Fall Detection Device Using Machine Learning on Embedded Systems

Siddarth Nandyala

Project Abstract

This project introduces an Embedded Fall Detection System using machine learning algorithms for rapid response to protect individual safety. By employing advanced sensors like the gyroscope and accelerometer IMU with real-time data analysis capabilities, an alarm will sound immediately upon detection of falls by users as soon as a system alarm triggers instant alarms for them; notifications will then be sent out by both email and notification system to designated contacts.

At its heart lies machine learning models trained on a fall dataset, which recognize patterns on efficient low-power embedded hardware to deliver discreet yet reliable daily life integration. When falls occur, immediate alarms provide user feedback, while email and notification alerts provide essential details that enable swift assistance from emergency services or caregivers.

This project pioneers assistive technology by embedding intelligent fall detection using machine learning on low-power systems. This technology can increase accuracy and quickly adapt to evolving situations, ensuring timely interventions to mitigate fall injuries and enhance the safety of at-risk individuals.

Background and Related Work

Problem

Falling is the highest reason for hospitalization for older people in the United States. According to the National Institutes of Health, around 250,000 injuries and 11,000 deaths happen in the United States alone due to falling. During falls leading to severe injuries, receiving immediate help and assistance is vital and can be the difference between life and death. An estimated 9,000,000 citizens report falls annually, revealing a need for fall-detection devices (Verma et al., 2016).

Current Solutions

Current solutions implemented include smart watches, necklaces, and more. These traditional methods of fall detection and alerts fall under a subset of a few disadvantages.

1. Cost effectiveness is a vital factor. The cost of the item can inhibit people from having access to innovative and potentially lifesaving devices.
2. Implementation of what the device can detect. Many devices can get falsely triggered by walking up and down stairs. This leads to a loss of accuracy, leading to alarms being triggered when superfluous and not being triggered when necessary. Furthermore,

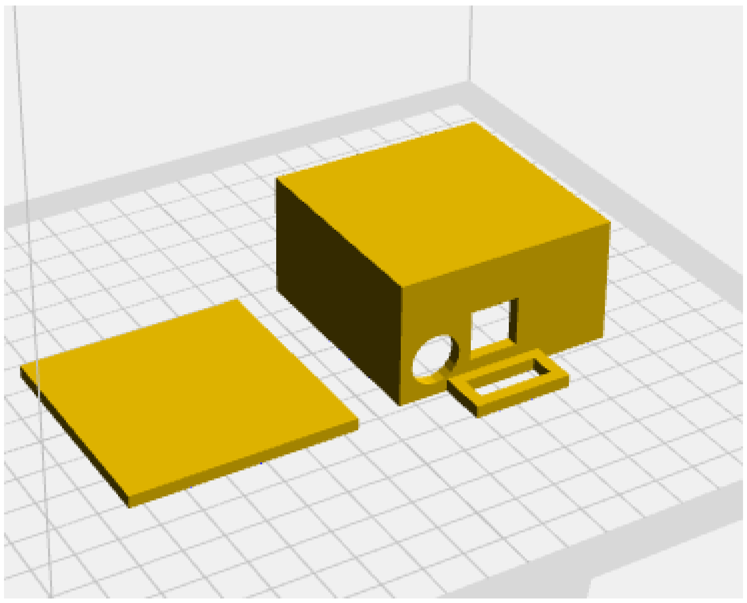
devices such as the most common devices for this application cannot detect falls from a bed, which is another widespread fall leading to injuries.

Methods

The solution to traditional flaws with fall detection can be an efficient microcontroller running a neural network trained to detect falls. The solution to this would work using gyroscopic and accelerometer data to classify the user's movement and act based on the model's inference.

Creating Band

The band was created to house all the components and have access to connect the Arduino to the computer to receive samples for data collection. The band also had sized openings to house the button and buzzer for this project. Below is a visual of the design that was 3d printed in PLA. You can download the 3D model at https://www.printables.com/model/754533-fall-detection-module.

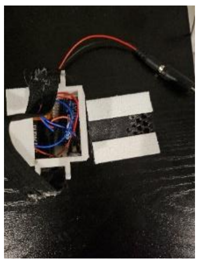
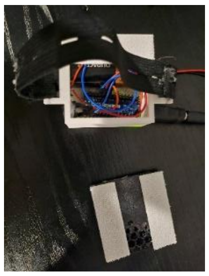
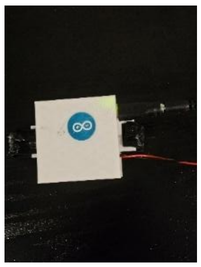
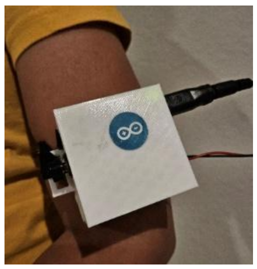


Hardware

The hardware for this model would have to be scalable, cost-effective, and efficient. The hardware would also have to be compact. The Arduino Nano RP2040 Connect would be the best choice for this project. The Arduino's onboard IMU system can capture data for classification and detection. The Arduino Nano RP2040 Connect also can run machine learning and TensorFlow Lite. The ability to run TensorFlow Lite and interpret the machine learning model allows the computation and results to have a high level of accuracy. This Arduino also has Wi-Fi functions for connectivity with different devices. Connection to the cloud through the Arduino can also send notifications in the case of a fall, making this Arduino Nano RP2040 Connect with the RP2040 microcontroller running the Arm Mbed OS a perfect choice.

The piezo buzzer and a push button are other components used in this project. The piezo could buzz during fall to notify nearby individuals that assistance is required. The button can either stop the buzzer or intercept the fall notification, stopping it from sending. This feature could be helpful in the case of a false prediction or a fall that did not injure or hurt the individual.

Connecting the hardware proves to be a relatively easy task. The Arduino will be placed into the housing so the connector goes outward to the designated connect port area. This next step is to connect a micro-USB to barrel jack converter to the Arduino. Add the 9-volt battery onto the other side of the Arduino. Lastly, we will connect the wiring to the buzzer and button. Connect the buzzer's negative to the Arduino's ground pin and the power pin (labeled with the plus sign) to digital pin number 2. The button should be connected to 3.3 volts and digital pin number 3 on the Arduino Board. These components can be added to their designated spots. The Arduino can be held in place simply using a small double-sided tape strip. The housing can be closed by attaching Velcro straps to the lid, and an elastic band can be added through the loop holder for the band to hold onto the arm of the user. The build should look something like the image below.

