# Getting started with machine learning using ModusToolbox™ and Imagimob Studio

## Human activity recognition and baby crying detection

## About this document

### Scope and purpose

This application note introduces machine learning (ML) on the edge and describes how to use ModusToolbox™ machine learning solution with Imagimob to collect data, train, optimize, and deploy machine learning models onto an embedded device. This is shown through the use of code examples and Imagimob reference examples, supporting both an inertial measurement unit (IMU) and Pulse-Density Modulation (PDM) microphone.

### Intended audience

This document is intended for design and application engineers interested collecting data, training a model, and deploying it to an embedded device. The application note is for machine learning beginners to experts generating models using Imagimob Studio and deploying using ModusToolbox™.

# Table of contents

# 1 Introduction to machine learning on the edge

Machine learning on the edge allows tiny, embedded devices to run applications that collect sensor data and process the data with some sort of machine learning algorithm. In the past, machine learning algorithms were typically run in the cloud. By moving the algorithm to the embedded device, a few benefits might apply:

- **Low latency:** Because the sensor data is processed locally, there is no need to send data to the cloud, which allows processing it in real time.
- **Privacy:** Keeping the sensor data locally solves some of the security concerns related to sending sensitive data to the cloud.
- **Data bandwidth:** Rather than sending the sensor data to the cloud, which might require a significant bandwidth from the connectivity block, the device can simply send the results of the data processed locally to the cloud.
- **Reliability:** Even if the connection between the embedded device and the cloud breaks, the system might still be able to collect data and run the machine learning algorithm on the device.
- **Lower cost:** By having edge-based inferencing, it allows the device to do early intervention for critical costly issues and reduces the cloud broker and compute cost for sending raw data.

When running a machine learning algorithm on the edge, the focus is on executing the inference engine, which is the process of passing sensor data into a machine learning model to calculate an output. The machine learning model needs to be trained by using a good-quality and quantitative data set. The initial training phase typically runs on a high-level computing system, for example, collecting motion data for human activity such as “jumping”, “sitting”, and “walking” from multiple individuals. Later, a training algorithm can be applied to generate a trained model and then the model can be deployed to the device so that it can detect the activity performed. Figure 1 shows a standard embedded ML flow for a gesture detect model.

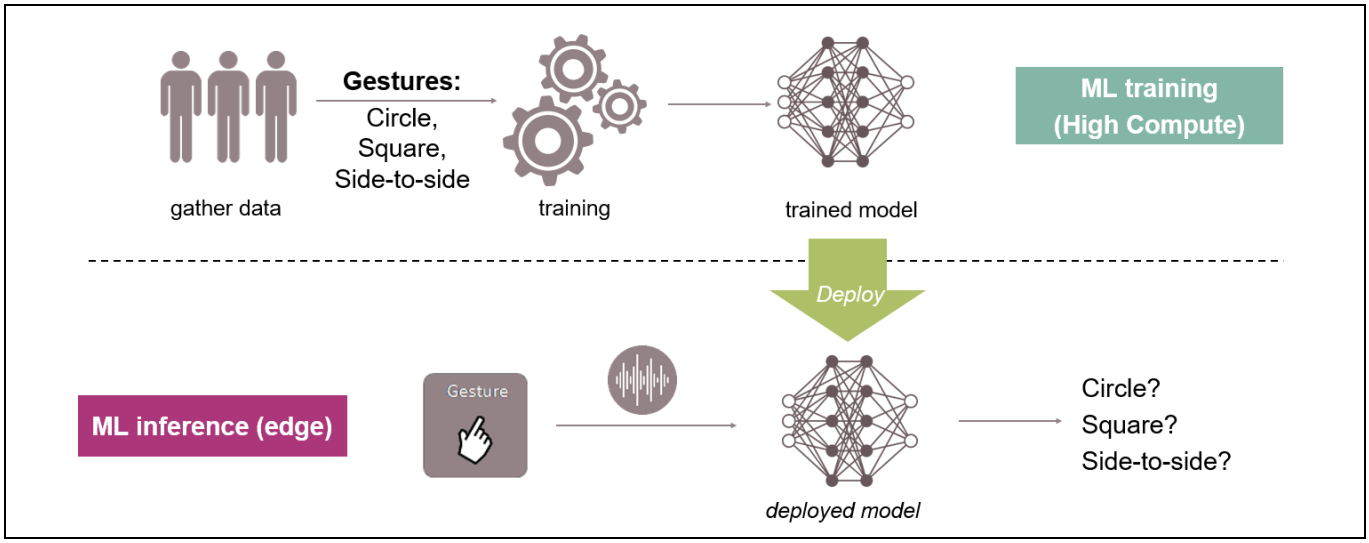


**Figure 1 Standard embedded ML flow for a gesture detect model**

The Standard development section describes the development flow to create such a machine learning application.

# 2 Imagimob Studio

Imagimob Studio is a tool that supports the general flow of machine learning development. Imagimob Studio supports importing and labeling data, pre-processing data, designing and training of models, model evaluation, and model deployment. Imagimob Studio helps users move through the Standard development by offering an easy to use interface with configurations to support a wide variety of use cases such as predictive maintenance, audio applications, gesture recognition, signal classification, fall detection, and material detection.

This application note uses Imagimob Studio to go through the development process and develop two different models.

*Note:* *All content on Imagimob account creation, Studio install, and project creation can be found in Imagimob's developer documentation.*

## 2.1 Registering with Imagimob

To download and install Imagimob Studio, a user account is required. To create an Imagimob account:

1. Navigate to Imagimob account creation.

The following window appears.

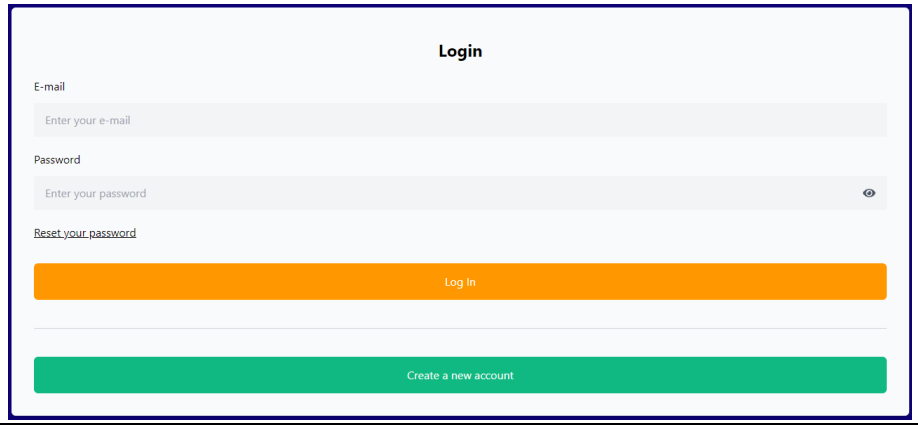


**Figure 2 Imagimob account login**

2. Click **Create a new account** for account creation.

The following window appears.

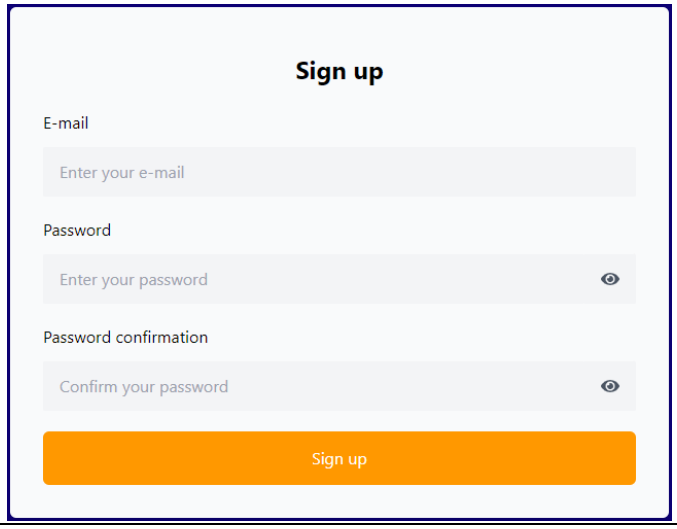


**Figure 3 Imagimob account sign up**

3. Insert a valid email address and password then select **Sign up**.

To ensure that a valid email was used, a confirmation email is sent to your email address. Complete the verification to finish the account creation process.

## 2.2 Imagimob Studio installation

Once an account is created, Imagimob Studio can be downloaded and installed.

1. Navigate to Imagimob account creation and sign in.
2. Once signed in, the following page will be shown. Select **Download (64-bit)** to start the download.


**Figure 4 Imagimob Studio download**

*Note: Imagimob Studio currently only supports Windows.*

3. Locate and run the Imagimob Studio executable file as administrator. The Imagimob Setup wizard appears.
4. Follow the prompts in the Imagimob Setup wizard to complete the installation.

The installation is now completed; the Imagimob Studio icon appears on the desktop.

## 2.3 Imagimob project creation

Imagimob Studio supports creating empty projects, but also has a selection of starter projects to get users up and running. The starter projects allow using a machine learning model that is already trained by Imagimob for different use cases. Use a starter project as a starting point and fine-tune the model as per the requirements. Two of the starter projects will be used throughout this application note, human activity recognition (IMU based) and baby crying detection (PDM based). Before creating a project, let us understand the concept of workspaces and projects in Imagimob Studio.

A workspace is a top-level directory in Imagimob Studio containing one or more project directories that helps in organizing and managing projects. A workspace can be created locally, when creating or downloading a project from Imagimob Studio.

A project is a directory in workspace containing data, models, project file and other resources that helps in managing the assets to build, train and deploy the machine learning model. Create an empty classification project, start from a starter project, or import an existing project to Imagimob Studio.

An empty classification project allows you to create a machine learning model from scratch. You can also bring your personal dataset and build, train and deploy the machine learning model in Imagimob Studio as per your business scenarios.

To create an empty classification project or starter project, perform the following steps:

1. Open Imagimob Studio,
2. On the **Welcome** page select **New Project**.

   The welcome screen appears when you open the Imagimob Studio for the first time.

   OR

   Go to File and select New Project.
   The New Project window appears.
3. Select the project type that meets your requirement.

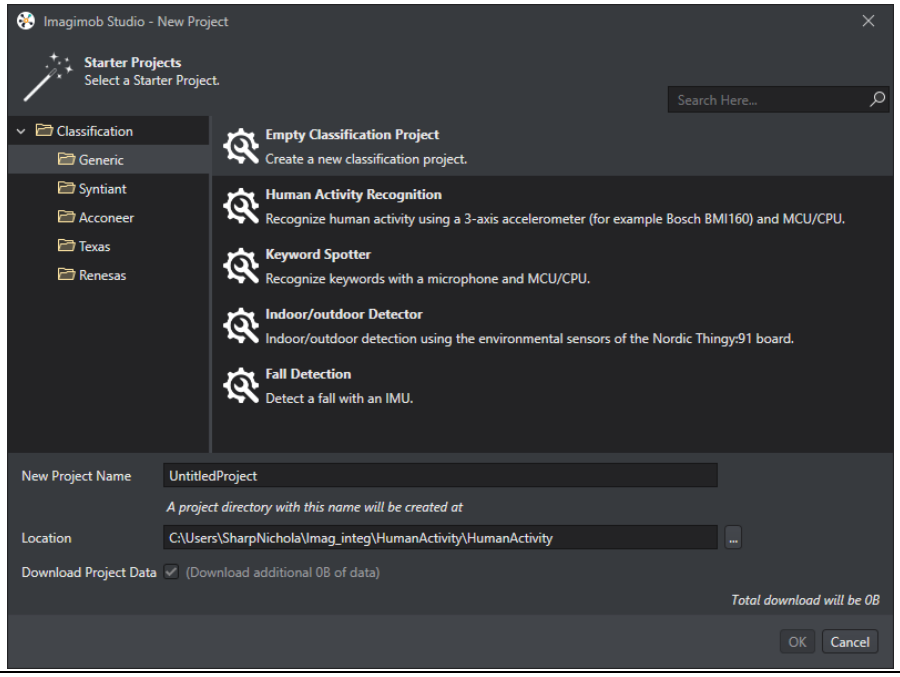


**Figure 5 Select project**

- To create a machine learning model from scratch or upload an existing model, select the project type as Empty Classification Project.
- To start with a pre-trained machine learning model, select the starter project that meets the business requirements. The Human Activity Recognition and Baby Crying Detection projects work out of the box with the ModusToolbox™ Reference Examples.

4. In the **New Project Name** field, enter the name of the project.
5. In the **Location** field, specify the location where you want to create the workspace and the project directory.
6. Select **Download Project Data** checkbox to download the project data for that project.
   The project directory is downloaded to the workspace in Imagimob Studio.
7. Click **OK** to create the project.

# 3 Imagimob reference examples

Imagimob offers a selection of starter examples to get users started quickly. Two of the supplied starter examples (Human Activity and Baby Crying detection) are used throughout this application note to show the standard ML flow using both ModusToolbox™ for Machine Learning (MTB-ML) and Imagimob Studio. It is recommended that an empty project is started after going through the starter projects and the ML flow is understood.

