# Application Software Pack: Machine Learning State Monitor

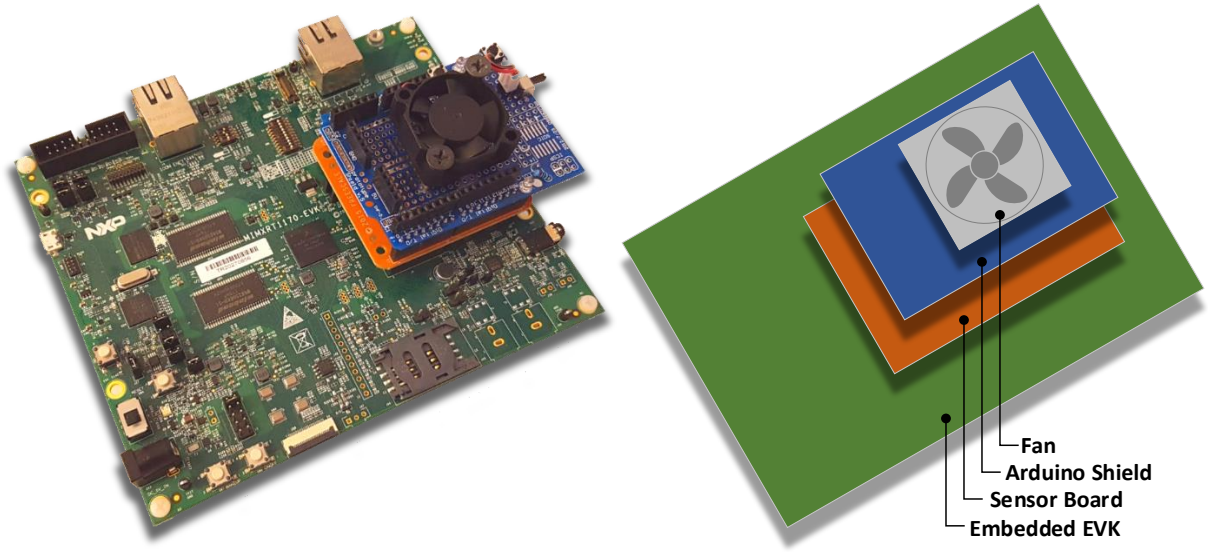
ML Dataset Creation Guide - Revision 2

# Contents

# 1 Introduction

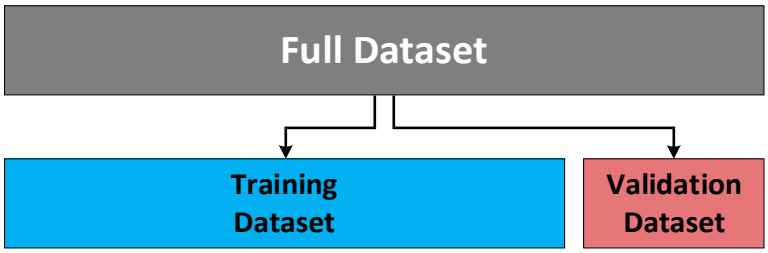
Machine learning (ML) gives systems the ability to learn from examples and generates rules to solve a particular problem based on past experience and historical data. In other words, an ML-based system learns from data to identify patterns and generates decisions without human intervention.
The presented project relies on the Deep Learning (DL) subfield of ML and builds a Convolutional Neural Network (CNN) with free parameters that is trained over a set of data to generate a fit model with fixed parameters that will be used to make accurate predictions and monitor the state of a system. The process of training relies on Supervised Learning (SL) method to learn from the input labelled dataset and to adjust the parameters until the model has been fitted properly to make accurate predictions on new unseen examples.

This guide presents how the provided example dataset was collected to solve the exemplified demo and will also detail how to collect data and build a new dataset.
The demo setup was built around of 5V fan placed on top of the embedded board and its purpose is to analyse the vibration pattern and to determine the state of the system.

