UM3053

User manual

Getting started with the AEKD-AICAR1 evaluation kit for car state classification

Introduction

The AEKD-AICAR1 is a versatile deep learning system based on a long-short term memory (LSTM) recurrent neural network (RNN). It is able to classify the car state:

- car parked
- car driving on a normal condition road
- car driving on a bumpy road
- car skidding or swerving

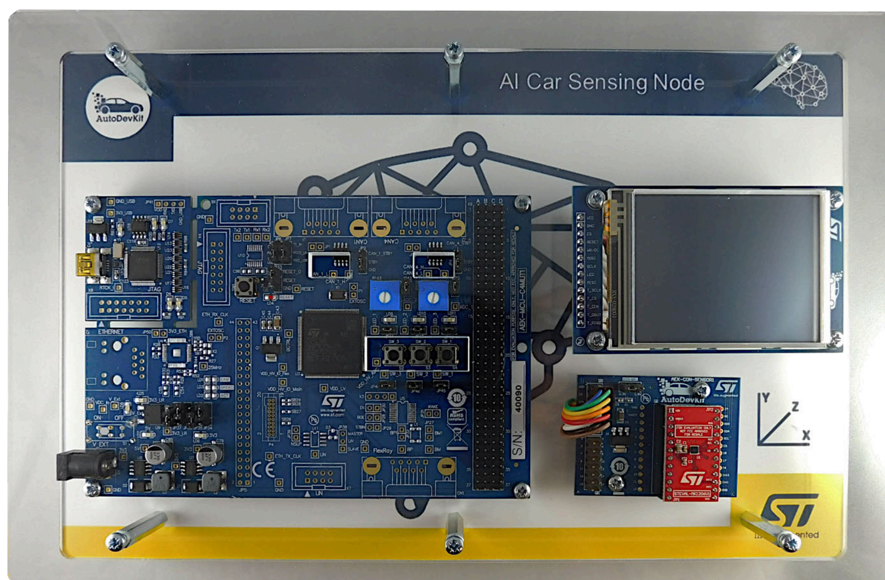
The main idea is to define an AI-car sensing node ECU with an embedded artificial intelligence processing.

The system hosts an SPC58EC Chorus 4M microcontroller, which is able to acquire discrete acceleration variations on a three-axis reference system.

The AIS2DW12 motion sensor mounted on the AEK-CON-SENSOR1 board retrieves inertial data. The acquired data are transmitted to the LSTM RNN, which classifies the car state. The classification result is shown on the AEK-LCD-DT028V1 LCD touch display.

The LSTM RNN has been implemented and trained using the TensorFlow 2.4.0 framework (Keras) in the Google Colab environment. The AI-SPC5Studio plug-in has been used to convert the resulting trained neural network into an optimized C code library, which can run on an MCU with limited power computing resources.

Figure 1. AEKD-AICAR1 evaluation kit



The LSTM RNN training has been performed with several time-series acceleration waveforms recorded on a real vehicle in motion. The resulting prediction accuracy, calculated by the confusion matrix, is about 93%. Field tests carried-out under all road conditions with a sedan confirm the adherence of the computed results compared with the real road conditions.

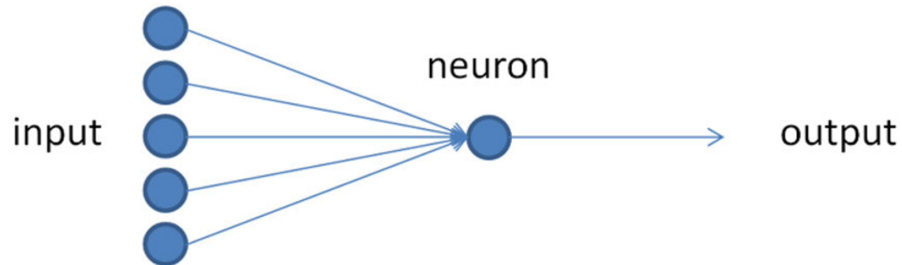
Note: The AEKD-AICAR1 is an evaluation tool for R&D laboratory use only. It is not destined for use inside a vehicle.

1 Neural network basic principles

1.1 Artificial neural network

The artificial neural network works as the human brain neural network. Data are transferred to the neuron through the inputs. Then, they are sent as an output after processing.

Figure 2. **Artificial neural network**



Artificial neural networks are built on layers of different neural units. Each unit consists of three parts:

- an input part that receives the data
- a hidden part that uses the neuron weight to calculate the result
- an output part that receives the calculation results and applies an eventual bias

The weight associated with each neuron determines its firing probability. The bias is the measure of assumption made by the form of the output.

This architecture is typical for deep learning processes like data classification and pattern recognition neural networks.

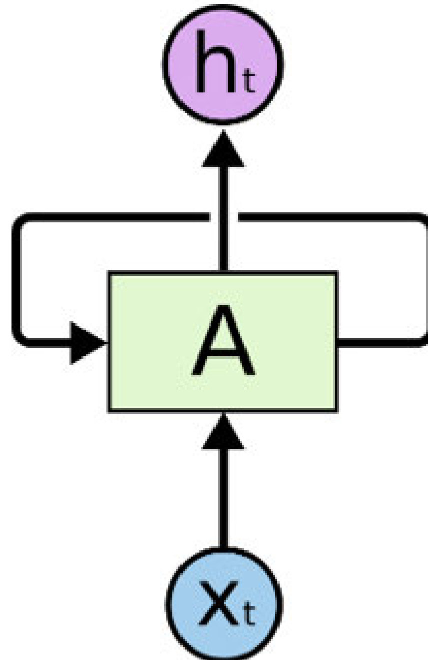
1.2 Long short-term memory recurrent neural network (LSTM RNN)

Traditional artificial neural networks keep no memory of what happened in the past. They take their decision only on the data provided instant by instant. These architectures are well suited for data classification and pattern recognition.

For other applications, like speech recognition, the proper classification requires a memory of the context, that is, the prior words in the speech recognition application.

A recurrent neural network (RNN) is a class of artificial neural network that includes neurons connected in a loop.

Figure 3. Recurrent neural network



In most cases, the output values of an upper layer are used as the input for a lower level one. This interconnection enables the use of one of the layers as the state memory and allows modeling a dynamic time behavior dependent on the information previously received.

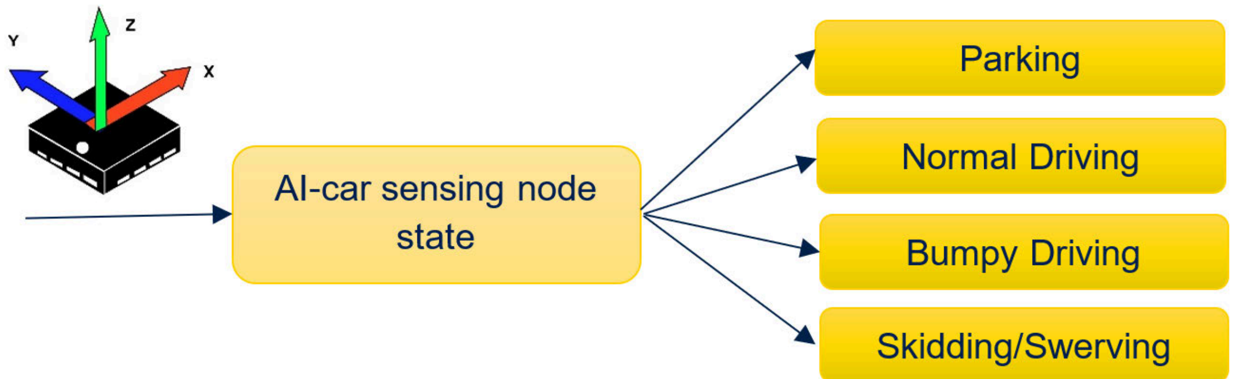
A typical recurrent neural network able to keep the information received is the long short-term memory (LSTM) recurrent network. Unlike traditional artificial neural networks, LSTM has feedback connections. It can process not only single data points but also entire sequences of data. For this reason, the LSTM neural network architecture is the best candidate to model a deep learning system for an AI-car sensing node, using a collection of time-series based on acceleration values.

2 Designing an AI-car sensing node

2.1 Tool-set introduction

An AI-car sensing node is an AI-deep learning system based on an LSTM recurring neural network, which can provide a car state classification: parking, driving on a normal condition road, driving on a bumpy road, and car skidding or swerving.

Figure 4. **AI-car sensing node: car state classification**



The LSTM RNN has been modeled with TensorFlow (Keras framework). This is an open-source software library for machine learning, which provides optimized modules to implement AI algorithms related to the classification problem.

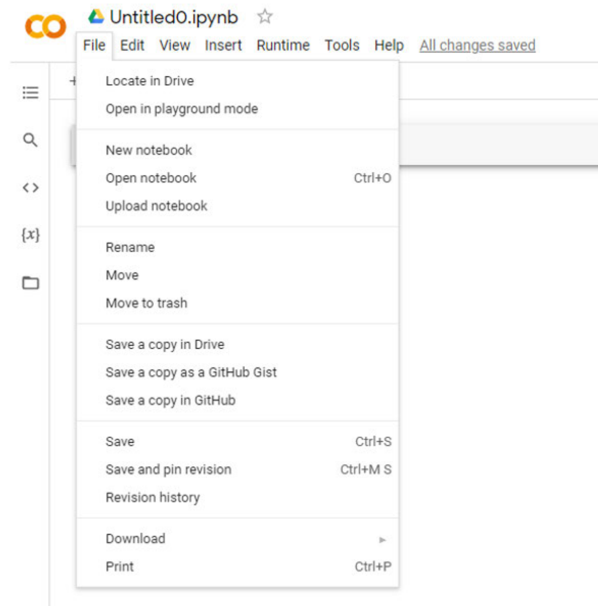
A significant amount of computing power is required to implement sufficiently robust and efficient models. For initial tests, we can rely on a standard machine. As the dataset size increases, the execution of complex training algorithms becomes rapidly prohibitive.

To address this issue, there are several cloud services that offer computing power. Google Colab is an alternative platform, which allows running the code directly on the cloud, even if with some limitations.

