RZ/A2M

Automatic Number-Plate Recognition (ANPR)
(eAI demo applying “Renesas eAI translator” and “TensorFlow Lite for MCU”)

Application Note

Introduction

This document describes the RZ/A2M Automatic Number-Plate Recognition (ANPR) demo software.

The letter recognition is processed by a Convolutional Neural Network (CNN) that has been trained with the TensorFlow Framework. It is implemented in 4 different flavours prepared with Renesas “eAI translator” and “TensorFlow Lite for MCU”.

The pre-processing of the camera image is accelerated with the Dynamically Reconfigurable Processor IP (DRP).

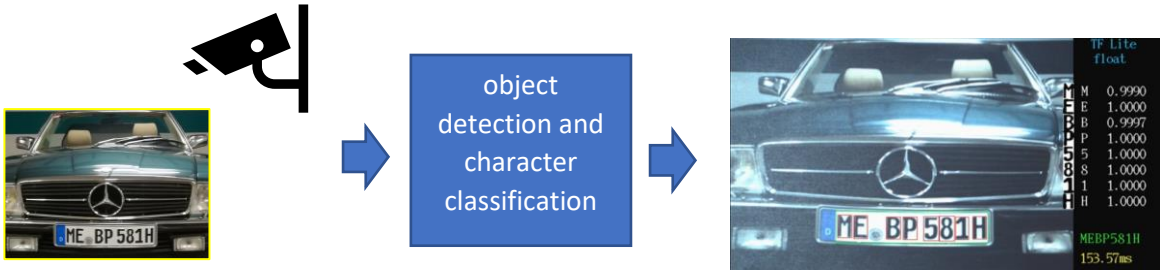
Target Device

RZ/A2M

Contents

1. Overview

The demo detects number plates and their characters from images taken by a camera. Additionally, the characters are classified to predict the number plate content. The result is shown on a display (HDMI monitor).



camera RZ/A2M monitor

The ANPR demo focusses on typical German, single-row number plates.
Only one number plate per image is used for classification. It is simply the number plate with the highest number of characters.

The ANPR demo software is available for 2 different platforms:

1. RZ/A2M Evaluation Board Kit
   https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/rz-cortex-a-mpus/rza2m-evaluation-kit-rza2m-evaluation-kit
   e²studio project name: “ANPR_RZA2M_EBK“

2. Gadget Renesas board “GR-MANGO”
   https://www.renesas.com/eu/en/products/gadget-renesas/boards/gr-mango
   e²studio project name: “ANPR_RZA2M_GR_MANGO”

Used e²studio version: e²studio 2021.04 (2021-04 (21.4.0))
eAI translator 2.0.0 (e²studio plugin)

There are a few platform-dependant differences. These are marked with the e²studio project name.

The user switches are used to select different options:

	ANPR_RZA2M_EBK	ANPR_RZA2M_GR_MANGO
toggle the Auto Exposure (AE) on/off	SW3	n.a.
select the image and information shown at the display (*1)	SW4	SW2
select one of 4 different neural network representations (*2)	SW5	SW3
reset	n.a.	SW1

(*1) Besides the main, coloured image with number plate detection result, the images generated within the pre-processing can be shown.
(*2) implemented neural network representations:

1. eAI translator output (32bit floating point model)
2. TensorFlow Lite for MCU (32bit floating point model)
3. TensorFlow Lite for MCU (post-training quantization model (8bit integer))
4. TensorFlow Lite for MCU (quantization aware training model (8bit integer))

The LED on the CPU board toggles after each processed image to indicate the overall performance. If no number plate is found, the LED blinking frequency increases, because the routines to find and classify characters are skipped.
The LED blinks in 2 different colours. If the last 6 consecutive number plate predictions are identical, the LED turns green. Otherwise, it stays or turns red.

The demo software is based on the “Object Detection Sample Program” (rza2m_drp_dynamic_sample3_freertos_gcc).
Please check it’s document “Object Detection Sample Program Application Note”
/ANPR_RZA2M_* /doc/r01an4787ej0113-rza2m-freertos-object-detection-gcc.pdf
for additional information.
The “Object Detection Sample Program” has been released within the “RZ/A2M Group RZ/A2M Simple Applications Package” that is available for the RZ/A2M Evaluation Board Kit (EBK) and GR-Mango board.
The latest version of the “simple applications package” is available for download at
https://www.renesas.com/eu/en/software-tool/rza2m-freertos-software-package

2. Operation Confirmation Conditions

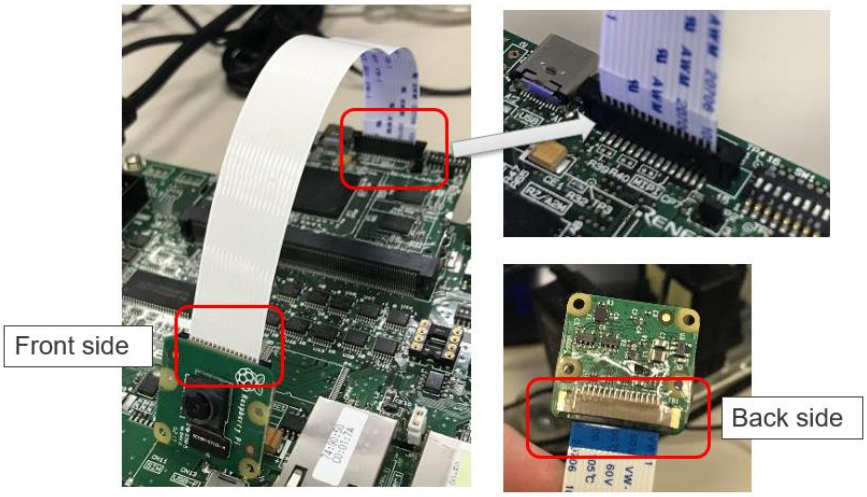
2.1. Evaluation Board Kit setup

Please setup the development kit board as described in
ANPR_RZA2M_EBK/doc/readme-e.txt
Especially the setting for DIP switches and jumpers needs to be followed.

