive learning rates to ensure efficient and stable convergence during neural network training. It computes the first moment (mean) and second moment (uncentered variance) of gradients for each parameter, applies bias correction, and updates the parameters with a controlled step size. Let the hyper parameters of this algorithm be:

Learning rate α.

First moment decay β1 = 0.9.

Second moment decay β2 = 0.999.

Numerical stability constant $\varepsilon = 10^{-7}$.

Then, the update rule for parameter θ (weights or biases) at step t is:

$$g_t = \nabla_\theta \mathcal{L}_t \quad (24)$$

$$m_t = \beta_1 \,.\, m_{t-1} + (1 - \beta_1) \,.\, g_t \quad (25)$$

$$v_t = \beta_2 \,.\, v_{t-1} + (1 - \beta_2) \,.\, g_t{}^2 \quad (26)$$

$$M_t = \frac{m_t}{1 - \beta_1{}^t} \quad (27)$$

$$V_t = \frac{v_t}{1 - \beta_2{}^t} \quad (28)$$

$$\theta_{t+1} = \theta_t - \alpha \,.\, \frac{M_t}{\sqrt{V_t} + \varepsilon} \quad (29)$$

Where

g is gradient.

m is first moment estimate; M is its bias corrected version.

v is second moment estimate; V is its bias corrected version.

θ_{t+1} is the parameter update.

The initial values of both moment estimates are set to zero. The accuracy of the batch is computed as:

$$Acc = \frac{1}{B} \,.\, \sum_{i=1}^{B} \mathbb{I}\left(arg\,max\,(P_i) = arg\,max\,(Y_i)\right) \quad (30)$$

Where

I (·) is the indicator function, which equals 1 if the condition inside is true (correct prediction) and 0 otherwise.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

arg max (P_i) is the index of the class with the highest predicted probability.

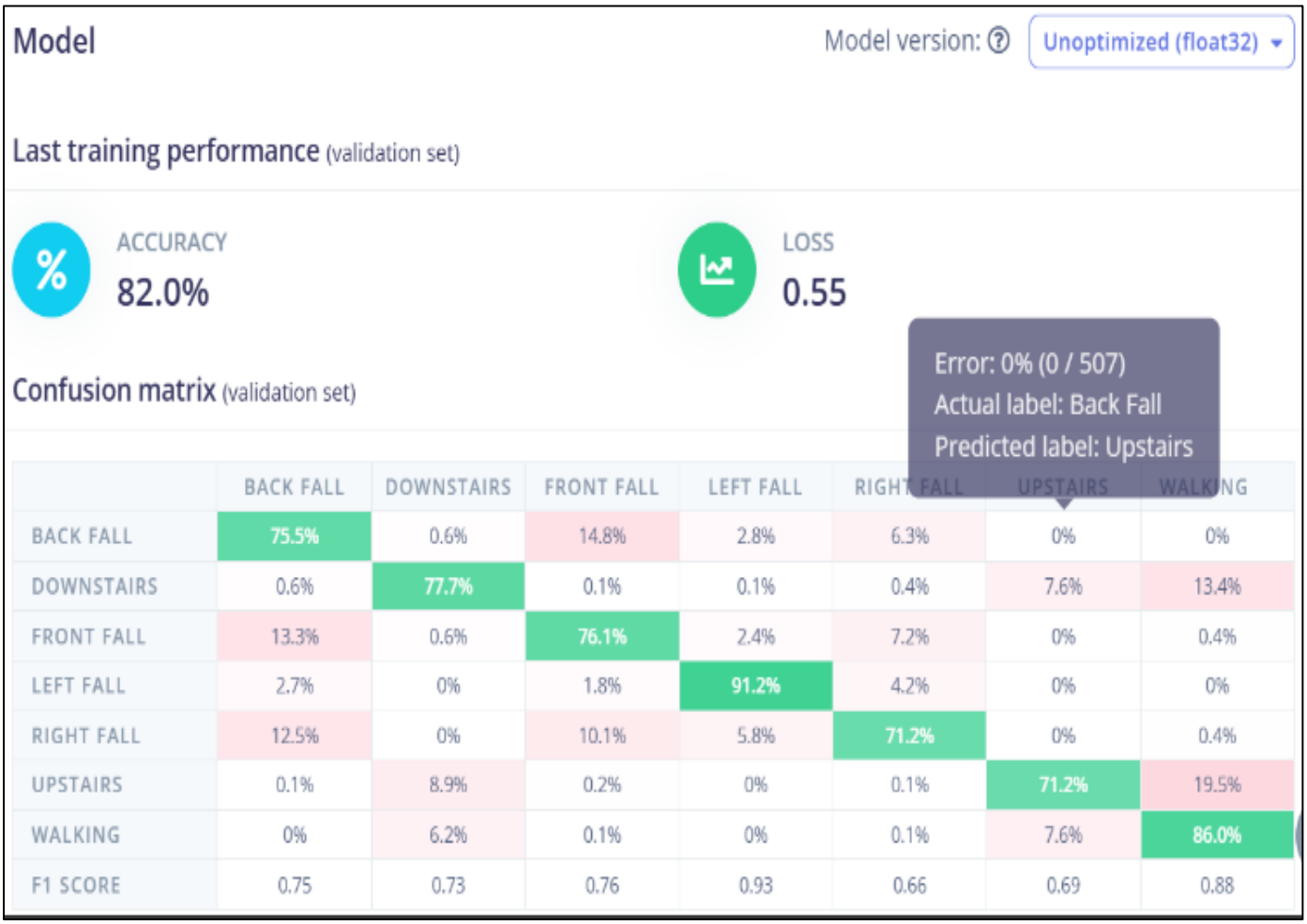
arg max (Y_i) is the index of the actual class label.