2.2 Creating a Google Colab notebook

To use the Google Colab platform, you need a Google account to login. After logging in, create a new “Colab notebook” project file.

Figure 5. Project file



The document that you are creating is not a static web page. It is an interactive environment that lets you write and execute your Python code to implement an LSTM RNN by using the TensorFlow framework.

2.3 Colab notebook setup and package importing

To implement and train an LSTM RNN, install the Tensorflow 2.4.0 framework on your Google Colab notebook by using the packet index package (PIP) command.

```
%pip install tensorflow==2.4.0
```

Note: PIP is already installed on your Google Colab notebook.

Using the Python commands, import the following packages:

- **Pandas**
  Open-source software library for data manipulation and analysis.
  ```
  import pandas as pd
  ```
- **Numpy**
  Open-source software library, adding support for large, multidimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
  ```
  import numpy as np
  ```
- **Tensorflow**
  Open-source software library for machine learning, which provides optimized modules to implement artificial intelligence (AI) algorithms.
  ```
  import tensorflow as tf
  ```
- **Sklearn**
  A package providing several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.
  ```
  from sklearn.preprocessing import StandardScaler, MinMaxScaler
  from sklearn.model_selection import train_test_split , StratifiedShuffleSplit
  from sklearn.metrics import confusion_matrix
  ```
- **OS**
  The Python language OS module has several useful functions to make the program interact with the computer operating system.
  ```
  import os
  ```

- **Random**

  This module implements pseudo-random number generators for various distributions.

  ```
  import random
  ```
- **Seaborn**

  Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface to draw attractive and informative statistical graphics.

  ```
  import seaborn as sn
  ```
- **Matplot**

  Matplotlib is a comprehensive library to create static, animated, and interactive visualizations in Python.

  ```
  import matplotlib.pyplot as plt
  ```

2.4 AI-car sensing node life cycle

The following steps define the life cycle of the AI-car sensing node implementation as a deep learning model for classification:

1. Model definition
2. Model training
3. Model fitting and compilation
4. Model evaluation

2.4.1 Model definition

To define the model, we have chosen the topology of the LSTM network. The AI-car sensing node network is based on the same network architecture of a speech recognition system. It consists of a neural convolution kernel as an input layer, able to elaborate a convolution function with the input vector over a temporal dimension.

The input vector is a time sequence of `TIMESERIES_LEN` number of samples, which consist of discrete acceleration variations on a three-axis (x, y, z) reference system: Δax, Δay, and Δaz.

`TIMESERIES_LEN` represents the minimum size of the temporal window for the car state classification. With an acquisition sampling time equal to 100 msec, the value of `TIMESERIES_LEN` has been fixed to 50 samples (5 seconds per acquisition).

A dense function implements the output layer. This function can provide an output shape of four-dimensional vectors (one for each expected status: parking, normal, bumpy, skid). The `Softmax` function activates this layer and converts the output vector values to a probability distribution.

From an implementation perspective, this involves a model layer architecture built with a connection topology into a cohesive model:

```
## Conv1D based model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=16, kernel_size=3, activation='relu',
input_shape=(TIMESERIES_LEN, 3)),
  tf.keras.layers.Conv1D(filters=8, kernel_size=3, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(4, activation='softmax')
])

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 48, 16)            160

 conv1d_1 (Conv1D)           (None, 46, 8)             392

 dropout (Dropout)           (None, 46, 8)             0

 flatten (Flatten)           (None, 368)               0

 dense (Dense)               (None, 64)                23616

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 24,428
Trainable params: 24,428
Non-trainable params: 0
```

As described in the above code block table, 24,428 trainable parameters define the AI-car sensing node network model. These parameters are related to the LSTM RNN topology chosen.

2.4.2 Model training

The AI-car sensing node network training has been performed by acquiring datasets of acceleration variations in the time domain:

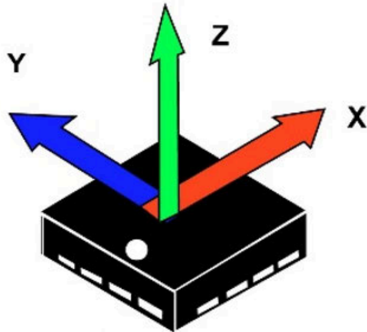
- over a three-axis reference system (Δax, Δay, Δaz)
- with a frequency equal to 10 Hz
- a sample time of 100 msec

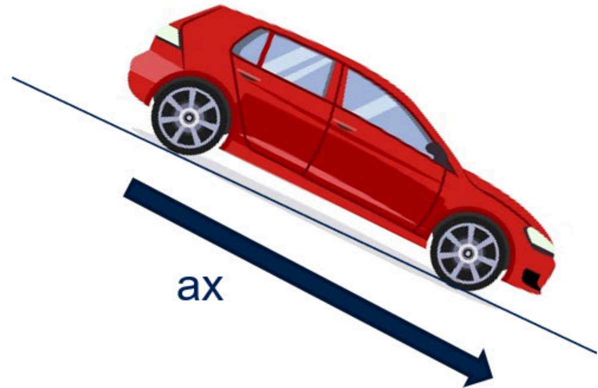
Each dataset targets a specific car state: parking, normal driving, bumpy driving or skidding.

The use of acceleration variations instead of raw acceleration data avoids misinterpretations of the car state. If the car is parked on a road with a negative (or positive) slope (as shown in the figure below), the IMU registers a nonzero X-axis acceleration that would not lead to the parking state. Instead, if we consider an acceleration variation in the time domain, the result would be 0, leading to the correct parking state.

Figure 6. **Acceleration variations**

Car speed Vx = 0 Km/sec;
Car acceleration ax! = 0;
Temporal variation of acceleration Δax = 0





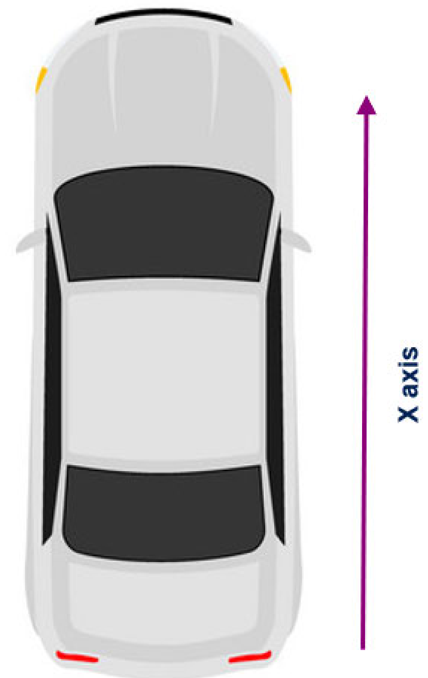
2.4.2.1 *Training dataset*

The acquisition of training datasets can be performed using a specific configuration of the AI-car sensing node mock-up (acquisition mode). A proper source code `#define` enables the acquisition mode. In this mode, the application sends each acceleration sample read from the sensor to a serial port with a baud rate set to 38400. The serial data stream can be read through a common serial client (Tera Term or Putty). The data acquisition and parsing in a CSV file are managed via a specific Python script (see Section Appendix C).

The following figure shows an example of a time-based dataset generated with the Python script.

Figure 7. **Time-based dataset example**

	A	B	C	D
1	Time	Acc_x	Acc_y	Acc_z
2	100	-0.28792	0.0244	-0.11468
3	200	-0.28792	0.0366	-0.11956
4	300	-0.29036	0.0366	-0.11468
5	400	-0.29036	0.02684	-0.10736
6	500	-0.2806	0.02684	-0.10492
7	600	-0.29524	0.04636	-0.11712
8	700	-0.29768	0.0366	-0.11468
9	800	-0.29524	0.02928	-0.11468
10	900	-0.2928	0.03172	-0.11224
11	1000	-0.29524	0.0366	-0.11224
12	1100	-0.2806	0.0244	-0.10492
13	1200	-0.28304	0.03172	-0.12444
14	1300	-0.28792	0.0366	-0.11956
15	1400	-0.28792	0.02928	-0.10492
16	1500	-0.28304	0.0366	-0.11956
17	1600	-0.28304	0.03904	-0.11468
18	1700	-0.29036	0.02928	-0.1098
19	1800	-0.27572	0.02196	-0.11468
20	1900	-0.28304	0.0244	-0.10736
21	2000	-0.28548	0.04148	-0.11956
22	2100	-0.27084	0.04148	-0.12444
23	2200	-0.28792	0.02196	-0.10492
24	2300	-0.29768	0.0244	-0.1098
25	2400	-0.2928	0.03904	-0.11956
26	2500	-0.2928	0.03172	-0.11224
27	2600	-0.29768	0.02196	-0.11224



The time column contains the recorded acceleration time samples. The other three columns contain the acceleration values measured on the X, Y, and Z axis, respectively (without the gravity offset).

To prepare the file for the network training, you have to:

- Compute the acceleration variations (excel cell operation: B3 = B3–B2 for X axis).

- Add a status column for the car state for each specific acquisition (P = parking; N = normal; B = bumpy; S= skid).

Figure 8. **Adding the status column**

	H	I	J	K	L
1	Time	Acc_x	Acc_y	Acc_z	Status
2	100	0	0.0122	-0.00488	P
3	200	-0.00244	0	0.00488	P
4	300	0	-0.00976	0.00732	P
5	400	0.00976	0	0.00244	P
6	500	-0.01464	0.01952	-0.0122	P
7	600	-0.00244	-0.00976	0.002441	P
8	700	0.00244	-0.00732	0	P
9	800	0.00244	0.00244	0.002439	P
10	900	-0.00244	0.00488	0	P
11	1000	0.01464	-0.0122	0.007321	P
12	1100	-0.00244	0.00732	-0.01952	P
13	1200	-0.00488	0.00488	0.00488	P
14	1300	0	-0.00732	0.01464	P
15	1400	0.00488	0.00732	-0.01464	P

- Create a CSV file for each car state through a specific acquisition task.

Figure 9. **Creating CSV files**

- Merge all CSV files by coping and pasting each row without changing the time column values.
- Add a new dummy row at the end of the file with the time value equal to 100.
- Add random values for Ax, Ay, and Ax status.

