PROJECT TITLE: MOTION DETECT – AI

MOTION DETECTION USING AI

A DEEPER INSIGHT INTO ML ON EMBEDDED SYSTEMS

Instructor:

Prof. Shawn Hymel

Student Name:

Muhammad Haris

ABSTRACT:

This project explores the integration of machine learning with embedded systems to enhance understanding of both fields. The focus is on motion detection using neural networks deployed on the Arduino Nano BLE Sense 33, leveraging Edge Impulse's user-friendly software. This implementation not only demonstrates practical applications of machine learning but also highlights the process of deploying AI models on constrained hardware. The project encapsulates key learnings, including data collection, model training, and evaluation, providing a comprehensive overview of how machine learning can be effectively utilized in embedded systems.

ACKNOWLEDGEMENT:

I cannot express enough thanks to my instructor Prof Shawn Hymel for his support through the course “Introduction to Embedded Machine Learning” on Coursera. I offer my sincere appreciation for such learning opportunities provided to students by Coursera.org.

TABLE OF CONTENTS:

LINK TO THE COURSE: coursera.org/learn/introduction-to-embedded-machine-learning/lecture/iIpqG/welcome-to-the-course

UNDERSTANDING THE BASICS:

What is ‘Data Science’ and ‘Artificial Intelligence’?

Data Science is the use of mathematical and statistical methods, and specialized programming/algorithms that help to uncover actionable insights in a specific form of data to thus make better decisions and predictions.

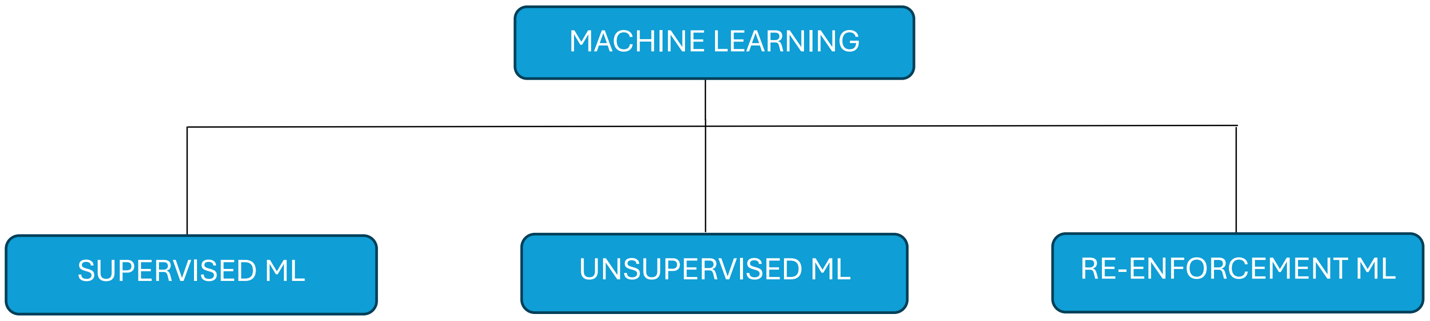
Artificial Intelligence is a subset of data science. It is the simulation of human intelligence in machines that programmed to think and act like humans. Learning, reasoning, problem solving, perception, and language comprehension are all examples of cognitive abilities that are begin replicated into a machine.

What is ‘Machine Learning’ and ‘Deep Learning’?

Machine Learning is the sub-set of AI. It is the use of algorithms and statistical models to extract patterns from an input data, which helps to come up with rules and parameters to analyze and draw smart inferences from that data

Deep Learning is the sub-set of ML. It is the use of complex Machine learning models which learn patterns successively in layers.

Categories of ML models:



Supervised ML is the type which involves the use of labelled data for training. Labelled data refers that each training data paired with an output label. Labels guide the model during training, allowing it to learn the relationship between the input data (like images, text, or numerical features) and the correct output. Supervised ML finds a function that maps new/unseen input data to some output that could be a prediction or a classification.

In unsupervised ML, the data does not come with labels. The model tries to find patterns, structures, or groupings in the data on its own. For example, clustering algorithms try to group similar data points together without knowing what those groups should represent beforehand.

Summarizing above stated information as; labels in supervised learning are crucial because they serve as the "correct answers" that the model tries to learn from, while unsupervised learning works without them, focusing instead on identifying patterns and structures in the data.

Reinforcement learning is a type of machine learning where an agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties, with the goal of maximizing cumulative rewards over time. Reinforcement learning is like teaching a dog new tricks using treats.

- **Agent**: The dog (or a computer program) that needs to learn something.
- **Environment**: The space where the dog (or program) operates, like a room or a game.
- **State**: What's happening around the dog right now, like whether it's sitting or standing.
- **Action**: What the dog decides to do next, like sitting, jumping, or rolling over.
- **Reward**: The treat you give the dog when it does the right thing, like sitting when you say "sit."
- **Policy**: The dog's strategy for deciding what to do, based on whether it wants more treats.
- **Exploration vs. Exploitation**: Sometimes the dog tries new things to see if it gets a treat (exploration), and sometimes it does what it knows already gets treats (exploitation).