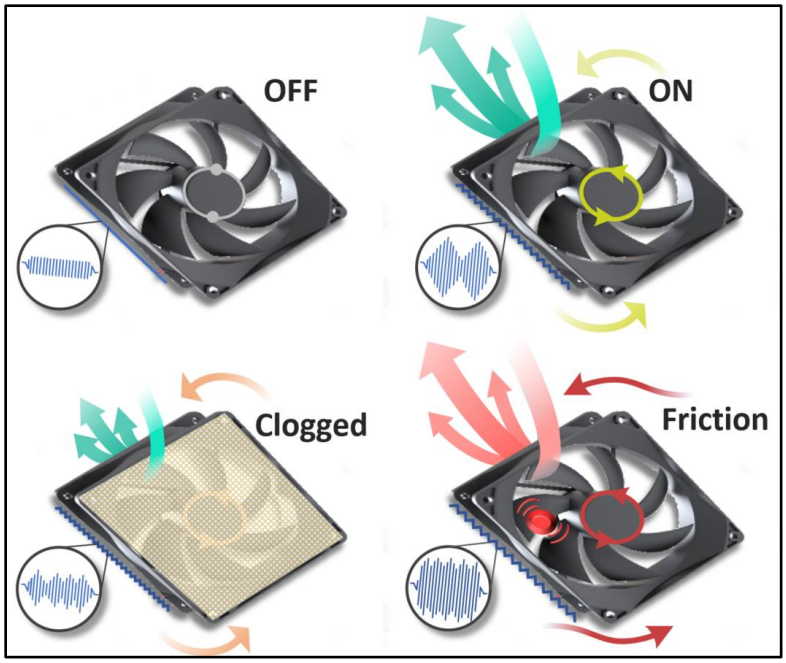



# 2 Example dataset

The ultimate goal of any machine learning model is to learn from examples (training data) in such a way that the model is capable to generalize the learning to new unseen instances (validation/testing data). Therefore, we need to collect data covering as much variation as possible following a training and validation split. For the provided example dataset an approx. 80%-20% split was used.