Note: A ready-to-use dataset (Diff_profile.csv) has been created to train the network for the car state classification.

2.4.2.2 *Neural network training with Google Colab*

Import the training dataset contained in the CSV file into the Colab notebook environment. Then, run a parser function on the imported data to build an input vector compliant with the LSTM RNN.

The following code block shows the Google Colab script, which imports and loads the CSV file.

```
from google.colab import files
uploaded = files.upload()

db = pd.read_csv('Diff_profile.csv',sep=',')
```

Parse each column of the CSV file (status, Acc_x, Acc_y, Acc_z, time).

```
states = db['Status'].value_counts()
ts_status = db.Status
ts_diff_Ax = (db.Acc_x.to_numpy().reshape(-1,1))/9.81
ts_diff_Ay = (db.Acc_y.to_numpy().reshape(-1,1))/9.81
ts_diff_Az = (db.Acc_z.to_numpy().reshape(-1,1))/9.81
ts_time = db['Time']
```

Use a simple script to reshape the dataset to be compliant with the shape of the LSTM input vector (50 samples of a three-dimensional vector). The number of LSTM input vectors is computed from the total number of samples acquired in all the states, choosing window waveforms of 50 samples.

If for the parking state we have 350 samples, 350/50 = 7 window waveforms are generated.

```
TIMESERIES_LEN = 50
Y_LABELS={'P':0,'N':1,'B':2,'S':3}

def trip_framing(trip,label,frame_size,db_x,db_y):
    a = np.array( trip )
    for i in np.arange( 0, a.shape[0]-frame_size, frame_size ):
        x = a[i:i+frame_size]
        db_x.append( x )
        db_y.append( Y_LABELS[label] )
rows = ts_status.shape[0]
db_x = []
db_y = []

for states_id in states.keys():
    trip = []
    cnt = 0
    for i in range(rows):
        if ts_time[i] == 100:
            if len(trip) > 0:
                trip_framing( trip, states_id, TIMESERIES_LEN, db_x, db_y)
                trip=[]
        if ts_status[i] == states_id and cnt < 7500:
             trip.append( [ts_diff_Ax[i],ts_diff_Ay[i],ts_diff_Az[i]] )
             cnt += 1
```

Note: *The cnt variable contains the number of available window waveforms according to the dataset acquired during each specific car status.*

2.4.3 Model fitting and compilation

Firstly, for model fitting, select the training configuration:

- Training test percentage:
  - Percentage of the window waveform available over the total, used to perform the training task.
- Batch size:
  - Percentage of the window waveform available over the total, used to perform the estimation of the network accuracy (model error).
- Number of epochs:
  - Number of loops through the training dataset.

```
x_train, x_test, y_train, y_test = train_test_split(db_x, db_y, test_size=0.4,
random_state=21,stratify=db_y)
x_train = np.asarray(x_train)[:,:,:,0]
x_test = np.asarray(x_test)[:,:,:,0]
y_train = np.asarray( y_train )
y_test = np.asarray( y_test )
db_stats = pd.Series( y_test )
```

To train the model, an algorithm is required to minimize the accumulated errors in identifying the correct car state. This algorithm consists of a loss function to be passed to the `model.compile` API to generate the complete runnable code.

The chosen loss function is the `sparse categorical cross-entropy` able to correlate the state label (for example, “P” with the neural network prediction).

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=1000)
```

Fitting and compilation for the RNN generate a complete model. This procedure is quite slow. It can take few seconds up to days, depending on the model complexity and the training dataset size.

For the AI-car sensing node, set 1000 epochs and 60% of the available window waveform used for the training. This results in a computation time of around 70 seconds.

Figure 10. Complete model generation

```
Epoch 990/1000
5/5 [==============================] - 0s 7ms/step - loss: 9.3488e-04 - accuracy: 1.0000
Epoch 991/1000
5/5 [==============================] - 0s 7ms/step - loss: 9.1694e-04 - accuracy: 1.0000
Epoch 992/1000
5/5 [==============================] - 0s 7ms/step - loss: 9.4312e-04 - accuracy: 1.0000
Epoch 993/1000
5/5 [==============================] - 0s 10ms/step - loss: 0.0012 - accuracy: 1.0000
Epoch 994/1000
5/5 [==============================] - 0s 9ms/step - loss: 6.8372e-04 - accuracy: 1.0000
Epoch 995/1000
5/5 [==============================] - 0s 8ms/step - loss: 4.1409e-04 - accuracy: 1.0000
Epoch 996/1000
5/5 [==============================] - 0s 8ms/step - loss: 0.0027 - accuracy: 1.0000
Epoch 997/1000
5/5 [==============================] - 0s 10ms/step - loss: 5.2084e-04 - accuracy: 1.0000
Epoch 998/1000
5/5 [==============================] - 0s 9ms/step - loss: 5.4034e-04 - accuracy: 1.0000
Epoch 999/1000
5/5 [==============================] - 0s 8ms/step - loss: 0.0015 - accuracy: 1.0000
Epoch 1000/1000
5/5 [==============================] - 0s 8ms/step - loss: 0.0034 - accuracy: 1.0000
Model: "sequential"
```

2.4.4 Model evaluation

To evaluate the model accuracy, choose a holdout window waveform. This should be based on data not used in the training process, to get an unbiased estimation of the model performance when making a prediction on “new” data.

The model evaluation speed is proportional to the amount of data used for it, but faster than the training phase.

A confusion matrix performs the model accuracy evaluation. This is a technique to summarize the algorithm classification performance. As the datasets have different sizes, using a pure classification accuracy could be misleading. Therefore, calculating a confusion matrix can give you a better idea of what your classification model is getting right and what miss classification is obtaining.

Use the following code to build a confusion matrix.

```
Y_pred = model.predict(x_test)
y_pred = np.argmax(Y_pred, axis=1)
confusion_matrix = tf.math.confusion_matrix(y_test, y_pred)

plt.figure()
sns.heatmap(confusion_matrix,
            annot=True,
            xticklabels=Y_LABELS,
            yticklabels=Y_LABELS,
            cmap=plt.cm.Blues,
            fmt='d', cbar=False)
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

The confusion matrix for the AI-car sensing node is:

- 26 actual parking window waveforms used for the evaluation have been predicted as parking (no miss classifications).

- 36 actual normal window waveforms used for the evaluation have been predicted as normal, whereas four have been classified as bumpy.
- 18 actual bumpy window waveforms used for the evaluation have been predicted as bumpy (no miss classifications).
- Only one actual skid window waveform has been recognized as skid, whereas three have been confused with bumpy.

Figure 11. Confusion matrix

True label \ Predicted label	P	N	B	S
P	26	0	0	0
N	0	36	4	0
B	0	0	18	0
S	0	0	3	1

The LSTM RNN accuracy for the AI-car sensing node is about 93%. The miss classification for the skid state impact the calculated accuracy value, due to the very low number of window waveforms used for training (only four).

Despite the above considerations, we consider the accuracy value acceptable and defer a new accuracy calculation for the skid state to the field tests to perform on the sedan.

3 AutoDevKit ecosystem

3.1 SPC5-STUDIO-AI plugin

The LSTM AI-car sensing node network implemented with the TensorFlow framework inside the Google Colab environment is then converted in a C library compliant with SPC5-STUDIO.

On the Google Colab notebook, the LSTM AI-car sensing node network architecture is saved as an **.h5* file. This file is then fed into SPC5-STUDIO-AI to obtain a fast and optimized version of the RNN.

```
model.save('model_car_sts.h5')
```

SPC5-STUDIO-AI is the artificial intelligence plug-in for SPC5-STUDIO able to generate automatically a pretrained neural network into an efficient “ANSI C” library to be compiled, installed, and executed on SPC58 microcontrollers.

You can easily import pretrained neural networks in the SPC5-STUDIO-AI from the most widely used deep learning frameworks, such as Keras, TensorFlow Lite, Lasagne, Caffe, ConvNetJS, and ONNX.

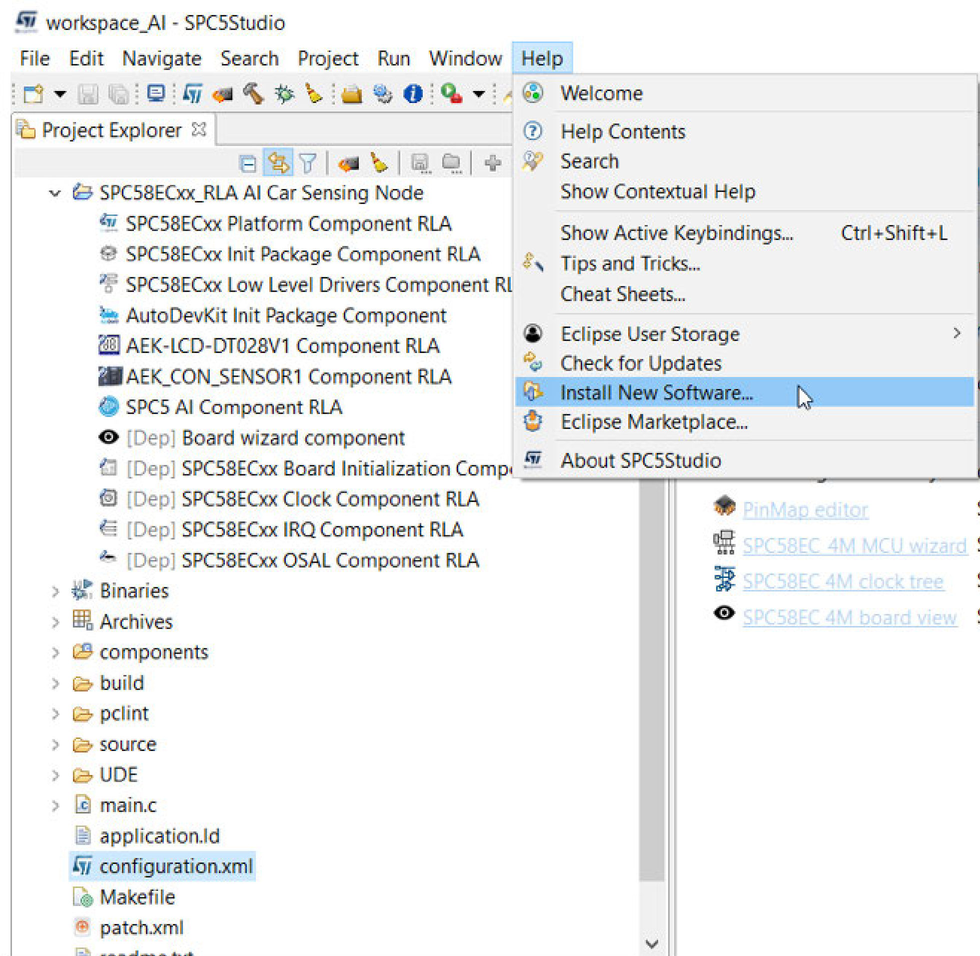
SPC5-STUDIO-AI provides validation and performance analysis features to validate and characterize the converted neural network and measure key metrics such as validation error, memory requirements (flash memory and RAM), and execution time.