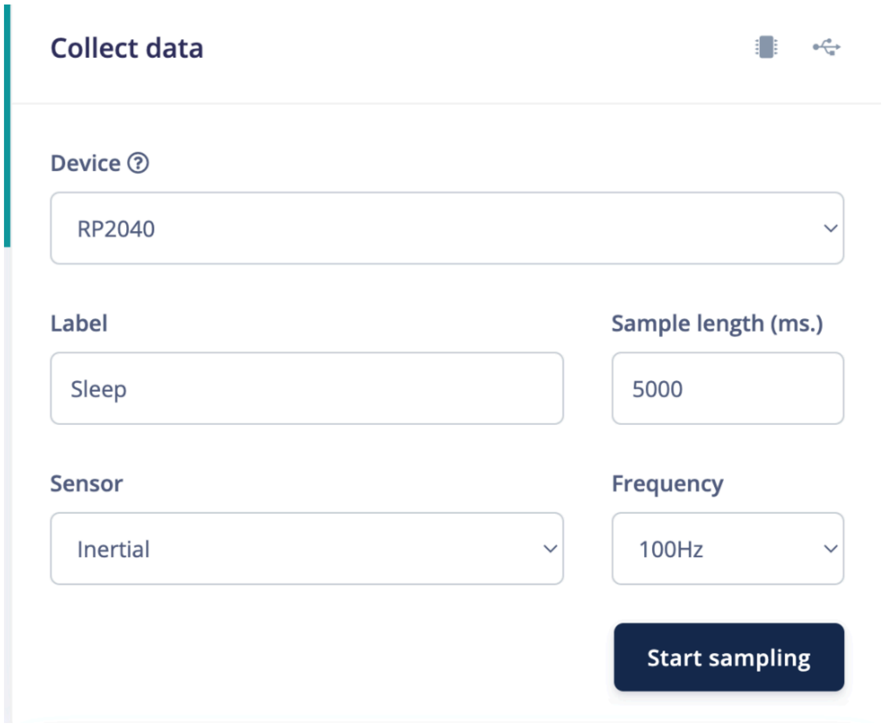





Machine Learning Tool

The device would have the neural network run using Arduino ML Tools, interfacing with Edge Impulse. Based on TensorFlow Lite, Edge Impulse allows the Arduino to run the model (Hymel et al., 2023).

Curating Samples for Training

The Arduino Nano RP2040 will be connected to the computer by pasting *edge-impulse-data-forwarder* into the terminal (*Data Forwarder*, n.d.). More about the serial forwarder can be found at https://docs.edgeimpulse.com/docs/tools/edge-impulse-cli/cli-data-forwarder. Alternatively, you can connect the board to the computer and paste *edge-impulse-daemon* into the terminal (*Serial Daemon*, n.d.). More about this can be found at https://docs.edgeimpulse.com/docs/tools/edge-impulse-cli/cli-daemon. Once this process is done, the Arduino can collect data. The band can be worn under the shoulder, and data can be collected. Each sample's size will be 5000 milliseconds, and the frequency will be 100Hz. The data will be received from the gyroscope and accelerometer. The class of the data collected will also be specified while collecting the data needed. The setup of the data should look like the image below.


Collect data

Device: RP2040

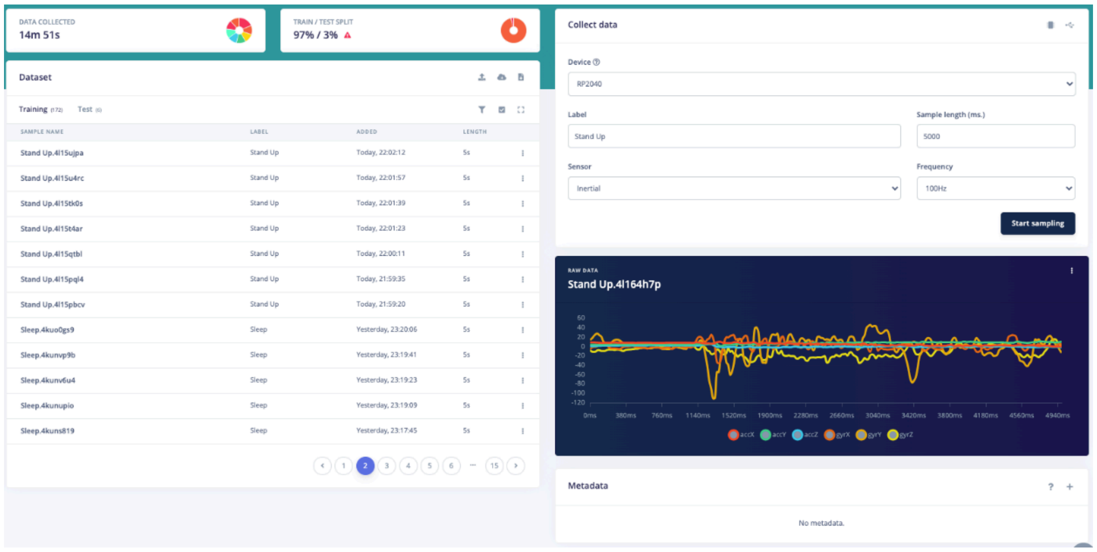
Label: Sleep

Sample length (ms.): 5000

Sensor: Inertial

Frequency: 100Hz

One hundred seventy-two samples were taken for the categories fall, nothing, walk, stairs up, stairs down, stand up, sleep, and sit. While training the fall section, multiple types of falls were trained on. Side falls, front and back falls, and bed falls. After visualizing the data from the raw data visualization, almost identical data was seen for all of them, and they were all placed in a single category. This would make the model smaller and more accurate in the real world. The visualization of the data should look like the image below.