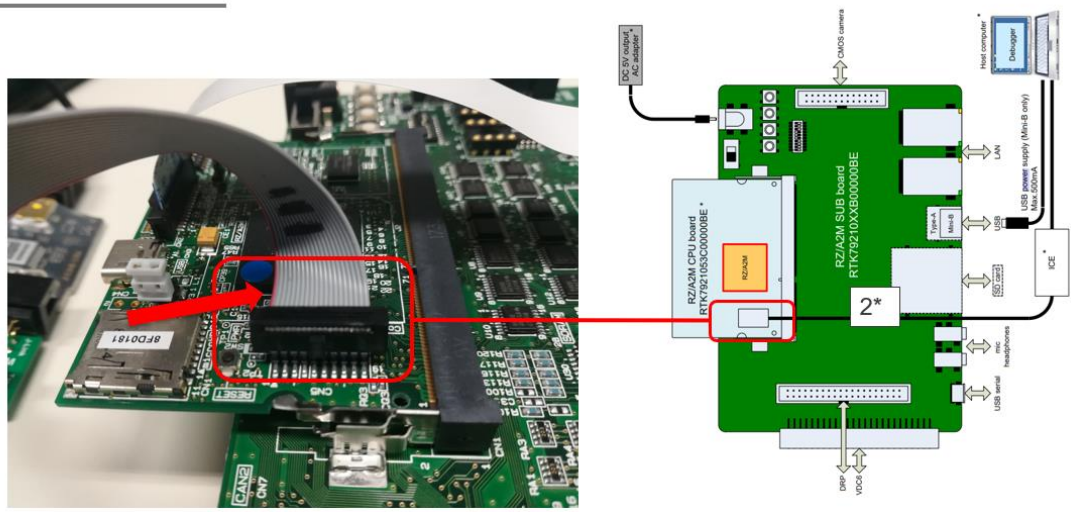
Additional information can be found in the “Object Detection Sample Program Application Note”
ANPR_RZA2M_EBK/doc/r01an4787ej0113-rza2m-freertos-object-detection-gcc.pdf
.

2.1.1. Evaluation Board Kit setup (quick start)

MIPI CAMERA CONNECTION



J_LINK CONNECTION

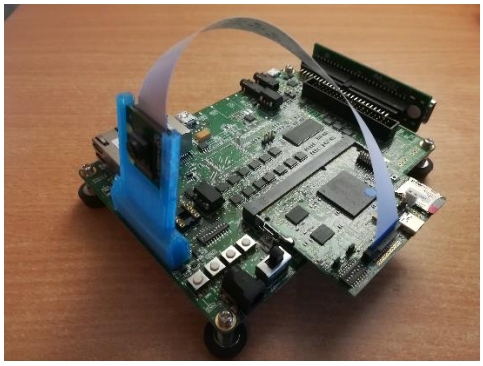


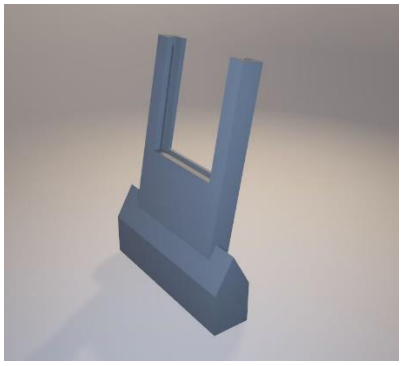
2.1.2. Raspberry PI camera holder (3D model)

The following camera holder is quite useful to stabilize the camera.

RZ/A2M development kit gadget: Raspberry PI camera holder (3D model)

https://renesasrulz.com/rz/m/rz_files/3399





2.2. GR-MANGO board setup

Please setup the development kit board as described in
ANPR_RZA2M_GR_MANGO/doc/readme-e.txt

"(4) Downloading sample code

Connect GR-MANGO CN1 and PC via USB cable.
PC detects the GR-MANGO as MBED drive.
Drag-and-drop the binary file
(e.g. ANPR_RZA2M_GR_MANGO/HardwareDebug/ANPR_RZA2M_GR_MANGO.bin)
to the MBED drive.

(5) Executing sample code

Connect the camera to GR-MANGO CN13 and connect display to CN9 via HDMI cable.

Ensure that complete to download program, and push reset button on GR-PEACH."

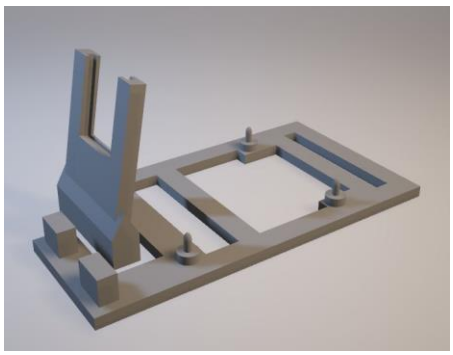
Additional information can be found in the "Object Detection Sample Program Application Note"
ANPR_RZA2M_GR_MANGO/doc/r01an4787ej0113-rza2m-freertos-object-detection-gcc.pdf
.

2.2.1. Raspberry PI camera holder (3D model)

The following camera holder is quite useful to stabilize the camera.