Both projects include training, validation, and test data that can be used to go through the ML development flow. The projects also include the pre-processing and model architecture so that users can quickly train a model without deciding the best architecture for these use cases.

See Imagimob starter projects documentation for more information.

## 3.1 Human Activity Recognition

The Human Activity Recognition starter project provides a pre-trained machine learning model that predicts human activities such as running, standing, walking, sitting, or jumping. Data for the project has been collected and is stored in “Data” folder. The project uses the BMI160 or BMX160 Inertial Measurement Unit (IMU), setup to collect data at 50 Hz using ± 8g for the accelerometer scale.

### 3.1.1 Pre-processing

The Human Activity Recognition project uses a “Sliding Window” for the pre-processing. The sliding window means that each data set that is fed to the model or inference engine overlaps with the previous sample. This makes sure that a pattern that the model is looking for is not broken up between two different data sets. The preprocessor is fed an input shape of 3 for the three values of the IMU (Accel_x, Accel_y, Accel_z). The output is a [50, 3] buffer that can then be fed to the inference engine at a rate of 3 Hz. With a 50 Hz IMU sample frequency this means the model is evaluating over a period of 1 second. The sliding window is configured as shown in Figure 6.

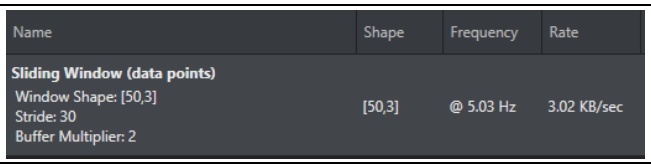


**Figure 6 Sliding window configuration**

### 3.1.2 Model architecture

The Human Activity Recognition model is a standard convolutional neural network (CNN) model consisting of five convolutional groups. These groups are divided into three groups of convolutional blocks (one or two layers), a batch normalization layer, a max pooling layer, and a dense layer at the end acts as a classifier. The complete model architecture can be seen in Figure 7.

| Name | Shape | Parameters |
|---|---|---|
| Input Layer | [50,3] | 0 |
| Convolution 1D | [25,16] | 144 |
| Batch Normalization | [25,16] | 64 |
| Activation | [25,16] | 0 |
| Convolution 1D | [25,16] | 768 |
| Convolution 1D | [25,16] | 768 |
| Batch Normalization | [25,16] | 64 |
| Activation | [25,16] | 0 |
| Max pooling 1D | [12,16] | 0 |
| Convolution 1D | [12,32] | 1536 |
| Convolution 1D | [12,32] | 3072 |
| Batch Normalization | [12,32] | 128 |
| Activation | [12,32] | 0 |
| Max pooling 1D | [6,32] | 0 |
| Global average pooling 1D | [32] | 0 |
| Dense | [6] | 198 |
| Activation | [6] | 0 |

**Figure 7 Human Activity Recognition model architecture**

## 3.2 Baby Crying Detection

The Baby Crying Detection project provides a machine learning mode that predicts if a baby is crying or not. Data for the project has been collected and is stored in the “Data” folder. The project uses dual-channel pulse density-modulated audio data collected at 16 kHz to train the model.

### 3.2.1 Pre-processing

The Baby Crying Detection pre-processor is fed one PDM sample at a time at a frequency of 16 kHz. The pre-processor used in this starter project generates a Mel spectrogram of the input audio data. Such a spectrogram is obtained by (1) calculating the Fast Fourier Transform (FFT) of a window of raw audio data and (2) by applying a Mel filter bank to it. This is a commonly used technique in audio pre-processing, and it helps the model in focusing on the relevant features of the audio signal thereby reducing the amount of data that the model needs to process. A sliding window is used to input the spectrogram data to the actual neural network. The output is a [60, 20] buffer that can be produced at 3 Hz. The complete pre-processing configuration can be seen in Figure 8.

| Name | Shape | Frequency | Rate |
|---|---|---|---|
| **Sliding Window (data points)**<br>Window Shape: [512]<br>Stride: 160<br>Buffer Multiplier: 1 | [512] | @ 100 Hz | 204.8 KB/sec |
| **Hann smoothing**<br>Symmetric: True | [512] | @ 100 Hz | 204.8 KB/sec |
| **Real Discrete Fourier Transform**<br>Axis: 0 | [257,2] | @ 100 Hz | 205.6 KB/sec |
| **Frobenius norm**<br>Axis: 0 | [257] | @ 100 Hz | 102.8 KB/sec |
| **Mel Filterbank**<br>Number of Filters: 20<br>Sample Rate (Hz): 16000<br>Low Frequency Cutoff (Hz): 300<br>High Frequency Cutoff (Hz): 8000<br>HTK formula: True<br>Librosa formula: False | [20] | @ 100 Hz | 8 KB/sec |
| **Clip**<br>Min: 0.000316227766016<br>Max: 3.40282347E+38 | [20] | @ 100 Hz | 8 KB/sec |
| **Logarithm**<br>Logarithm base: 0 | [20] | @ 100 Hz | 8 KB/sec |
| **Sliding Window (data points)**<br>Window Shape: [60,20]<br>Stride: 660<br>Buffer Multiplier: 1 | [60,20] | @ 3.03 Hz | 14.55 KB/sec |

**Figure 8 Pre-processing configuration**

### 3.2.2 Model architecture

The Baby Crying Detection model is a standard 2D convolutional neural network (CNN) model consisting of five convolutional groups. These groups are divided into three groups of convolutional blocks (one or two layers), a batch normalization layer, a rectified linear unit (ReLU) activation layer, and an average pooling layer. The complete model architecture can be seen in Figure 9.

| Name | Shape | Parameters |
|---|---|---|
| **Input Layer** | [60,20] | 0 |
| **Reshape** | [60,20,1] | 0 |
| **Convolution 2D** | [30,10,12] | 300 |
| **Batch Normalization** | [30,10,12] | 48 |
| **Activation** | [30,10,12] | 0 |
| **Convolution 2D** | [15,5,24] | 7200 |
| **Convolution 2D** | [8,3,24] | 14400 |
| **Batch Normalization** | [8,3,24] | 96 |
| **Activation** | [8,3,24] | 0 |
| **Convolution 2D** | [8,3,32] | 6912 |
| **Convolution 2D** | [8,3,32] | 9216 |
| **Batch Normalization** | [8,3,32] | 128 |
| **Activation** | [8,3,32] | 0 |
| **Global average pooling 2D** | [32] | 0 |
| **Dense** | [2] | 66 |
| **Activation** | [2] | 0 |

**Figure 9 Baby Crying Detection model architecture**

# 4 ModusToolbox™

ModusToolbox™ is a set of tools that enable integration of devices into existing development methodology. Out of the box, ModusToolbox™ comes with an Eclipse-based Integrated Development Environment (IDE) that supports application configuration and development. ModusToolbox™ also supports development with IAR, Keil, and Visual Studio toolchains.

See the instructions in the ModusToolbox™ tools package installation guide for how to download and install ModusToolbox™.

## 4.1 ModusToolbox™ machine learning (ML) solution

The ModusToolbox™ ML solution is a set of tools, libraries, and middleware that helps to build, evaluate, and benchmark pre-trained ML models. ModusToolbox™ ML libraries easily and efficiently run inferencing on an Infineon MCU. These libraries and tools help to rapidly deploy neural network (NN)-based classification-type ML applications.

The solution also provides a configurator to import pre-trained ML models and generate an embedded model implementation (as C code or binary file). This generated model can be used with the Modus Toolbox™ ML library along with the user application code for a target device. The tool also lets you optimize the pre-trained model of choice and evaluate its performance.

To get started with the ModusToolbox™ ML solution, see the Machine Learning user guide and download the required tools.

# 5 ModusToolbox™ reference examples

To support the Imagimob Studio's standard ML flow, three code examples are included in ModusToolbox™. While Imagimob Studio focus' on pre-processing, training, and validation, the ModusToolbox code examples focus on data collection and deployment. These code examples will be used throughout the application note in support of the ML development flow.

Imagimob Studio can produce a C implementation of the preprocessing, or a C implementation of the pre-processing and model (combined). The Deploy with MTB-ML code example uses only the preprocessor along with the .h5 file that Imagimob produces. The .h5 model file is passed to the MTB-ML configurator to generate TensorFlow Lite for Microcontrollers (TFLM) source code. The Deploy with Imagimob code example uses the Imagimob produced C implementation of both the pre-processor and model, which means the MTB-ML configurator is not used.

## 5.1 Data collection

The mtb-example-ml-imagimob-data-collection code example demonstrates how to collect data using Imagimob's Capture Server to train a model within Imagimob Studio. The code example supports collecting data from two different sources, data can be collected from an IMU or from a digital microphone using PDM to PCM. The data is transmitted using UART to the Capture Server where it stores it as a .data file for IMU or a .wav file for PDM data. The data can then be used in the Human Activity Detection or Baby Crying Detection Imagimob starter projects or to generate a new model.

For more information, view the mtb-example-ml-imagimob-data-collection readme on GitHub.

## 5.2 Deploy with MTB-ML

The mtb-example-ml-imagimob-MTB-ML-deploy code example demonstrates how to deploy an Imagimob generated machine learning model with the MTB-ML flow. The code example includes two models that are generated from Imagimob Studio's starter projects. The first model, Human Activity Detection uses data from an IMU which is then sent to the model to detect specific motions (sitting, standing, walking, running, or jumping). The Human Activity Detection model is setup to run out of the box. The second model, Baby Crying Detection uses data from the PDM which is sent to the model to detect whether a baby is crying or not. Both models require pre-processing produced by Imagimob Studio's, Human Activity Detection uses *imu_model.c/h* while Baby Crying Detection uses *pdm_model.c/.h*. New models based on IMU (BMX160 or BMI160) or PDM data can be dropped into this project as-is.

For more information, view the mtb-example-ml-imagimob-MTB-ML-deploy readme on GitHub.

## 5.3 Deploy with Imagimob

The mtb-example-ml-imagimob-deploy code example demonstrates how to deploy an Imagimob generated machine learning model. It comes pre-configured with a model generated from the Human Activity Recognition starter project in Imagimob Studio. The code example collects accelerometer data from an IMU which is then sent to the machine learning model to detect specific motions (sitting, standing, walking, running, or jumping). It uses the model.c/h files generated from within Imagimob Studio directly. New models based on the Human Activity Recognition project can be dropped into the project as-is.

For more information view the mtb-example-ml-imagimob-deploy readme on GitHub.

# 6 Standard development flow

A typical development flow starts with the sensor data and ends with an ML application for the target device. There are four major stages in this flow:

- Data collection**:** Data is collected on an Infineon MCU and transmitted to the Imagimob Capture Server. Once data is captured, it can be imported into Imagimob Studio and labeled. Imagimob also supports data visualization to ensure proper labeling.
- Pre-processing**:** Data signal processing techniques are applied to the sensor data, such as windowing and filtering techniques are applied to reduce the noise, and finally, features of the sensors data are extracted.
- Model training**:** A model might be created from scratch layer-by-layer or Imagimob's Auto ML feature can be used. Using the data collected, begin fitting the data to the model. After the model is trained, it is tested to ensure accuracy.
- Deployment**:** Imagimob supports generating C implemented code to produce either combined pre-processing and model files, or just pre-processing files with a .h5 file. With either selection, an embedded device can be programmed to run the inferencing.

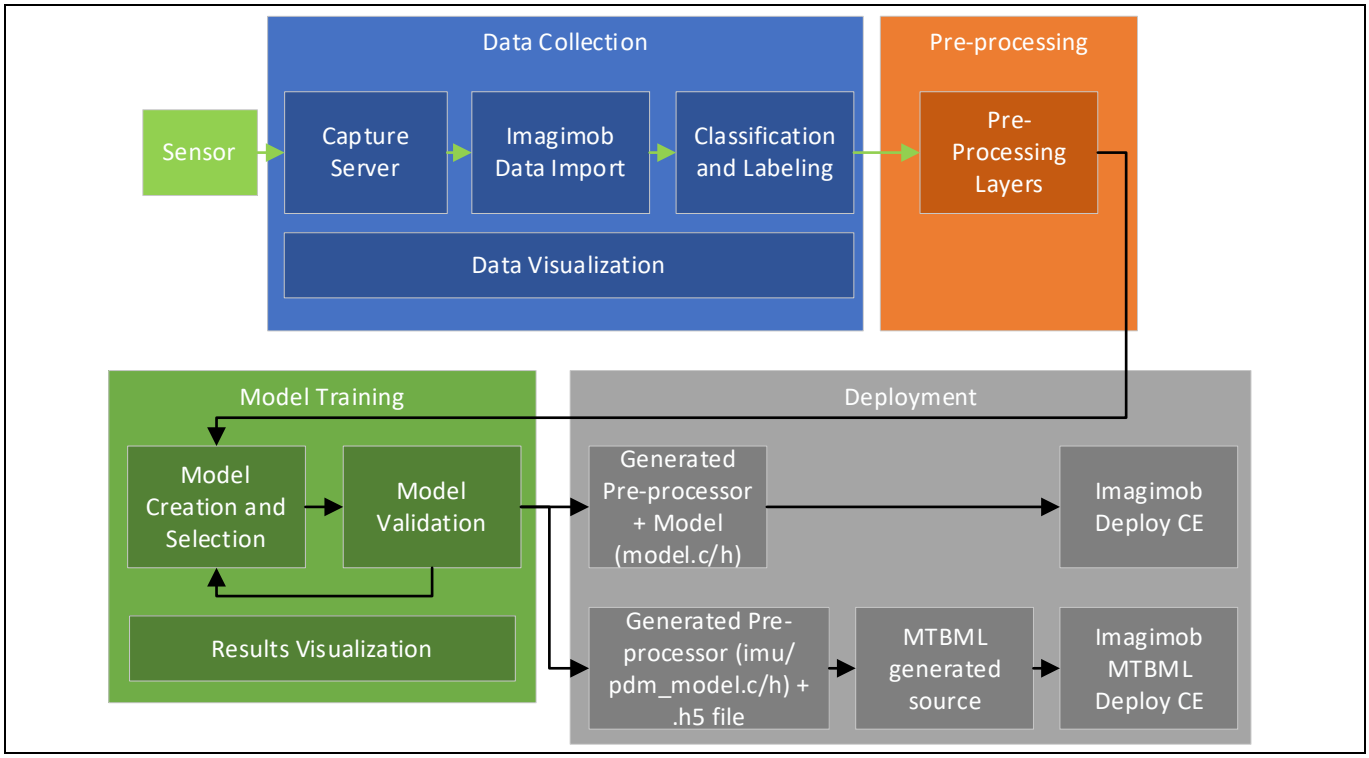


**Figure 10 Standard development flow**

## 6.1 Data collection

The first step to start the ML application journey is to obtain the data from sensors. To do so, it is recommended to collect the data from the same hardware that is planned to run the inference engine. This is to avoid any discrepancy between the data used to train the model and the one to run the inference engine.