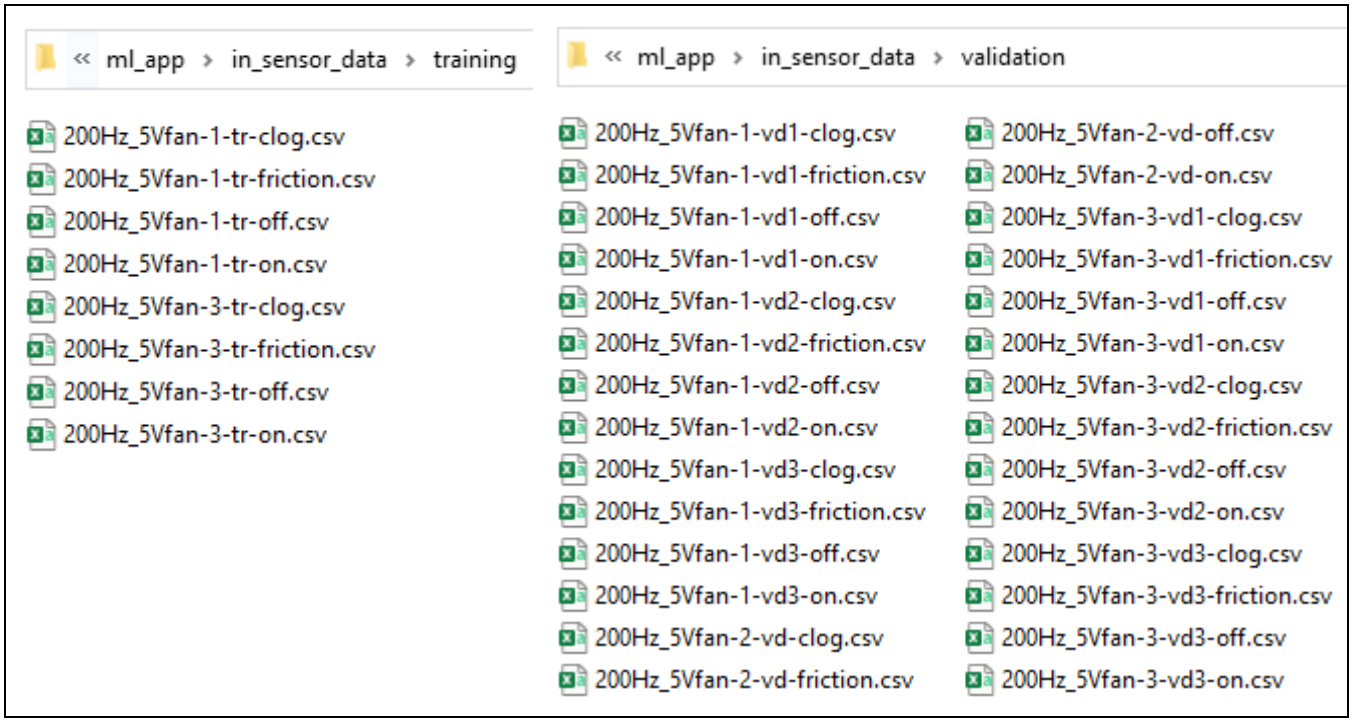


In this example dataset there are four fan states that were recorded: FAN-OFF, FAN-ON (normal operation), FAN-CLOG (something is covering the fan, so it does not get the normal airflow), FAN-FRICTION (something is creating drag on the blades like a piece of tape or cardboard). The setup was physically configured for each class and the data were collected for every particular one.




The main rule for collecting the data is to cover as much as possible every corner case and to not use the same data for training and validation. Other than that, any pattern for duration and how to switch between classes can be followed, depending on the use case and the analysis behind that.

For the presented case study the data were collected in three sessions of recording, cumulating 7440000 samples collected in 10 hours and 20 minutes using a 200Hz sampling frequency.



Each recording session collected a balanced dataset (data collected equally improves the model performance by providing equal distribution and priority for each class) trying to cover as much variation as possible. Sessions 1 and 3 of recording, cumulating 10 hours of recording, contain both training and validation data while session 2 contains only validation data recorded for 20 minutes.

The recording in sessions 1 and 3 was started by collecting a validation dataset for a short period of time (5mins), followed by a dataset for training and validation collected for a longer period (1h + 5mins) and ending with a validation dataset collected again for a short period of time (5mins). For the demonstrated case study was quite easy to collect data for the classes OFF and ON while for the Clogged class a thin piece of cardboard was used to block the airflow and for the Friction class a corner of the cardboard was gently pressed on the blades to produce friction.


The dataset files are stored in CSV (Comma Separated Value) format and each instance (features row) within these files captures the state of the fan, the sample time, the X, Y, and Z axis of the accelerometer, the X, Y, and Z axis of the magnetometer, and the temperature (T). Our model will only use the time and accelerometer data, but other applications may find those other fields useful for their deep learning models.

| class | time[ms] | ax | ay | az | Bx | By | Bz | T |
|---|---|---|---|---|---|---|---|---|
| FAN-CLOG | 5 | 49 | 98 | 4162 | 42 | 93 | -11 | 24 |
| FAN-CLOG | 10 | 81 | 153 | 4140 | 57 | 96 | -2 | 24 |
| FAN-CLOG | 15 | 24 | -113 | 4038 | 51 | 92 | 0 | 25 |
| FAN-CLOG | 20 | -41 | -256 | 3958 | 32 | 89 | 4 | 24 |
| FAN-CLOG | 25 | 4 | -69 | 4094 | 50 | 101 | 5 | 22 |
| FAN-FRICTION | 30 | 15 | -184 | 3989 | 36 | 82 | 8 | 25 |
| FAN-FRICTION | 35 | 117 | 13 | 4102 | 44 | 101 | 8 | 26 |
| FAN-FRICTION | 40 | -9 | -25 | 4078 | 52 | 82 | -3 | 25 |
| FAN-FRICTION | 45 | -21 | -97 | 3975 | 37 | 87 | 0 | 25 |
| FAN-FRICTION | 50 | 40 | 64 | 3888 | 49 | 77 | 34 | 26 |
| FAN-OFF | 50 | 40 | -64 | 4065 | 58 | 83 | 18 | 25 |
| FAN-OFF | 55 | 37 | -64 | 4069 | 64 | 74 | 33 | 24 |
| FAN-OFF | 60 | 29 | -66 | 4092 | 73 | 91 | 33 | 25 |
| FAN-OFF | 65 | 40 | -70 | 4070 | 64 | 100 | 25 | 25 |
| FAN-OFF | 70 | 30 | -73 | 4085 | 74 | 83 | 27 | 25 |
| FAN-OFF | 75 | 30 | -63 | 4100 | 58 | 94 | 41 | 24 |
| FAN-ON | 80 | -72 | -58 | 4079 | 46 | 87 | 13 | 24 |
| FAN-ON | 85 | 114 | -73 | 4049 | 65 | 86 | 21 | 26 |
| FAN-ON | 90 | -1 | -38 | 4145 | 47 | 83 | -6 | 25 |
| FAN-ON | 95 | 30 | -95 | 3892 | 56 | 87 | -2 | 24 |
| FAN-ON | 100 | 76 | -64 | 4240 | 42 | 88 | 6 | 25 |