RZ/A2M GR-MANGO board gadget: board mount with Raspberry PI camera holder (3D model)

https://renesasrulz.com/rz/m/rz_files/3402



3. Software

3.1. Folder Structure

```
ANPR_RZAeM_EBK or ANRPR_RZA2M_GR_MANGO
+---bootloader
+---doc
+---generate
|   +---compiler
|   +---configuration
|   +---drivers
|   +---os_abstraction
|   +---sc_drivers
|   |   +---r_adc
|   |   +---r_cbuffer
|   |   +---r_ceu
|   |   +---r_drp
|   |   |   +---doc
|   |   |   +---drp_lib
|   |   +---r_drp_custom
|   |   |   +---doc
|   |   |   \---drp_lib_custom
|   |   +---r_mipi
|   |   +---r_ostm
|   |   +---r_riic
|   |   +---r_rvapi
|   |   +---r_scifa
|   |   \---r_vdc
|   \---system
\---src
    +---config_files
    +---FreeRTOS
    +---neural_networks
    |   +---tensorflow_lite
    |   |   +---TF_lite_float
    |   |   +---TF_lite_post_training_quantization
    |   |   \---TF_lite_quantization_aware_training
    |   \---Translator
    +---renesas
    |   +---application
    |   |   +---common
    |   |   |   +---camera
    |   |   |   +---console
    |   |   |   +---perform
    |   |   |   +---port_settings
    |   |   |   \---render
    |   |   +---GR-MANGO      (ANRPR_RZA2M_GR_MANGO only)
    |   |   +---inc
    |   |   |   +---camera
    |   |   |   \---lcd
    |   |   \---subBoard      (ANRPR_RZA2M_EBK only)
    |   \---hwsetup
    +---tensorflow_lite
    |   +---patches
    |   +---tensorflow
    |   \---third_party
    \---user_prog
```

FreeRTOS is open-source software distributed under the MIT license.
Regarding the MIT license,please refer to
https://opensource.org/licenses/mit-license.php
FreeRTOS is a real-time operation system kernel for embedded microcomputer.
This sample program is based on FreeRTOS OS Abstraction Version 3.5

The TensorFlowLite library is licensed under the Apache License, Version 2.0.
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
In this sample program, version 2.4.0-ALPHA is used.

3.2. Display type definition

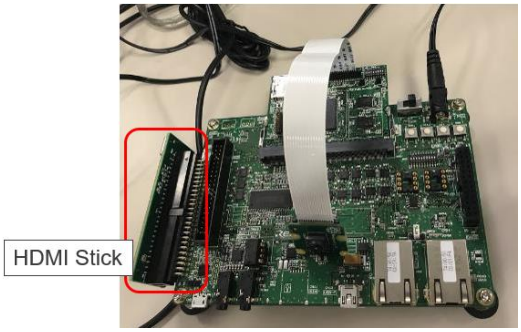
The RZ/A2M Evaluation Board Kit offers different options to connect a display.
To change the display type, please open header file

ANPR_RZA2M_EBK/src/renesas/application/inc/lcd_panel.h

and map LCD_PANEL with one of the pre-defined options (e.g. LCD_PANEL_RSK or LCD_PANEL_DVI).

LCD_PANEL_RSK is the label for a Renesas touch display that can be connected to the Evaluation board kit directly.

LCD_PANEL_DVI is the label for HDMI displays that can be connected via the HDMI interface stick that needs to be connected to the development board.



RZ/A2M Evaluation Board Kit and GR-MANGO board:

If LCD_PANEL_DVI (HDMI) is selected, please check header file

ANPR_RZA2M_EBK/src/renesas/application/inc/lcd/pc_monitor.h

and modify the define for SELECT_MONITOR to select the most suitable configuration for the connected monitor.

3.3. Serial Terminal

For de-bugging and TAT evaluation purposes, several messages can be transferred to a serial terminal (e.g., Tera Term):

- On the RZ/A2M Evaluation board kit, please connect CN5 on the RZ/A2M SUB board to a Windows™ PC, this provides a USB virtual serial port.
  For GR-MANGO, connect CN1 and PC via USB cable.
- Start a serial terminal program (such as PuTTY, HyperTerminal or Tera Term) using the following configuration:
  Baud Rate: 115200
  Data Bits: 8
  Parity: None
  Stop Bits: 1
  Flow Control: None
  COM Port: As shown in Windows™ Device Manager.
- To activate or de-activate certain messages, please configure the define statements for "SerialTerminal_*" given in
  src/renesas/application/inc/r_serial_terminal.h.

3.4. DRP libraries

The Software uses a mixture of released DRP libraries and modified DRP libraries that have been developed at Renesas Electronics Europe.

The libraries are given in
/ANPR_RZA2M_*/generate/sc_drivers/r_drp
/ANPR_RZA2M_*/generate/sc_drivers/r_drp_custom

Please check the documents in sub-directories
r_drp/doc,
r_drp_custom/drp_lib_custom/doc and
r_drp_custom/drp_lib_custom/drp_lib_custom/<lib_name>/doc
for detailed information.

3.5. Extra-ordinary font extension

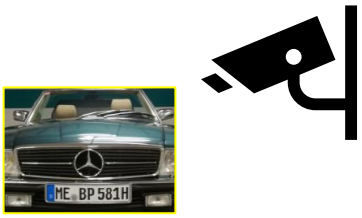
The font used to print text on the display is defined for the standard printable characters.
For the number plate demo that focusses on German number plates, additional font entries for "Ä", "Ö" and Ü" have been added to
src/renesas/application/common/render/fontdata_12x24.c

The extra-ordinary font extension does not follow the ASCII standard.

Ä ≙ "\x80", Ö ≙ "\x81", Ü ≙ "\x82"

4. Image Pre-Processing

4.1. Pre-Processing before step “find contours”



Camera (Raspberry Pi camera V2.1)

Bayer to RGB conversion
(DRP library “r_drp_bayer2argb_color_correction”)

+

Gaussian Blur filter
(DRP library “r_drp_lrgb2l_gaussian_blur”)

Canny Calculate filter
(DRP library “r_drp_canny_calculate”)