This plugin is integrated within the SPC5-STUDIO (version 6.0.0 or higher) development environment.

To install the SPC5-STUDIO-AI, follow the procedure below.

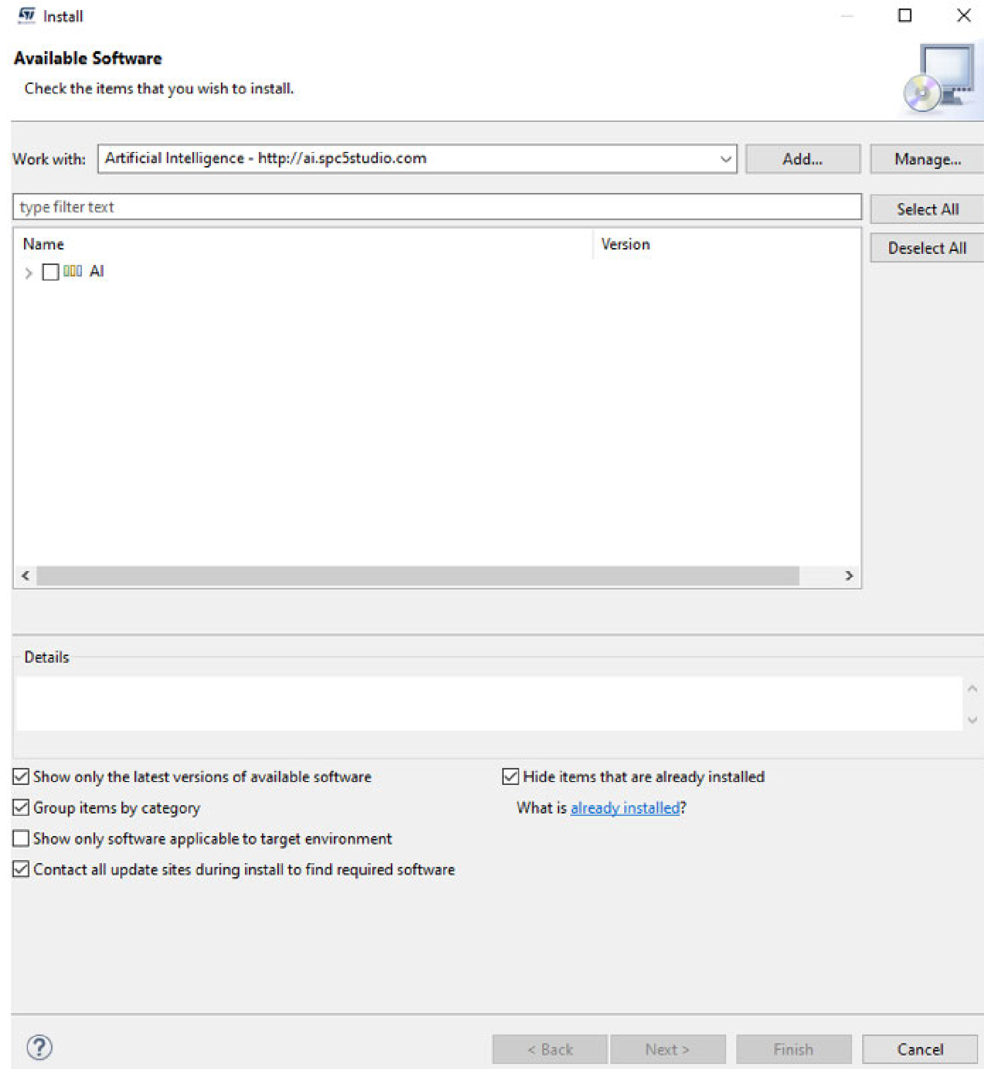
Step 1. Select [**Install new software**] from [**Help**].

Figure 12. Installing the plugin



Step 2. Insert *http://ai.spc5studio.com* in the [**Work with**] field.

Figure 13. Install new software



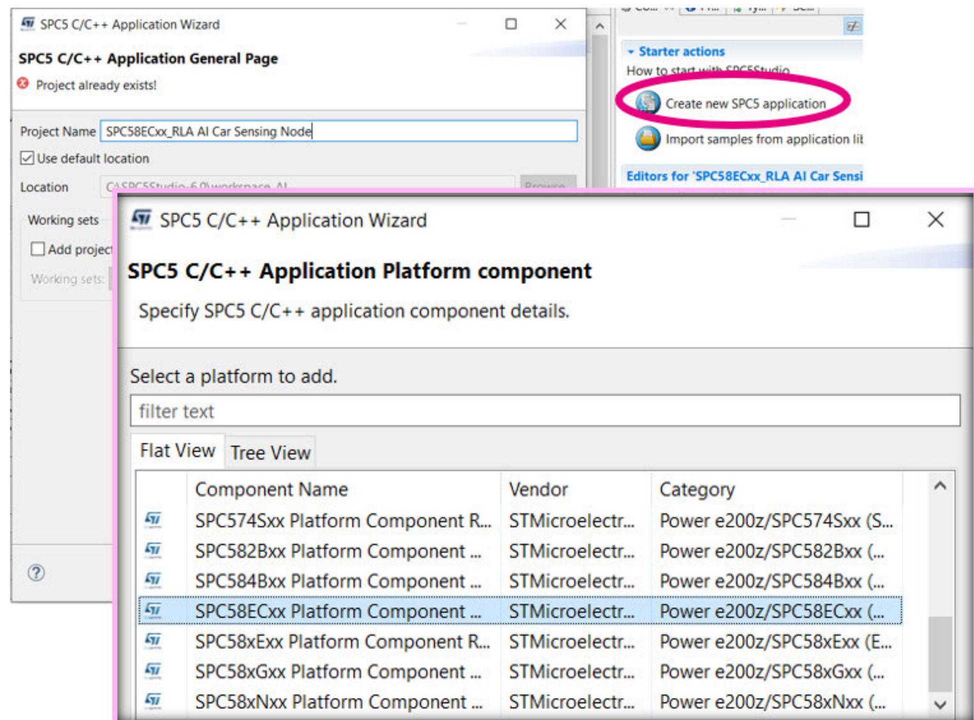
Step 3. Click on [**Select all**] and [**Next**].

Step 4. Install SPC5-STUDIO-AI.

3.1.1 How to import SPC5-STUDIO-AI using the standard importing procedure

Step 1. Create a new SPC5-STUDIO application based on the SPC58ECxx platform.

Figure 14. Creating a new SPC5-STUDIO application

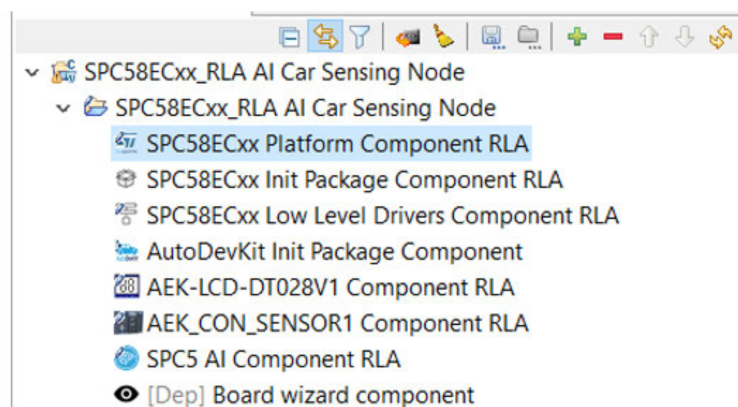


Step 2. Import the following components:

– *AutodevKit Init Package*
– *SPC58ECxx Init Package Component RLA*

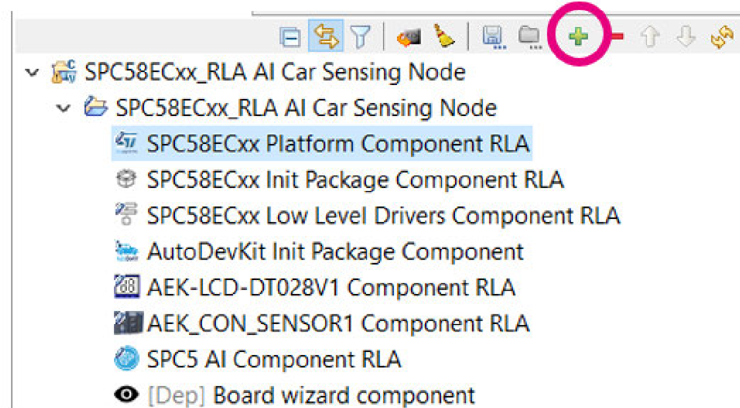
Step 3. Select [**SPC58ECxx Platform Component RLA**].

