AN12766

Anomaly Detection with eIQ using K-Means clustering in Tensor Flow Lite

Rev. 0 — 18 March 2020 Application Note

Contents

1 Introduction

This document provides step by step instructions to enable a machine condition monitoring application using anomaly detection. The application creates the model using k-means clustering algorithm with Tensorflow framework and python. Once the model is trained and developed, then it is converted to Tensorflow Lite (TF Lite) and finally to header file so that it can be exported to i.MX RT1050 board.

2 Applications support

NXP eIQ Software Development Environment (SDE) can support several types of applications such as:

- Facial Recognition
- Anomaly detection
- Object detection
- Voice recognition
- Image processing
- Sensor Drivers
- Vision Apps
- Audio Front End

This application note focuses on Anomaly Detection by collecting accelerometer data for Machine Condition Monitoring.

3 Application setup

For the training data, the i.MX RT1050 EVKB is mounted on a table fan and collect the vibration data as shown in Figure-1. The data is exported to an xls sheet for different conditions. The collected data is the raw X, Y, data on which we require to do further analysis. We need to get the real-time data of machine for monitoring its condition in healthy state and in fault state. For monitoring the machine health, we use FXOS8700 accelerometer available on the i.MXRT1050 which communicates with i.MX RT1050 over I2C.

3.1 Hardware

NXP i.MX RT1050 EVKB

Figure 1. i.MX RT board setup

3.2 Software

SDK_2.6.0_EVKB-IMXRT1050 or later

The application monitors the fan to detect anomalies in vibration. For the training data, the i.MXRT1050 EVKB is mounted on a table fan to collect the vibration data.

The data is exported to an xls sheet from the serial console for different conditions by a python script. The collected data is the raw X, Y, Z axis data on which we require to do further analysis.

4 Model selection

Once you have the application and requirement in place, you have to select the type of learnings for the model that best suits your application. There are different types of learnings like:

- Supervised Learning: Training data includes desired outputs
- Unsupervised Learning: Training data does not include desired outputs
- Semi-supervised Learning: Training data includes a few desired outputs
- Reinforcement Learning: Rewards from a sequence of actions

We have taken references from several white papers for anomaly detection and found that most of them are using unsupervised or semi supervised learning for this kind of application. So we decided to try both and then conclude which suits best for our application.

Here we have the application of anomaly detection where we first selected unsupervised learning where we would train the model with only healthy data and detect as anomaly in case we receive any data out of healthy conditions. But the drawback of this

approach would be that the model would detect anomaly for very less change in vibrations and it would be difficult for us to create such an atmosphere where there is no change in vibrations.

Due to the above reason, we switched to the semi-supervised learning where we include some anomalies along with the healthy data for training the model. So in this case the model is trained with both healthy and anomalous data and it would detect anomalies of all types.

References

- https://www.intechopen.com/books/artificial-intelligence-emerging-trends-and-applications/artificial-intelligence-application-in-machine-condition-monitoring-and-fault-diagnosis
- http://wiredspace.wits.ac.za/bitstream/handle/10539/5482/VILAKAZI_DISSERTATION.pdf?sequence=1&isAllowed=y
- https://towardsdatascience.com/machine-learning-for-anomaly-detection-and-condition-monitoring-d4614e7de770

5 Data collection

The vibration data on which we are going to train the model needs to be analyzed before applying it to the model. The first step is to collect the real-time vibration data. As we have selected semi-supervised learning, so we would use the application (Data_collection.py script) that we have developed for data collection and create anomalies in between by tapping from different sides.

The application for data collection integrates the FXOS800CQ configured with the following settings:

- Mode: Accelerometer only
- Output Data Rate: 400 HZ
- I2C Baud Rate: 1 MBPS
- I2C read rate: 1.5 Ms

The application constantly reads the vibration data from accelerometer over I2C as per the read rate and dump the data on serial console for all the three X, Y, & Z axis. We have developed a “Data_collection.py” python script which reads the data from serial console, parses it and creates a workbook where it copies this data of all three axis in separate columns. You can configure the amount of time for what you want to collect the data. The script would collect the data for that much time and then automatically close the console and save the data. You can kill the process at any time as the sheet is saved every 1 minute so that you do not lose data in case you kill the program before configured time.