+

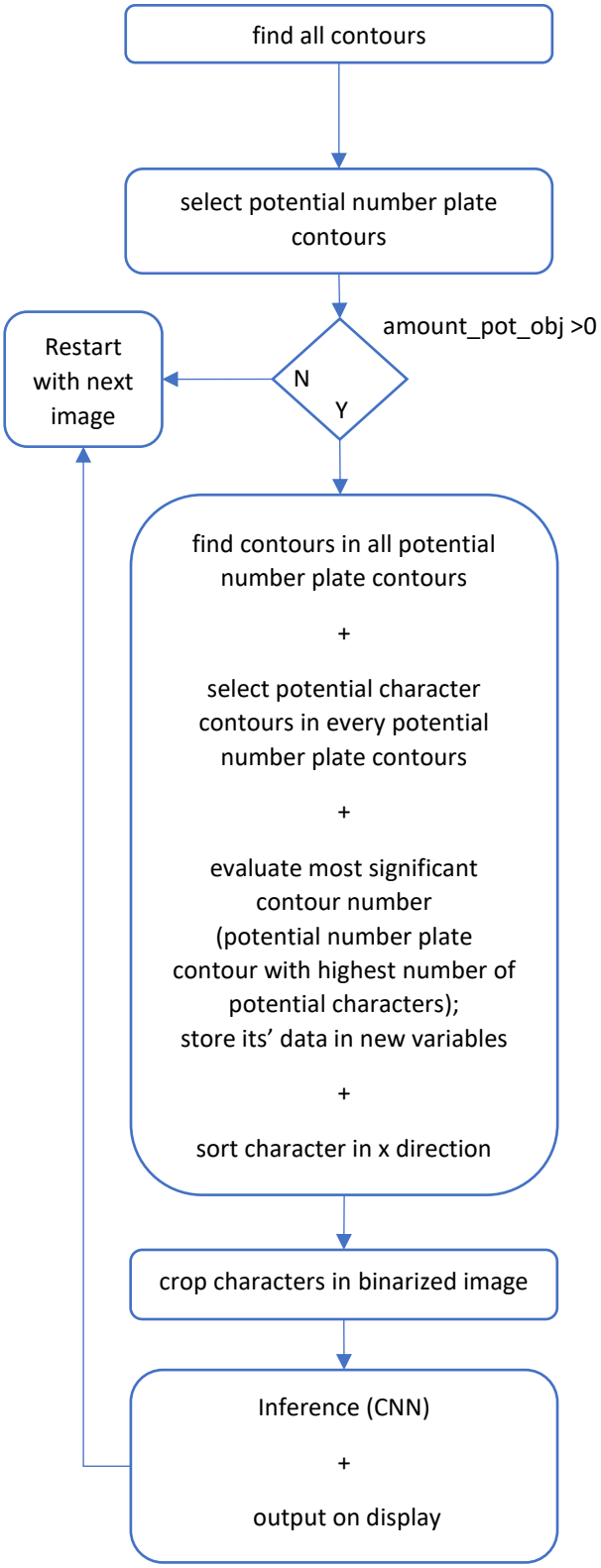
Canny Histeresis filter
(DRP library “r_drp_canny_hysterisis”)

Dilate filter
(DRP library “r_drp_dilate”)

find contours

4.2. Pre-Processing step “find contours”

find contours with DRP library “r_drp_find_contours_crop”.



Find up to FIND_CONTOURS_NUM (15) contours in image (Dilate result).
Result stored in variable contours_rect_adr[]

Select contours with dedicated shape dimensions.
Result:
amount_pot_obj :
amount of potential number plate contours
pot_obj_rect_number[] :
list with indices of potential number plate contours

Find up to FIND_CONTOURS_NUM (15) contours within all potential number plate contours.
Result:
amount_pot_char[]:
amount of potential character for each potential number plate

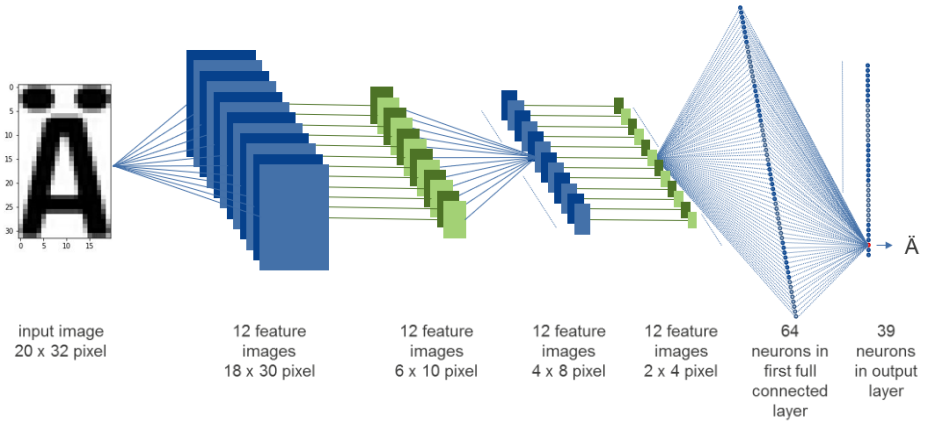
pot_char_rect_number[]
list with indices of potential character contours

Evaluate number plate contour with highest amount of potential characters
mscn:
most significant contour number;
index of potential number plate with highest amount of potential characters
number_plate_contours_rect_adr[]:
character contour data from most significant contour number
number_plate_char_rect_number[]:
list with indices of potential character contours from most significant contour

DRP library “r_drp_resize_bilinear_crop”.

5. Inference with a Convolutional Neural Network (CNN)

CNN architecture



The CNN is described and trained, optimized and quantized with different TensorFlow environments:

- CNN training with TensorFlow 2.1.0 to be compatible with Renesas eAI translator 2.0.0.
- TensorFlow Lite for MCU conversion with TensorFlow 2.4.1.
- Quantization aware training with TensorFlow 2.6.0-dev20210505 (“nightly” release).