Figure 15. Selecting the component



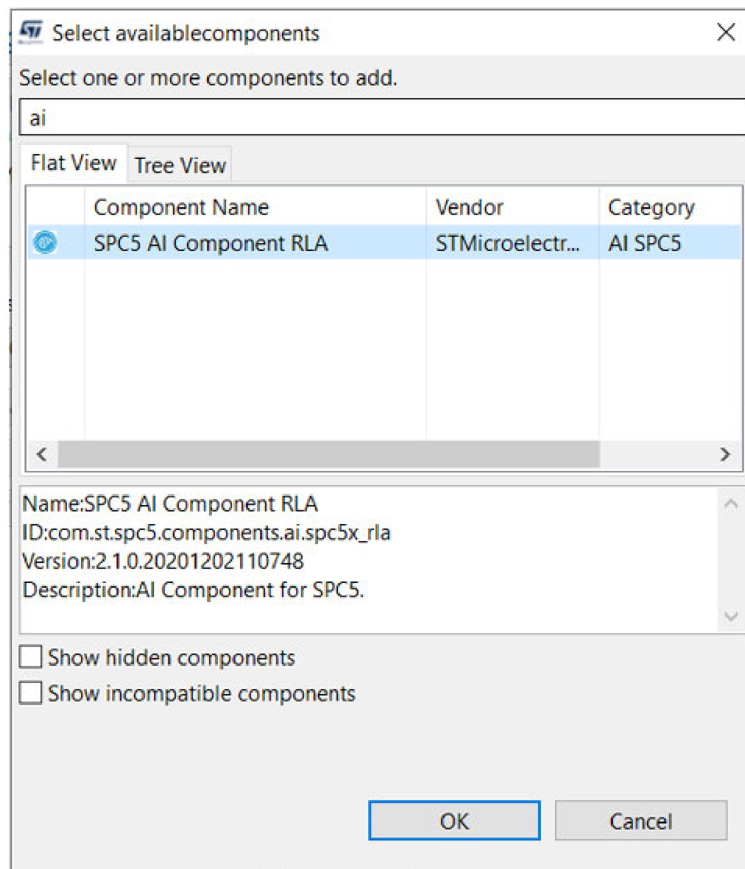
Step 4. Click on [**+**] to add a new component.

Figure 16. Adding a new component



Step 5. Select [**SPC5 AI Component RLA**].

Figure 17. Selecting [SPC5 AI Component RLA]



3.1.2 How to import the pretrained LSTM neural network

The following procedure shows how to import the pretrained LSTM neural network for the AI-car sensing node.

Step 1. Select the [**SPC5 AI Component RLA**] in the project explorer.
A new window is opened with the network list imported.

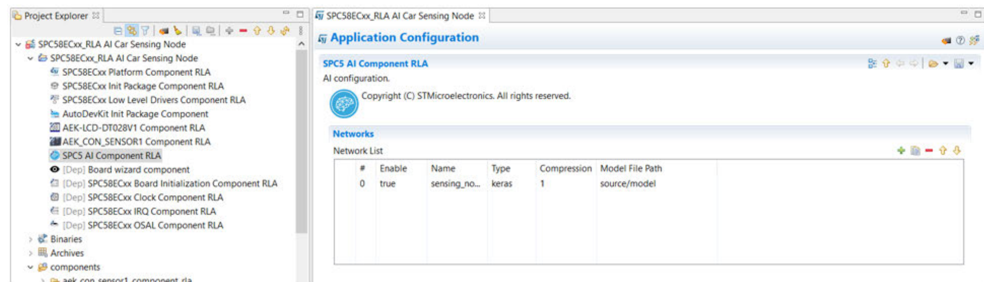


Figure 18. **Network list imported**

Step 2. Add a new network by clicking on [**+**].

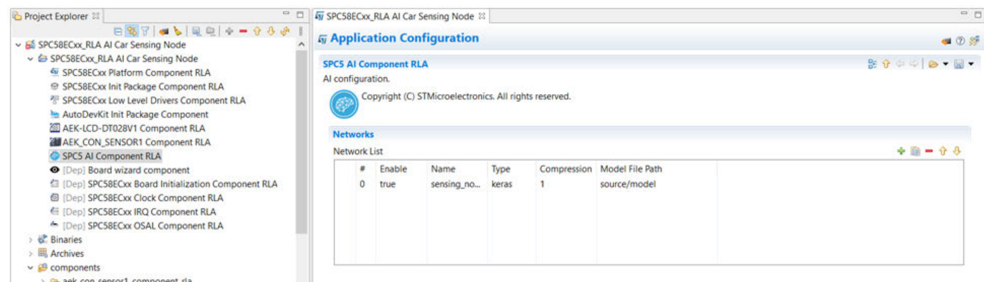


Figure 19. **Adding a new network**

Step 3. Double-click on the new network preconfigured with the default option.

Step 4. Configure the parameters for the LSTM AI-car sensing node.

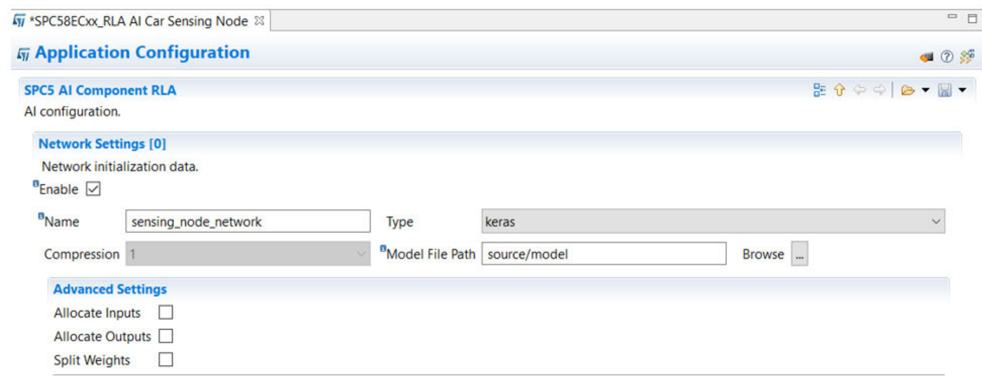


Figure 20. **Configuring the parameters**

Step 4a. Enable the network initialization data.

Step 4b. Put a valid neural network name.

Step 4c. Select the framework used for the training (Keras).

Step 4d. Select a compression equal to 1 (it indicates the expected global factor of compression applied to dense layers).

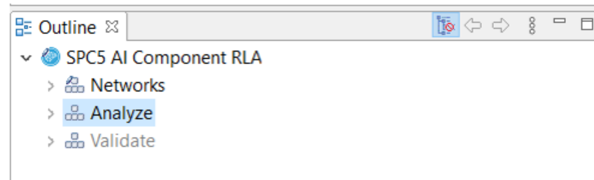
Step 4e. Select a valid model file path that contains the *.h5 file.

3.1.3 How to analyze the pretrained LSTM neural network

To analyze the pretrained LSTM neural network for the AI-car sensing node, follow the procedure below.

Step 1. Select [**Analyze**] in the [**Outline**] tab.

Figure 21. **Selecting [Analyze]**



Step 2. Click on [**Analyze**] in the newly opened window.