6 Data analysis

During collection of this data, we also keep a track of the anomalies; for example, the number of times we created anomalies in one sheet, total number of anomalies created, direction of anomaly. Once the data collected, it needs to be analyzed so that we can decide filtering and feature extraction method required before feeding it to the model. We have done the frequency and time domain analysis of the collected data.

6.1 Time domain analysis

In the time domain analysis, we have plotted the graph of captured data in the sheet and observed the peaks of data. By doing time domain analysis of vibration data, we found that RMS value of waveform is useful feature to detect anomaly in the vibration.

6.2 Frequency domain analysis

For the frequency domain analysis, we have written a python script to generate the Fast Fourier Transform (FFT) of original and filtered data. You can find the script for FFT in the release package under “Scripts” Folder. Observing the harmonics of healthy and anomalous data from the FFT graphs we found that the overall vibration has the frequency, ranging from 100 – 300 HZ. As per the Nyquist criteria, the sampling rate should be greater than 2 times of the frequency. So as per this calculation we have kept the sampling rate as 666.66 HZ which leads to 1.5 milliseconds of data collection. We also observed few harmonics which need to be removed for effective feature abstraction from vibration data.

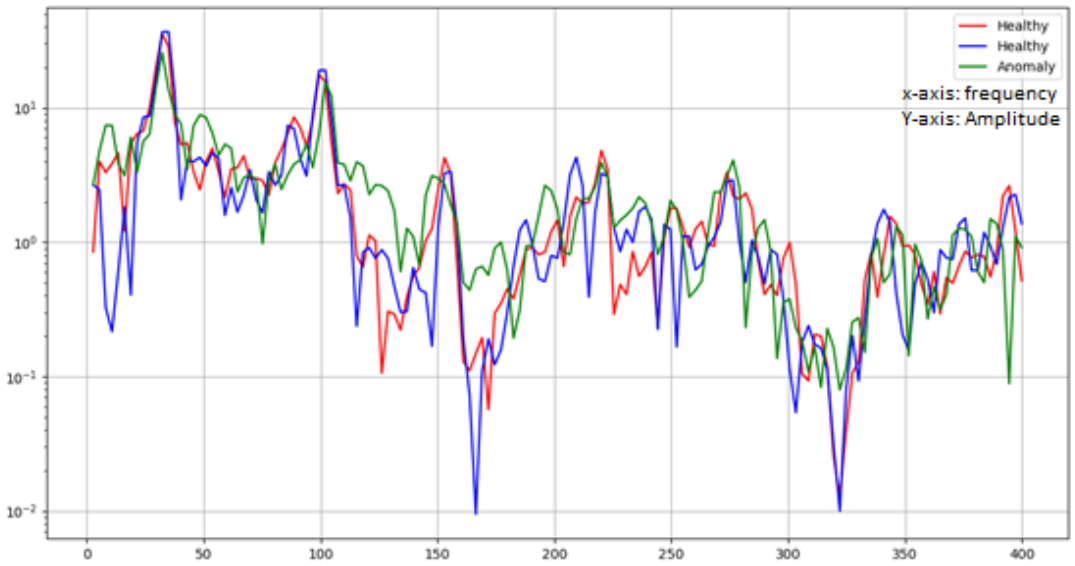

Figure 2. X axis Filtered

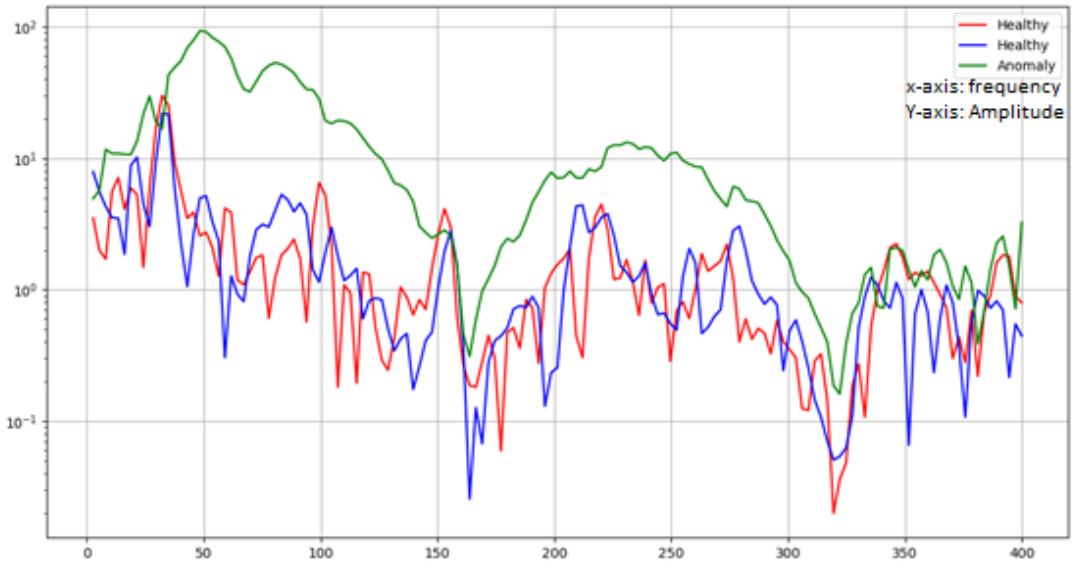

Figure 3. Y axis Filtered

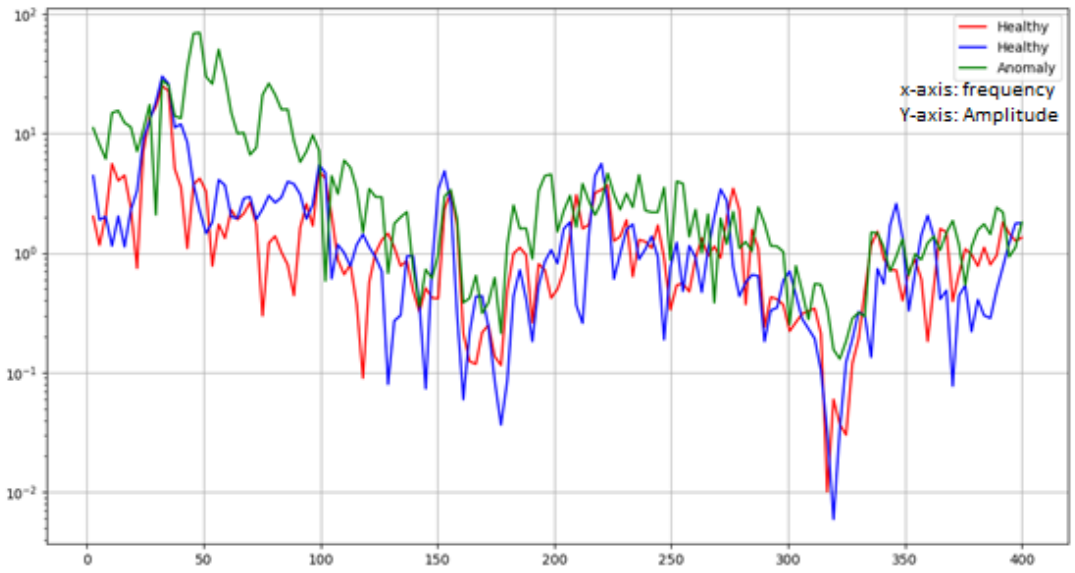

Figure 4. Z axis Filtered

7 Data pre-processing

7.1 Low-Pass filtering

After collecting the data, it needs to be filtered which removes the unwanted noise and glitches from the data. Here after doing the frequency domain analysis we have decided to apply the moving average filter taking the average of 5 samples. After applying this filter, we had plotted the graph of the filtered data and observed that the data was much smoother, and all unwanted glitches were removed. This graph was plotted in the excel sheet as shown in Figure 5.