DATA COLLECTED
14m 51s

TRAIN / TEST SPLIT
97% / 3%

Dataset

Training (172) Test (6)

SAMPLE NAME	LABEL	ADDED	LENGTH
Stand Up.4l15ujpa	Stand Up	Today, 22:02:12	5s
Stand Up.4l15u4rc	Stand Up	Today, 22:01:57	5s
Stand Up.4l15tk0s	Stand Up	Today, 22:01:39	5s
Stand Up.4l15t4ar	Stand Up	Today, 22:01:23	5s
Stand Up.4l15qtbl	Stand Up	Today, 22:00:11	5s
Stand Up.4l15pql4	Stand Up	Today, 21:59:35	5s
Stand Up.4l15pbcv	Stand Up	Today, 21:59:20	5s
Sleep.4kuo0gs9	Sleep	Yesterday, 23:20:06	5s
Sleep.4kunvp9b	Sleep	Yesterday, 23:19:41	5s
Sleep.4kunv6u4	Sleep	Yesterday, 23:19:23	5s
Sleep.4kunupio	Sleep	Yesterday, 23:19:09	5s
Sleep.4kuns819	Sleep	Yesterday, 23:17:45	5s

Collect data

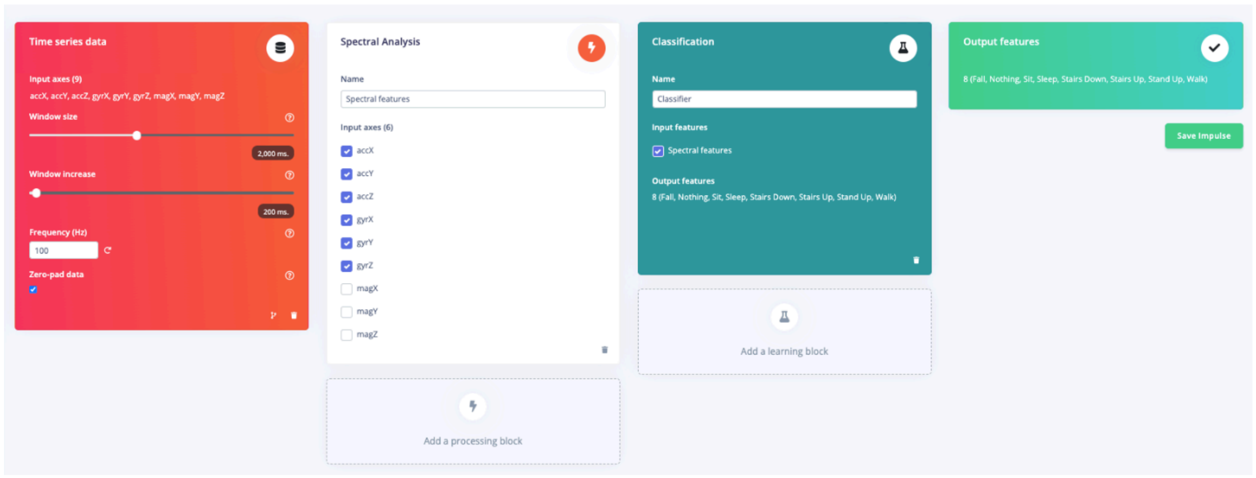
Device: RP2040
Label: Stand Up
Sample length (ms.): 5000
Sensor: Inertial
Frequency: 100Hz

Metadata

No metadata.

Creating Impulse

The next step of this project is to create an impulse. The impulse takes the raw data feature, extracts it, and classifies it. The data was taken over a series of times, making us use the *Time Series Data* block with a window size of 2000ms and window increase of 200ms. Due to the data samples being at 100Hz, the frequency will be the same. The data will also be zero-padded. The next block will be *Spectral Analysis* because this will allow us to tailor the impulse for accelerometer and gyroscopic data. The next step is to select the input axes we want to use. We will use *AccX, AccY, AccZ*, *GyrX*, *GyrY*, and *GyrZ*. This will allow us to use the accelerometer and gyroscope for making the prediction. Now, we must add the *Classifier* block. This will enable us to classify the data. We will use the *Spectral Features* as our input, with eight output features. The impulse should look like the image below.



Time series data

Input axes (9)
accX, accY, accZ, gyrX, gyrY, gyrZ, magX, magY, magZ
Window size: 2,000 ms.
Window increase: 200 ms.
Frequency (Hz): 100
Zero-pad data: ✓

Spectral Analysis

Name: Spectral features
Input axes (6)
✓ accX
✓ accY
✓ accZ
✓ gyrX
✓ gyrY
✓ gyrZ
☐ magX
☐ magY
☐ magZ

Classification

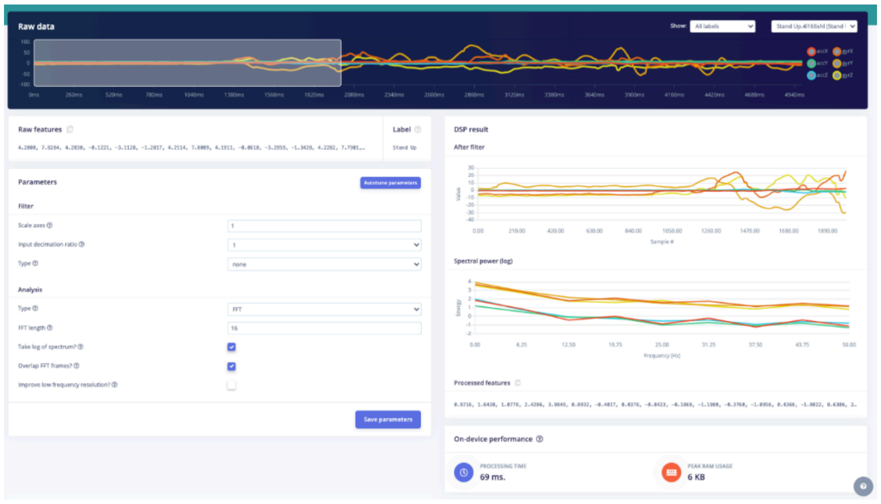
Name: Classifier
Input features
✓ Spectral features
Output features
8 (Fall, Nothing, Sit, Sleep, Stairs Down, Stairs Up, Stand Up, Walk)

Output features

8 (Fall, Nothing, Sit, Sleep, Stairs Down, Stairs Up, Stand Up, Walk)

Viewing Data

The data can be visualized in the *Spectral Features* tab. This will allow us to view the data for the features and apply filters to the data. We can adjust the parameters in the data to app filters and more. The *Scale Axes* and the *Input Decimation Ratio* will be set to 1. The type specifies the type of filter we want to use. We will not be using a filter. Moving on to the analysis of the parameters, the *Type* can be set to *FFT*, and the *FFT Length* will be set to 16. We will want to take the log of the spectrum and overlap *FFT Frames*, so we need to check those boxes. The image below should show what the *Spectral Features* tab and the parameters should look like.