Because the training is not performed in an embedded system, some mechanism to upload the data to a computer is required (Imagimob Capture Server setup). The mtb-example-ml-imagimob-data-collection code example supports data collection of both IMU and PDM data and streams the data using UART.

When collecting data, it is recommended to capture data from multiple different users to train a robust model.

### 6.1.1 Data collection with PSoC™

The mtb-example-ml-imagimob-data-collection code example supports data collection of both IMU and PDM data and streams the data using UART. This code example supports using the Imagimob Capture Server for data collection.

See the README.md file from the code example to download, configure the example, and program the device.

### 6.1.2 Imagimob Capture Server setup

The Imagimob Capture Server is a collection of python scripts that simplifies the integration of sensor data with Imagimob Studio. The Imagimob Capture Server includes many example scripts to collect sensor data over different transmission protocols including Bluetooth® Low Energy, UART, USB, and TCP.

To install the Imagimob Capture Server run the following commands from the command prompt:

```
git clone git@bitbucket.org:imagimob/captureserver.git
cd captureserver
python setup.py install
```

The *setup.py* script downloads the Python modules listed in Table 1.

**Table 1 Python modules**

| Python module | Version | Description |
|---|---|---|
| *Bleak* | 0.20.2 | GATT client software for Bluetooth® LE connection |
| *Keyboard* | 0.13.4 | Capture keyboard inputs |
| *Numpy* | 1.18.2 | Data analysis and computing |
| *Opencv-python* | 4.4.0 | Capture video |
| *Pandas* | 2.0.3 | Data analysis and manipulation tool |
| *PySerial* | 3.5 | Python serial port library |
| *Wave* | 0.0.2 | Wav file manipulation |

The Imagimob Capture Server is now setup and ready to use.

### 6.1.3 Imagimob Capture Server data collection

The Imagimob Capture Server supports collecting data over different communication protocols, as the data collection code example supports UART, the *generic_local_capture_interface.py* is used to capture and format data.

Change to the following directory *{Imagimob Capture Server cloned repo}/captureserver/examples/generic*. The script supports many different configurations, for more information see the README file.

The commands are used in this application note are listed in Table 2.

**Table 2 Imagimob Capture Server commands**

| Command | Description | IMU | PDM |
|---|---|---|---|
| *--data-filename* | Specify the file name for the collected sample | accel | pdm |
| *--output-dir* | Specify the directory to store the data | data | data |
| *--protocol* | Specify the protocol for data transmission | serial | serial |
| *--COM-port* | COM port used by the device | *Device dependent | *Device dependent |
| *--baudrate* | Select connection baud rate. PDM collection requires a higher baud rate for the 16 kHz sample rate | 115200 | 921600 |
| *--data-format* | Select output file format as .data or .wav | Default (.data) | .wav |
| *--data-type* | Select sample data type | f - float | h - short |
| *--samples-per-packet* | Specify the number of samples sent in a packet. IMU sends one sample of x, y, z acceleration data. PDM sends packets of 1024 samples at a time. | 3 | 1024 |
| *--features* | Select number of features/channels or types of data. IMU has three features, x, y, z acceleration data. PDM is dual channel, one feature per channel. | 3 | 2 |
| *--sample-rate* | The number of samples to attempt to collect per second | Default (50) | 16000 |
| *--video-disabled* | Disables collecting video while recording (required for PDM data collection) | Not Used (Can be used if video is not required) | Video Disabled |

*Note: The video feed must be disabled for PDM data collection to increase data speeds.*

### 6.1.3.1 Code example is configured for IMU data collection

Once the code example is configured for IMU data collection and the device is programmed and connected to the PC, run the following command after changing the '**xxx**' to the COM port used by the developer kit.

```
python generic_local_capture_interface.py --data-filename accel --output-dir
data --protocol serial --COM-port COMxxx --baudrate 115200 --data-type f --
samples-per-packet 3 --features 3
```

The script will start the video feed, this may take up to 10 seconds. The collected video can be used in Imagimob Studio for reference while labeling. The commands in Table 3 are available to start recording, stop recording, and to quit the capture server. Press **USER BTN1** on the kit after the server is started to trigger the device to start transmitting data. This is done to sync the capture server. Each time the server is run, reset the kit and press "USER BTN1".

**Table 3 Video feed commands**

| Command | Description |
|---|---|
| *r* | Start recording data |
| *s* | Stop active recording |
| *q* | Quit the data capture server |

Press ‘r’ and complete one of the various activities sitting, standing, walking, running, or jumping and press ‘s’ once the action is complete. Pressing ‘r’, completing an activity, then pressing ‘s’ can be repeated multiple times and will results in multiple files (the user button on the kit only needs to be pressed before selecting ‘r’ for the first time). For proper collection, the sensor on the board must be oriented in the same general manner as the rest of the training data. The orientation for sitting and standing are pictured in the following figures. For walking, running, and jumping the board is held the same as orientation as standing with the normal arm movements associated with the respective activity.

For sitting, the board should be upside down with the KitProg USB port facing forward.

**Figure 11 Board orientation for sitting**

For standing, the shield should be facing toward the body with the KitProg USB toward the top of the board.

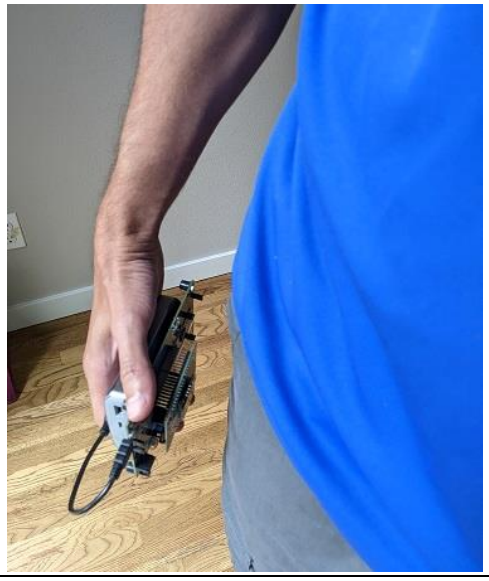

**Figure 12 Board orientation for standing**

When the required data collection is completed, press ‘q’ to quit the capture server. If the server is restarted, the kit should be reset, and the user button pressed after the server is restarted. All collected data will be found in the directory *{Imagimob Capture Server cloned repo}/captureserver/examples/generic/data.* The output files will include *label.label*, *accel.raw*, *video.mp4*, and *accel_parsed.data*. These files will be imported into Imagimob Studio.

### 6.1.3.2 Code example is configured for PDM data collection

Once the code example is configured for PDM data collection and the device is programmed and connected to the PC, run the following command after changing the ‘**xxx**’ to the COM port used by the developer kit.

```
python generic_local_capture_interface.py --data-filename pdm --output-dir
data --protocol serial --COM-port COMxxx --baudrate 1000000 --data-format
.wav --data-type h --samples-per-packet 1024 --features 1 --sample-rate 16000
--video-disabled
```

Press **USER BTN1** on the kit after the server is started to trigger the device to start transmitting data. This is done to sync the capture server. Each time the server is run, reset the kit and press "USER BTN1”.

The commands in Table 4 be available to start recording, stop recording, and to quit the capture server.

**Table 4 Record data commands**

| Command | Description |
|---|---|
| *r* | Start recording data |
| *s* | Stop active recording |
| *q* | Quit the data capture server |

Press ‘r’ and play baby crying sounds near the sensor shield, press ‘s’ to stop data collection. Pressing ‘r’, collecting data, then pressing ‘s’ can be repeated multiple times and will results in multiple data files (the user

button on the kit only needs to be pressed before selecting 'r' for the first time). When the required data collection is completed, press 'q' to quit the capture server. If the server is restarted, the kit should be reset, and the user button pressed after the server is restarted. All collected data will be found in the directory *{Imagimob Capture Server cloned repo}/captureserver/examples/generic/data.* The output files will include *label.label*, *pdm.raw*, and *pdm_parsed.wav*. These files will be used Imagimob Studio.

### 6.1.4 Imagimob Studio data import

Imagimob Studio allows for easy importing of new data into a project. At this point a workspace and project should be created, if this has not been done see Imagimob project creation. To add data to a starter project, either the Human Activity Recognition or Baby Crying Detection starter project should be used.

Imagimob Studio supports importing the file types in Table 5.

**Table 5 Data types**

| Data type | File type |
|---|---|
| *Audio* | .wav |
| *Data* | .csv, .data |
| *Video* | .mp4 |

To import data:

1. Copy the collected data from *{Imagimob Capture Server cloned repo}/captureserver/examples/generic/data* to the *Data* folder in the starter project.
2. Open the project file (.improj) present in the project folder.

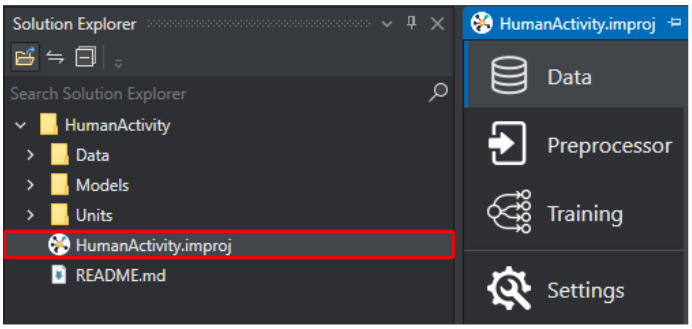


**Figure 13 Project file**

3. In the **Data** tab, click **Add Data** button.

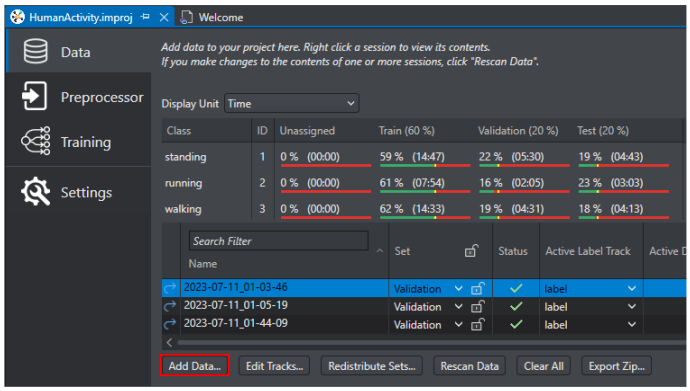


**Figure 14 Add project data**

4. Once the **Select import mode** window appears, select the **Add** icon.

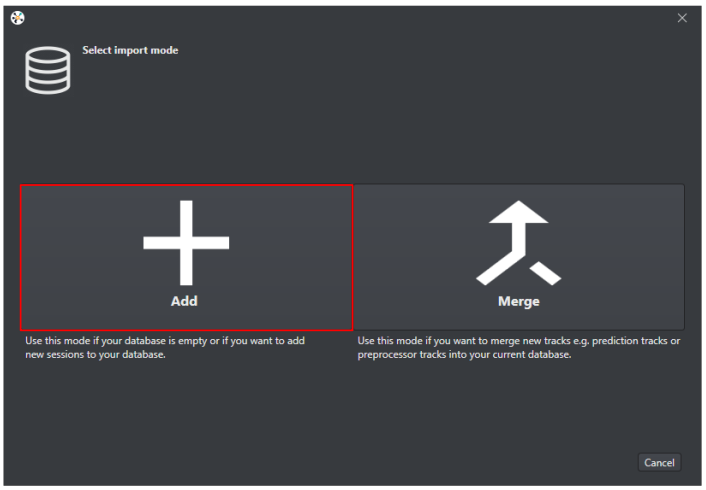


**Figure 15 Add database**

5. Select the *Data* folder in the current working project as the source for data.

6. If importing data from the capture server, select the **Nested structure** and click **Next.**

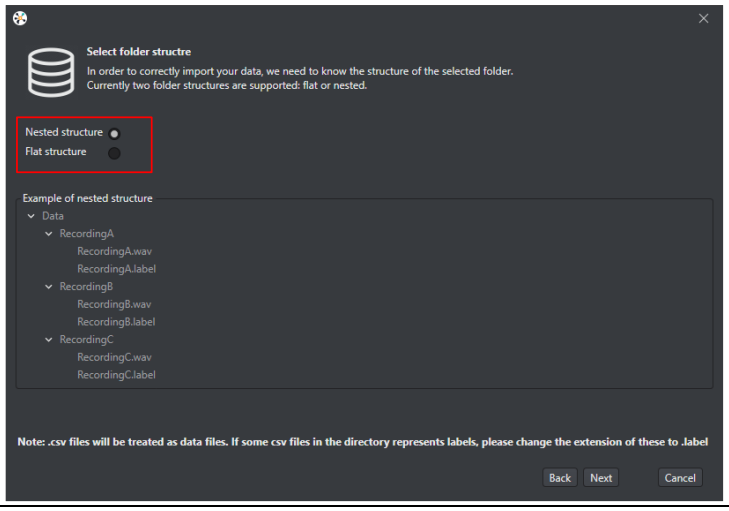


**Figure 16 Nested folder structure**

7. The data import tool searches through the folders and recognizes any folders containing *.data*, *.label*, or *.mp4 files*.