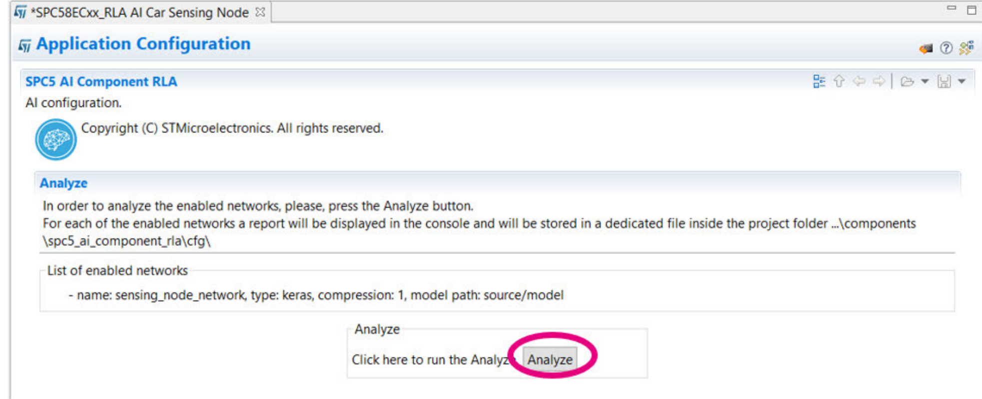


Figure 22. **Clicking on [Analyze]**

If the importing procedure for the LSTM AI-sensing node is correct, a new report is generated. This new report shows the architecture of the neural network and the ROM and RAM memory usage.

```
Neural Network Tools for STM32 v1.4.0 (AI tools v5.2.0)
-- Importing model
-- Importing model - done (elapsed time 0.574s)
-- Rendering model
-- Rendering model - done (elapsed time 0.077s)

Creating report file C:\SPC5Studio-6.0\workspace_AI\SPC58ECxx_RLA AI Car Sensing
Node\components\spc5_ai_component_rla\cfg\sensing_node_network_analyze_report.txt

Exec/report summary (analyze dur=0.65s err=0)
------------------------------------------------------------------------------------------------------------------------
model file         : C:\SPC5Studio-6.0\workspace_AI\SPC58ECxx_RLA AI Car Sensing Node\source\model\model_car_sts.h5
type               : keras (keras_dump) - tf.keras 2.4.0
c_name             : sensing_node_network
compression        : None
quantize           : None
workspace dir      : C:\SPC5Studio-6.0\workspace_AI\SPC58ECxx_RLA AI Car Sensing Node\components\stm32ai_ws
output dir         : C:\SPC5Studio-6.0\workspace_AI\SPC58ECxx_RLA AI Car Sensing
Node\components\spc5_ai_component_rla\cfg

model_name         : model_car_sts
model_hash         : 10794f1c230799b2a1cda67171827db2
input              : input_0 [150 items, 600 B, ai_float, FLOAT32, (50, 1, 3)]
inputs (total)     : 600 B
output             : dense_1_nl [4 items, 16 B, ai_float, FLOAT32, (1, 1, 4)]
outputs (total)    : 16 B
params #           : 24,428 items (95.42 KiB)
macc               : 49,668
weights (ro)       : 97,712 B (95.42 KiB)
activations (rw)   : 3,136 B (3.06 KiB)
ram (total)        : 3,752 B (3.66 KiB) = 3,136 + 600 + 16

------------------------------------------------------------------------------------------------------------------------
id  layer (type)                output shape      param #     connected to             macc           rom
------------------------------------------------------------------------------------------------------------------------
0   input_0 (Input)             (50, 1, 3)
    conv1d (Conv2D)             (48, 1, 16)       160         input_0                  7,696          640
    conv1d_nl (Nonlinearity)    (48, 1, 16)                   conv1d
------------------------------------------------------------------------------------------------------------------------
1   conv1d_1 (Conv2D)           (46, 1, 8)        392         conv1d_nl                18,040         1,568
    conv1d_1_nl (Nonlinearity)  (46, 1, 8)                    conv1d_1
------------------------------------------------------------------------------------------------------------------------
3   flatten (Reshape)           (368,)                        conv1d_1_nl
------------------------------------------------------------------------------------------------------------------------
4   dense (Dense)               (64,)             23,616      flatten                  23,552         94,464
    dense_nl (Nonlinearity)     (64,)                         dense                    64
------------------------------------------------------------------------------------------------------------------------
5   dense_1 (Dense)             (4,)              260         dense_nl                 256            1,040
    dense_1_nl (Nonlinearity)   (4,)                          dense_1                  60
------------------------------------------------------------------------------------------------------------------------

model_car_sts p=24428(95.42 KBytes) macc=49668 rom=95.42 KBytes ram=3.06 KiB io_ram=616 B


Complexity per-layer - macc=49,668 rom=97,712
------------------------------------------------------------------------------------------------------------------------
id      layer (type)                macc                                     rom
------------------------------------------------------------------------------------------------------------------------
0       conv1d (Conv2D)             ||||||||||                       15.5%   |                                  0.7%
1       conv1d_1 (Conv2D)           |||||||||||||||||||||||          36.3%   |                                  1.6%
4       dense (Dense)               |||||||||||||||||||||||||||||||  47.4%   |||||||||||||||||||||||||||||||   96.7%
4       dense_nl (Nonlinearity)     |                                 0.1%   |                                  0.0%
5       dense_1 (Dense)             |                                 0.5%   |                                  1.1%
5       dense_1_nl (Nonlinearity)   |                                 0.1%   |                                  0.0%
------------------------------------------------------------------------------------------------------------------------
```

```
elapsed time (analyze): 0.65s
```

*Note: If the report does not appear properly filled, check the generated *.h5 file as it could be corrupted or generated with a Tensorflow framework version different from 2.4.0. In the latter case, uninstall and reinstall Tensorflow 2.4.0 in the Google Colab, as follows:*

– `%pip uninstall tensorflow`
– `%pip install tensorflow==2.4.0`

3.2 SPC5-STUDIO-AI API

The SPC5-STUDIO-AI plugin generates the C library files for all the enabled networks within the network list. Then, users can design and develop specific applications based on these C library APIs.

The generation process output creates the following files (C library files) within the folder / spc5_ai_components/cfg/:

- *sensing_node_network.c*
- *sensing_node_network.h*
- *sensing_node_network_data.c*
- *sensing_node_network_data.h*

Moreover, the generation process output includes a report shown in the SPC5-STUDIO console while the command is in execution. The same report is also stored in a *.txt* file within the same folder. Even if an error occurs during the processing of one of the enabled networks, the generation task continues to process the other enabled networks.

In the generated files you can find:

- `AI_SENSING_NODE_NETWORK C #define`
  – Several C *#defines* are generated in the *.h* and *_data.h* files. Most of these *#defines* are used to instruct the compiler preprocessor on how to configure properly the dimension and allocation of the neural network data structure.

Table 1. List of *#defines*

C language #define	Description
`AI_SENSING_NODE_NETWORK_MODEL_NAME`	A C string with the C name of the model
`AI_SENSING_NODE_NETWORK _IN/OUT_NUM`	It indicates the total number of input/output tensors.
`AI_ SENSING_NODE_NETWORK _IN/OUT`	A C table (*ai_buffer* type) to describe the input/output tensors (see *ai__run()* function
`AI_SENSING_NODE_NETWORK _IN/OUT_SIZE`	A C table (integer type) to indicate the number of items by input/output tensors (= H x W x C)
`AI_SENSING_NODE_NETWORK _IN/OUT_1_SIZE`	It indicates the total number of items for the first input/ output tensor
`AI_SENSING_NODE_NETWORK_IN/OUT_1_SIZE_BYTES`	It indicates the size in bytes for the first input/output tensor (see *ai__run()* function)
`AI_SENSING_NODE_NETWORK_DATA_ACTIVATIONS_SIZE`	It indicates the minimal size in bytes provided by a client application layer as a working buffer (see *ai__init()* function)
`AI_SENSING_NODE_NETWORK_DATA__WEIGHTS_SIZE`	It indicates the size in bytes of the generated weights/ bias buffer segment
`AI_SENSING_NODE_NETWORK _INPUTS_IN_ACTIVATIONS`	It indicates that the input buffers are usable from the activations buffer. It is defined only if the --*allocate-inputs* option is used
`AI_SENSING_NODE_NETWORK_OUTPUTS_IN_ACTIVATIONS`	It indicates that the outputs buffers are usable from the activations buffer. It is defined only if the'--*allocate-outputs* option is used

- `ai_sensing_node_network_create`

```
ai_error ai_sensing_node_network_create(ai_handle* network, const ai_buffer*
network_config);
ai_handle ai_sensing_node_network_destroy(ai_handle network);
```

This is the initial function invoked by the application to create an instance of the neural network. The `ai_handle` is updated after the network creation with the pointer to the entire structure. From this moment, this handle is passed to all the other neural network related functions. The `network_config` parameter is a specific network configuration buffer (opaque structure) coded as *ai_buffer* type. The network configuration is decided before the code generation. Therefore, it cannot be dynamically changed by the application.

When the instance is no more used by the application, the `ai_sensing_node_network_destroy()` function should be called to release the possible allocated resources.

- `ai_sensing_node_network_init`

```
ai_bool ai_sensing_init(ai_handle network, const ai_network_params* params);
```

This mandatory function is used by the application to initialize the internal runtime data structures and to set the activations buffer and weights buffer.

The `params` parameter is a *structure* (*ai_network_params* type) to pass the references of the generated weights (*params* field).

The `network` handler should be a valid handle (see `ai_sensing_node_network_create()` function).

- `ai_sensing_node_network_run`

```
ai_i32 ai__run(ai_handle network, const ai_buffer* input, ai_buffer* output);
```

This function is called to feed the neural network. The `input` and `output` buffer parameters (*ai_buffer* type) provide the input tensors and store the predicted output tensors. The returned value is the number of the input tensors processed.

4 AI-car sensing node

4.1 Precautions for use

The EMC performance of the AEKD-AICAR1 is detailed below:

- for emission, full compliance with EN IEC 61000-6-3 and EN 55032 standards
- for immunity, partial compliance with EN IEC 61000-6-1 and EN 55035 standards, as the kit is not immune to the following types of disturbance:
  – continuous RF electromagnetic field disturbances at the frequency of 260 MHz (in vertical polarization 3 V/m) and 340 MHz (in horizontal polarization 3 V/m);
  – continuous induced RF disturbances at the frequency of 43MHz @ 3V.

If exposed to the above disturbances, the kit might change its operation mode without recovering the previous working condition.

During immunity tests, the kit achieved level C. This means that the kit was not damaged during the test, but required the intervention of an operator to reset it. If the kit is exposed to these frequencies and shows changes in the working condition, turn the kit off and switch it on again.

4.2 Hardware overview

The AEKD-AICAR1 AI-car sensing node consists of:

1. The AEK-MCU-C4MLIT1, designed to address automotive and transportation applications and other applications requiring automotive safety and security levels. The board exploits the functionality of the SPC58EC80E5 32-bit automotive grade ASIL-B microcontroller with 4 MB flash memory, full access to the two MCU cores, GPIOs and peripherals such as ISO CAN FD (with a transceiver), and UART. The AEKD-AICAR1 uses the AEK-MCU-C4MLIT1 evaluation board to process discrete acceleration variations (acquired from the three-axis IMU system) in order to run a pretrained neural network. Moreover, the board drives the AEK-LCD-DT028V1 LCD display board to show neural network results.
2. The AEK-CON-SENSOR1 with the STEVAL-MKI206V1 evaluation board, which embeds the AIS2DW12 IMU sensor. The AEK-CON-SENSOR1 connector board is designed to interface MEMS sensor boards in a DIL 24 socket with SPC58 Chorus MCUs. The connector board includes a 1.8 V LDO voltage regulator to supply the AIS2DW12 inertial measurement sensor. The AIS2DW12 sensor is used to acquire acceleration values on a three-axis reference system.
3. The AEK-LCD-DT028V1, which hosts a 2.8” LCD display with resistive touch to display the graphical user interface (GUI) and interact with the SPC58 Chorus family microcontrollers. The TFT LCD display has a resolution of 240x320 pixels and features the resistive touch managed by an on-board touch screen controller. Both the display and the touch are controlled via SPI by the MCU. The AEK-LCD-DT028V1 display is used to show results of the LSTM neural network computation.
4. The power supply unit.