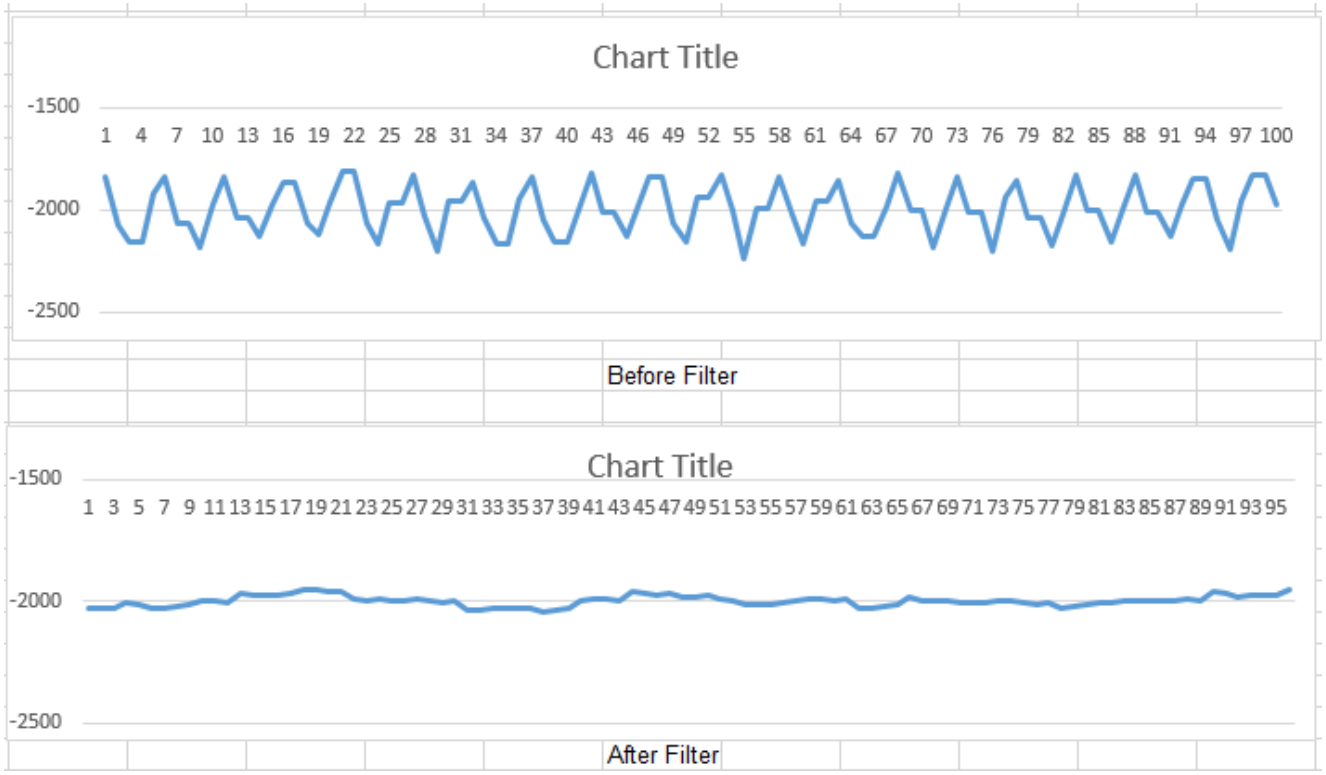

Figure 5. Low-pass filer graph

8 Feature extraction

After the filtering process, we have smooth data so now it is ready for feature extraction. By doing time domain analysis of vibration data, we found that RMS value of waveform is useful feature to detect anomaly in the vibration. We have taken the RMS of 10 samples. This number is decided on trial and error basis after observing the results for different values. This number is to be decided in such a way that the difference between RMS values of healthy data and anomalous data is large enough to be easily separated by the model.

The application reads the accelerometer raw data of X, Y& Z axis every 1.5 ms and sends it to PC over UART interface.

Next we have developed a python script which reads the data from UART and parses it to get the data of X, Y, Z axis and dump it in xls file.