Over time, the dog (or computer program) learns what actions lead to treats (rewards) and gets better at doing the right things to earn those rewards.

What is embedded machine learning?

Embedded ML refers to the integration of machine learning algorithms and models directly into hardware devices, enabling these devices to perform intelligent tasks without relying on cloud computing or external servers.

This approach is especially useful for devices with limited computing power, memory and energy resources such as IoT devices, wearables and smartphones.

What is the difference between a deterministic and probabilistic system?

In a deterministic system, the outcome is entirely determined by the initial conditions and rules governing the system. If you know the starting point and the rules, you can precisely predict the future state of the system.

In a probabilistic system, the outcome is not fully determined by the initial conditions; instead, there is an element of randomness or uncertainty. The system can produce different outcomes, even with the same starting point.

What is neural network (NN)?

A neural network is a type of machine learning model that is inspired by the structure and function of a human brain. The fundamental unit of a neural network is a perceptron. The perceptron model was introduced in the 1950s; inspired from the biological neuron, it acts as a binary classifier. It takes multiple input values, multiplies them by weights, sums them, and passes the result through an activation function (often a step function) to produce a single output. It's the building block for more complex neural networks. It consists of layers of interconnected nodes, or “neurons” that work together to process data and make decisions or predictions.

Basic Structure of a Neural Network:

1. Input layer: This layer receives raw data or input features. Each neuron in this layer represents one feature of the data.
2. Hidden layer: These layers are immediately between the input and the output layers. They perform various transformations onto the input data by applying weights and biases. They are called ‘Hidden’ because they are not visible or exposed as outputs but crucial in processing the data.

3. Output layer: This layer produces the final output or prediction. The number of neurons in this layer corresponds to the number of outcomes or classes being solved.

What are the types of neural networks?

- **Feedforward Neural Networks (FNN):** The simplest type of NN, where the data moves in one direction >from input to output>without loops.
- **Convolutional Neural Network (CNN):** Specialized for image and video processing, with layers that detect features like edges and textures. Usually used for tasks like image classification and object detection.
- **Recurrent Neural Networks (RNN):** Designed for sequential data, such as time series or language, where each neuron can use its output as part of its input for the next step. It is used for time series forecasting and natural language processing.
- **Generative Adversarial Networks (GANs):** Generative Adversarial Networks (GANs) are a type of machine learning model designed to generate new, synthetic data that closely resembles real data. They consist of two neural networks, a **generator** and a **discriminator**, that are trained together in a process where they compete against each other. GANs are particularly popular for tasks like image generation, video synthesis, and even creating realistic audio or text.

What is an activation function?

They are used to introduce non-linearity into a NN that allows the network to model complex relationships between inputs and outputs. It also controls whether a neuron should be activated or not, by the following procedure:

- When a neuron processes input, it computes a weighted sum of the inputs.
- The activation function then processes this sum and determines if it meets a certain threshold.
- If the output passes the threshold (in functions like ReLU, sigmoid, or step functions), the neuron "activates" and passes the signal forward.
- If it doesn't pass the threshold, the neuron remains "inactive" and might output zero or a small value.

What are weights and biases?

Weight: These are the key parameters that multiply with the input values. They determine the importance of each input in making predictions. During training, the neural networks adjust the weights to minimize the error between the predicted and actual output.

Biases: These are the additional parameters added to the weighted sum of the inputs before applying the activation function. They allow the network to shift the activation function, helping it to fit the data better. Biases provide the model with more flexibility, enabling to learn pattern that don't pass through the origin.

What are dense and deep neural networks?

- Dense Neural Network: This refers to a type of neural network where each neuron in one layer is connected to every neuron in the next layer. These networks are also called fully connected networks or fully connected layers. The term "dense" specifically describes the connectivity pattern between layers.
- Deep Neural Network: This refers to a neural network with multiple layers between the input and output layers. "Deep" indicates that the network has a substantial depth, meaning it has several hidden layers. A deep neural network can consist of dense layers, but it can also include other types of layers like convolutional layers, recurrent layers, etc.

So, a dense neural network can be part of a deep neural network, but a deep neural network isn't necessarily just dense layers—it can have a combination of different types of layers.

What is back propagation in the NN training process?

Backpropagation is an algorithm used in neural networks to minimize the error in predictions by adjusting the weights and biases of the network. It works by propagating the error backward through the network from the output layer to the input layer.

- **Gradient Descent**: This is an optimization algorithm used to minimize the loss function by iteratively adjusting the weights and biases. During backpropagation, the gradients (derivatives of the loss with respect to each parameter) indicate the direction and magnitude of the weight updates.

- **Learning Rate**: This is a hyperparameter that controls the size of the weight updates. If the learning rate is too high, the network might overshoot the optimal values, leading to instability. If it's too low, the training process can be very slow. The learning rate determines how quickly or slowly the model learns by scaling the magnitude of the weight updates during gradient descent.