Raw features
4.2000, 7.8284, 4.2030, -0.1221, -3.1128, -1.2817, 4.2114, 7.8009, 4.1911, -0.0610, -3.2959, -1.3428, 4.2282, 7.7901,…

Label
Stand Up

Parameters

Filter

Scale axes	1
Input decimation ratio	1
Type	none

Analysis

Type	FFT
FFT length	16
Take log of spectrum?	✓
Overlap FFT frames?	✓
Improve low frequency resolution?	

DSP result

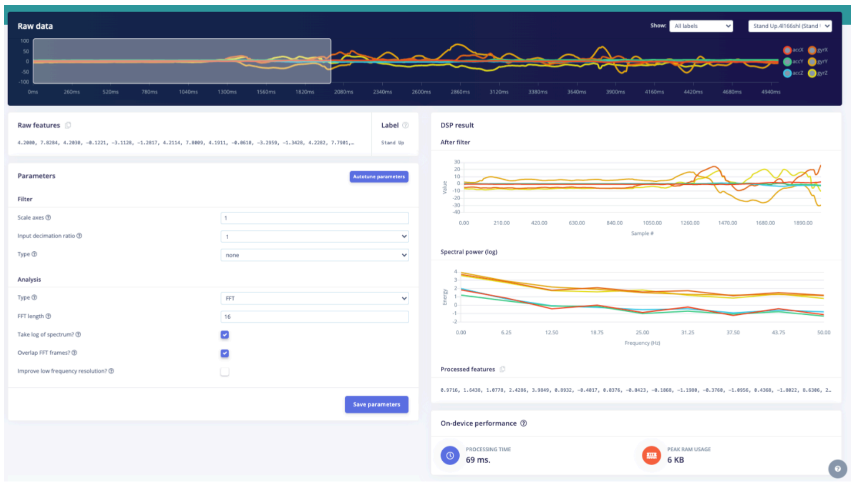
Processed features
0.9716, 1.6438, 1.0778, 2.4286, 3.9849, 0.8932, -0.4017, 0.0376, -0.8423, -0.1868, -1.1980, -0.3760, -1.0956, 0.4368, -1.0022, 0.6386, 2…

On-device performance

PROCESSING TIME 69 ms.

PEAK RAM USAGE 6 KB

After we save the parameters, the screen will move to the *Generate Features* tab. Select *Calculate Feature Importance* to view the features' importance for each prediction. Now, you must select *Generate Features* to start the process. The image below documents how the screen should look after generating the features.


Raw features
4.2000, 7.8284, 4.2030, -0.1221, -3.1128, -1.2817, 4.2114, 7.8009, 4.1911, -0.0610, -3.2959, -1.3428, 4.2282, 7.7901,…

Label
Stand Up

Parameters

Filter
Scale axes 1
Input decimation ratio 1
Type none

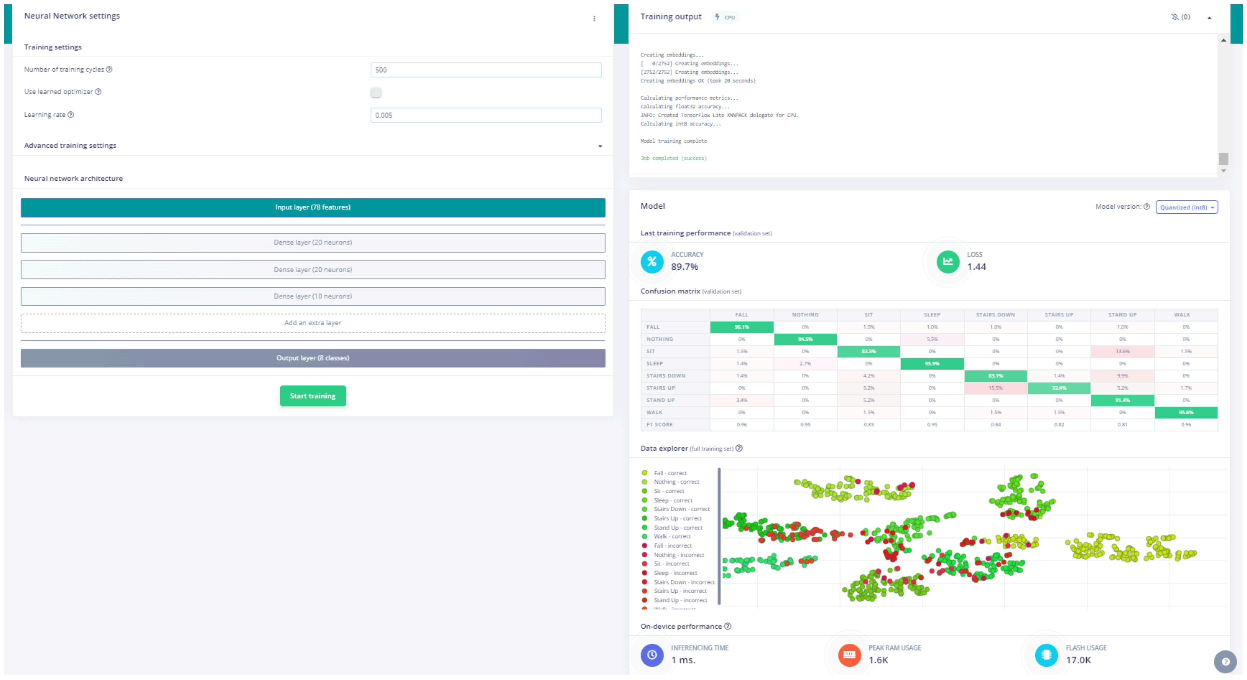
Analysis
Type FFT
FFT length 16
Take log of spectrum? ✓
Overlap FFT frames? ✓
Improve low frequency resolution?