After successfully creating the xls file with parsed X, Y, Z axis, data we prepare an insertion chart for all the three axis to analyze the data. We run this test many times and collect the data in different conditions like normal condition and anomalous condition like taping in between, moving the board horizontally, moving the board vertically. After collecting the data in different conditions, we analyze it manually by plotting the graphs and capturing the anomalies there.

Model development with Tensorflow is shown in Figure 6

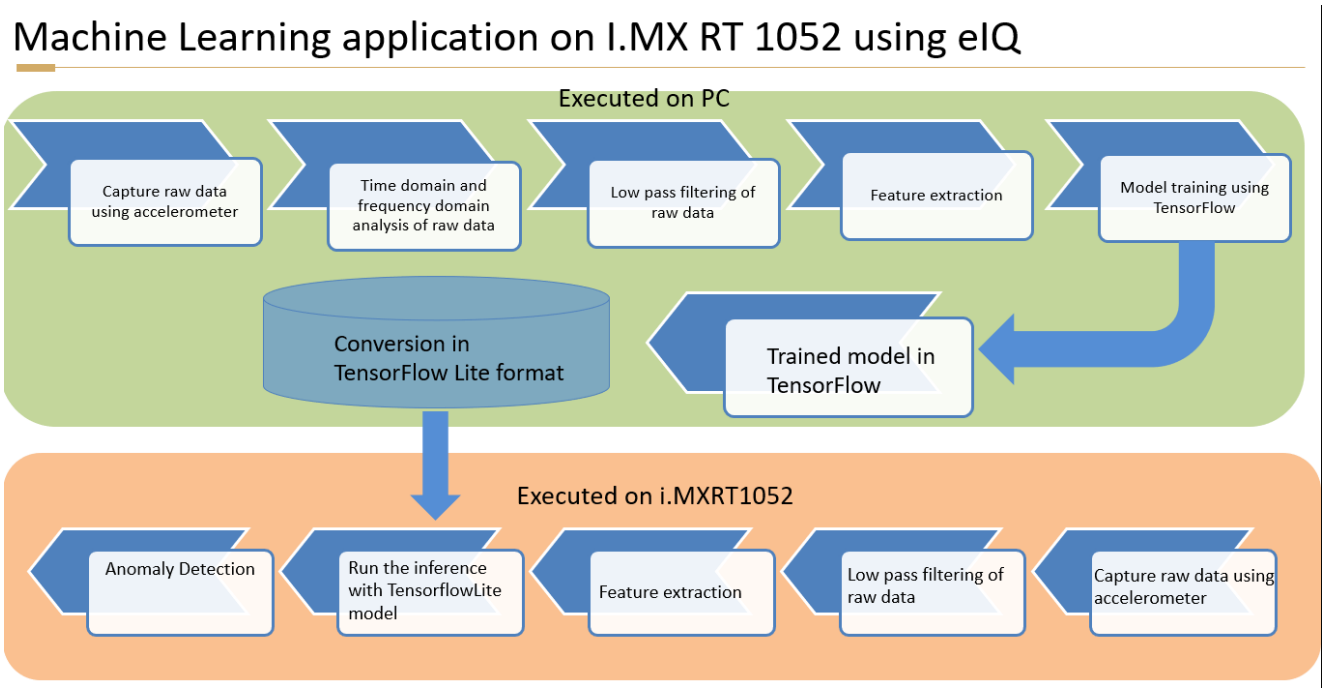

Figure 6. Model development with Tensorflow

9 Model development in Tensorflow

In this demo, we use K-Means Clustering algorithm for anomaly detection. The model is developed in python using Tensorflow. It takes raw accelerometer data as input and extracts different features from it by calculating RMS. After analyzing the Data, it is feed to the K-Means clustering which provides output in terms of clusters.

Here you can train your model as per the requirement of your application in python. The model should be developed in Tensorflow so that it can be converted into Tensorflow Lite format for i.MXRT1050 processing.

The model in python should be developed keeping in mind the input to be feed and the output expected. This same model would be saved as .pb format, converted to .tflite format and imported on the application running on i.MXRT1050.