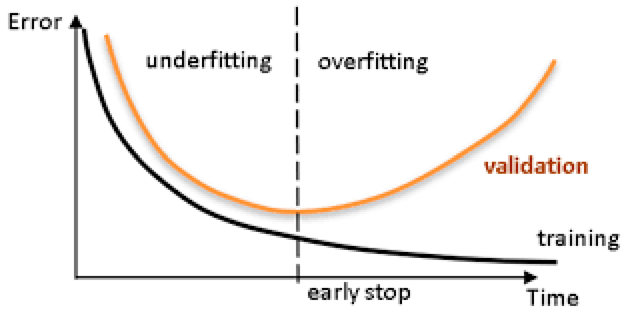
What is a confusion matrix?

A confusion matrix is a table used in machine learning to evaluate the performance of a classification model. It provides a summary of the prediction results on a classification problem, showing the count of correct and incorrect predictions, broken down by each class.

What is the under-fitting and over-fitting of a model?

Under-Fit: The model fails to capture/understand any trends/grouping of data.

Over-Fit: The model captures the training data trends but fails on unseen data.



What are the methods to fix under-fitting and over-fitting?

Under-Fit:

1. Get more data.
2. Try different features.
3. Training for a longer period.
4. Try a more complex model.

Over-Fit:

1. Get more data.
2. Early stopping.
3. Reduce Model Complexity.
4. Add regularization terms.

5. Add dropout layer.

Adding Dropout layers is a technique that randomly ignores the output of some nodes during the training, this allows the remaining nodes/neurons to take the responsibility of the fitting of data.

UNDERSTANDING EMBEDDED MACHINE LEARNING:

Feasibility of Embedded ML.

The feasibility of ML on embedded systems is increased over the years due to the following factors:

1. Newer low-powered models.
2. Optimized Frameworks.

Example: The MobileNetV2, a type of convolutional neural network (CNN) that is based on TensorFlow lite for the embedded system, which requires only 250kb of memory and 7 million MAC operations.

Role of optimized libraries in Embedded ML.

CMSIS-NN: is an optimized software library by ARM that supports TensorFlow lite and focuses on optimizing key compute-intensive operations.

However, while using these libraries, following two problems may needed to be addressed:

1. Cycle-Bound Problems:
   Problems related to the performance of an algorithm where the primary constraint is the number of clock-cycles required to execute a NN on a micro-controller.

2. Memory-Bound Problems:
   Refer to performance issues where the speed of execution is limited by the memory system rather than the processing power of the CPU. These problems occur when the processor must spend a significant amount of time waiting for data to be read from or written to memory, rather than performing computations.

STEPS INVOLVED IN EMBEDDED ML:

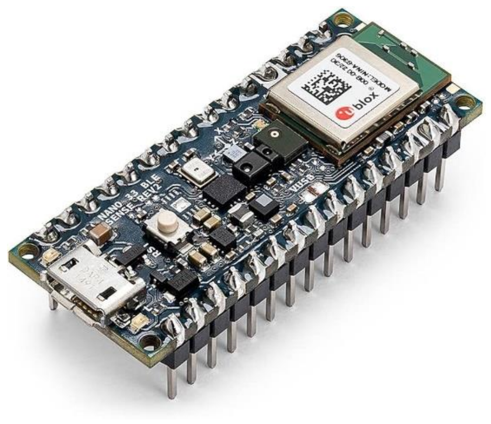
It is to be noted that all the steps are advised to be performed on the Edge Impulse site, as it provides a user friendly and easy to understand interface for project understanding and completion.

The detailed tutorial for greater understanding of the software and hardware use can be found through the course link mentioned at the start of this project file.

My project progress through edge impulse following the below mentioned steps have also been mentioned with relevant screen shots below:

STEP 1: **DATA COLLECTION**

Record the data needed for the relative NN using the Arduino Nano 33 BLE-sense or a smart phone (if containing required sensing devices) etc.



STEP 2: **DATA ACQUISATION | HOLDOUT METHOD**

1. **DATA SPLITTING PROCESS:**
   Random Shuffle: Randomly shuffle the dataset.
   Training Set: Set aside most of the data (60-80%) for training.
   Validation Set: Set aside the remaining data (10-20%) for validation.
   Test Set: Reserve (10-20%) for testing which should remain untouched.

2. **TRAINING AND VALIDATION:**
   Training: Use the training set to update the model parameters, setting it accurately as required.
   Validation: Evaluate the model performance on the validation set. Adjust hyper-parameters if needed and retrain.
   Hyper-Para-metering: Tweak the model size, shape, training set and learning rates.

3. **FIND TESTING:**
   Use the final untouched test set to test the model for the final performance.