CNN definition:

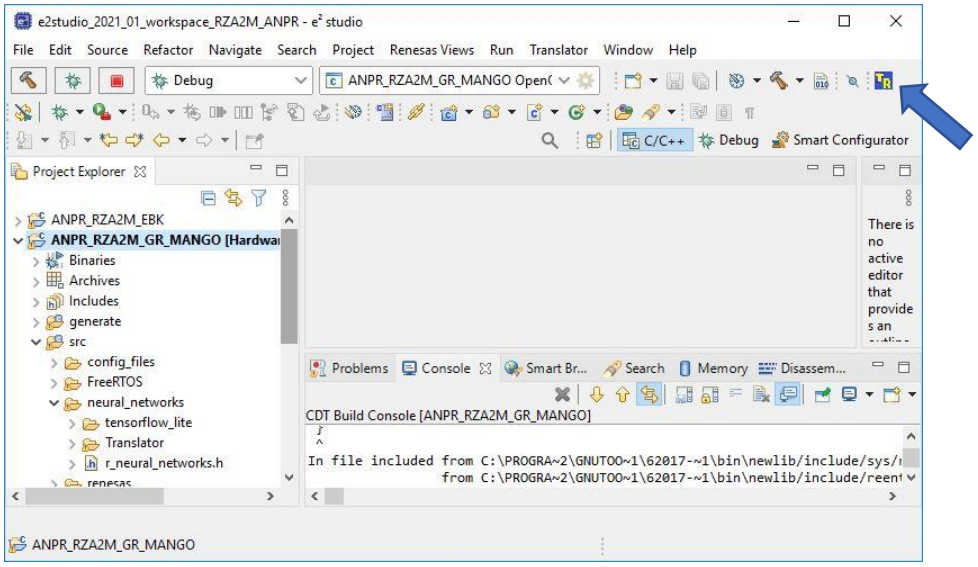
```
model = Sequential()

model.add(Conv2D(12, (3, 3), activation='relu', input_shape=(train_height,train_width,1)))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(12, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dense(len(char_list), activation='softmax'))


sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer=sgd)
```

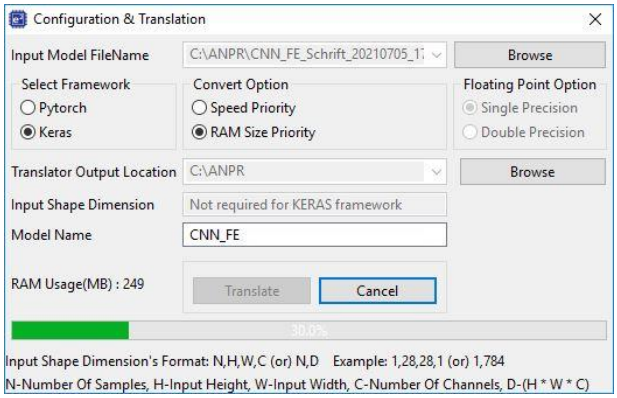
‘Softmax’ has been chosen as activation function for the output layer.
This amplifies the prediction accuracy of the ‘best fit’, but decreases the ability to differentiate between a ‘week’ and ‘strong’ predictions.
(Changing the activation function from ‘softmax’ to ‘relu’ can be an alternative here.)

6. eAI translator Flow

6.1. NN translation (eAI translator)

The frozen model has been translated with the Renesas eAI translator (e²studio plugin).
(e-AI Translator V2.0.0 translates Keras or PyTorch model format to C code.)





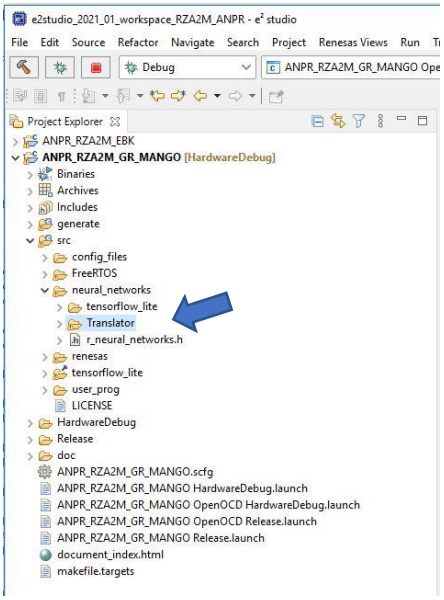
Enter “Input Model FileName” and “Translator Output Location”.

Defining a “Model Name” is optional if your project has one neural network only. If the project has more than one neural network, please define different names.

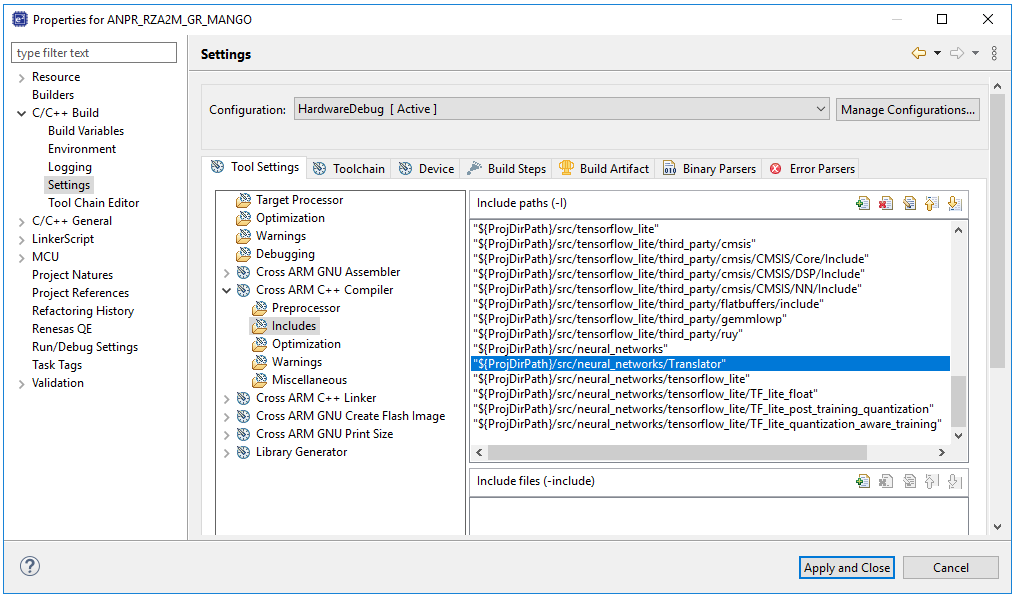
⇨ start “Translate”