On-device performance measured via the EON (Edge Optimized Neural network) compiler showed an inference time of 6ms, peak RAM usage of 2.4 KB, and Flash usage of 86.9 KB, demonstrating the model's suitability for embedded deployment on the receiver ESP32. The inference on the test set was evaluated using the final training output parameters, to confirm its ability to generalize to unseen data. This step validated that the model retained its performance on data it had never seen during training or validation. This tested model was then loaded and deployed onto the receiver via the generated Arduino library, enabling real-time inference on incoming sensor data from the sender module

.

V. RESULTS AND DISCUSSION

The final training output was defined as the combination of the lowest validation loss, its corresponding validation accuracy, and the confusion matrix obtained for that epoch. These results were obtained using the unoptimized full-precision 32-bit floating-point (FP32) model, without applying quantization (unlike INT8 quantized models, which reduce model size and inference latency by using 8-bit integers at the cost of numerical precision). The achieved loss was 0.55, with a corresponding accuracy of 82%. This is outlined in Fig. 7.



	BACK FALL	DOWNSTAIRS	FRONT FALL	LEFT FALL	RIGHT FALL	UPSTAIRS	WALKING
BACK FALL	75.5%	0.6%	14.8%	2.8%	6.3%	0%	0%
DOWNSTAIRS	0.6%	77.7%	0.1%	0.1%	0.4%	7.6%	13.4%
FRONT FALL	13.3%	0.6%	76.1%	2.4%	7.2%	0%	0.4%
LEFT FALL	2.7%	0%	1.8%	91.2%	4.2%	0%	0%
RIGHT FALL	12.5%	0%	10.1%	5.8%	71.2%	0%	0.4%
UPSTAIRS	0.1%	8.9%	0.2%	0%	0.1%	71.2%	19.5%
WALKING	0%	6.2%	0.1%	0%	0.1%	7.6%	86.0%
F1 SCORE	0.75	0.73	0.76	0.93	0.66	0.69	0.88

Fig 7 Final Training Output

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

	BACK FALL	DOWNSTAIRS	FRONT FALL	LEFT FALL	RIGHT FALL	UPSTAIRS	WALKING	UNCERTAIN
BACK FALL	74.8%	0.8%	13.8%	4.3%	6.1%	0%	0.3%	0%
DOWNSTAIRS	0.5%	78.3%	0.2%	0%	0%	7.5%	13.4%	0%
FRONT FALL	11.0%	0%	76.1%	5.0%	7.4%	0%	0.5%	0%
LEFT FALL	2.9%	0.1%	3.0%	91.0%	3.0%	0%	0%	0%
RIGHT FALL	7.9%	0.9%	9.5%	3.8%	77.5%	0%	0.3%	0%
UPSTAIRS	0.2%	9.0%	0.2%	0%	0.2%	74.8%	15.6%	0%
WALKING	0.1%	6.2%	0.2%	0%	0.1%	7.5%	85.9%	0%
F1 SCORE	0.76	0.73	0.75	0.92	0.71	0.71	0.89	