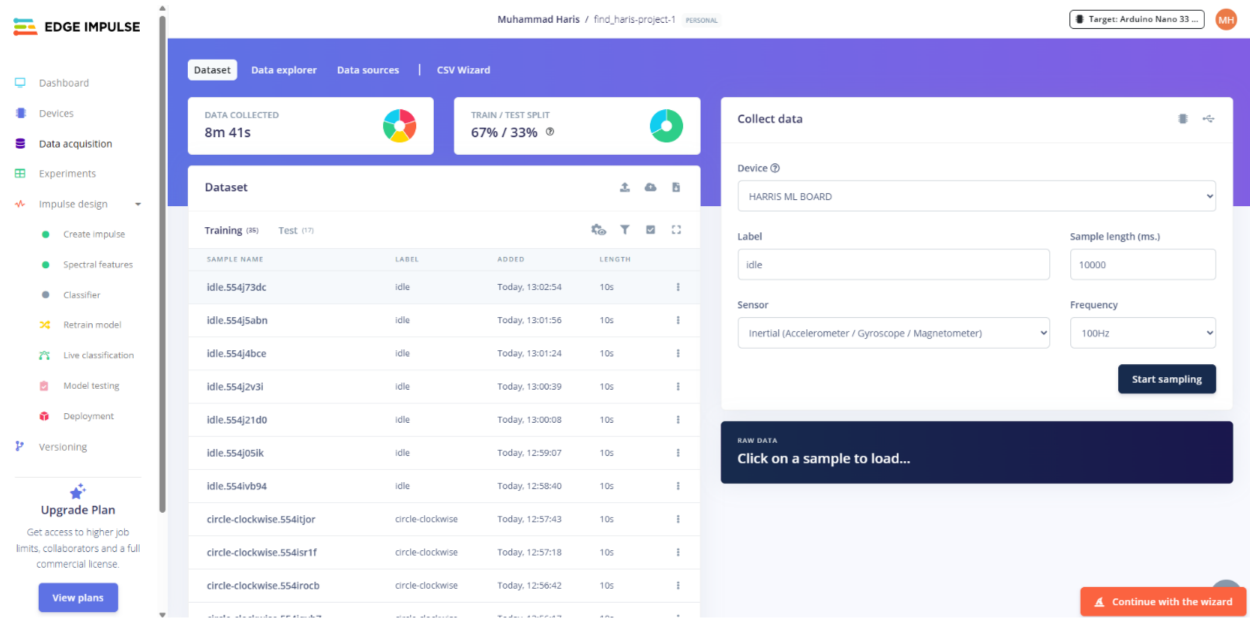
NOTE:

Ensure that the training data set represents real world applications. Example: training a dog classifier requires diverse dog images, not just a specific breed or background.

4. **IMBALANCED DATA:**
   Balancing: is aiming for equal number of samples per class.
   An imbalanced data can lead to a naiver classifier which will always predict the majority class.

NOTE:

If balancing is difficult, consider using an alternative method like anomaly detection. Anomaly detection is the process of identifying data points, patterns, or events that deviate significantly from much of the data. These "anomalies" are rare or unusual instances that do not confirm to expected behavior.



DATA COLLECTED
8m 41s

TRAIN / TEST SPLIT
67% / 33%

Dataset

Training (35) Test (17)

SAMPLE NAME	LABEL	ADDED	LENGTH
idle.554j73dc	idle	Today, 13:02:54	10s
idle.554j5abn	idle	Today, 13:01:56	10s
idle.554j4bce	idle	Today, 13:01:24	10s
idle.554j2v3i	idle	Today, 13:00:39	10s
idle.554j21d0	idle	Today, 13:00:08	10s
idle.554j05ik	idle	Today, 12:59:07	10s
idle.554ivb94	idle	Today, 12:58:40	10s
circle-clockwise.554itjor	circle-clockwise	Today, 12:57:43	10s
circle-clockwise.554isr1f	circle-clockwise	Today, 12:57:18	10s
circle-clockwise.554irocb	circle-clockwise	Today, 12:56:42	10s

Collect data

Device: HARRIS ML BOARD

Label: idle

Sample length (ms.): 10000

Sensor: Inertial (Accelerometer / Gyroscope / Magnetometer)

Frequency: 100Hz

RAW DATA
Click on a sample to load...

STEP 3: **FEATURE EXTRACTION**

A feature is an individual measurable property or characteristic of a phenomenon being observed. It is crucial for training and inference in machine learning.

DEEP NEURAL NETWORKS:

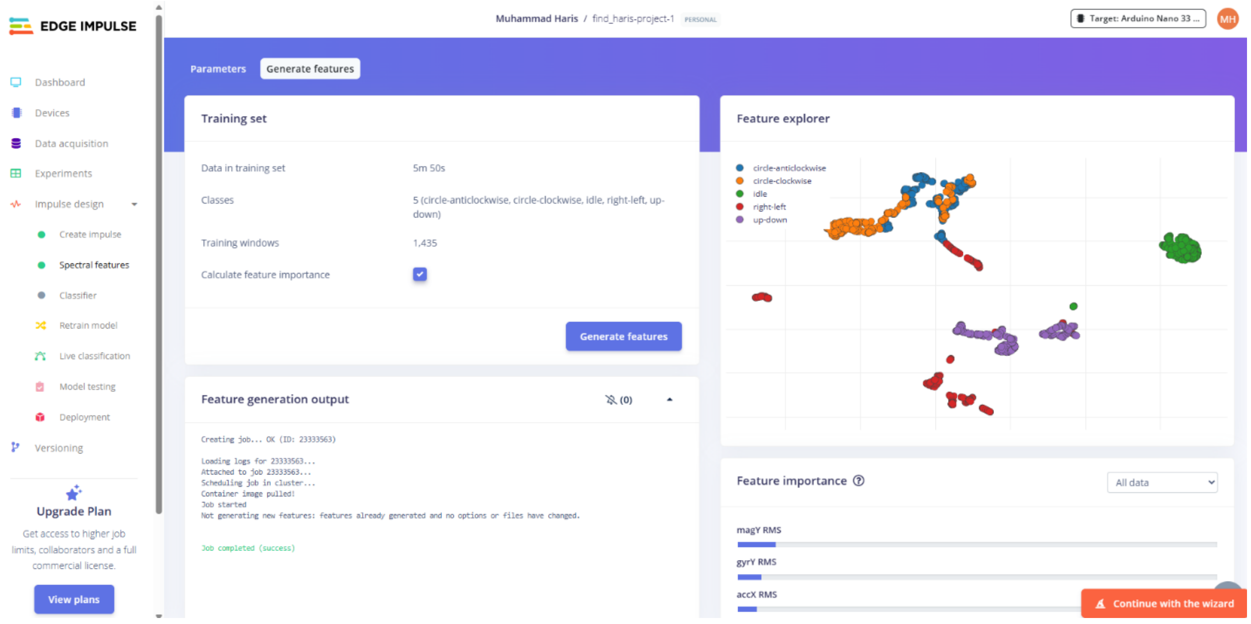
Deep neural networks (DNN) can automatically extract and learn features from raw data. However, there are two problems related to this approach:

1. Analyzing a large data set for feature extraction will require large computational power.
2. DNN requires a lot of data for its action.

NOTE:

These limitations are not feasible for embedded systems.

Instead, here are some strategies that can be employed to address these limitations:
1. Use Pre-trained Models and Transfer Learning.
2. Use Lightweight Neural Network Architectures (etc.)



Training set

Data in training set	5m 50s
Classes	5 (circle-anticlockwise, circle-clockwise, idle, right-left, up-down)
Training windows	1,435
Calculate feature importance	✓

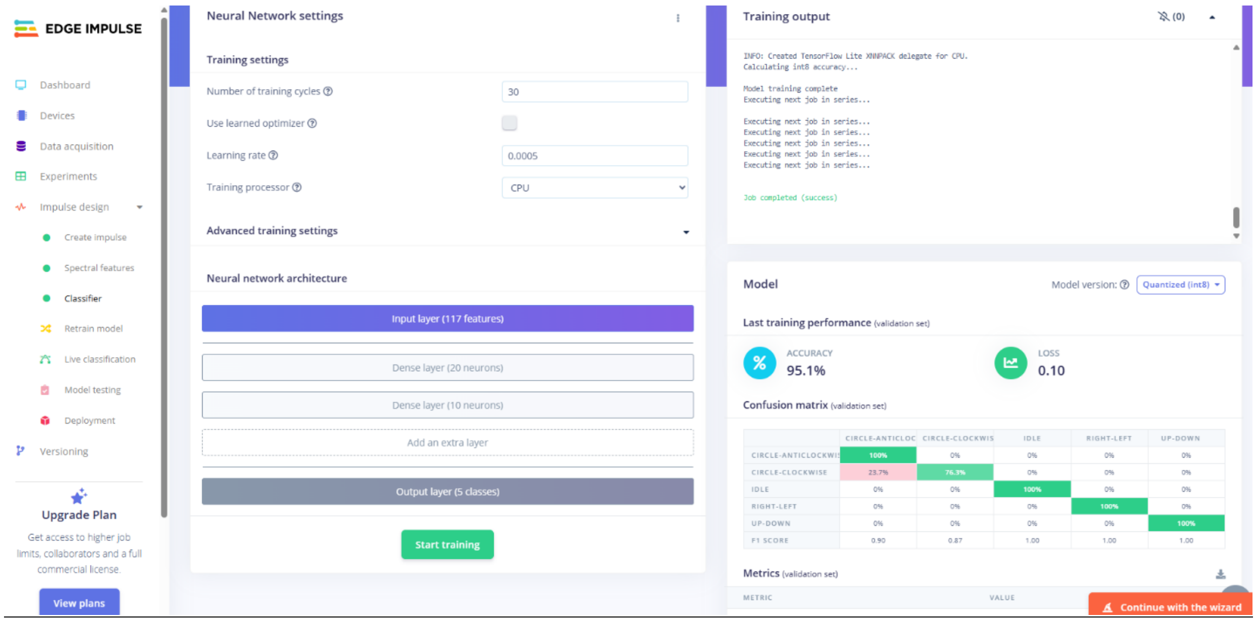
```
Creating job... OK (ID: 23333563)

Loading logs for 23333563...
Attached to job 23333563...
Scheduling job in cluster...
Container image pulled!
Job started
Not generating new features: features already generated and no options or files have changed.


Job completed (success)
```

<u>STEP 4:</u> TRAINING A NEURAL NETWORK

Following are the steps that are involved in training a Neural Network:

1. <u>Initialize</u>: Start with random weights and biases.
2. <u>Forward Pass</u>: Passing the inputs through the NN to get predictions.
3. <u>Calculate Loss</u>: Comparing the results with the actual results to calculate loss.
4. <u>Backward Pass</u>: Adjust weight and biases to reduce the loss.