6.2. NN integration (e²studio)

Copy the eAI translator result (directory ‘Translator’) into the project directory



Add “Translator” path to the list of Include paths
right-click at project RZA2M_number_plate_detection -> Properties



“Apply and Close”

integrate neural network
e.g.:

```
#include "Typedef.h"
#include "r_neural_networks.h"

float   prediction_init[39]  = { (float)0.0 };
float   *prediction          = &prediction_init[0];
TsInt   dnn_compute_CNN_FE_errorcode;
TsInt   *dnn_compute_CNN_FE_error   = &dnn_compute_CNN_FE_errorcode;

static float  frame_RAM_CCN_float[NN_IN_HEIGHT * NN_IN_WIDTH];


prediction  = dnn_compute_CNN_FE(&frame_RAM_CCN_float[0], dnn_compute_CNN_FE_error) ;
```

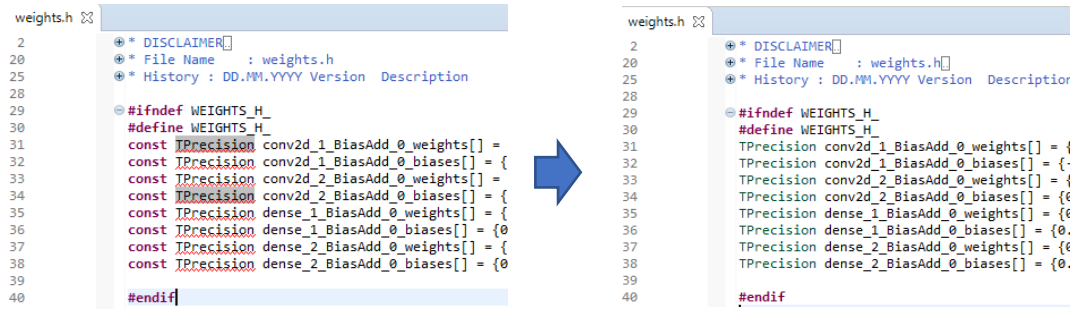
In this demo, the inference is called from

ANPR_RZA2M_*/src/renesas/application/r_bcd_main.c

6.3. NN TAT optimisation (e²studio, gcc)

1. increase inference performance by loading the weights and biases into the local RAM

   Open
   RZA2M_number_plate_detection_TensorFlow_Lite\
   src\neural_networks\Translator\weights.h
   and change the variable definition from “const TPrecision” to “TPrecision”.

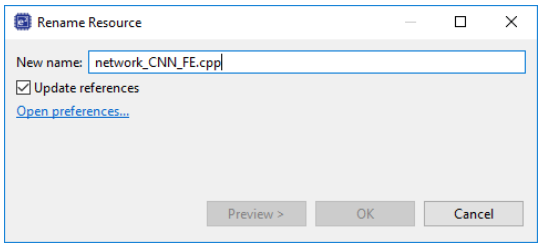


   (This step has been skipped in the demo to be able to implement 4 different CNNs in parallel.)

2. Increase inference performance by adding optimization options for loop vectorization on trees

   i. rename Translator files from .c to .cpp
   src/neural_networks/Translator/dnn_compute_CNN_FE.c
   to src/neural_networks/Translator/dnn_compute_CNN_FE.cpp
   and
   src/neural_networks/Translator/network_CNN_FE.c
   to src/neural_networks/Translator/network_CNN_FE.cpp

   right-click on the file -> Rename (tick “Update references”)



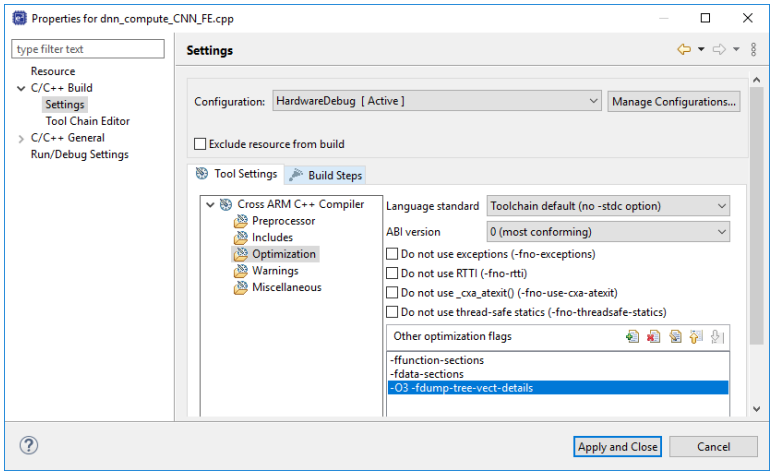
   ii. add Wrapper around dnn_compute.cpp
   Step 1 : change function name in
   src/neural_networks/Translator/dnn_compute_CNN_FE.cpp
   add declaration:
   TPrecision* dnn_compute_CNN_FEPre(TPrecision*, TsInt *);
   modify
   TPrecision* dnn_compute_CNN_FE(TPrecision* conv2d_input,
   TsInt *errorcode)
   to
   TPrecision* dnn_compute_CNN_FEPre(TPrecision* conv2d_input,
   TsInt *errorcode)

Step 2 : prepare a new .cpp file
src/neural_networks/Translator/dnnInit.cpp

content:

```
#include "Typedef.h"
TPrecision* dnn_compute_CNN_FEPre(TPrecision*, TsInt*) ;

extern "C" float* dnn_compute_CNN_FE(float*, TsInt*) ;

float* dnn_compute_CNN_FE(float* inPointer, TsInt* errorPointer) {
  return dnn_compute_CNN_FEPre((TPrecision*)inPointer, (TsInt*)errorPointer) ;
}
```

iii. add compile options to activate loop vectorization on trees
right-click separately on file
src\neural_networks\Translator\dnn_compute_CNN_FE.cpp
and
src\neural_networks\Translator\network_CNN_FE.cpp
to open the “Properties” window.
Select C/C++ Build -> Settings -> Optimization
add “Other optimization flags”:
-O3 -fdump-tree-vect-details



iv. add “__restrict” parameter to all pointer used as function parameter of file
src\neural_networks\Translator\network_CNN_FE.cpp
e.g.:

```
void padding(TPrecision *__restrict__ dData, TPrecision *__restrict__ dPad, TsInt
*__restrict__ iShapes)
…
void relu(TPrecision *__restrict__ dData, TPrecision *__restrict__ dOut, TsInt iShapes )
```

“Clean” and “Build” the project.