   When importing from the capture server, the following files are imported, label.label, imu_parsed.data or pdm_parsed.wav, and imu.raw or pdm.raw. **The .raw files should not be included in the import**, they are called imu and pdm in the importer. Simply deselect the imu and pdm file. Select or deselect what files should be imported into the project and select **OK**.

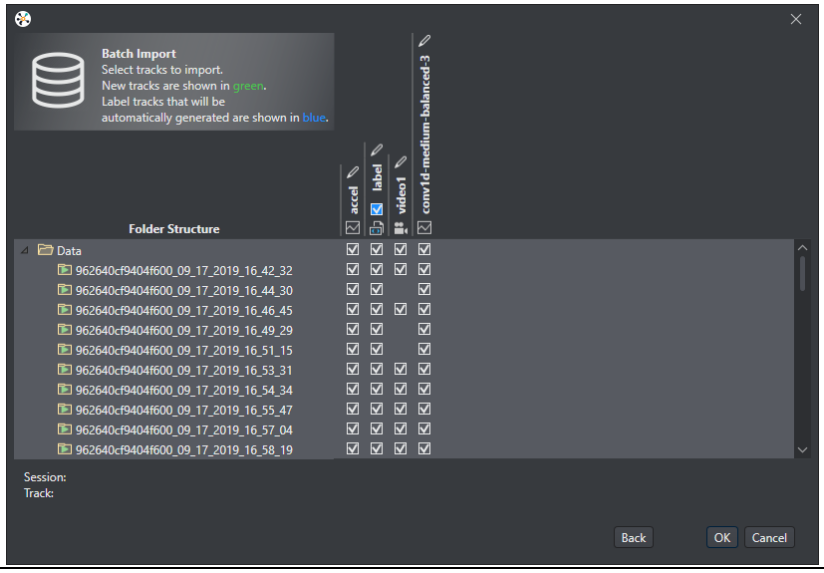


**Figure 17 Data folder file selection**

The data was imported into Imagimob Studio. After the data was imported a session file (.imsession) and an empty label file (.label) is automatically generated for every data file in the *Data* directory. These new files will be used for labeling the data.

### 6.1.5 Data labeling

Data labeling allows labeling the raw data with the expected classification. When the model is being trained, the data is used, and the model gives a classification which is then compared to the label given by the user. The model weights are then adjusted depending on whether the correct classification was achieved.

To make labeling simple, when adding data to a project, Imagimob Studio adds a session file and an empty label file for every data file in the data directory. The session file displays data file and label file as data track and label track, respectively. Add labels in the label track, whenever needed.

To add labels in the label track, follow the steps:

1. Navigate to your project directory and open the session file (.imsession).
   OR
   Double-click the session file (.imsession) to open from the **Data** tab.

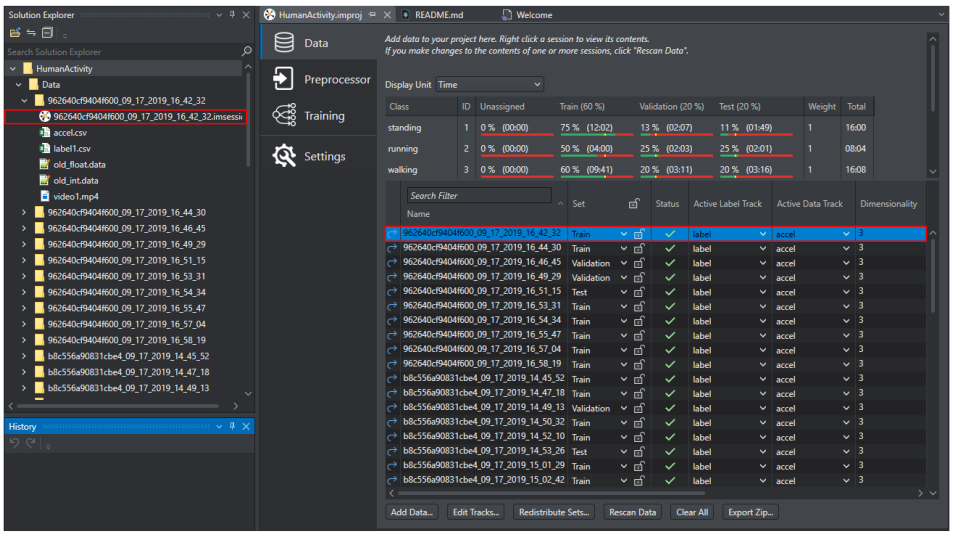


**Figure 18 Open session file**

   The session file opens a new tab where the collected data and label are viewed.

2. Locate and select the empty **label** track in the session file.

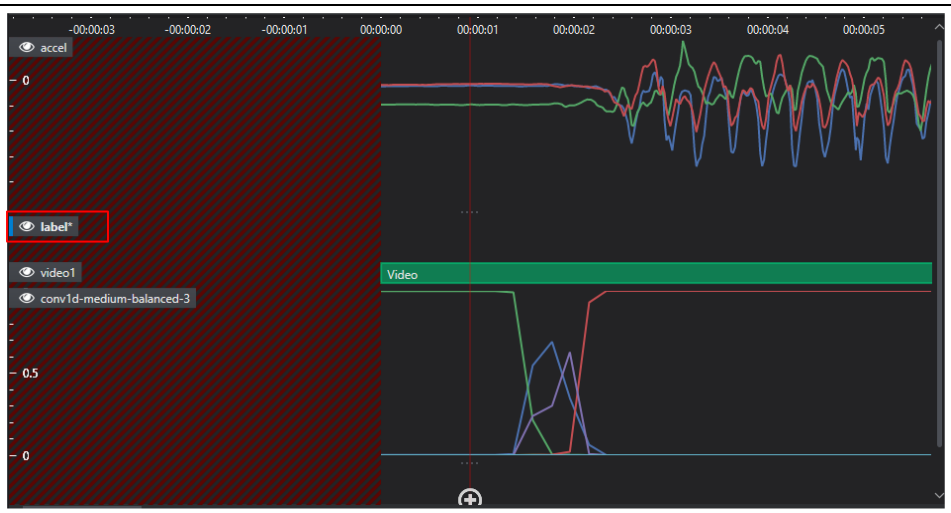

**Figure 19 Select label track**

3. Zoom-in or zoom-out until the entire event is seen and navigate to the part to label.

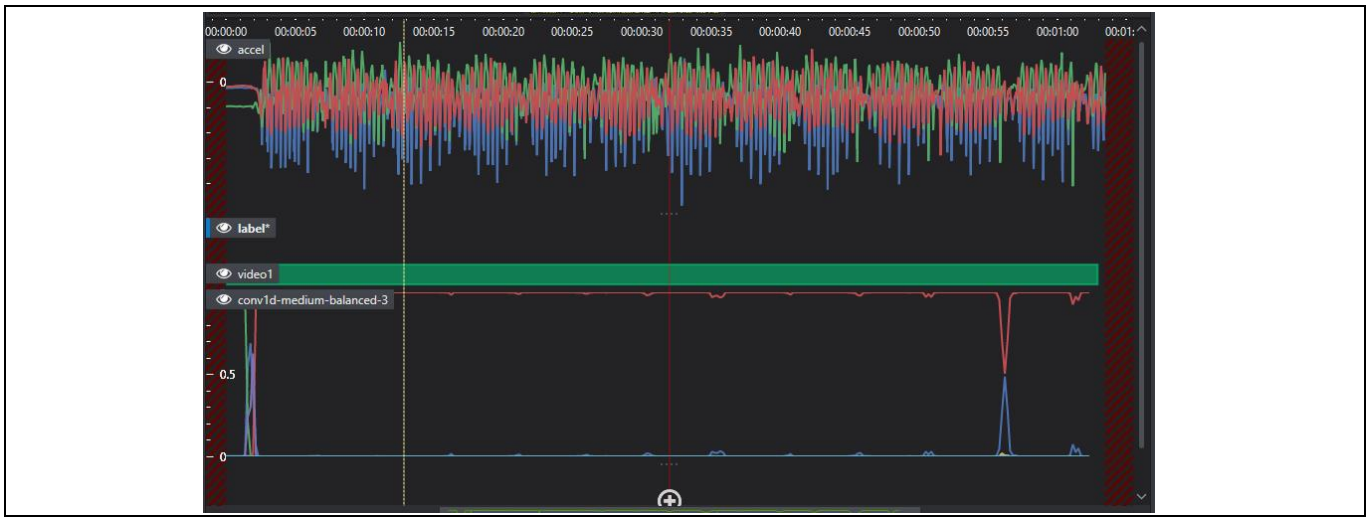


**Figure 20 Part labeling**

4. Move the cursor to the label track region, right-click and select **New Label** from the list of options. The **Add New Label** window appears.

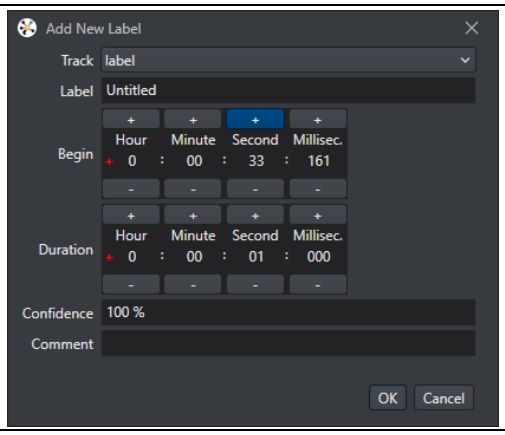


**Figure 21 Add new label**

5. Configure the following parameters:

**Table 6 Label parameters**

| Parameter | Description |
|---|---|
| *Track* | Select the label track in which you want to add the labels |
| *Label* | Enter the name of the label to add for that specific piece of data in the track |
| *Begin* | Set the timestamp from where the label starts |
| *Duration* | Set the timestamp for how long the label lasts |
| *Confidence* | Enter the confidence percentage for the input label |
| *Comment* | Comment for label, if required |

If collecting data for the Human Activity Detection starter project, the following labels are supported, standing, running, walking, sitting, and jumping.

If collecting data for the Baby Crying Detection starter project, the supported label is baby_cry.

New labels can be added to the Human Activity Recognition and Baby Crying Detection starter projects, by labeling the data with the new desired label.

6. A new label is created, once the label is created it can be manually adjusted by dragging the sides to make it longer or grabbing the middle of the label to move it.


**Figure 22 Adjust label manually**

7. When all data is labeled, open the project file (*.improj) present in the project folder. Select **Save All** then select **Rescan Data** to ensure all labeling is registered by the project.

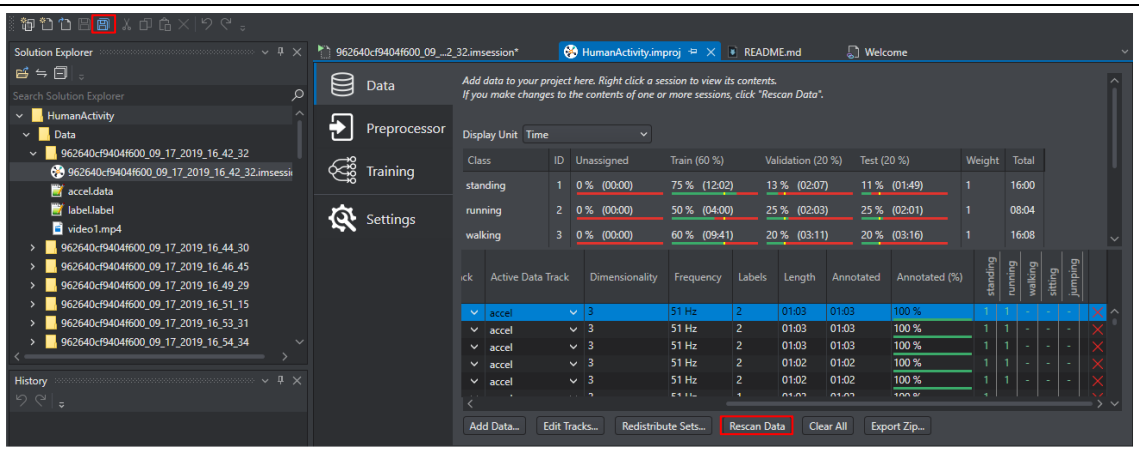


**Figure 23 Rescan project data**

Repeat steps until all new data is labeled. The data is now ready for use to train the model.