Processed features
0.9716, 1.6438, 1.0778, 2.4286, 3.9849, 0.8932, -0.4017, 0.0376, -0.8423, -0.1868, -1.1980, -0.3760, -1.0956, 0.4368, -1.0022, 0.6386, 2…

On-device performance
PROCESSING TIME 69 ms.
PEAK RAM USAGE 6 KB

Building the Neural Network

We must navigate to the *Classifier* tab to adjust the neural network settings. We will adjust the number of epochs or the number of training cycles to 500. This will make the classifier run the training data 500 times through it to train the model. Next, set the learning rate of the model to 0.005. We should now adjust the dense layers of the network. This will allow the model to adjust the weights by the set learning rate and continue to adjust until we reach convergence. Select *Add an Extra Layer* and then select the *Dense Layer* option. The first and second layers should consist of 20 neurons each. The third layer should consist of 10 neurons. The *Starting Training* button can be clicked, and the process will start. This could take some time as the model is trained. The result should look something like the image below. The accuracies may vary as the data is not the same. If you want to recreate the exact values, you can get the data set at https://mltools.arduino.cc/public/330362/live.


Neural Network settings

Training settings

Number of training cycles: 500

Use learned optimizer: ☐

Learning rate: 0.005

Advanced training settings

Neural network architecture

Input layer (78 features)

Dense layer (20 neurons)

Dense layer (20 neurons)

Dense layer (10 neurons)

Add an extra layer

Output layer (8 classes)

Start training

Training output CPU

```
Creating embeddings...
[    0/2752] Creating embeddings...
[2752/2752] Creating embeddings...
Creating embeddings OK (took 20 seconds)

Calculating performance metrics...
Calculating float32 accuracy...
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
Calculating int8 accuracy...

Model training complete

Job completed (success)
```

Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 89.7%

LOSS 1.44

Confusion matrix (validation set)

	FALL	NOTHING	SIT	SLEEP	STAIRS DOWN	STAIRS UP	STAND UP	WALK
FALL	96.1%	0%	1.0%	1.0%	1.0%	0%	1.0%	0%
NOTHING	0%	94.5%	0%	5.5%	0%	0%	0%	0%
SIT	1.5%	0%	83.3%	0%	0%	0%	13.6%	1.5%
SLEEP	1.4%	2.7%	0%	95.9%	0%	0%	0%	0%
STAIRS DOWN	1.4%	0%	4.2%	0%	83.1%	1.4%	9.9%	0%
STAIRS UP	0%	0%	5.2%	0%	15.5%	72.4%	5.2%	1.7%
STAND UP	3.4%	0%	5.2%	0%	0%	0%	91.4%	0%
WALK	0%	0%	1.5%	0%	1.5%	1.5%	0%	95.6%
F1 SCORE	0.96	0.95	0.83	0.95	0.84	0.82	0.81	0.96

Data explorer (full training set)

On-device performance

INFERENCING TIME 1 ms.

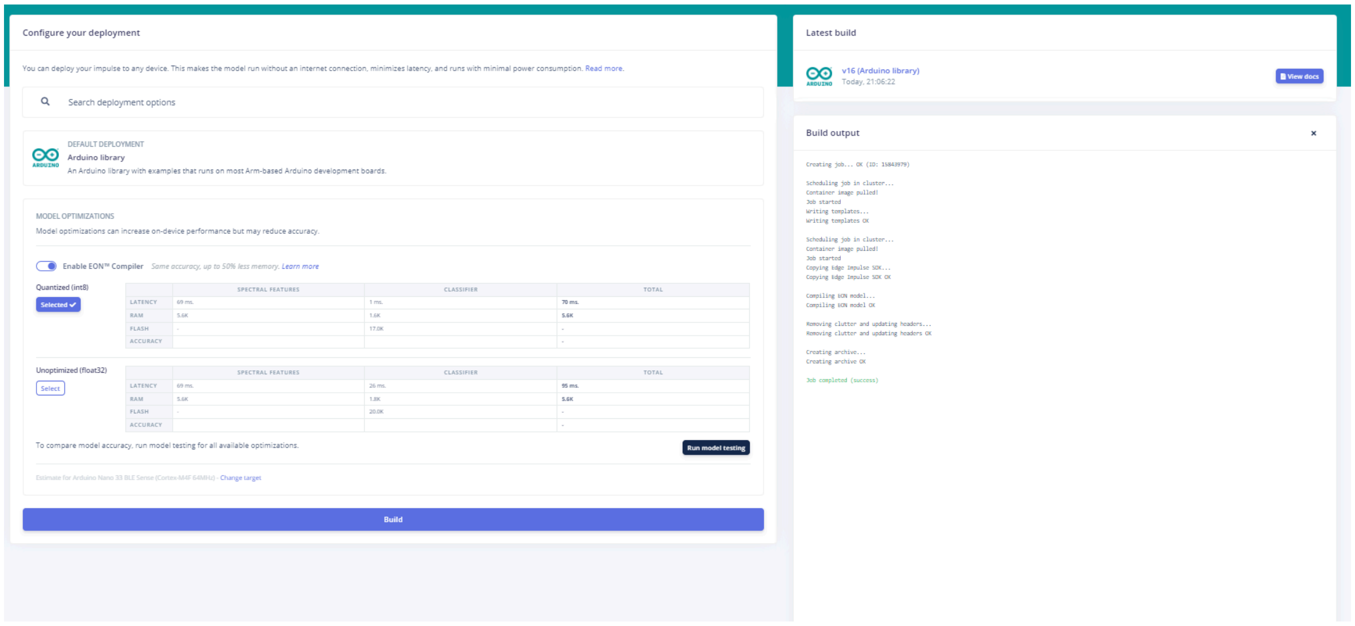
PEAK RAM USAGE 1.6K

FLASH USAGE 17.0K

After finishing the training process, we can see our results. Our overall accuracy is 89.7%, and the accuracy for detecting falls is 96.1%! This allows us to accurately classify whether the user has had a fall or not.

Deployment

The next step is to deploy the model onto the Arduino. The Edge Impulse model can be cloned at https://mltools.arduino.cc/public/330362/live. Navigate to the *Deployment* tab. There, you can select your deployment method to be the Arduino library. This will create a library you can add to your Arduino IDE to interface with the model. We will enable the *EON Compiler* as it can help lower the model's size. Now you can click build model. It should look something like the image below.



Configure your deployment

You can deploy your impulse to any device. This makes the model run without an internet connection, minimizes latency, and runs with minimal power consumption. Read more.

Search deployment options

DEFAULT DEPLOYMENT
Arduino library
An Arduino library with examples that runs on most Arm-based Arduino development boards.

MODEL OPTIMIZATIONS
Model optimizations can increase on-device performance but may reduce accuracy.