7. “TensorFlow Lite for Microcontroller” (TFLite for MCU) Flow

“TensorFlow Lite for RZ/A2M” is based on the TensorFlow Guide „TensorFlow Lite for Microcontrollers “: https://www.tensorflow.org/lite/microcontrollers

As mentioned in the Guide, the TFLite for MCU workflow consists of 5 steps:

1. **Create or obtain a TensorFlow model**
   The model must be small enough to fit on your target device after conversion, and it can only use supported operations. If you want to use operations that are not currently supported, you can provide your own implementations.
2. **Convert the model to a TensorFlow Lite FlatBuffer**
   You will convert your model into the standard TensorFlow Lite format using the TensorFlow Lite converter. You may wish to output a quantized model, since these are smaller in size and more efficient to execute.
3. **Convert the FlatBuffer to a C byte array**
   Models are kept in read-only program memory and provided in the form of a simple C file. Standard tools can be used to convert the FlatBuffer into a C array.
4. **Integrate the TensorFlow Lite for Microcontrollers C++ library**
   Write your microcontroller code to collect data, perform inference using the C++ library, and make use of the results.
5. **Deploy to your device**
   Build and deploy the program to your device.

This application note does not describe Steps1-3 in detail.
As the flow is changing rapidly, it’s recommended to check the latest TensorFlow guides.
For this demo, 3 different FlatBuffer files have been generated, converted to C byte arrays and finally added to the project. Related files and variables follow the following naming style.

a) Standard “float” with 32bit floating point precision
   src/neural_networks/tensorflow_lite/TF_lite_float
   (file and variable names with “float”)
b) “Post-Training Quantization” to 8bit integer format precision
   src/neural_networks/tensorflow_lite/TF_lite_post_training_quantization
   (file and variable names with “PTQ”)
c) “Quantization Aware Training” to 8bit integer format precision
   src/neural_networks/tensorflow_lite/TF_lite_quantization_aware_training
   (file and variable names with “QAT”)

7.1. Step 4: Integrate the TensorFlow Lite for MCUs C++ library

The Tensorflow repository contains all required libraries, several tools and reference projects.

First, prepare the TensorFlow Lite library tree (example as used for the demo):

```
# step 1:       clone tensorflow "tensorflow" repository

git clone https://github.com/tensorflow/tensorflow.git
export branch="r2.6“
cd tensorflow
git checkout ${branch}

# step 2:       generate all projects to get all required downloads
#       (the python script to create the tflm tree may fail to download certain libraries)

make -f tensorflow/lite/micro/tools/make/Makefile TARGET_ARCH=cortex-a9 \
        OPTIMIZED_KERNEL_DIR=cmsis_nn generate_projects

# step 3: prepare tflm tree (with cmsis_nn)

python3 tensorflow/lite/micro/tools/project_generation/create_tflm_tree.py \
        --makefile_options="TARGET_ARCH=cortex-a9 OPTIMIZED_KERNEL_DIR=cmsis_nn" \
        "tensorflow_lite“
```

The whole script has been copied to the e²studio project:

src/tensorflow_lite/prepare_TFLITE.bash

For this demo, the person detection example has been used as reference.
Reference:
prj/person_detection_int8/tensorflow_lite/examples/person_detection/person_detection.ino

ANPR demo:
src/neural_networks/tensorflow_lite/TF_lite_float/tensorflowlite_float_dnn_compute.cc

Directory

tensorflow/lite/micro/tools/make/gen/linux_x86_64/prj/person_detection/tensorflow_lite

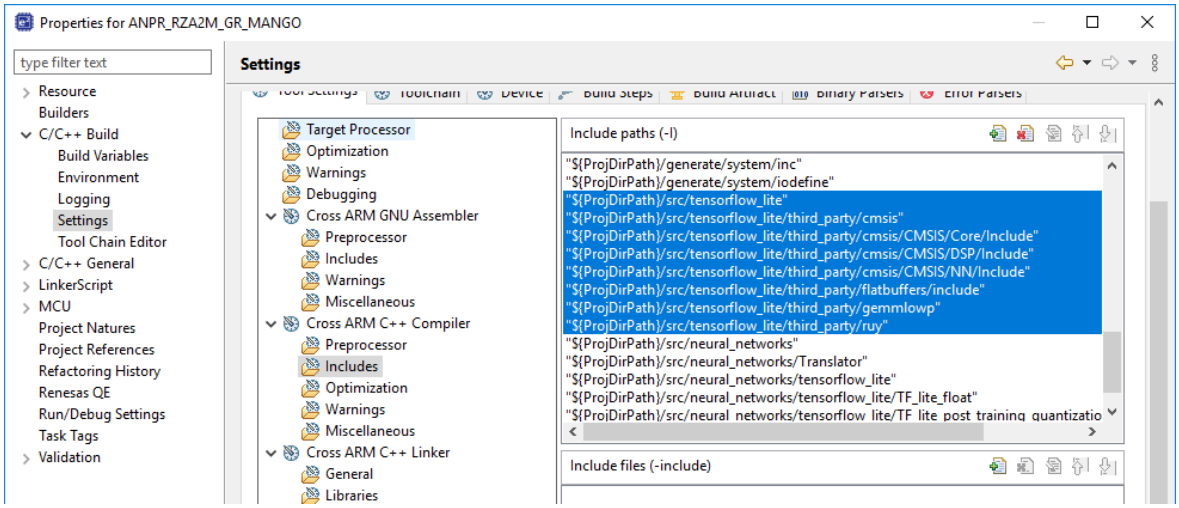
has been copied to e²studio project directory

src/

Additionally, the path

src/tensorflow_lite

and a few third_party library paths need to be added to list of include directories.