The process is then repeated until the NN is completely trained to give accurate results.



Neural Network settings

Training settings

Number of training cycles	30
Use learned optimizer	☐
Learning rate	0.0005
Training processor	CPU

Advanced training settings

Neural network architecture

Training output

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
Calculating int8 accuracy...

Model training complete
Executing next job in series...

Executing next job in series...
Executing next job in series...
Executing next job in series...
Executing next job in series...
Executing next job in series...


Job completed (success)
```

Model — Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 95.1%

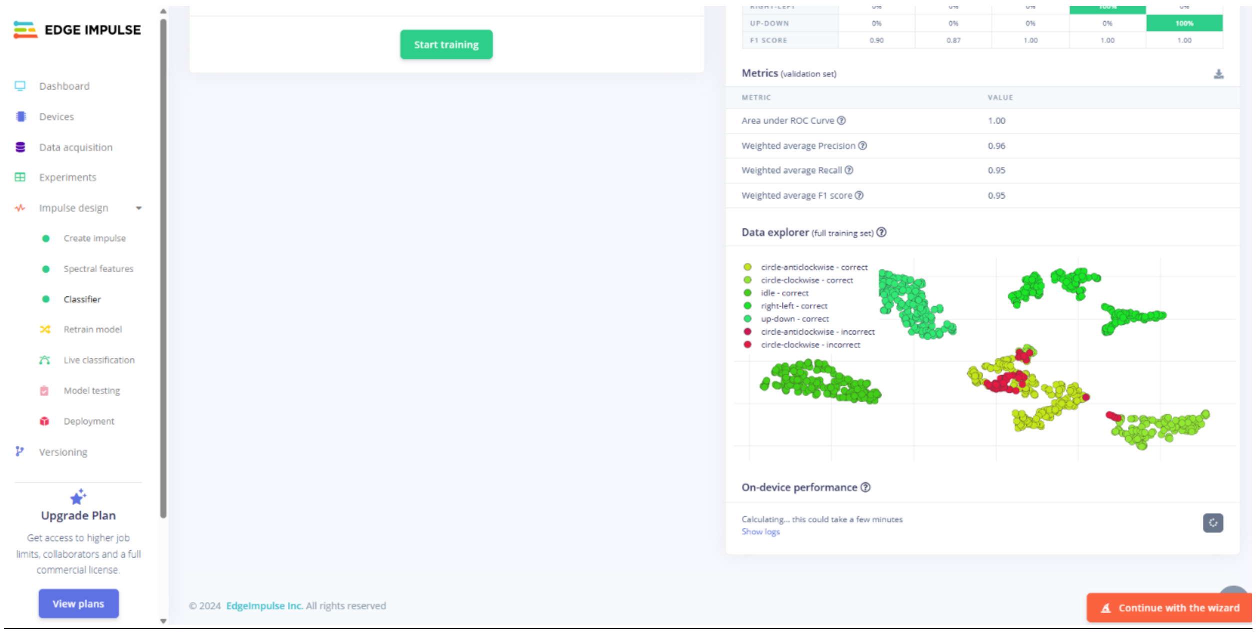
LOSS 0.10

Confusion matrix (validation set)

	CIRCLE-ANTICLOC	CIRCLE-CLOCKWIS	IDLE	RIGHT-LEFT	UP-DOWN
CIRCLE-ANTICLOCKWIS	100%	0%	0%	0%	0%
CIRCLE-CLOCKWISE	23.7%	76.3%	0%	0%	0%
IDLE	0%	0%	100%	0%	0%
RIGHT-LEFT	0%	0%	0%	100%	0%
UP-DOWN	0%	0%	0%	0%	100%
F1 SCORE	0.90	0.87	1.00	1.00	1.00

Metrics (validation set)