Enable EON™ Compiler *Same accuracy, up to 50% less memory. Learn more*

Quantized (int8) — Selected ✓

	SPECTRAL FEATURES	CLASSIFIER	TOTAL
LATENCY	69 ms.	1 ms.	**70 ms.**
RAM	5.6K	1.6K	**5.6K**
FLASH	-	17.0K	-
ACCURACY			-

Unoptimized (float32) — Select

	SPECTRAL FEATURES	CLASSIFIER	TOTAL
LATENCY	69 ms.	26 ms.	**95 ms.**
RAM	5.6K	1.8K	**5.6K**
FLASH	-	20.0K	-
ACCURACY			-

To compare model accuracy, run model testing for all available optimizations.

Estimate for Arduino Nano 33 BLE Sense (Cortex-M4F 64MHz) - Change target

Latest build
v16 (Arduino library)
Today, 21:06:22

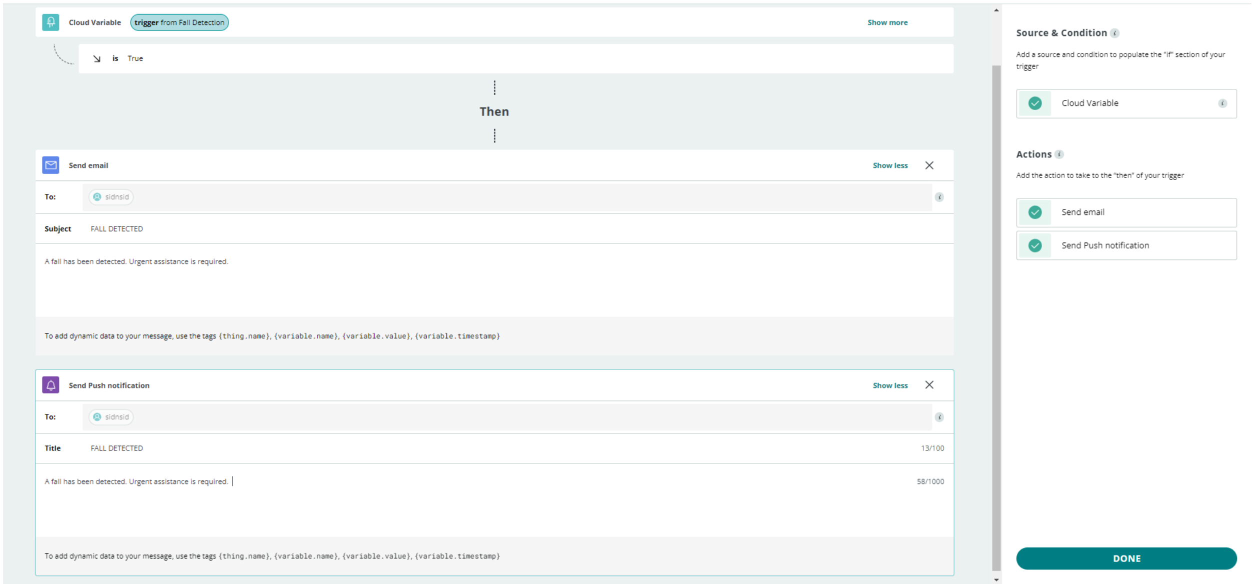
Build output

```
Creating job... OK (ID: 15843979)

Scheduling job in cluster...
Container image pulled!
Job started
Writing templates...
Writing templates OK

Scheduling job in cluster...
Container image pulled!
Job started
Copying Edge Impulse SDK...
Copying Edge Impulse SDK OK

Compiling EON model...
Compiling EON model OK

Removing clutter and updating headers...
Removing clutter and updating headers OK

Creating archive...
Creating archive OK

Job completed (success)
```

A file now will download, which contains the library holding the model. Now, we will deploy the model onto the device through the cloud. Navigate to the Arduino cloud: https://app.arduino.cc/. The Arduino Maker Plan for the cloud must be purchased for the notification triggers.

Click on the *Things* tab and add a thing (indicated with + *Thing*) to connect and code the device. Now, click on the linking icon, which says*, Select Device*. Now, you must download Arduino Create Agent and follow the steps to connect the board to your computer. The board should be plugged into the computer, and it should detect it and allow you to connect. Now click on the linking icon that says, *Configure*. Selecting this will enable you to connect the board to Wi-Fi to send the SMS notifications. The Wi-Fi can be set to the person's phone's hotspot if they disconnect from their home Wi-Fi. Now, you will create the cloud variable. This variable will change the send notifications when the fall is detected. Select the *Add* button to create the variable. It would be best to name the variable trigger and set the variable type. To do this, select *Basic Variable* and then *Boolean* from the drop-down menu once you try to choose the variable type. Still inside the *Thing* section, move to the next tab, the sketch. Click *Open Full Editor* at the top of the screen. This will take you to the main editor with more functionality available. Click on the *Import* button at the top left of the screen and upload the zip file containing the model. This will download the library with the model onto the code. The C++ code can be found at https://github.com/siddarthnandy/FallDetection. This code can be copied and pasted into the main editor. This code will run the interpretation of the model's prediction, send the notifications, and ring the buzzer. It will also allow the button to be pressed to intercept the message and stop the buzzer.

The next step is to create a mechanism to send the user the notifications. Navigate to the homepage of the Arduino Cloud. From here, navigate to the *Triggers* tab. This is where we will set up what the cloud will do when the state of the trigger during the event that falls is detected. Select *Create Trigger* and *Cloud Variable*, then the sketch and the cloud variable you made inside it. Now click *Link Variable*. Click on the *Send Email* and the *Send Push Notification* options. This states what function the cloud must perform when the trigger variable is too high when a fall is detected. Type what you want to send in the email and push a notification when the system has detected the fall. Now select the *Done* button at the bottom of the screen. The system setup should look something like the image below.



To: sidnsid

Subject FALL DETECTED

A fall has been detected. Urgent assistance is required.