### 6.1.6 Train-Validation-Test distribution

Now that all the data is labeled and included in the project, the data needs to be split into the following three groups.

**Train data**: Data used to train the model with

**Validation data**: Data that is used to test the model during training to determine how well the model is performing

**Test data**: Data used at the end of training to determine how well the model performs with data that the model has not been trained with

The standard split between the three data sets is 60 percent training data, 20 percent validation data, and 20 percent test data.

To distribute the data into the three sets:

1. Open the project file (*.improj) present in the project folder and select **Redistribute Sets**.

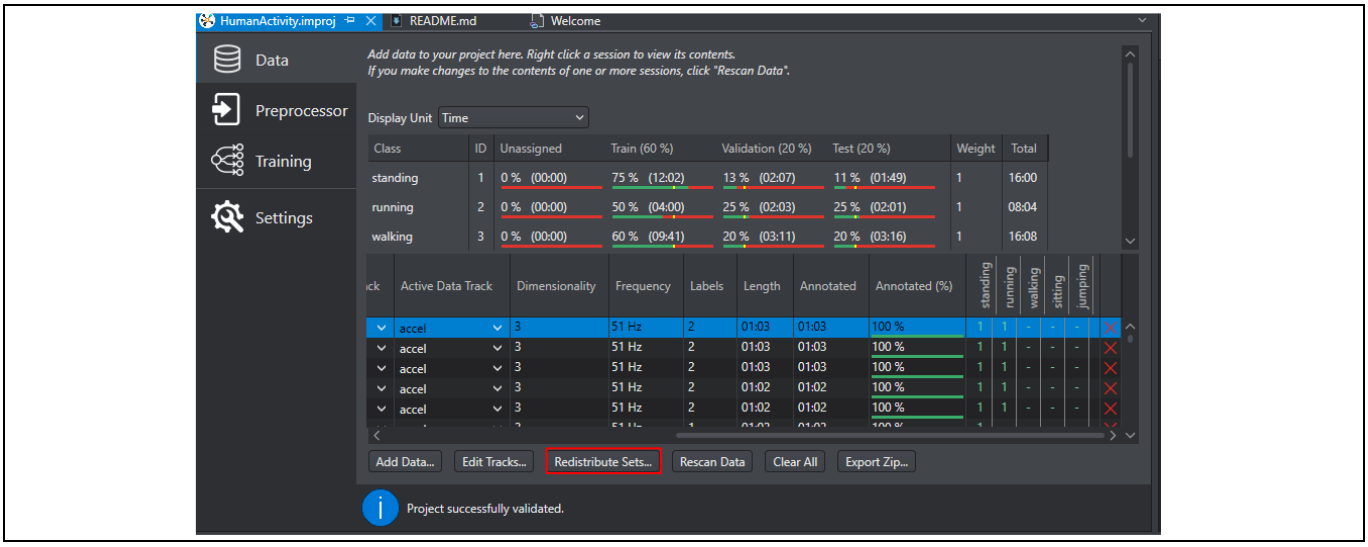


**Figure 24 Select Redistribute Sets**

2. Set **Scope** as either **Redistributing data from All Sets** or from **Unassigned sets**. Use the graphical tool to distribute the data into different data sets.

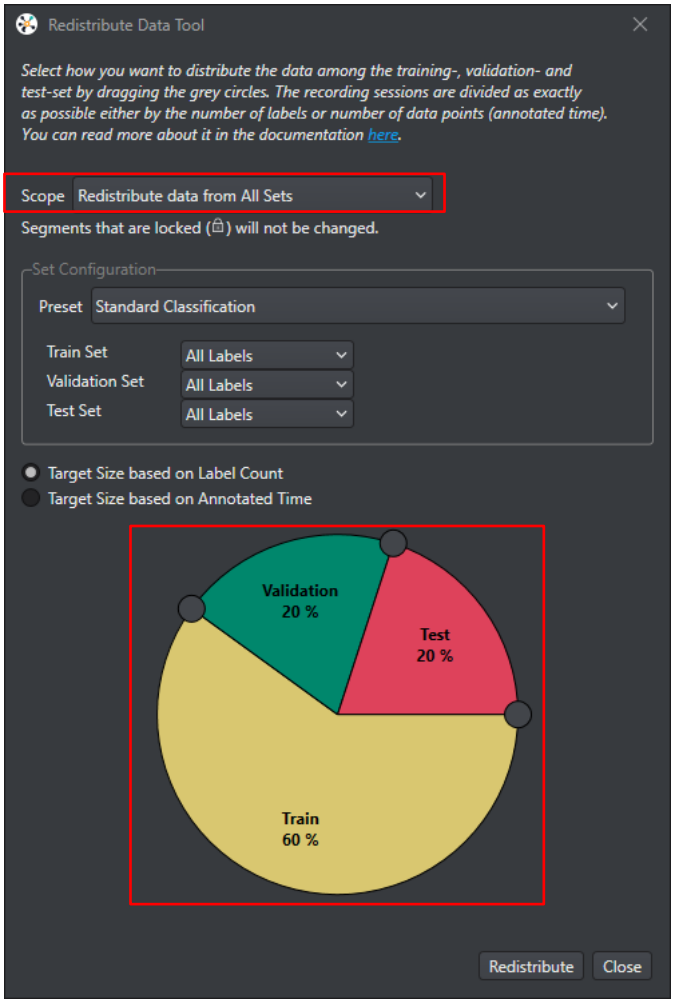


**Figure 25 Redistribute data**

3. Select **Redistribute** and the data in the selected scope will be distributed among the three data sets.

## 6.2 Pre-processing

The second main step in producing a model is pre-processing. This is all the processing done on the collected data before using it to train a model and to run the inference engine.

All data that is going to be passed into the model needs to be processed. Doing this can give a model better accuracy without changing the architecture of the model. Imagimob offers a wide variety of pre-processing layers that can easily be added and configured into your pipeline and later deployed onto an embedded device.

### 6.2.1 Pre-processing configuration

Imagimob Studio allows users to introduce pre-processing into the pipeline using the **Preprocessor** tab after selecting the **.improj* file. Imagimob supports adding many different processing layers that will be used before training the model and can also be exported into C code for an embedded device.

For the Human Activity Recognition and Baby Crying Detection starter projects, the pre-processing layers are already added. More information on the starter project's pre-processing layers can be viewed in Imagimob reference examples section.

At the top of the **Preprocessor** tab are two locked variables called Input Shape and Input Frequency. These are auto generated based on the imported data.

To create a pre-processing pipeline:

1. Select the **Preprocessor** tab.

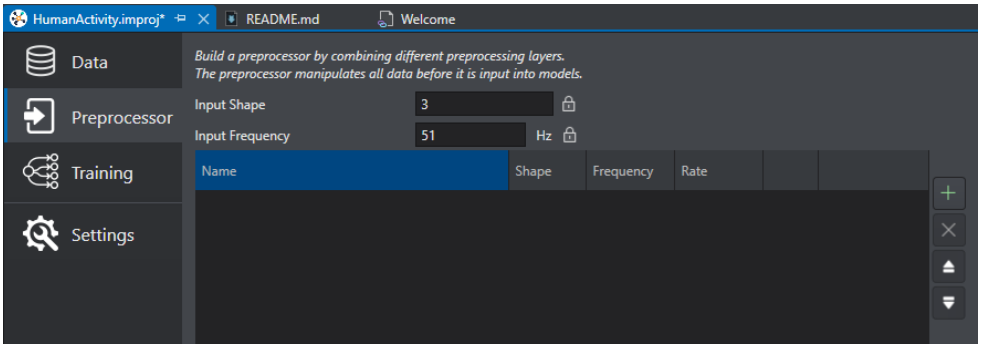


**Figure 26 Preprocessor tab**

2. To add a layer into the pipeline, click the **plus icon** on the right

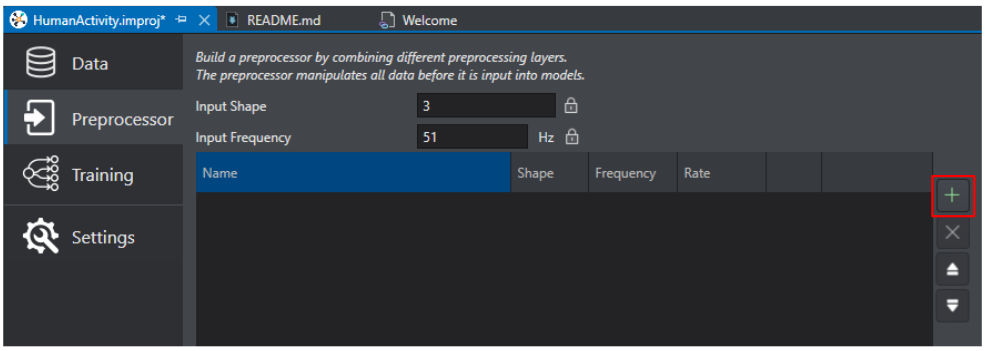


**Figure 27 Add layer into the pipeline**

3. A list of supported pre-processing layers shows up that can be configured and added to the pipeline.

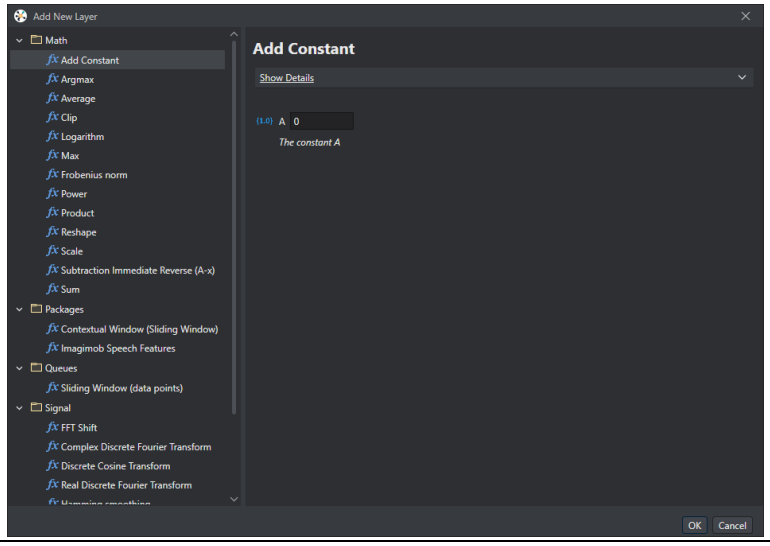


**Figure 28 Supported pre-processing layers**

4. Select the pre-processing layer required. Configure it based on the data being used and select **OK** to bring the layer into the pipeline.

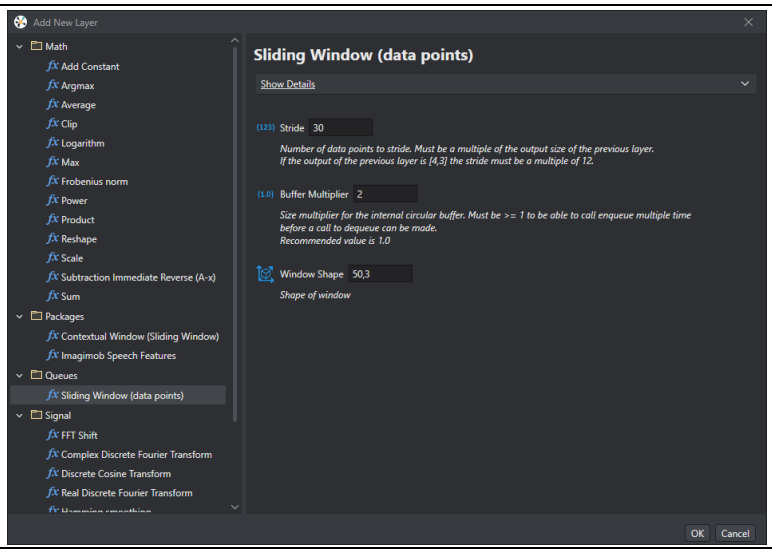


**Figure 29 Configure pre-processing layer**

The sliding window for the Human Activity recognition starter code is shown in the figure. This pre-processor shifts data by 30 data points (10 x, y, z samples), can store two complete data sets, and is configured to produce a window shape of 50 by 3.

5. The pre-processor gives information for the output shape, the frequency at which data will be ready, and the data rate.

6. Steps 2 through 4 can be repeated to build multiple layers in a pipeline which can be seen in the Baby Crying Detection starter project.