# 3 Create a new dataset

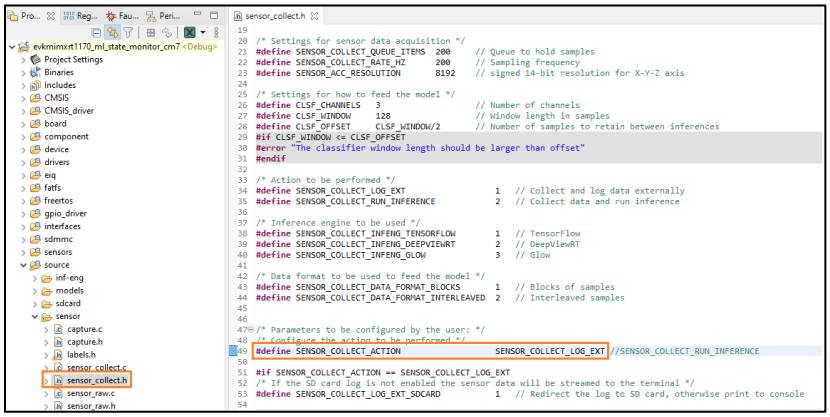
Unlike image or audio, time series sensor data are often unique for the product setup, depending on sensor type, sensor placement, location, surface, etc. Because of this we cannot use datasets already available so we need to create (collect and annotate) a dataset relevant for a particular setup.

New datasets can be collected by enabling the embedded application to log the sensor data on the SD card or over the serial debugging interface and then transfer the recorded data to the host machine. The following following sections are focused on the SD card method and detail how to collect new datasets and prepare them for model training on the host machine.

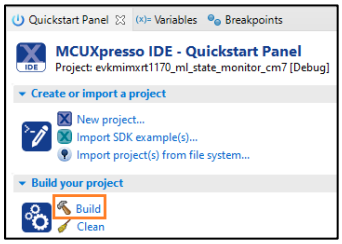
## 3.1 Collect data

The project will need to be modified to collect accelerometer data and store it on an SD card.

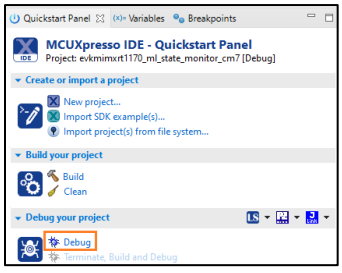
1. Put in a microSD card into the board and set the fan on top of the board
2. In MCUXpresso IDE, open the **sensor_collect.h** file and on line 49 change the #define for **SENSOR_COLLECT_ACTION** to **SENSOR_COLLECT_LOG_EXT** as shown in the image below.