METRIC VALUE



	CIRCLE-ANTICLOC	CIRCLE-CLOCKWIS	IDLE	RIGHT-LEFT	UP-DOWN
UP-DOWN	0%	0%	0%	0%	100%
F1 SCORE	0.90	0.87	1.00	1.00	1.00

Metrics (validation set)

METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	0.96
Weighted average Recall	0.95
Weighted average F1 score	0.95

Data explorer (full training set)

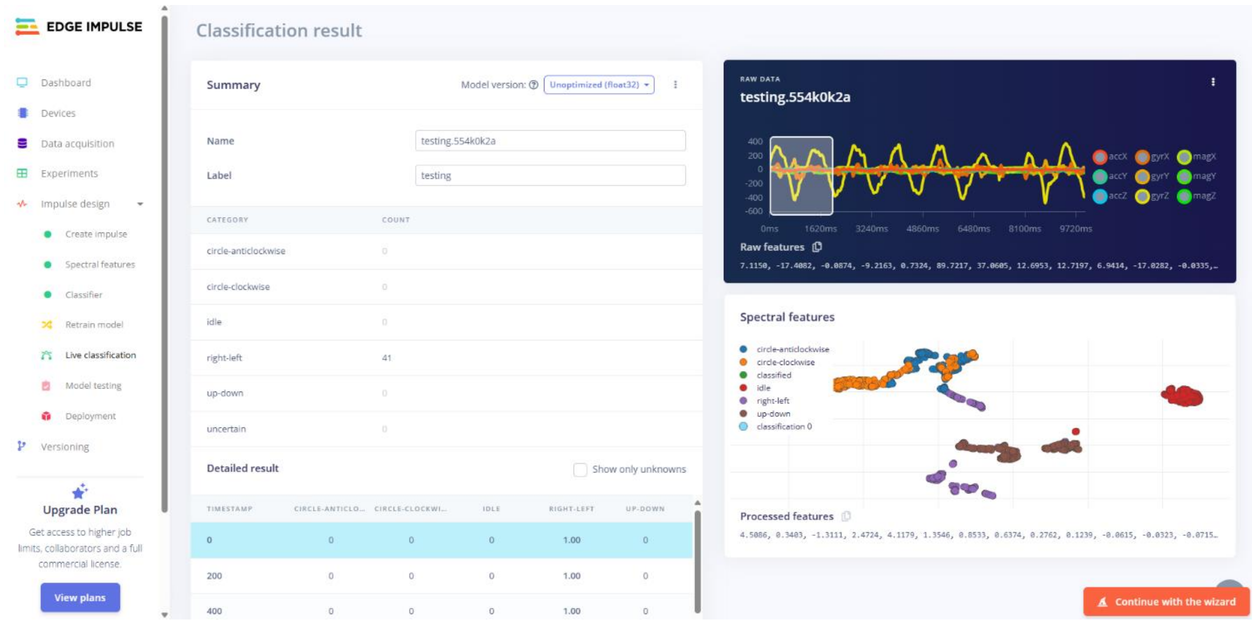
On-device performance

Calculating... this could take a few minutes
Show logs

© 2024 EdgeImpulse Inc. All rights reserved

STEP 5: **MODEL EVALUTATION**

This step is to determine if the model is capable enough to meet our needs. The process is usually carried out with the help of a confusion matrix.



Classification result

Summary

Model version: Unoptimized (float32)

Name	testing.554k0k2a
Label	testing

CATEGORY	COUNT
circle-anticlockwise	0
circle-clockwise	0
idle	0
right-left	41
up-down	0
uncertain	0

Detailed result

Show only unknowns

TIMESTAMP	CIRCLE-ANTICLO...	CIRCLE-CLOCKWI...	IDLE	RIGHT-LEFT	UP-DOWN
0	0	0	0	1.00	0
200	0	0	0	1.00	0
400	0	0	0	1.00	0

RAW DATA

testing.554k0k2a

Raw features