For more information on setting up a pre-processing pipeline, see Imagimob's pre-processor documentation.

## 6.3 Model training

Once all data has been collected and the pre-processor is built, data needs to be passed to the model for training. Before the model can be trained the layers of a model must be configured. Imagimob gives users an Auto ML wizard which can generate multiple different models to be trained and compared to find the best fitting model for a given use case. Imagimob Studio also allows for users to select their own model layers and configure each layer individually allowing for a wide range of expertise.

### 6.3.1 Supported layers (MTB-ML)

If using the MTB-ML deployment flow, it is important to note that Imagimob and MTB-ML do not support all the same layers. If a model is generated with layers that are not supported by MTB-ML, then an error is produced when trying to generate source for the model in the MTB-ML configurator. The Imagimob deployment flow generates model source code with Imagimob Studio, which means all layers all supported.

**Table 7 Supported model layers and flow**

| Flow | Supported model layers |
|---|---|
| *MTB-ML* | Activation / Add / AveragePooling1D / AveragePooling2D / BatchNormalization / Clipped RELU / Concatenate / Conv1D / Conv2D / Dense / DepthwiseConv2D / Dropout / Flatten / GlobalAveragePolling1D / GlobalAveragePooling2D / GlobalMaxPooling1D / GlobalMaxPooling2D / InputLayer / LeakyReLU / MaxPooling1D / MaxPooling2D / ReLU / Reshape / SeparableConv2D / Softmax / Transpose / Upsampling |
| *Imagimob* | Activation / AveragePooling1D / AveragePooling2D / BatchNormalization / Convolution1DTranspose / Conv1D / Conv2D / Dense / Dropout / Flatten / GlobalAveragePooling1D / GlobalAveragePooling2D / GlobalMaxPooling1D / GatedRecurrentUnit/ InputLayer / LeakyRelu / LSTM / MaxPooling1D / MaxPooling2D / Rescaling / Reshape / TimeDistributed |

### 6.3.2 Auto ML model wizard

Imagimob Studio's Auto ML wizard allows for the creation of multiple model architectures that are then trained and compared to find the best performing model. The Auto ML wizard can be configured for different model families, classifiers, sizes, and learning rate. The wizard then generates models based on the configuration. This speeds up the model development phase by prioritizing the main features of a model.

For the Human Activity Recognition and Baby Crying Detection starter projects, multiple models with different architectures are included in the project. The pre-defined models can be trained with the newly imported and labeled data by skipping to Model training and results.

To generate a model:

1. Navigate to your project directory and double-click the project file (.improj).
2. Click on the **Training** tab on the left pane.

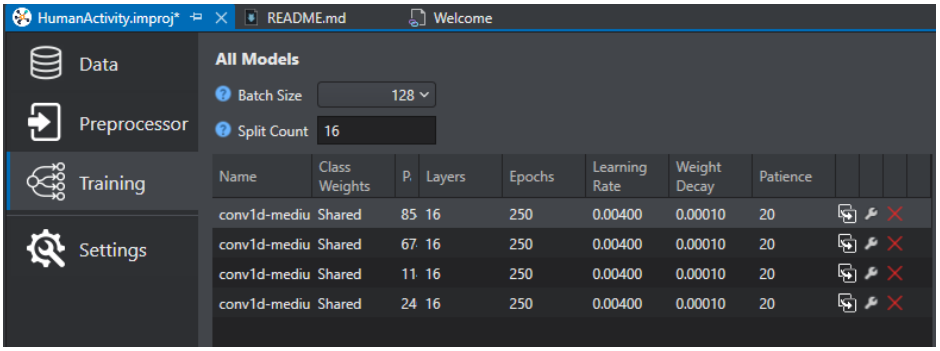


**Figure 30 Training tab**

Figure 30 shows an image from the Human Activity Recognition project which includes defined models. In an empty project this will show no models.

3. Open the Auto ML wizard by selecting the **Generate Model List**.

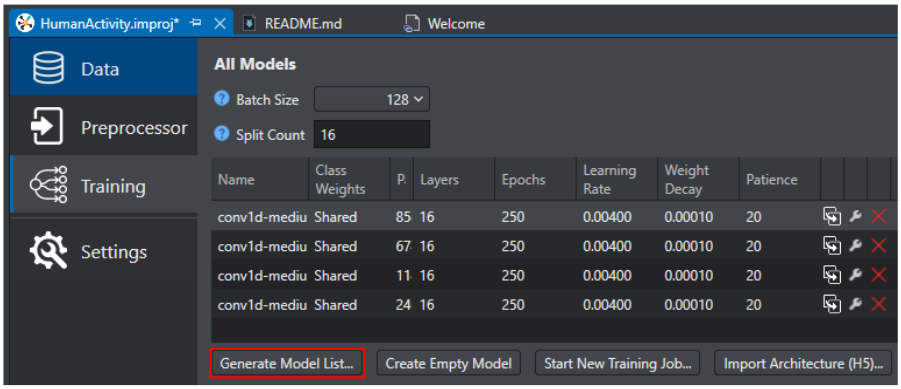


**Figure 31 Generate model list**

4. The **Auto ML** wizard will appear with the following configurable parameters.

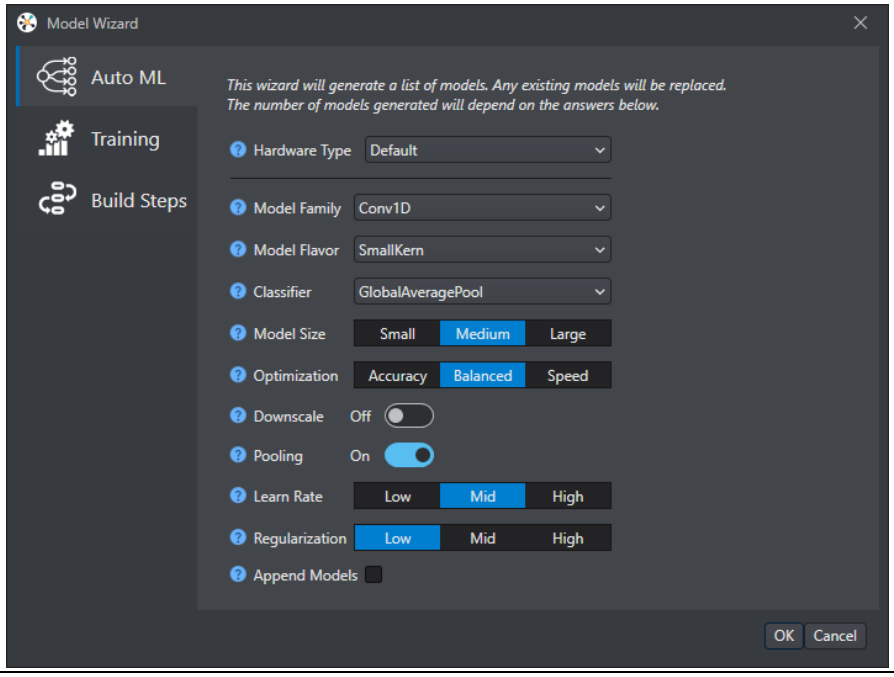


**Figure 32 Auto ML parameters**

**Table 8 Auto ML configurable parameters**

| Parameter | Options | MTB-ML flow |
|---|---|---|
| *Hardware Type* | Default, Syntiant | Default |
| *Model Family* | Conv1D, Conv1DLSTM, Conv2D | Conv1D, Conv2D |
| *Model Flavor* | SmallKern, LargeKern | Application dependent: SmallKern – Quicker Inference time, LargerKern – Higher Accuracy |
| *Classifier* | GlobalAveragePool, Hybrid, Dense | GlobalAveragePool, Dense |
| *Model Size* | Small, Medium, Large | Application dependent: Dependent on how much flash the MCU has available. Larger models tend to have a higher accuracy. |
| *Optimization* | Accuracy, Balanced, Speed | Application dependent: Does the application require high accuracy or quick inference time |
| *Downscale* | On, Off | Application dependent: When turned on, increases model speed and accuracy when the model identifies large features in the input data, otherwise the accuracy can drop |

| Parameter | Options | MTB-ML flow |
|---|---|---|
| *Pooling* | On, Off | Recommended: On - increases speed, lowers memory cost, with minor impact to accuracy |
| *Learn Rate* | Low, Mid, High | Recommended: Mid |
| *Regularization* | Low, Mid, High | Used to reduce over-fitting to the training data |

5. Imagimob Studio offers hyperparameter tuning, these are options such as number of epochs, batch size, and loss function selection. For more information, see Imagimob's Generating model documentation.

6. Select **OK** and the model list is generated.

### 6.3.3 Layer configuration

All models generated by the Auto ML wizard can be changed layer by layer. This allows for fine tuning of the models for more advanced users. The adding, removing, and configuring of model layers works like the pre-processing step.

1. Select one of the models generated by the Auto ML model wizard. This brings up the model architecture.

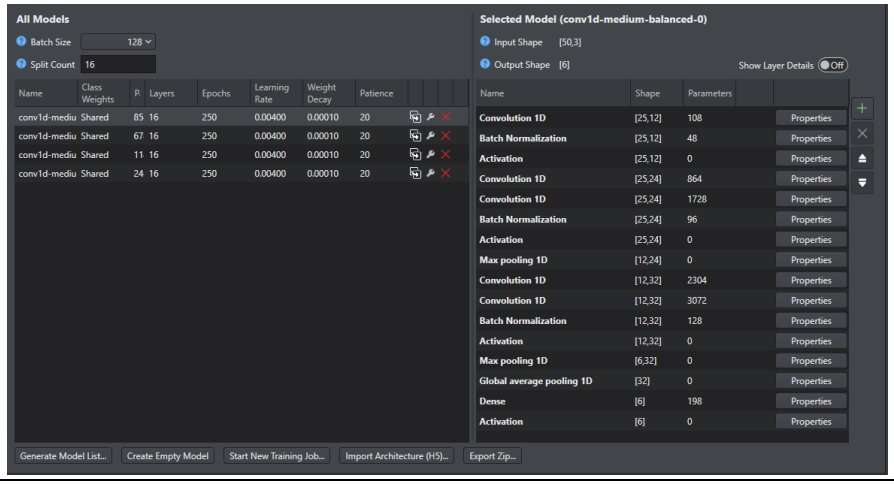


**Figure 33 Selected model architecture**

2. Each of these layers can be removed by selecting the layer and pressing the **delete button (X)** or configured by pressing the **Properties** button.

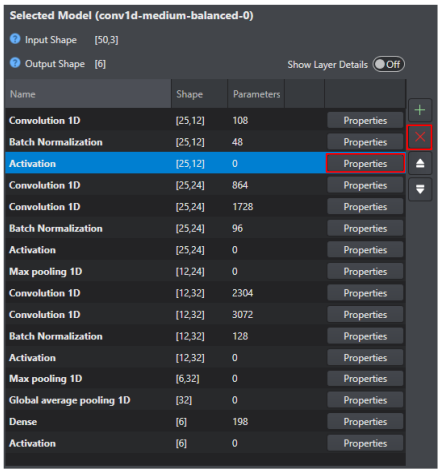


**Figure 34 Remove or configure layers**

3. A layer can be added to the model by pressing the **plus (+)** button. This will bring up the **Add New Layer** window where any supported layer can be configured and added to the model.

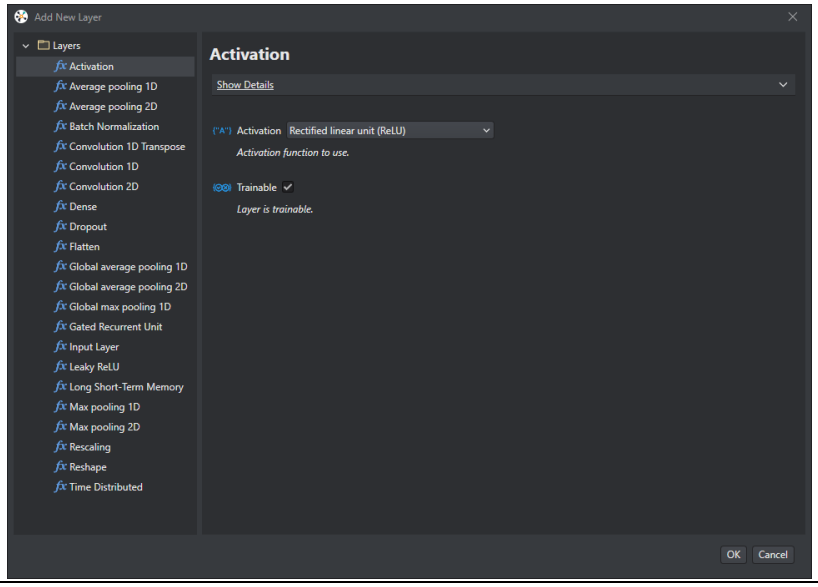


**Figure 35 Add new layer**

## 6.3.4 Model training and results

Imagimob Studio offers many different training statistics and metrics to analyze the strengths and weakness of a model. The statistics include accuracy of validation and test data as well as confusion matrix to see what classifications performed well.