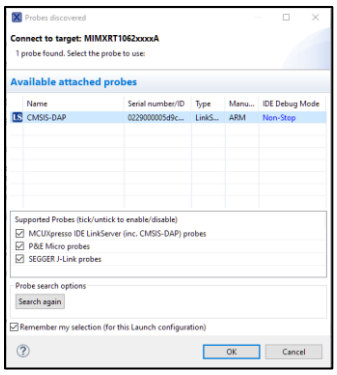

3. Next build the project by clicking on “Build” in the Quickstart Panel and make sure there are no errors.



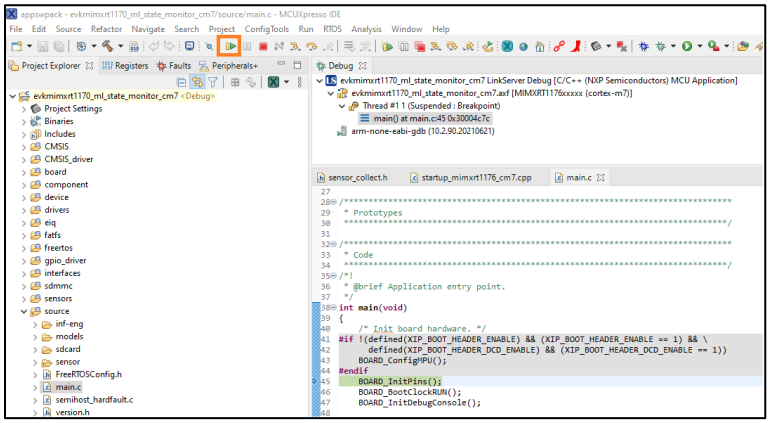

4. Plug the micro-B USB cable into the board at J11 on the i.MXRT1170 board.
5. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.
6. Debug the project by clicking on “Debug” in the Quickstart Panel.



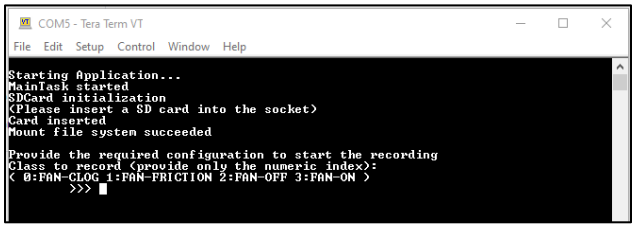
7. It will ask what interface to use. Select CMSIS-DAP.



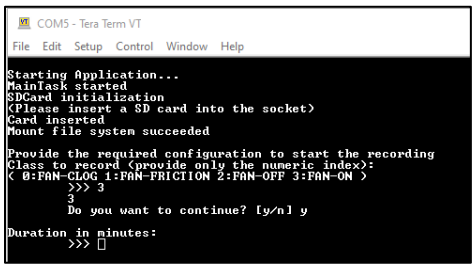

8. The debugger will download the firmware and open up the debug view. Click on the Resume button to start running.



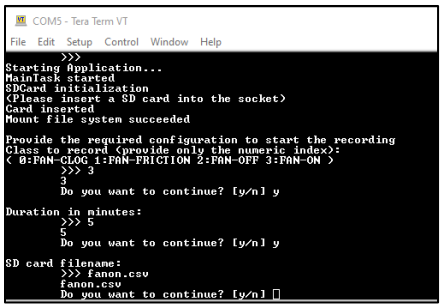
9. The following text should appear in the terminal program:



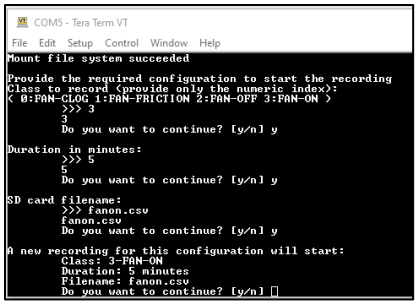

10. For this demo there are four fan states that are going to be detected:
    - FAN-OFF
    - FAN-ON (normal operation)
    - FAN-CLOG (something covering the fan so it does not get the normal airflow)
    - FAN-FRICTION (something creating drag on the blades like a piece of tape or cardboard)
11. Physically put the fan in the desired state. For this first run, lets put the fan into the normal ON state.
12. In the terminal, type of the number of the fan state that corresponds with what the physical fan state is. So for this example, type ‘3’ for FAN-ON.
13. The next selection is how many minutes to gather data. There is no hard and fast rule on how much data needs to be collected, but at a minimum several minutes worth will be required. For this simple example we’ll select 5 minutes, but for more complex examples it could be significantly longer to ensure good training results. The key goal is to cover every corner case as much as possible and to not use the same data for training and validation.