When you run the model on i.MXRT1050, it expects to feed the input in same format and size as developed in the python. For example if you look at the developed model for anomaly detection with this package, the input size is 2005. This is an empirical number which is fixed on several trials and errors while developing the model in python. Based on several trials we came to a conclusion that the model gives the expected output if the input size is 200 samples. Before feeding the samples to model, we apply low pass filter to the data which is done using moving average. After that we calculate RMS values of samples (10 samples). Low pass filter and RMS calculation are provided below.

1. Applying low pass filter to data:

   Moving average: $A_1 = (x1+x2+x3+x4+x5)/5$

   $A_2 = (x2+x3+x4+x5+x6)/5$

   will go till A_n

   Here we apply low pass filter by taking moving average of 5 samples so,

   Output size = (input size – 5)

2. Calculating RMS values of filtered data:

   Final output size = (size of input to RMS)/10

   Apply RMS on filtered data:

Output of filter = Input of RMS

Hence:

Final output size = (input size -5)/10

Now this final output after calculating RMS should be provided as input to the model. The model requires 200 inputs then only it gives expected output.

For the final output size to be 200, from the above calculations the input size of raw sample should be 2005. So we have taken 2005 raw samples as input in the application.

9.1 Training and saving the model

The developed model should be trained with healthy and some anomalies data and saved using the SavedModel in Tensorflow. First we train the model and get the trained data. In our example this is done using “kmeans_development_training.py” and as per our algorithm of kmeans clustering we take trained data as centroids.

This trained data should be saved to the model that is running on the i.MX RT1050 platform. This model should be given the data that is filtered and analyzed as the input and then saved in the SavedModel format. This is performed by the script “Export_Trained_model.py”.

The SavedModel saves the model in ProtocolBuffer (.pb) format. The APIs for SavedModel are as below:

1. tf.compat.v1.saved_model.simple_save
2. tf.saved_model.builder.SavedModelBuilder&builder.add_meta_graph_and_variables

For further details about the SavedModel format, go through the below link:

https://www.tensorflow.org/guide/saved_model

9.2 Converting model in TensorFlowLite format

The trained and saved model should be converted into a TensorFlowLite compatible format. For this we have the below steps:

1. Convert the saved model to tflite format:

   export_dir = "<Path to saved model>/savedmodel"

   converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)

   tflite_model = converter.convert()

   open("converted_model.tflite", "wb").write(tflite_model)

2. Convert the tflite file to converted_model.h file

   This can be done using the xxd utility like below:

   xxd -i ./converted_model.tflite> ./converted_model.h

   Your model is ready in TensorFlowLite format. Now add “converted_model.h” header file in the TensorflowLite application project. Now project must have below structure as shown in Figure-6.