To start training and review the training statistics:

1. Start the model training by selecting **Start New Training Job**.

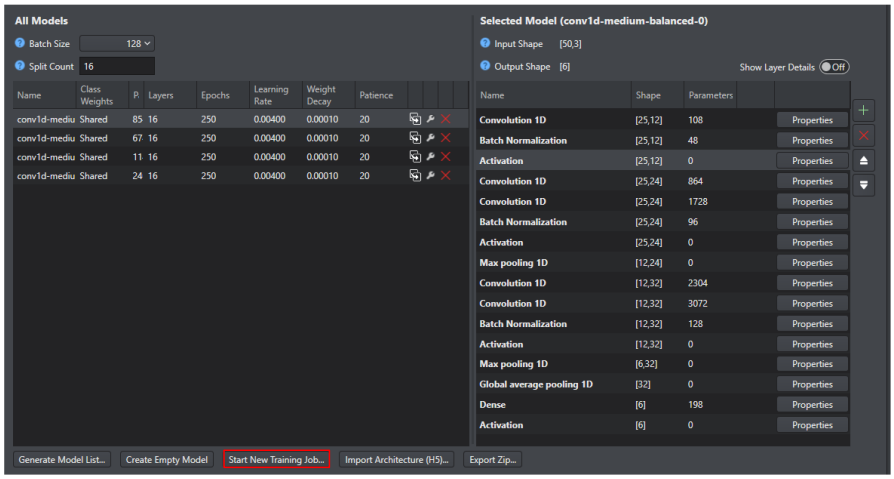


**Figure 36 Start new training job**

2. A prompt directs to the Imagimob account sign-in to start the training.
3. A **New Training Job** window will come up showing the user their available credits and a **Description** window to add comments about the training job. Select **OK** to start the training job.
4. Once a training job is started a window will come up asking if the user wants to open the current job, select **Yes** and the following screen shows up giving stats on the training progress.

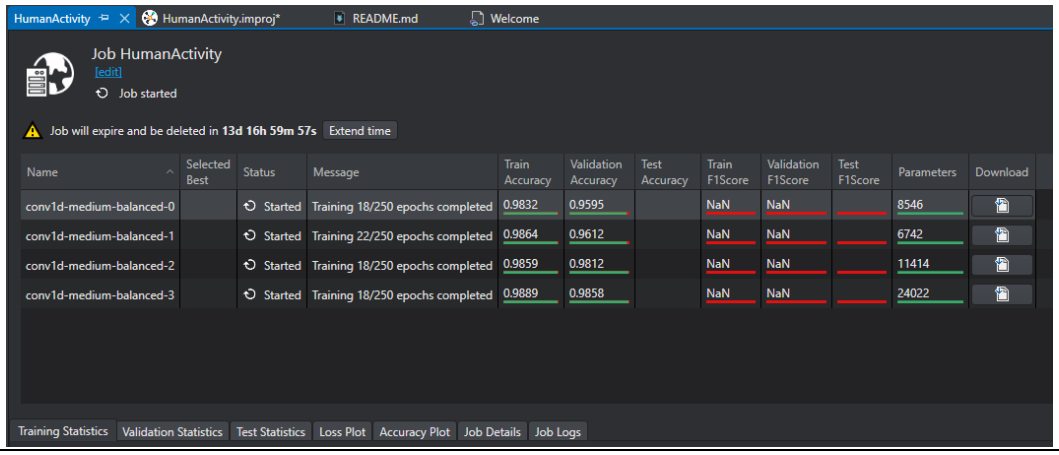


**Figure 37 Training progress stats**

5. Once the **Status** for each model has changed from **Started** to **Completed** the model's results can be validated. Select one of the models and the training statistics for that model as shown in Figure 38.

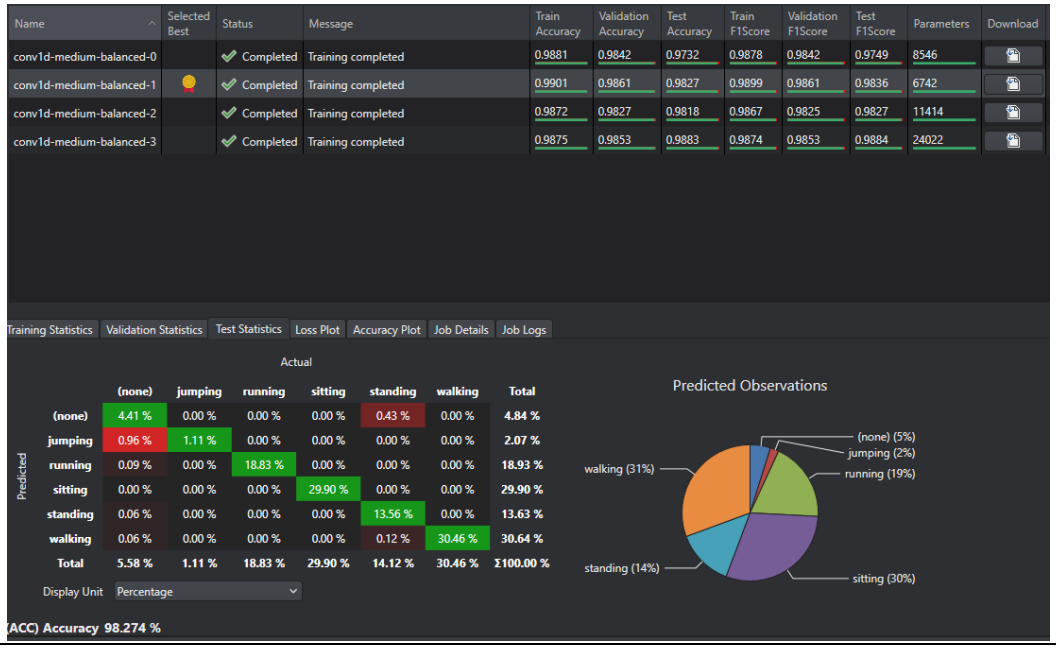


**Figure 38 Model training statistics**

Here users can see the overall accuracy of the model and a confusion matrix for training, validation, and test data. The confusion matrix shows when data was misclassified and may mean that the model layers need to be changed or that more training data needs to be added. For this particular model, jumping was classified as none at a high rate while jumping only made up 2 percent of the predicted observations. This means that more jumping data should be collected for training data to increase the model accuracy.

6. Imagimob Studio compares the model statistics and recommends the best model by putting a star in the **Selected Best** category.

7. Model development is a reiterative process and if a model does not perform as needed this may mean that more training data needs to be added, the pre-processor needs re-configuring, or the model layers should be changed.

8. When a model meets the user's criteria the model can be downloaded using the download icon next to the model and the **Download Model Files** comes up. Select **Download** and pick the *Model* folder in the Imagimob project, then select **OK**.

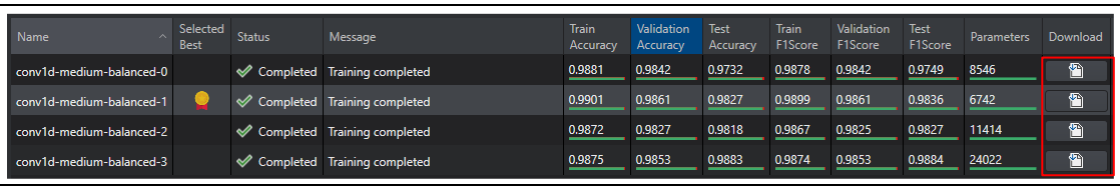


**Figure 39 Download model files**

The model is now downloaded in the *Model* directory of the project.

## 6.4 Deployment

The last step is to deploy the model, which takes a trained model and generates the source code to run on the target embedded device. There are two different flows to deploy the generated model on the embedded device:

- The MTB-ML flow that uses the Imagimob generated model passed through the MTB-ML tooling to get the C implementation
- The Imagimob flow that uses the Imagimob C implementation of the generated model

Each deployment option has a corresponding code example for quick deployment and learning.

When generating the Imagimob Studio source, three functions are provided to interact with the pre-processor or combined pre-processor and model.

**Table 9 Functions to interact with pre-processor or combined pre-processor and model**

| Function | Pre-processor (MTB-ML flow) | Pre-processor + Model (Imagimob Flow) |
|---|---|---|
| *[C_Prefix]_init* | Initializes the pre-processor | Initializes the pre-processor and model |
| *[C_Prefix]_enqueue* | Feeds the pre-processor data | Feeds the pre-processor and model data |
| *[C_Prefix]_dequeue* | Returns a buffer with the pre-processed data, this is then fed to the MTB-ML generated source to get an inference | Returns a model with the confidence of each classification |

For more information see Imagimob's API documentation.

### 6.4.1 Deploy model with ModusToolbox™ ML flow

To deploy with the ModusToolbox™ ML flow, the Deploy with MTB-ML code example is used. See the README.md file from the code example to download, configure the example, and program the device.

The code example requires both the generated .h5 model file as well as the Imagimob generated model.c/.h (pre-processor only) files.

To generate the files required for deployment:

1. Download the model files as seen in Model training and results.
2. Double-click the *.h5 model in the project in the directory *Models/{model name}*

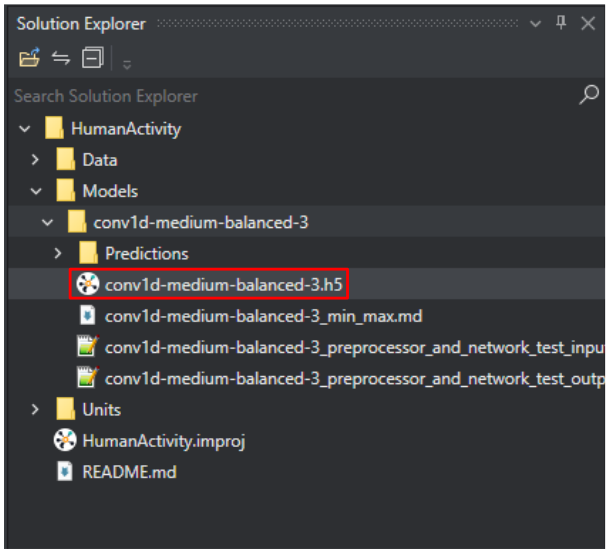


**Figure 40 Select *.h5 model**

3. This will bring up a new page where the **Preprocessor**, **Network**, **Evaluation**, and **Edge** tabs can be accessed. Select the **Edge** tab to bring up the options as shown in Figure 41.

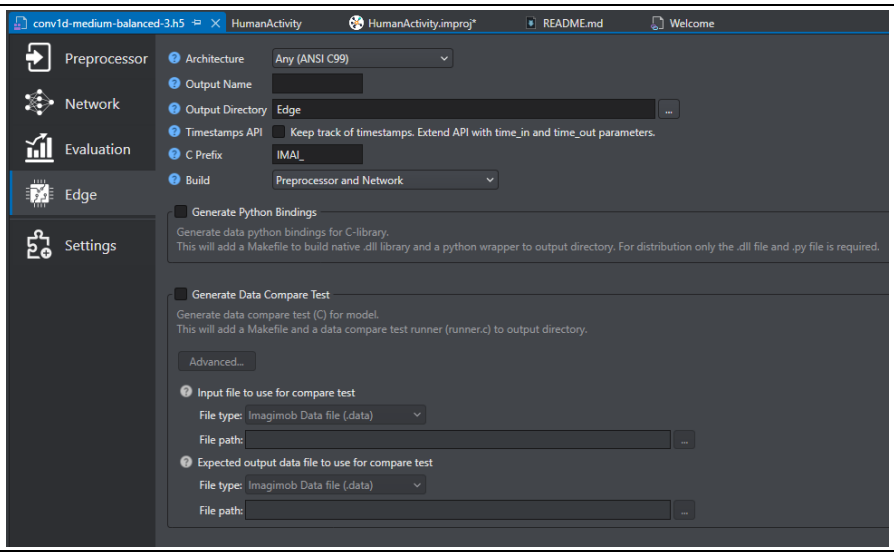


**Figure 41 Edge tab**

The **Edge** tab allows users to generate output that can then be used on an embedded device.

4. To configure the generated models to work with the code example the options need to be configured as given in Table 12.

**Table 10 Build edge parameters**

| Build Edge parameters | IMU based | PDM based | Description |
|---|---|---|---|
| *Output Name* | imu_model | pdm_model | The preprocessor is generated as a .c/h file with the names defined here |
| *C Prefix* | IMAI_IMU_ | IMAI_PDM_ | Defines the prefix for the three functions required to use the preprocessor (init, enqueue, dequeue) |
| *Build* | Preprocessor | Preprocessor | Only the pre-processor is generated in the .c/.h files |

5. Select **Build Edge** to generate the source, an **Edge Generation Report** comes up, select **OK**.
6. The pre-processing .c/h files are generated under Models/{model name}/Edge folder.