Fig 8 Final Testing Output

```
17:48:33.934 -> Connected to Wi-Fi: n[redacted]G
17:48:35.343 -> Current date and time (IST): 11/07/2025, 17:48:37
17:50:06.670 -> AX(m/s²):-1.06,AY(m/s²):1.52,AZ(m/s²):9.70,GX(°/s):-0.44,GY(°/s):-0.36,GZ(°/s):-2.35,Label:Upstairs
17:50:18.478 -> AX(m/s²):-4.56,AY(m/s²):4.34,AZ(m/s²):8.77,GX(°/s):7.19,GY(°/s):-0.03,GZ(°/s):-4.44,Label:Walking
17:51:00.631 -> AX(m/s²):-5.52,AY(m/s²):-0.15,AZ(m/s²):9.16,GX(°/s):-17.48,GY(°/s):-9.13,GZ(°/s):18.70,Label:Upstairs
17:51:05.641 -> AX(m/s²):-9.75,AY(m/s²):-0.97,AZ(m/s²):7.27,GX(°/s):9.40,GY(°/s):25.90,GZ(°/s):-58.94,Label:Downstairs
17:51:25.652 -> AX(m/s²):0.75,AY(m/s²):2.03,AZ(m/s²):-2.61,GX(°/s):197.60,GY(°/s):118.06,GZ(°/s):377.25,Label:Front Fall
17:51:25.763 ->
17:51:25.763 -> Fall detected, sending an email...
17:51:34.696 -> An alert email sent to d[redacted]6@gmail.com. Please check the inbox immediately!
17:51:45.698 -> AX(m/s²):-0.50,AY(m/s²):-2.83,AZ(m/s²):9.76,GX(°/s):0.46,GY(°/s):0.29,GZ(°/s):-3.49,Label:Walking
17:52:03.795 -> AX(m/s²):-4.82,AY(m/s²):3.10,AZ(m/s²):9.05,GX(°/s):-50.77,GY(°/s):-144.04,GZ(°/s):496.06,Label:Front Fall
17:52:03.795 -> Fall detected during cooldown – email skipped.
17:52:55.117 -> AX(m/s²):9.52,AY(m/s²):-12.05,AZ(m/s²):11.73,GX(°/s):-13.47,GY(°/s):-43.89,GZ(°/s):29.68,Label:Downstairs
17:53:19.226 -> AX(m/s²):9.77,AY(m/s²):-10.75,AZ(m/s²):12.11,GX(°/s):-27.72,GY(°/s):-44.82,GZ(°/s):27.05,Label:Downstairs
17:53:31.195 -> AX(m/s²):-4.60,AY(m/s²):-1.69,AZ(m/s²):-18.02,GX(°/s):-12.98,GY(°/s):-48.22,GZ(°/s):26.93,Label:Back Fall
17:53:31.293 ->
17:53:31.293 -> Fall detected, sending an email...
17:53:41.481 -> An alert email sent to d[redacted]6@gmail.com. Please check the inbox immediately!
17:53:44.393 -> AX(m/s²):-4.62,AY(m/s²):-1.89,AZ(m/s²):-18.10,GX(°/s):-13.46,GY(°/s):-44.57,GZ(°/s):29.65,Label:Back Fall
17:53:44.401 -> Fall detected during cooldown – email skipped.
17:54:04.305 -> AX(m/s²):9.49,AY(m/s²):11.32,AZ(m/s²):-20.24,GX(°/s):-7.95,GY(°/s):-42.44,GZ(°/s):33.10,Label:Left Fall
17:54:04.305 -> Fall detected during cooldown – email skipped.
17:54:11.305 -> AX(m/s²):0.64,AY(m/s²):5.49,AZ(m/s²):8.04,GX(°/s):-20.17,GY(°/s):-153.08,GZ(°/s):-1.08,Label:Downstairs
17:54:59.516 -> AX(m/s²):-17.58,AY(m/s²):-1.85,AZ(m/s²):-5.59,GX(°/s):-62.97,GY(°/s):-101.28,GZ(°/s):-118.75,Label:Back Fall
17:54:59.516 ->
17:54:59.516 -> Fall detected, sending an email...
17:55:07.572 -> An alert email sent to d[redacted]6@gmail.com. Please check the inbox immediately!
17:56:06.674 -> AX(m/s²):10.11,AY(m/s²):-16.88,AZ(m/s²):7.85,GX(°/s):-12.85,GY(°/s):-43.67,GZ(°/s):29.79,Label:Downstairs
17:56:14.713 -> AX(m/s²):10.30,AY(m/s²):-17.01,AZ(m/s²):7.99,GX(°/s):-14.53,GY(°/s):-43.81,GZ(°/s):29.66,Label:Downstairs
```