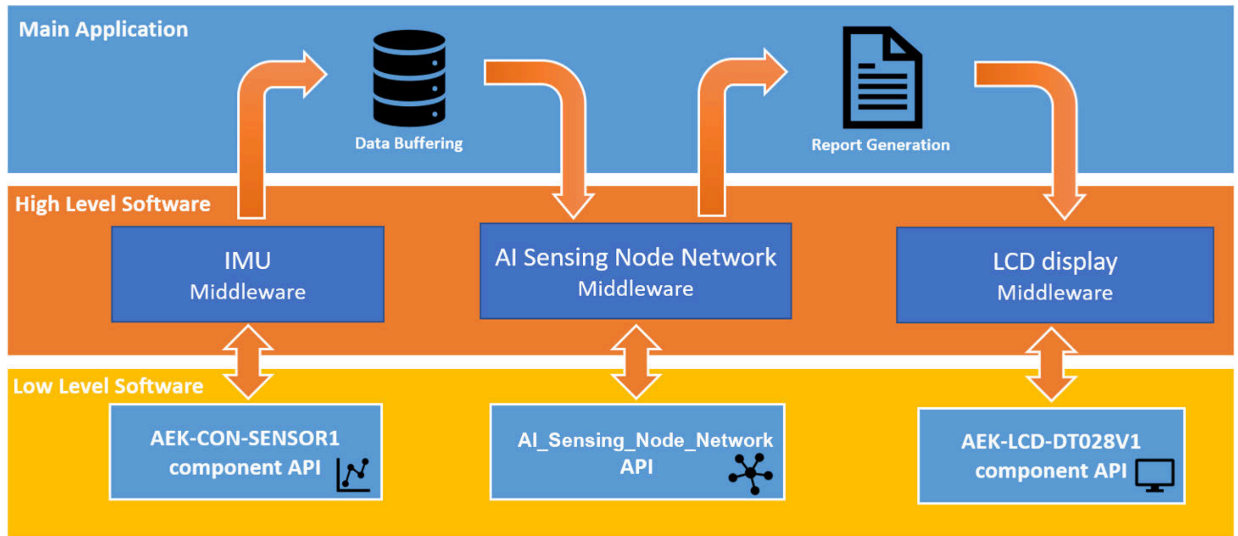
   To power the AEKD-AICAR1, use the switch to choose one of the following options:

   a. the internal battery pack with six AAA batteries;
   b. an external 12 V DC power supply directly connected to the AEK-MCU-C4MLIT1.

4.3 Software architecture overview

The AEKD-AICAR1 software architecture consists of three layers: low-level software, high-level software, and the main application.

Figure 23. AEKD-AICAR1 software architecture



4.3.1 Low-level software

This is the lower layer of the software architecture. It is responsible for interfacing the MCU with the other boards. This layer includes:

- AEK-CON-SENSOR1 AutoDevKit component APIs
- AI sensing node neural network API
- AEK-LCD-DT028V1 AutoDevKit component API

4.3.1.1 *AEK-CON-SENSOR1 AutoDevKit component APIs*

The AEK-CON-SENSOR1 MEMS sensor drivers are included in a component that belongs to the AutoDevKit software (STSW-AUTODEVKIT). The library is written in MISRA C. The target software is generated automatically according to the code generation and pin allocation paradigm included in the AutoDevKit design flow. The drivers can be easily adapted to any ST MEMS sensor, even if not directly supported by the software library.

To configure the AEK-CON-SENSOR1 AutoDevKit component for the AEKD-AICAR1 application, follow the procedure below.

Step 1. Go on [**SPC58ECxx Platform Component RLA**] in the project explorer and add a new [**AEK-CON-SENSOR1 Component RLA**] from the list.

Step 2. Press on the allocation button to make the AutoDevKit automatically allocate all the required MCU pins.

Step 3. Use the *PinMap* editor to verify that all the relevant pins have been allocated (four for the SPI and two for the interrupts).

Step 4. Use the [**Board Viewer**] for information on how to connect the MCU board to the sensor board.

Then, the application uses the following available AEK-CON-SENSOR1 APIs:

– To initialize the MEMS sensor board: `void init_mems()`

– To configure the AIS2DW12 accelerometer: `ais2dw12_0.methods->configure_sensor(&ais2dw12_0, power_mode_12bit, continuous, _25Hz, _2g, low_pass_1_and_2_odr_div_20);`

– To get acceleration values: `ais2dw12_0.methods->get_accelerations(&ais2dw12_0, &accelerations);`

Refer to the AEK-CON-SENSOR1 user manual for further details.

4.3.1.2 *AI sensing node neural network API*

For detailed information on this API, see Section 3.2 .

4.3.1.3 *AEK-LCD-DT028V1 AutoDevKit component API*

All the development drivers related to the LCD display based on the AEK-LCD-DT028V1 board are included in a component that belongs to the AutoDevKit software (STSW-AUTODEVKIT) version 1.5.0 (or higher). The library is written in MISRA C. The target software is generated automatically according to the code generation and pin allocation paradigm included in the AutoDevKit design flow.

To configure the AEK-LCD-DT028V1 AutoDevKit component for the AEKD-AICAR1 application, follow the procedure below.

Step 1. Go to the [**SPC58ECxx Platform Component RLA**] in the project explorer and add a new [**AEK-LCD-DT028V1 Component RLA**] from the list.

Step 2. Select the font file to embed in your application. (font size = 12 pt).

Step 3. Go to the configuration portion of the component and add an entry on the board list, selecting the SPI port (DSPI) and the related chip selects (CS) for the display and touch screen.

Step 4. Use the [**Board Viewer**] to determine how to connect the MCU board to the LCD display board.

The following AEK-LCD-DT028V1 APIs are used in our system:

– To initialize the display: `aek_ili9341_init(AEK_LCD_DEV0);`

– To set the orientation: `aek_ili9341_setOrientation(AEK_LCD_DEV0, ILI9341_ROTATE_LANDSCAPE);`

– To draw display icons and show results:

```
aek_ili9341_fillRect(AEK_LCD_DEV0,102,20,150,160,BLUE1)
aek_ili9341_drawImage(AEK_LCD_DEV0,110, 20, (int16_t)parking.width,
(int16_t)parking.height, parking.pixel_data, PIXFMT_RGB_565);
aek_ili9341_drawString(AEK_LCD_DEV0, 102, 160, "PARKING", color,
font24pt);
```

4.3.2 High-level software

This layer includes:

- IMU middleware
- AI sensing node network middleware
- LCD display middleware

4.3.2.1 *IMU middleware*

The IMU middleware is responsible for:

- initializing the inertial measurement unit: `void init_IMU()`
- acquiring and removing the IMU offset values on the three-axis reference system due to the incorrect ECU positioning and gravity effect on the z -axis: `void get_offset_IMU(void)`
- collecting data from IMU and perform data buffering on the *buffer_difference_accelerations* array of the Δ acceleration on the three-axis:

```
void collect_data_IMU(void)
buffer_difference_accelerations[BUFFER_ACCELERATIONS_SIZE] = {
ΔAcc_x_sample1, ΔAcc_y_sample1, ΔAcc_z_sample1, ΔAcc_x_sample2, ΔAcc_y_sample2,
ΔAcc_z_sample2, ΔAcc_x_sample3, ΔAcc_y_sample3, ΔAcc_z_sample3, …….}
```

  where `BUFFER_ACCELERATIONS_SIZE = S * 1 * D;`

  S = 50 is the number of samples of LSTM network window.

  D = 3 is three times the Δ acceleration on the three-axis.

- returning the *buffer_difference_acceleration* array for the neural network processing: `float* get_diff_acceleration_buffer_IMU(void);`
- sending a frame of the raw acceleration array for the neural network training with the serial protocol through Google Colab: `void send_frame(void)`

Note: To evaluate the accuracy of the LSTM neural network, we have created an IMU_LOG.h file that contains a fixed buffer_difference_accelerations array. This array has been loaded on the flash memory and contains 3-axis Δ acceleration samples with the known car state (parking, normal, bumpy, skid).

If you want to read from the log file, use the following instructions:

- `void get_offset_LOG(void)`
- `void collect_data_LOG(void)`
- `float* get_diff_acceleration_buffer_IMU(void);`

4.3.2.2 *AI sensing node network middleware*

This middleware is responsible for:

- initializing the LSTM neural network: `void init_ai_sensing_node_CNN(void)`
- returning the LSTM neural network to initial data: `void deinit_ai_sensor_node_CNN(void)`
- running the LSTM neural network: `void run_ai_sensor_node_network(void)`

4.3.2.3 *LCD display middleware*

The LCD display middleware is responsible for the icons and messages to display according to the results of LSTM neural network. It is responsible also for:

- initializing the LCD display: `void infotainment_system_init(void);`
- printing a message (with a specific color) and drawing an icon: `void display_print(uint8_t msg, uint16_t color);`

4.3.3 Main application software

The AEKD-AICAR1 application has three operating working modes:

- Acquisition
- Real-time calculation (default)
- Offline calculation

4.3.3.1 *Acquisition*

This working mode is used to perform an LSTM neural network training.

If `#define ACQUISITION` is applied on the head of main application:

- the calculation of the LSTM is deactivated
- the serial protocol is initialized to send three axis acceleration samples with a baud rate of 38400 bps

4.3.3.2 *Real-time calculation*

This working mode is used when the ECU is installed on a car and the LSTM neural network is already pretrained. To activate this working mode, comment the `#define ACQUISITION`.

At power-up, press the SW_1 button on the AEK-MCU-C4MLIT1. If the button is well pressed, then the LED_2 switches off.

4.3.3.3 *Offline calculation*

This is the default operating mode at power-on.

The LSTM neural network is computed on the log file.

To activate this working mode, comment the `#define ACQUISITION`.

At power-up, press the SW_1 button on the AEK-MCU-C4MLIT1 until the LED_2 switches on.

5 Test and results

5.1 Environment setup

The AEKD-AICAR1 sensing node has been installed in a sedan car with a normal shock absorber system.

The x-axis of the AIS2DW12 accelerometer must have the same direction of the car motion.

5.2 Test performed

The AEKD-AICAR1 has been tested in several conditions:

- normal road: a straight or curved road with normal surface conditions
- bumpy road: a straight or curved road with a bumpy road surface
- skidding/swerving: on a normal road and on a bumpy road
- parking
- stopped car: with the engine on and off

5.3 Results

Table 2. Test conditions and results

Test condition	Number of tests	Number of failures	Failure percentage
Normal road	10	0	0%
Bumpy road	10	0	0%
Skid normal road	10	2 (detected as bumpy road)	20%
Skid bumpy road	10	2 (detected as bumpy road)	20%
Parking engine on	10	0	0%
Parking engine off	10	0	0%

Totally, we have performed 60 tests, with four failures. Then, the failure percentage is 6%.

As shown by the above tests, the AEKD-AICAR1 has some issues with the skid state recognition, due to the low number of test cases used during the training phase.