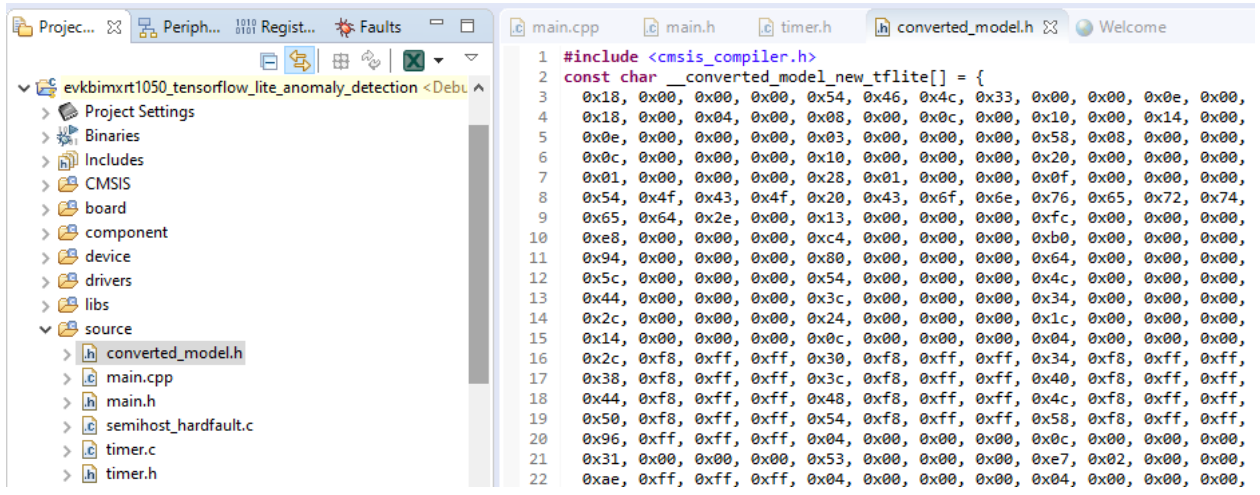

Figure 7. Anomaly detection model file in application project

9.3 Model verification using TensorFlow

At this point, we have the trained model in .pb (Protocol Buffer) format. Now convert this trained model to TensorFlowLite format. This can be done using the script “conversion.py” in the package. This script provides you the output in the form of “.tflite” file. This file is the converted model in TensorFlowLite format. Before running this model on edge using TensorFlowLite inference, it is necessary to verify the model on our machine using Tensorflow.

Find the below steps for running a “.tflite” file using TensorFlow

1. Loading the TensorFlow Lite model

   tf.lite.Interpreter

2. Get Input & Output Tensor from the model

   interpreter.get_input_details()

   interpreter.get_output_details()

3. Provide the new input
4. Reshape the input in required shape and type
5. Run the loaded model with new input

   interpreter.invoke()

Here you can validate the model by providing different inputs and observing the outputs. You can refer to the script “check_tflite.py” which is an example for the current model of anomaly detection. When check_tflite.py is run, it produces output (distance of each point from each centroids). Value for each point are mentioned in check_tflite.py and centroid value is fetch from trained model file. By mathematical calculation you can find distance between each point and centroids.

In this script all the above mentioned steps are followed. You can find the script at below location:

<path-to-ML-release-Package>/scripts/check_tflite.py

10 Application development on TensorFlowLite using eIQ

For developing any application on TensorFlow Lite using eIQ Software Development Environment(SDE), you can take reference of the available demo application. You can find the demo applications from the i.MX RT1050 SDK which can be downloaded from https://mcuxpresso.nxp.com.

For basic steps of developing any application on microcontroller using TensorFlow Lite, find the below link:

https://www.tensorflow.org/lite/microcontrollers/overview.

11 Important notes regarding eIQ SDE

There are some operators on TensorFlow that are supported by higher versions of TensorFlow Lite but are not supported by the TensorFlow Lite version(r1.11) that eIQ SDE supports. Find the below points:

- The tf.square operator is not a builtin operator in TensorFlow Lite version supported by the eIQ SDE and our model requires this operator.

Workaround:

As a workaround of the above unsupported operator we have currently used multiplication to perform square.

- The tf.argmin is a built-in operator of TensorFlow Lite. But we found some unexpected behavior in the output that if we include tf.argmin as a part of TensorFlow model and run it on TensorFlow Lite after necessary conversions, the dimensions of the output changes. For example, if we give a tensor with dimension (3,409) as input to tf.argmin (tensor,axis=0) then it should provide an output tensor with shape 409. This is providing the expected output on TensorFlow but if we convert it to .tflite and then to .h file and run the model on TensorFlow Lite on i.MXRT1050 we see that the output dimension is (3,1) instead of (409,1).

Workaround:

We have developed the logic of tf.armin on the application for anomaly detection running on i.MX RT1050 and we are able to detect the clusters with an anomaly on i.MX RT1050 board. The sam behavior is observed for tf.argmax operator.

12 Conclusion

This exercise demonstrates, how model is trained by using TensorFlow framework by collecting the sensor data from i.MX RT1050 board on computer through python script. And also demonstrate how trained model is converted to the supporting file which can be imported to i.MX RT1050 board.

This provides broader view to user, that how TensorFlow framework can be implemented on embedded board.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 18 March 2020

Document identifier: AN12766

arm