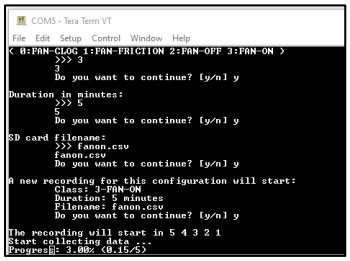

14. Next give a filename where this data will be stored in CSV (Comma Separated Value) format. It can be any name but should be descriptive and cannot be more than 12 characters in length. For this example lets use fanon.csv




15. Finally it will display your selections. Type ‘y’ to then start the data collection:



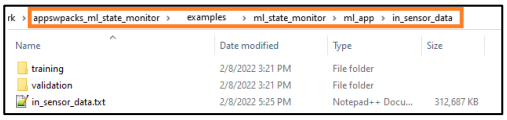

16. You will see the status as it collects the data.
17. Once completed, then you can either remove the SD card to look at the data on your computer or hit the Reset button on the board to start the process again to collect validation data for FAN-ON or to collect the other fan states. You’ll want to collect data for all 4 conditions and collect both Training data and Validation data so there should be at least 8 runs. The same amount of data collection time should be used for all data sets.



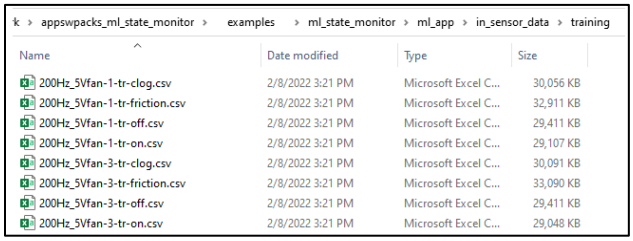
## 3.2 Copy Training Data to PC

Once all the data required has been saved to the microSD card, remove it from the RT1170 EVK and put it into a card reader for your PC so the data can be copied to the hard drive.

1. Inside the project folder there is a folder named **\appswpacks_ml_state_monitor\examples \ml_state_monitor\ml_app\in_sensor_data**. Locate that directory

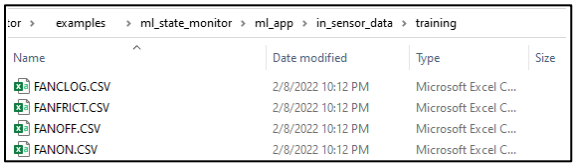



2. Inside that folder there are two subfolders, one named training and the other validation. By default they contain pre-created data store in .csv files that were gathered from a simple 5V fan.

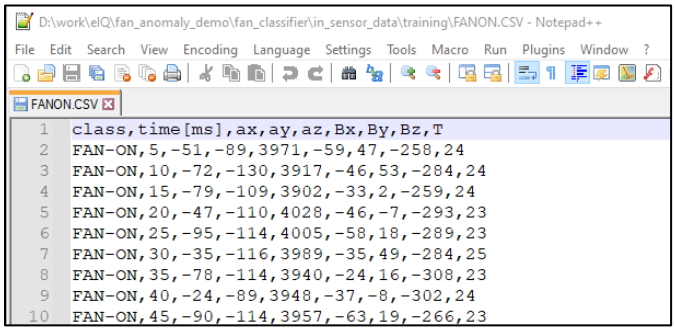



3. These files should be deleted, and the newly collected data placed inside these folders in order to create a new custom model. The file name does not matter but should end in .csv. The files should cover all the possible classes and should be updated for both the **training** and **validation** folders with their respective training and validation data.

   If you did not collect new data, you can use the pre-existing .csv files as that has data collected with a simple 5V brushless DC motor.




4. The Python scripts that will be used to load the data and train a model will assume the files will be in the following Comma Separated Value format, which is how the data was written to the files by using the projects from the previous section, so no modifications should be needed.




# 4 Conclusion

This guide demonstrated how the example dataset was collected and how to gather new sensor data to be used for to train new models.