The system has an accuracy of 94% when the ECU is directly installed on the sedan. Instead, with the offline simulation, the accuracy from the confusion matrix is 93%.

6 Schematic diagrams

Note: *The AEKD-AICAR1 kit consists of the following evaluation boards: AEK-MCU-C4MLIT1, AEK-CON-SENSOR1, AEK-LCD-DT028V1, and STEVAL-MKI206V1. You can find their detailed schematic diagrams at the related web pages:*

- *AEK-MCU-C4MLIT1 schematic diagrams*
- *AEK-CON-SENSOR1 schematic diagrams*
- *AEK-LCD-DT028V1 schematic diagrams*
- *STEVAL-MKI206V1 schematic diagrams*

7 Bill of materials

Note: The AEKD-AICAR1 kit consists of the following evaluation boards: AEK-MCU-C4MLIT1, AEK-CON-SENSOR1, AEK-LCD-DT028V1, and STEVAL-MKI206V1. You can find their detailed schematic diagrams at the related web pages:

- *AEK-MCU-C4MLIT1 bill of materials*
- *AEK-CON-SENSOR1 bill of materials*
- *AEK-LCD-DT028V1 bill of materials*
- *STEVAL-MKI206V1 bill of materials*

Appendix A Regulatory compliance

Notice for US Federal Communication Commission (FCC)

For evaluation only; not FCC approved for resale

FCC NOTICE

FCC NOTICE - This kit is designed to allow:

(1) Product developers to evaluate electronic components, circuitry, or software associated with the kit to determine whether to incorporate such items in a finished product and

(2) Software developers to write software applications for use with the end product.

This kit is not a finished product and when assembled may not be resold or otherwise marketed unless all required FCC equipment authorizations are first obtained. Operation is subject to the condition that this product not cause harmful interference to licensed radio stations and that this product accept harmful interference. Unless the assembled kit is designed to operate under part 15, part 18 or part 95 of this chapter, the operator of the kit must operate under the authority of an FCC license holder or must secure an experimental authorization under part 5 of this chapter 3.1.2.

Notice for Innovation, Science and Economic Development Canada (ISED)

For evaluation purposes only. This kit generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to Industry Canada (IC) rules.

À des fins d'évaluation uniquement. Ce kit génère, utilise et peut émettre de l'énergie radiofréquence et n'a pas été testé pour sa conformité aux limites des appareils informatiques conformément aux règles d'Industrie Canada (IC).

Notice for the European Union

This device is in conformity with the essential requirements of the Directive 2014/30/EU (EMC) and of the Directive 2015/863/EU (RoHS).

Notice for the United Kingdom

This device is in compliance with the UK Electromagnetic Compatibility Regulations 2016 (UK S.I. 2016 No. 1091) and with the Restriction of the Use of Certain Hazardous Substances in Electrical and Electronic Equipment Regulations 2012 (UK S.I. 2012 No. 3032).

Appendix B Google Colab code for the neural network training

```
%pip uninstall tensorflow
%pip install tensorflow==2.4.0

from google.colab import files
uploaded = files.upload()

import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split , StratifiedShuffleSplit
import os
import random
from sklearn.metrics import confusion_matrix
import seaborn as sn
import matplotlib.pyplot as plt

os.environ['PYTHONHASHSEED']=str(1)
tf.random.set_seed(2021)
np.random.seed(2021)
random.seed(2021)

Y_LABELS={'P':0,'N':1,'B':2,'S':3}
TIMESERIES_LEN = 50

def trip_framing(trip,label,frame_size,db_x,db_y):
    a = np.array( trip )
    for i in np.arange( 0, a.shape[0]-frame_size, frame_size ):
        x = a[i:i+frame_size]
        db_x.append( x )
        db_y.append( Y_LABELS[label] )

db = pd.read_csv('Diff_profile.csv',sep=',')
print(db.keys())
states = db['Status'].value_counts()

scaler = MinMaxScaler()
ts_status = db.Status

ts_diff_Ax = (db.Acc_x.to_numpy().reshape(-1,1))/9.81
ts_diff_Ay = (db.Acc_y.to_numpy().reshape(-1,1))/9.81
ts_diff_Az = (db.Acc_z.to_numpy().reshape(-1,1))/9.81
ts_time = db['Time']

rows = ts_status.shape[0]
db_x = []
db_y = []

for states_id in states.keys():
    trip = []
    cnt = 0
    for i in range(rows):
        if ts_time[i] == 100:
            if len(trip) > 0:
                trip_framing( trip, states_id, TIMESERIES_LEN, db_x, db_y)
                trip=[]
        if ts_status[i] == states_id and cnt < 7500:
             trip.append( [ts_diff_Ax[i],ts_diff_Ay[i],ts_diff_Az[i]] )
             cnt += 1

x_train, x_test, y_train, y_test = train_test_split(db_x, db_y, test_size=0.4,
random_state=21,stratify=db_y)
x_train = np.asarray(x_train)[:,:,:,0]
x_test = np.asarray(x_test)[:,:,:,0]
y_train = np.asarray( y_train )
y_test = np.asarray( y_test )
db_stats = pd.Series( y_test )
```

```
## Conv1D based model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=16, kernel_size=3, activation='relu',
input_shape=(TIMESERIES_LEN, 3)),
  tf.keras.layers.Conv1D(filters=8, kernel_size=3, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(4, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=1000)

model.summary()

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

Y_pred = model.predict(x_test)
y_pred = np.argmax(Y_pred, axis=1)
confusion_matrix = tf.math.confusion_matrix(y_test, y_pred)

plt.figure()
sns.heatmap(confusion_matrix,
            annot=True,
            xticklabels=Y_LABELS,
            yticklabels=Y_LABELS,
            cmap=plt.cm.Blues,
            fmt='d', cbar=False)
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

model.save('model_car_sts.h5')
```

Appendix C Python script code for the data acquisition and parsing

```
import serial as sr
import matplotlib.pyplot as plt
import numpy as np
import time
from csv import writer

def append_list_as_row(list_of_elem):
    # Open file in append mode
    with open(file_name, 'a+') as write_obj:
        # Create a writer object from csv module
        csv_writer = writer(write_obj, delimiter=',',lineterminator='\n')
        # Add contents of list as last row in the csv file
        csv_writer.writerow(list_of_elem)

def create_file_with_header(list_of_elem):
    # Open file in append mode
    with open(file_name, 'wb') as write_obj:
        # Create a writer object from csv module
        csv_writer = writer(write_obj)
        # Add contents of list as last row in the csv file
        csv_writer.writerow(list_of_elem)

s = sr.Serial('COM31', 38400);

#Write header
named_tuple = time.localtime() # get struct_time
time_string = time.strftime("%m_%d_%Y_%H_%M_%S", named_tuple)
file_name = 'data_logger_' + time_string + '.csv'

print('File Created: ' + file_name)
row_contents = ['Time','Acc_x','Acc_y','Acc_z']
create_file_with_header(row_contents)

while 1:
    string_line=s.readline()      #ascii
    string_line = string_line.strip('\n')
    line_as_list = string_line.split(b',')
    time_stamp = float(line_as_list[0])
    acc_x = float(line_as_list[1])
    acc_y = float(line_as_list[2])
    acc_z = float(line_as_list[3])
    row_contents = [time_stamp,acc_x,acc_y,acc_z]
    append_list_as_row(row_contents)
```

Revision history

Table 3. **Document revision history**

Date	Revision	Changes
20-Sep-2022	1	Initial release.

Contents

1 **Neural network basic principles** 2
1.1 Artificial neural network 2
1.2 Long short-term memory recurrent neural network (LSTM RNN) 2
2 **Designing an AI-car sensing node** 4
2.1 Tool-set introduction 4
2.2 Creating a Google Colab notebook 4
2.3 Colab notebook setup and package importing 5
2.4 AI-car sensing node life cycle 6
2.4.1 Model definition 6
2.4.2 Model training 7
2.4.3 Model fitting and compilation 11
2.4.4 Model evaluation 12
3 **AutoDevKit ecosystem** 14
3.1 SPC5-STUDIO-AI plugin 14
3.1.1 How to import SPC5-STUDIO-AI using the standard importing procedure 16
3.1.2 How to import the pretrained LSTM neural network 17
3.1.3 How to analyze the pretrained LSTM neural network 19
3.2 SPC5-STUDIO-AI API 21
4 **AI-car sensing node** 23
4.1 Precautions for use 23
4.2 Hardware overview 23
4.3 Software architecture overview 23
4.3.1 Low-level software 24
4.3.2 High-level software 25
4.3.3 Main application software 26
5 **Test and results** 27
5.1 Environment setup 27
5.2 Test performed 27
5.3 Results 27
6 **Schematic diagrams** 28
7 **Bill of materials** 29
Appendix A **Regulatory compliance** 30
Appendix B **Google Colab code for the neural network training** 31
Appendix C **Python script code for the data acquisition and parsing** 33

Revision history 34
List of tables 37
List of figures 38

List of tables

Table 1. List of *#defines* 21
Table 2. Test conditions and results 27
Table 3. Document revision history 34

List of figures

Figure 1. AEKD-AICAR1 evaluation kit 1
Figure 2. Artificial neural network 2
Figure 3. Recurrent neural network 3
Figure 4. AI-car sensing node: car state classification 4
Figure 5. Project file 5
Figure 6. Acceleration variations 8
Figure 7. Time-based dataset example 9
Figure 8. Adding the status column 10
Figure 9. Creating CSV files 10
Figure 10. Complete model generation 12
Figure 11. Confusion matrix 13
Figure 12. Installing the plugin 14
Figure 13. Install new software 15
Figure 14. Creating a new SPC5-STUDIO application 16
Figure 15. Selecting the component 16
Figure 16. Adding a new component 17
Figure 17. Selecting [**SPC5 AI Component RLA**] 17
Figure 18. Network list imported 18
Figure 19. Adding a new network 18
Figure 20. Configuring the parameters 18
Figure 21. Selecting [**Analyze**] 19
Figure 22. Clicking on [**Analyze**] 20
Figure 23. AEKD-AICAR1 software architecture 24

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2022 STMicroelectronics – All rights reserved