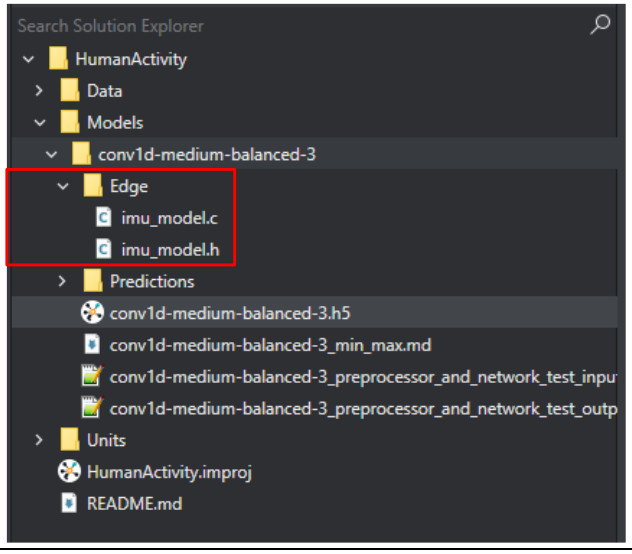


**Figure 42 Pre-processing files**

7. Copy the files as shown in Table 11 from the Imagimob Studio project into the mtb-example-ml-imagimob-MTB-ML-deploy project.

**Table 11 Files to be copied to the project**

| Files | From Imagimob Studio | To mtb-example-ml-imagimob-MTB-ML-deploy |
|---|---|---|
| *{imu/pdm}_model.c/.h* | Models/{model_name}/Edge | Edge/IMU or Edge/PDM |
| *{model_name}.h5* | Models/{model_name} | Edge/IMU or Edge/PDM |

8. The generated source for the .h5 file can now be generated using the ML configurator.

9. Open the **ML Configurator** in the eclipse quick panel or run "make ml-configurator" in the command console and point the **Pretrained model** input field to the model that was copied into the ModusToolbox™ project.

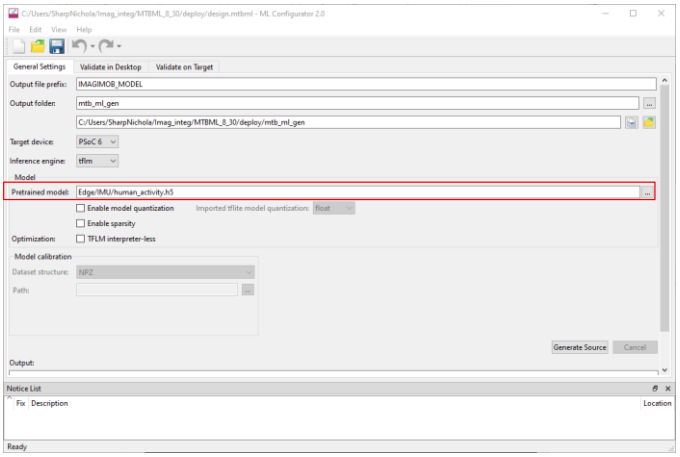


**Figure 43 Pretrained model**

10. Select **Generate Source** to generate model files in the *mtb_ml_gen* folder in the ModusToolbox™ project.
11. To configure the application to run an IMU based model, in the ModusToolbox™ project open *source/config*.h and set `INFERENCE_MODE_SELECT = IMU_INFERENCE`. To configure the application to run a PDM based model, set `INFERENCE_MODE_SELECT = PDM_INFERENCE`.
12. Program the device with the configured application.
13. Open a terminal program and select the KitProg3 COM port. Set the serial port parameters to 8N1 and 115200 baud.
14. The terminal displays the inferencing result as shown in Figure 44. The example uses Human Activity Recognition model.

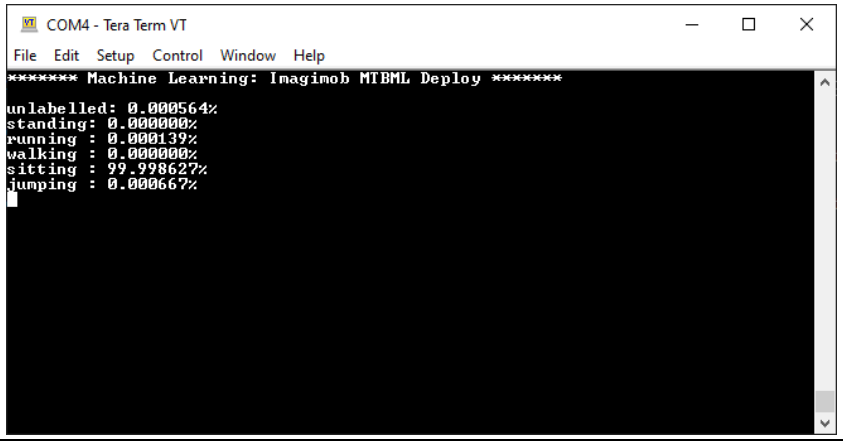


**Figure 44 Inferencing result**

15. If the Human Activity Recognition model was used, then perform one of the following activities standing, running, walking, sitting, or jumping. Use the figures in the Code example is configured for IMU data collection to determine how the device should be held. Watch the confidence change for each classification as an action is completed.

    If using the Baby Crying detection model, play baby crying sounds and watch the confidence change for baby crying as the audio is played.

    For more information see the code example README.md.

### 6.4.2 Deploy model with Imagimob ML flow

To deploy with the Imagimob flow, the Deploy with Imagimob code example is used. See the README.md file from the code example to download, configure the example, and program the device. This code example only supports models generated with IMU data. For PDM based models, see Deploy model with ModusToolbox™ ML flow.

The code example requires the Imagimob generated *model.c/.h* (pre-processor + model).

To generate the files required for deployment:

1. Download the model files as seen in Model training and results.
2. Double-click the *.h5 model in the project under Models/{model name}.

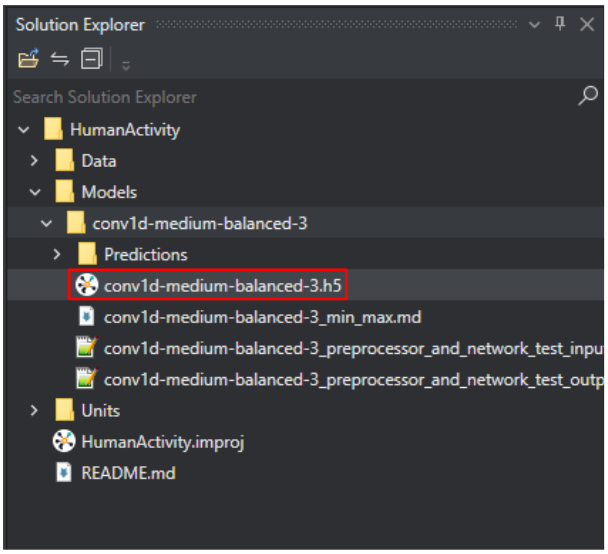


**Figure 45 Select *.h5 model**

3. This will bring up a new page where the **Preprocessor**, **Network**, **Evaluation**, and **Edge** tabs can be accessed. Select the **Edge** tab to bring up the options shown in Figure 46.

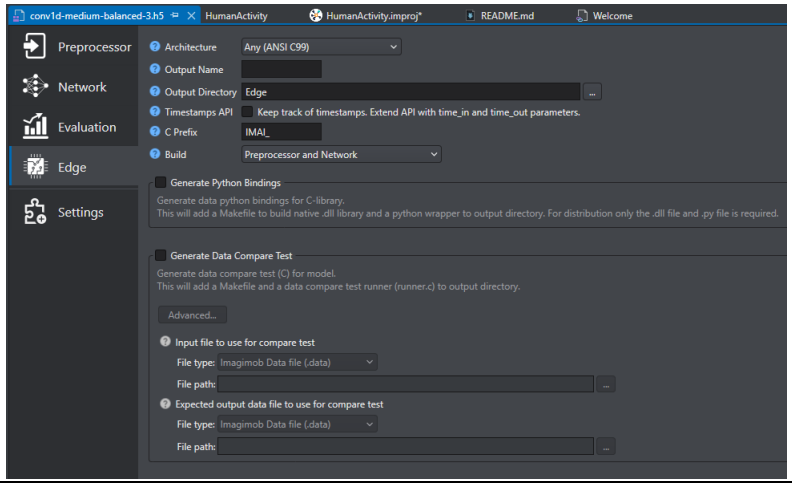


**Figure 46 Edge tab**

The **Edge** tab allows users to generate output that can then be used on an embedded device.

4. To configure the generated models to work with the code example the following options needs to be configured based on Table 12.

**Table 12 Build edge parameters**

| Build edge parameters | IMU based | Description |
|---|---|---|
| *Output Name* | model | The preprocessor is generated as a .c/h file with the names defined here |
| *C Prefix* | IMAI_ | Defines the prefix for the three functions required to use the preprocessor (init, enqueue, dequeue) |
| *Build* | Preprocessor and Network | Only the pre-processor is generated in the .c/.h files |

5. Select **Build Edge** to generate the source, an **Edge Generation Report** comes up, select **OK**.
6. The model .c/h files are generated under Models/{model name}/Edge folder.

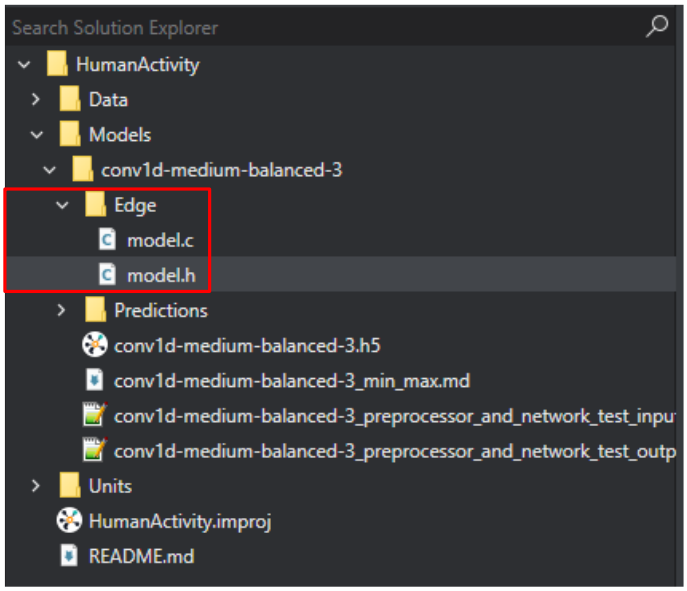


**Figure 47 Model *.c/h* files**

7. Copy the *model.c/h* files from the *Models/{model name}/Edge* folder in the Imagimob project to the *Edge* folder in the mtb-example-ml-imagimob-deploy project.
8. Program the device with the configured application.
9. Open a terminal program and select the KitProg3 COM port. Set the serial port parameters to 8N1 and 115200 baud.
10. The terminal will display the inferencing result as shown in Figure 48. The example uses Human Activity Recognition model.

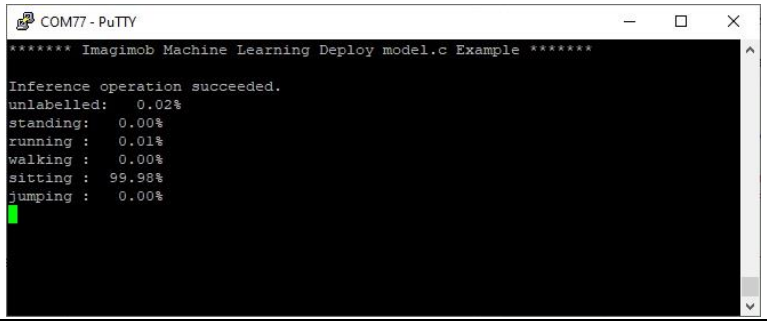


**Figure 48 Inferencing result**

11. If the Human Activity Recognition model was used, then perform one of the following activities standing, running, walking, sitting, or jumping. Use the figures in the Code example is configured for IMU data collection to determine how the device should be held. Watch the confidence change for each classification as an action is completed.

    If using the Baby Crying detection model, play baby crying sounds and watch the confidence change for baby crying as the audio is played.

## References

In this application note, you learned how to use the ModusToolbox™ machine learning solution and Imagimob to solve Machine Learning problems. If you want to learn more about our solution and stay on top of any latest information and releases, visit:

https://www.infineon.com/cms/en/design-support/tools/sdk/modustoolbox-software/modustoolbox-machine-learning/

For the complete list of machine learning code examples available, see the Infineon GitHub repository.

See the following for additional material:

Guides:

- ModusToolbox™ Machine Learning user guide
- ModusToolbox™ Machine Learning Configurator guide
- Imagimob's Developer guide

Code Examples:

- Machine Learning: Imagimob Data Collection
- Machine Learning: Imagimob MTB-ML Deployment
- Imagimob Machine Learning Model Deployment

Machine Learning Pack:

- https://softwaretools.infineon.com/tools/com.ifx.tb.tool.modustoolboxpackmachinelearning

GitHub libraries:

- https://github.com/Infineon/ml-inference
- https://github.com/Infineon/ml-middleware
- https://github.com/Infineon/ml-tflite-micro

# Revision history

| Document revision | Date | Description of changes |
|---|---|---|
| ** | 2023-09-12 | New version. |

**Trademarks**

All referenced product or service names and trademarks are the property of their respective owners.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc., and any use of such marks by Infineon is under license.

**Edition 2023-09-12**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**



**Do you have a question about this document?**

**Email:** erratum@infineon.com

**Document reference**

**002-38663 Rev. ****

**Important notice**

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

**Warnings**

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.