Fig 9 Final Inference Output

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

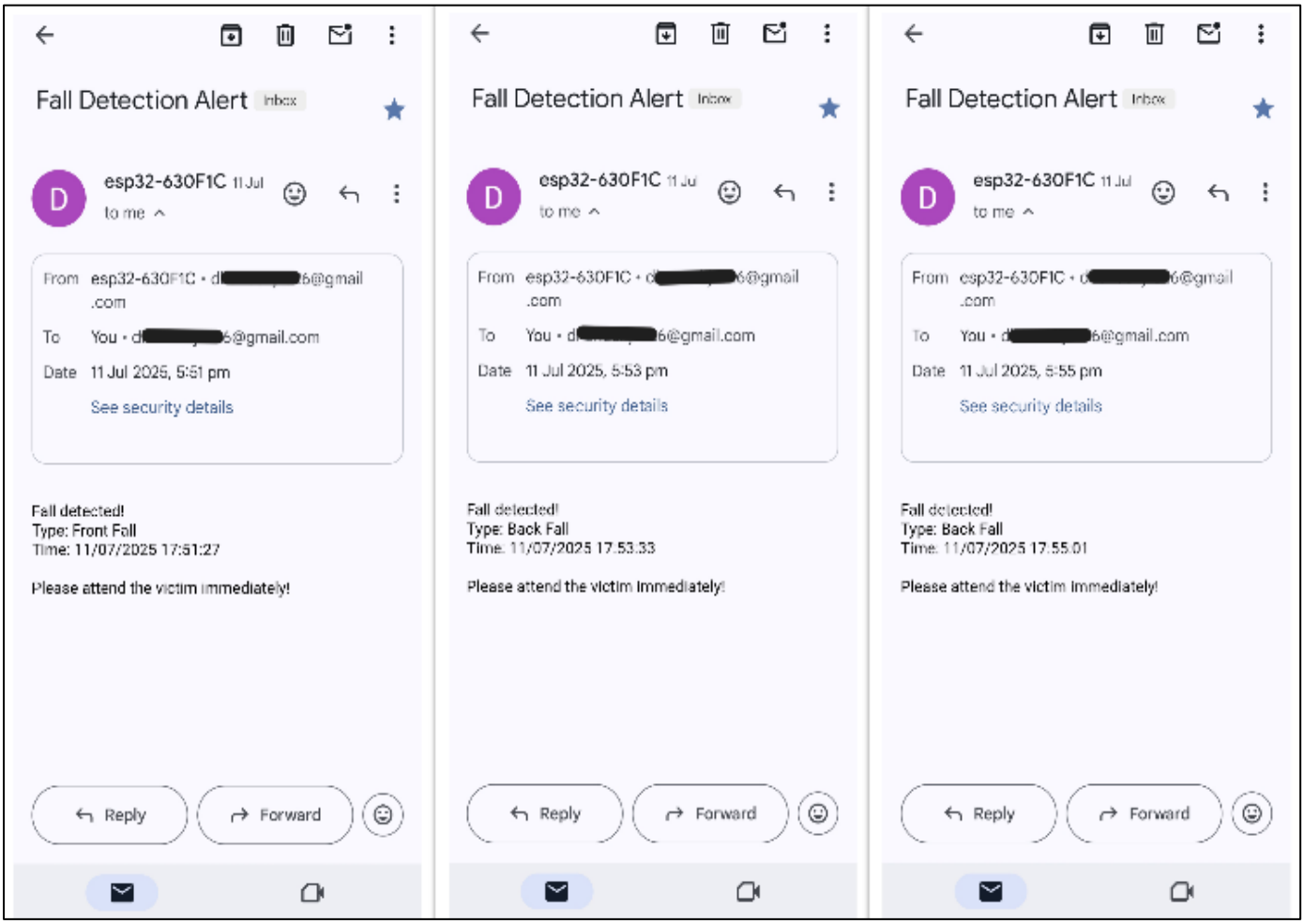


Fig 10 Final Mail Alert Output

The accuracy and confusion matrix were computed for the test set, with the minimum confidence threshold set to 0, ensuring that all predictions were mapped to a known label and no "uncertain" class occurred. An overall accuracy of 82.53% was achieved. Fig. 8 presents this.