The most important files are:

src/neural_networks/tensorflow_lite/TF_lite_*/tensorflowlite_*_dnn_compute.cc:

- setup of the TFLITE interpreter that calls the NN
- the file can be replaced with a more general (more memory consuming) interpreter usage given in tensorflowlite_float_dnn_compute.cc_AllOpsResolver

tensorflowlite_*_dnnInit.cc & tensorflowlite_*_dnnInit.h:

- wrapper to instantiate C++ code within C environment

tensorflowlite_model_settings.cpp & tensorflowlite_model_settings.h:

- defines a variable required by the interpreter (same for all TFLITE NNs)
- defines ‘global’ CNN result variables used in the main routine

r_neural_networks.h:

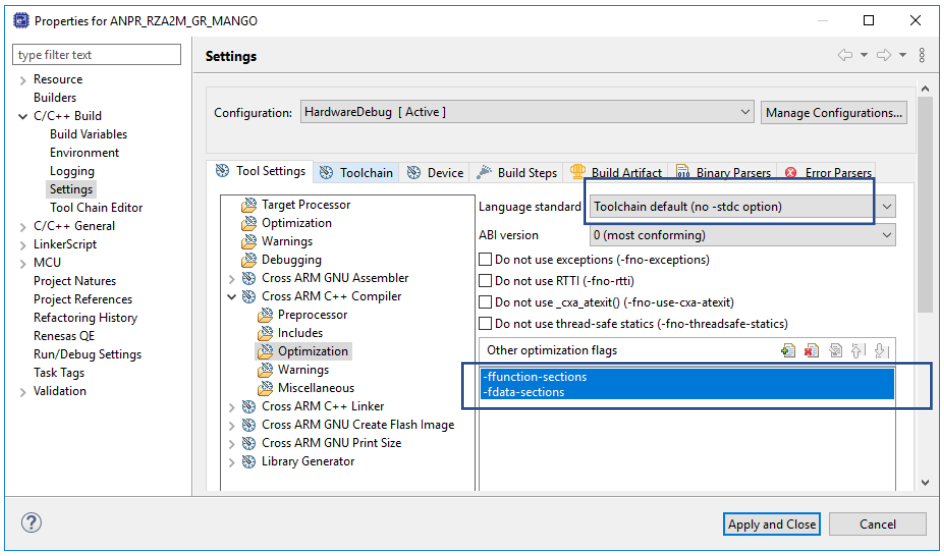
- header file with a few variables used in the main routine (global setting for all NN)

7.2. NN TAT optimization

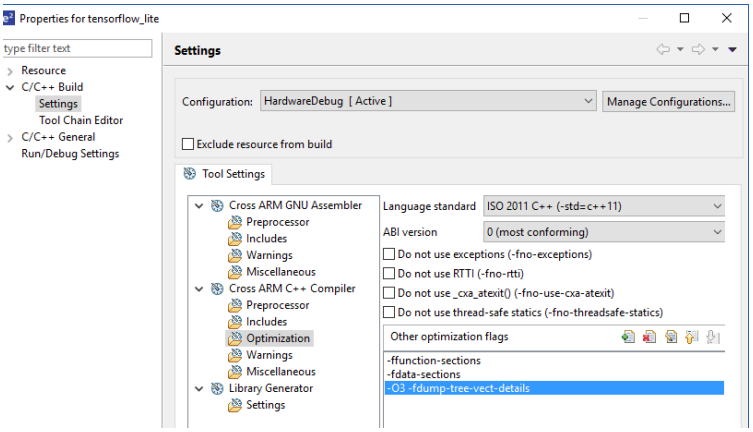
Optimization flags have been added for the whole project.

Language standard: Toolchain default (no -stdc option)

Other optimization flags: -ffunction-sections -fdata-sections



For subdirectory src/tensorflow_lite
change language standard to ISO 2011 C++ (-std=c++11)
and add optimization flag -O3 -fdump-tree-vect-details

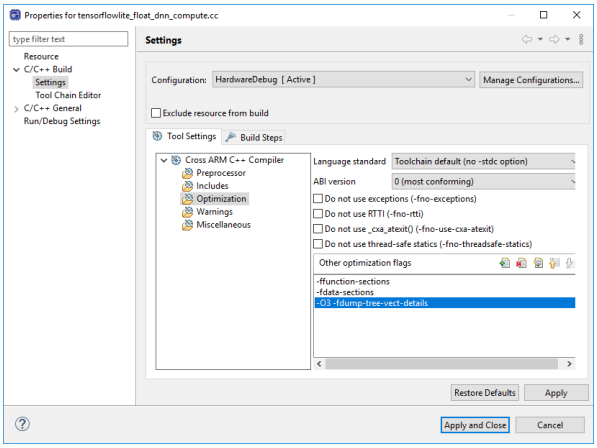


add optimization flag -O3 -fdump-tree-vect-details
for neural network files given in subdirectory
src/neural_networks/tensorflow_lite/

TF_lite_float/tensorflowlite_float_dnn_compute.cc
TF_lite_float/CNN_FE_Schrift_20210705_171657_float.cc

TF_lite_post_training_quantization/tensorflowlite_ptquant_dnn_compute.cc
TF_lite_post_training_quantization/CNN_FE_Schrift_20210705_171657_PTQ.cc

TF_lite_quantization_aware_training/tensorflowlite_quantdt_dnn_compute.cc
TF_lite_quantization_aware_training/CNN_FE_Schrift_20210702_183016_QAT.cc



“Release” configuration:

For performance and memory consumption optimization (without debugging),

add optimization flag -O3 -fdump-tree-vect-details
and linker optimization flag -Wl,--strip-all
to overall project.