```
7.1150, -17.4082, -0.0874, -9.2163, 0.7324, 89.7217, 37.0605, 12.6953, 12.7197, 6.9414, -17.0282, -0.0335,…
```

Spectral features

Processed features

```
4.5086, 0.3403, -1.3111, 2.4724, 4.1179, 1.3546, 0.8533, 0.6374, 0.2762, 0.1239, -0.0615, -0.0323, -0.0715…
```

NOTE:

‘Accuracy’ is not the best predictor for model evaluation.

STEP 6: **DEPLOYING A MODEL**

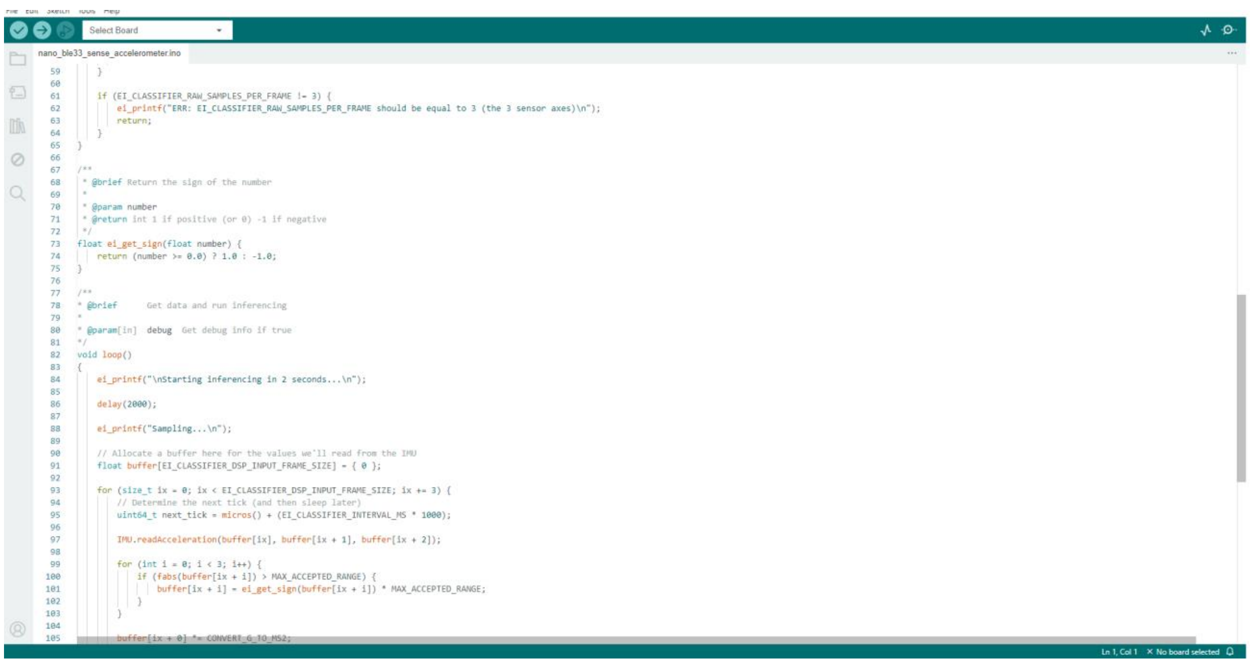
After the model had started to give satisfactory results, it is ready to be deployed into the embedded system.

1. **Prepare for Deployment:**
   - Go to the **Deployment** page of your Edge Impulse project.
   - Choose the **Arduino library** option for deployment.
   - Ensure the **EON Compiler** is enabled (recommended for space optimization).

2. **Build the Library:**
    - Click the **"Analyze"** button to get an estimate of inference time and resource requirements.
        - Note: Estimates are for ARM Cortex-M4 F at 80 MHz; your Arduino Nano 33 BLE Sense runs at 64 MHz, so expect slightly longer inference times.
        - The estimate does not include feature extraction time.
    - Click the **"Build"** button to compile the model into an Arduino library. Keep the resulting ZIP file intact.
3. **Install the Library in Arduino IDE:**
    - Delete any existing library folder for this project from Arduino's libraries directory:
        - Navigate to Documents/Arduino/libraries and delete the folder named <your_project_name>.
    - Open the Arduino IDE.
    - Go to **Sketch > Include Library > Add .ZIP Library**.
    - Select the ZIP file you downloaded and import it.
4. **Upload the Model to Arduino:**
    - Select **Tools > Board > Arduino Nano 33 BLE** in the Arduino IDE.
    - Go to **File > Examples**, find the library with your project name, and select the **nano BLE 33 sense accelerometer sketch**.
    - Plug your Arduino board into your computer.
    - Select the correct Serial port from **Tools > Port**.
    - Click **Upload**.
5. **Troubleshoot Upload Errors (if any):**
    - If you encounter a path length error, download the mbed platform.local.txt file from the Edge Impulse docs.

- Copy this file to C:\Users\<YourUsername>\AppData\Local\Arduino15\packages\arduino\hardware\mbed\1.1.4.
- Retry the upload process.

6. **Check and Run the Code:**
   - Once uploaded, change the Serial port back if it switches during upload.
   - Open the Serial Monitor to view outputs.
   - Observe sample collection and classification results.
7. **Verify Functionality:**
   - Move the Arduino left/right, up/down, and in circular motions to see if the predictions match expected labels.
   - Hold the board still and check if it labels the motion as idle.
8. **Review Model Metadata (optional):**
   - Navigate to Documents/Arduino/libraries/<your_project_name>/src/model_parameters and open model_metadata.h.
   - Check for important settings like sampling rates and label indices.
9. **Explore multi-threading (advanced):**
   - For more efficient data collection and inference, use the **Accelerometer Continuous Demo** sketch.
   - This example demonstrates how to use multi-threading with mbed OS for parallel data collection and inference.


File Edit Sketch Tools Help
Select Board
nano_ble33_sense_accelerometer.ino
Ln 1, Col 1 × No board selected

```
        }

        if (EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME != 3) {
            ei_printf("ERR: EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME should be equal to 3 (the 3 sensor axes)\n");
            return;
        }
}

/**
 * @brief Return the sign of the number
 *
 * @param number
 * @return int 1 if positive (or 0) -1 if negative
 */
float ei_get_sign(float number) {
    return (number >= 0.0) ? 1.0 : -1.0;
}

/**
* @brief      Get data and run inferencing
*
* @param[in]  debug  Get debug info if true
*/
void loop()
{
    ei_printf("\nStarting inferencing in 2 seconds...\n");

    delay(2000);

    ei_printf("Sampling...\n");

    // Allocate a buffer here for the values we'll read from the IMU
    float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };

    for (size_t ix = 0; ix < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; ix += 3) {
        // Determine the next tick (and then sleep later)
        uint64_t next_tick = micros() + (EI_CLASSIFIER_INTERVAL_MS * 1000);

        IMU.readAcceleration(buffer[ix], buffer[ix + 1], buffer[ix + 2]);

        for (int i = 0; i < 3; i++) {
            if (fabs(buffer[ix + i]) > MAX_ACCEPTED_RANGE) {
                buffer[ix + i] = ei_get_sign(buffer[ix + i]) * MAX_ACCEPTED_RANGE;
            }
        }

        buffer[ix + 0] *= CONVERT_G_TO_MS2;
```