During inference, each new sensor reading (AX, AY, AZ, GX, GY, GZ) is processed individually by the embedded model, which classifies it and prints the predicted label alongside the raw sensor values on the serial monitor. The system continuously monitors for fall-type events—Front Fall, Back Fall, Left Fall, or Right Fall—and automatically triggers an email alert via SMTP whenever such an event occurs, with a 60-second cooldown to prevent repeated notifications. Accurate timestamps are generated using the onboard real-time clock, ensuring that each alert is time-stamped. This is visualized in Fig. 9 and Fig. 10.

These results demonstrate that the model operates efficiently in a low-power, embedded environment, supporting autonomous, continuous classification and immediate fall notifications.

VI. CONCLUSION AND FUTURE WORK

This study introduces a real-time fall detection system that integrates an MPU6050 IMU with two ESP32-WROOM-32 MCUs communicating via ESP-NOW. Motion signals are processed directly on the receiver MCU using a regularized Deep MLP trained and fine-tuned on Edge Impulse. Across tests, the system recognized falls toward the front, back, left, and right with short response times, operating entirely within the MCU's processing and memory limits. Its small form factor and wearable nature make it applicable to elderly assistance, workplace hazard prevention, and intelligent healthcare systems. Performance gaps remain due to a compact dataset, limited coverage of daily activities, and occasional confusion between motions with similar dynamics, all affected by MCU hardware constraints. Future work will focus on enlarging the dataset with varied participants and movements, improving model robustness, enabling mobile connectivity for added capabilities, and assessing endurance through extended deployment in uncontrolled real-world conditions.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25sep394

ACKNOWLEDGMENT

The contribution of the four participants is gratefully acknowledged, as their involvement enabled the collection of reliable motion data, which was essential for the development and validation of the proposed fall-detection system.

REFERENCES

[1]. S. -T. Hsieh and C. -L. Lin, "Fall Detection Algorithm Based on MPU6050 and Long-Term Short-Term Memory network," 2020 International Automatic Control Conference (CACS), Hsinchu, Taiwan, 2020, pp. 1-5.

[2]. D. L. Cong, B. N. Quang, D. T. Minh, D. T. Cao and M. N. Ngoc, "Continuous Wearable-based Fall Detection using Tiny Machine Learning," 2024 9th International Conference on Applying New Technology in Green Buildings (ATiGB), Danang, Vietnam, 2024, pp. 339-344.

[3]. Jefiza, E. Pramunanto, H. Boedinoegroho, & M. Purnomo, "Fall detection based on accelerometer and gyroscope using back propagation", 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), p. 1-6.

[4]. G. Mahesh and M. Kalidas, "A Real-Time IoT Based Fall Detection and Alert System for Elderly," 2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT), Faridabad, India, 2023, pp. 327-331.

[5]. H. R. Kumar, S. Janardhan, D. Prakash and M. K. Prasanna Kumar, "Fall Detection System using Tri-Axial Accelerometer," 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 2018, pp. 1846-1850.

[6]. C. Nutsathaporn, S. Chomkokard, W. Wongkokua, N. Jinuntuya, S. Ruengittinun and S. Sasimontonkul, "Human Fall Prediction and Detection Using Low Price IMU Sensor," 2022 IEEE 4th Eurasia Conference on IOT, Communication and Engineering (ECICE), Yunlin, Taiwan, 2022, pp. 157-159.

[7]. N. A. Syafiqah Mohd Sharif, M. Zaki Ayob and S. B. Yusoff, "Development of Wearable Fall Detection Alert for Elderly," 2023 International Conference on Engineering Technology and Technopreneurship (ICE2T), Kuala Lumpur, Malaysia, 2023, pp. 311-315.

[8]. Kurniawan, A. R. Hermawan and I. K. E. Purnama, "A wearable device for fall detection elderly people using tri dimensional accelerometer," 2016 International Seminar on Intelligent Technology and Its Applications (ISITIA), Lombok, Indonesia, 2016, pp. 671-674.