To add dynamic data to your message, use the tags {thing.name}, {variable.name}, {variable.value}, {variable.timestamp}

To: sidnsid

Title FALL DETECTED 13/100

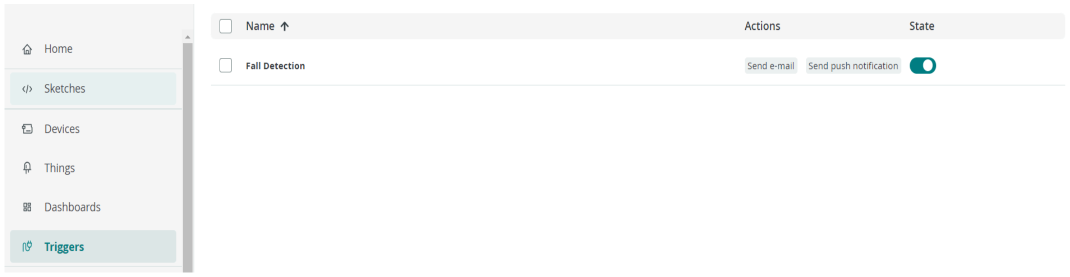
A fall has been detected. Urgent assistance is required. 58/1000

To add dynamic data to your message, use the tags {thing.name}, {variable.name}, {variable.value}, {variable.timestamp}

Add a source and condition to populate the "if" section of your trigger

Add the action to take to the "then" of your trigger

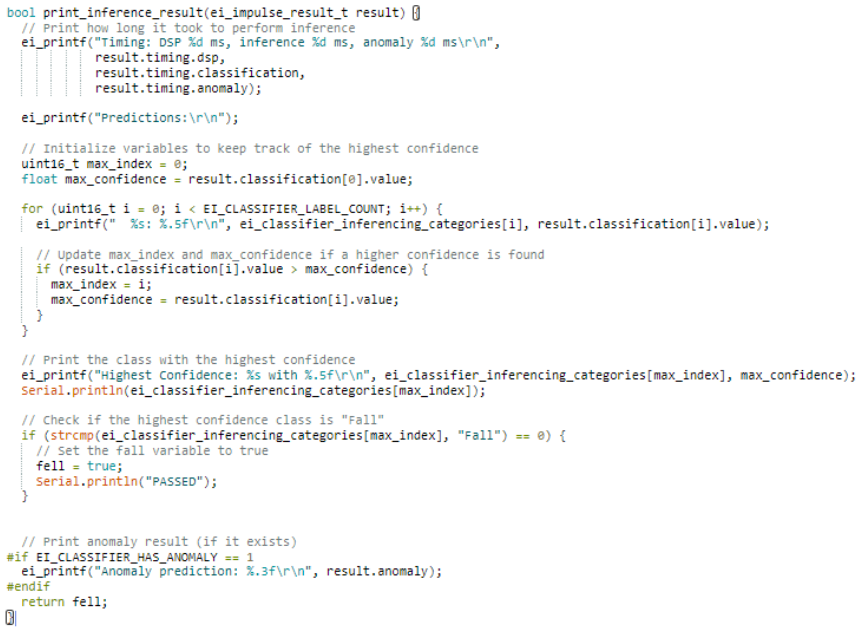
The email will be sent to the email address that your Arduino Account is linked to. The Arduino IOT Remote must be downloaded on both the armband recipient and the person who can help them and come to assist during a fall. Both devices should be logged onto the same account in the app that the Arduino Cloud was logged into. This will send the notifications to the devices. Now navigate back to the home page and then the *Triggers* page and set the state of the trigger to be on.



Now that the steps for deployment have been completed, we can look at how the code functions. The code is a fusion between the code Edge Impulse provides for the fusion model for the Arduino Nano RP2040 and the code we provide when creating a *Thing* in Arduino. The code that has been modified are the functions and the conditional blocks that decide what will happen when the fall has been detected. The standalone Edge Impulse code only provides the running of the model's inference and printing it to the Serial Monitor. The code taken from the Arduino is only the other files that we did not edit, which contain the cloud variable and the username and password to the Wi-Fi network that the Arduino has been connected to.

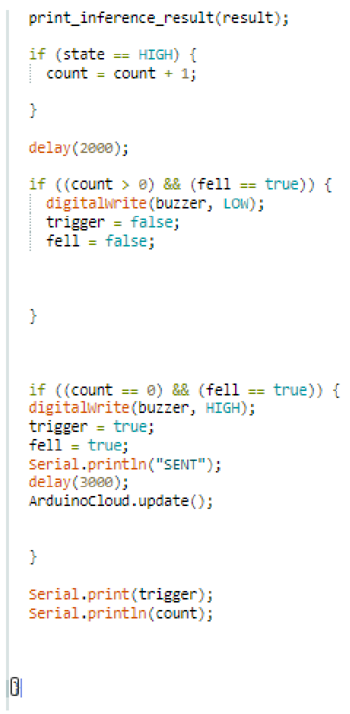
The first edit has been made to the *print_inference_result* function. We will change it to a boolean function, returning the value of the fell variable as either true or false. This edit allows the function to return the actual value if the inference prediction was a fall.

This is a snippet picture of the changed part of the code.



```
bool print_inference_result(ei_impulse_result_t result) {
  // Print how long it took to perform inference
  ei_printf("Timing: DSP %d ms, inference %d ms, anomaly %d ms\r\n",
            result.timing.dsp,
            result.timing.classification,
            result.timing.anomaly);

  ei_printf("Predictions:\r\n");

  // Initialize variables to keep track of the highest confidence
  uint16_t max_index = 0;
  float max_confidence = result.classification[0].value;

  for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
    ei_printf("  %s: %.5f\r\n", ei_classifier_inferencing_categories[i], result.classification[i].value);

    // Update max_index and max_confidence if a higher confidence is found
    if (result.classification[i].value > max_confidence) {
      max_index = i;
      max_confidence = result.classification[i].value;
    }
  }

  // Print the class with the highest confidence
  ei_printf("Highest Confidence: %s with %.5f\r\n", ei_classifier_inferencing_categories[max_index], max_confidence);
  Serial.println(ei_classifier_inferencing_categories[max_index]);

  // Check if the highest confidence class is "Fall"
  if (strcmp(ei_classifier_inferencing_categories[max_index], "Fall") == 0) {
    // Set the fall variable to true
    fell = true;
    Serial.println("PASSED");
  }


  // Print anomaly result (if it exists)
#if EI_CLASSIFIER_HAS_ANOMALY == 1
  ei_printf("Anomaly prediction: %.3f\r\n", result.anomaly);
#endif
  return fell;
}
```

The next edit made to the program was the if statements that automate sending the trigger for notification and the buzzer. Now, we will create a statement that will save the number of times the button has been pressed to know whether the user did not have a fatal fall or injury and if immediate help is needed. The next step is to create a statement that will send a trigger for the notification, turn on the buzzer, and update the Arduino cloud to send the notification instantly. A snippet picture of the edited code has been provided below.

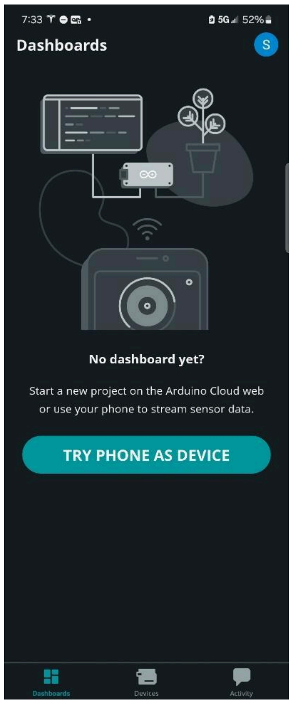


```
print_inference_result(result);

if (state == HIGH) {
  count = count + 1;

}

delay(2000);

if ((count > 0) && (fell == true)) {
  digitalWrite(buzzer, LOW);
  trigger = false;
  fell = false;


}


if ((count == 0) && (fell == true)) {
digitalWrite(buzzer, HIGH);
trigger = true;
fell = true;
Serial.println("SENT");
delay(3000);
ArduinoCloud.update();


}

Serial.print(trigger);
Serial.println(count);


}
```

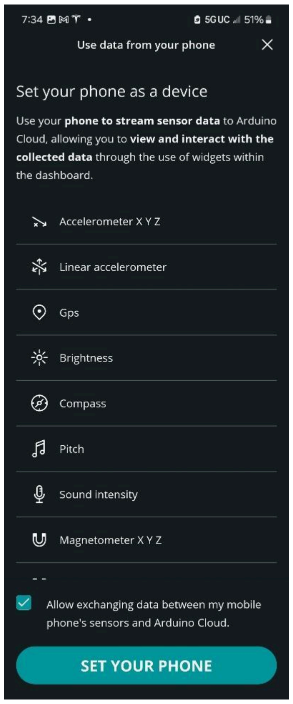
The only other changes to the code were defining and redefining the new variables and updating the Arduino Cloud every time the code ran. Through these changes, the code can be modified to work in detected falls, turning on the buzzer, detecting if the button has been pressed, indicating that the user is exemplary, and sending emails and notifications during the event where the fall has been detected. No action was taken from the user to intercept the call for help.

Location Tracking

The last step is to track the recipient's location using their phone in case of a fall. Opening the Arduino IOT Remote on the recipient's device, select the *Try Phone as Device* button. Allow the app to track data from the phone and click *Set Your Phone*. This will automatically build the connections and variables for the phone's sensors and create widgets for you to view the recipient's data remotely, including the location. Now select *Try Out* to activate.



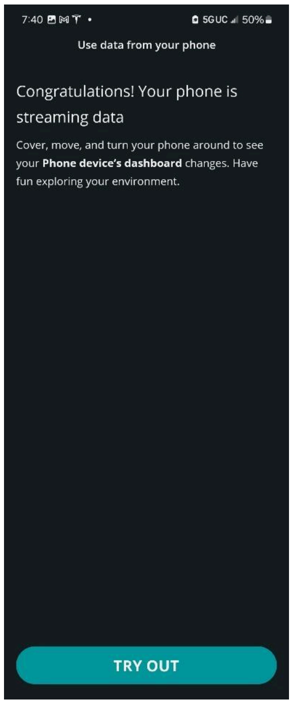
Start a new project on the Arduino Cloud web or use your phone to stream sensor data.



Set your phone as a device

Use your **phone to stream sensor data** to Arduino Cloud, allowing you to **view and interact with the collected data** through the use of widgets within the dashboard.

✓ Allow exchanging data between my mobile phone's sensors and Arduino Cloud.



Congratulations! Your phone is streaming data

Cover, move, and turn your phone around to see your **Phone device's dashboard** changes. Have fun exploring your environment.

Results and Testing

The device we tested proved to be very accurate in detecting falls. During the training step, we achieved a 96.1% accuracy in detecting falls. The device correctly detected 98 out of 102 falls during this training process! After live testing, the fall detection band could classify all ten falls correctly. When we tested the device for falling from a bed, it correctly classified all ten tests. This test helped us understand the accuracy of our model. Compared to competitors such as the Apple Watch lineup, which could not correctly classify falls out of eleven, the fall detection band is a better option (*Should You Use Apple Watch for Fall Detection in 2023?*, n.d.).

Conclusion

Through these steps, an accurate fall detection system can be achieved. Alerts and signals can be sent during the event of a fall, saving people all around the globe. This innovation can help to seek urgent assistance and is a crucial step in shaping a bright future for this overlooked danger in citizens worldwide. Through this invention, people can have peace of mind knowing the systems engaged.

Through the power of technology and artificial intelligence, problems such as this can be solved. With constant development in the technological field, many challenges can be found solutions. With breakthroughs in the machine learning space on embedded systems, new doors can be opened to change and shape the world for the better.

Bibliography

[1] Data forwarder. (n.d.). Retrieved January 31, 2024, from https://docs.edgeimpulse.com/docs/tools/edge-impulse-cli/cli-data-forwarder

[2] Hymel, S., Banbury, C., Situnayake, D., Elium, A., Ward, C., Kelcey, M., Baaijens, M., Majchrzycki, M., Plunkett, J., Tischler, D., Grande, A., Moreau, L., Maslov, D., Beavis, A., Jongboom, J., & Reddi, V. J. (2023). Edge Impulse: An MLOps Platform for Tiny Machine Learning.

[3] Serial daemon. (n.d.). Retrieved January 31, 2024, from https://docs.edgeimpulse.com/docs/tools/edge-impulse-cli/cli-daemonh

[4] Should You Use Apple Watch for Fall Detection in 2023? Experts Weigh In. (n.d.). NCOA Adviser. Retrieved February 10, 2024, from https://www.ncoa.org/adviser/medical-alert-systems/apple-watch-medical-alert-review/

[5] Verma, S. K., Willetts, J. L., Corns, H. L., Marucci-Wellman, H. R., Lombardi, D. A., & Courtney, T. K. (2016). Falls and Fall-Related Injuries among Community-Dwelling Adults in the United States. PLOS ONE, 11(3), e0150939. https://doi.org/10.1371/journal.pone.0150939