life.augmented

UM2753

User manual

SPC5Studio artificial intelligence

Introduction

This document is intended for software and hardware developers who want to understand how to configure and handle the SPC5Studio artificial intelligence component (SPC5-AI component) 2.1.2 for developing a project based on neural networks for the SPC58 automotive PowerPC device family.

1 Requirements

The SPC5-AI component 2.1.2 requires SPC5Studio 6.0 or later for Windows Operating System and it is compatible with the SPC58 automotive PowerPC device family. In particular, the devices that support the SPC5-AI component are listed in Table 1.

Table 1. SPC58 cores

Device	Name
SPC584B	SPC58 4B Line
SPC58xC	SPC58 C Line
SPC58xG	SPC58 G Line
SPC58xE	SPC58 E Line
SPC58xN	SPC58 N Line
SPC58xH	SPC58 H Line

2 Overview

The SPC5-AI component is a SPC5Studio plug-in that can be included in the SPC5Studio projects to analyze, validate and generate an optimized neural network C-library for the SPC58 device family. It is based on a graphic interface that allows to define a list of networks on which the project is based on and the type of processing (analyze, validate or generate) to execute. The SPC5-AI Component graphic interface is shown in Figure 1.

Figure 1. SPC5-AI component graphic interface

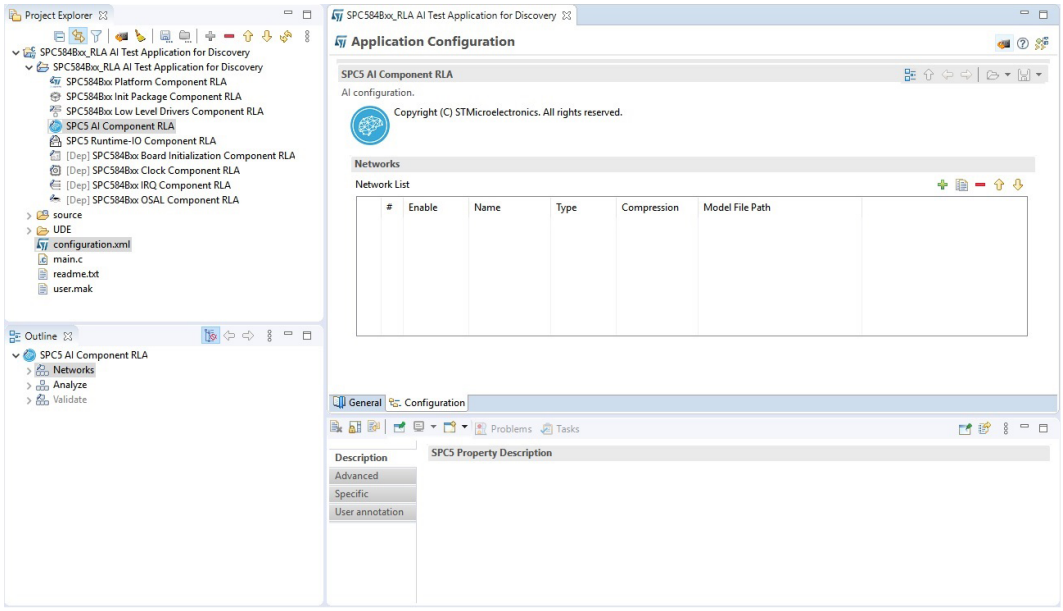
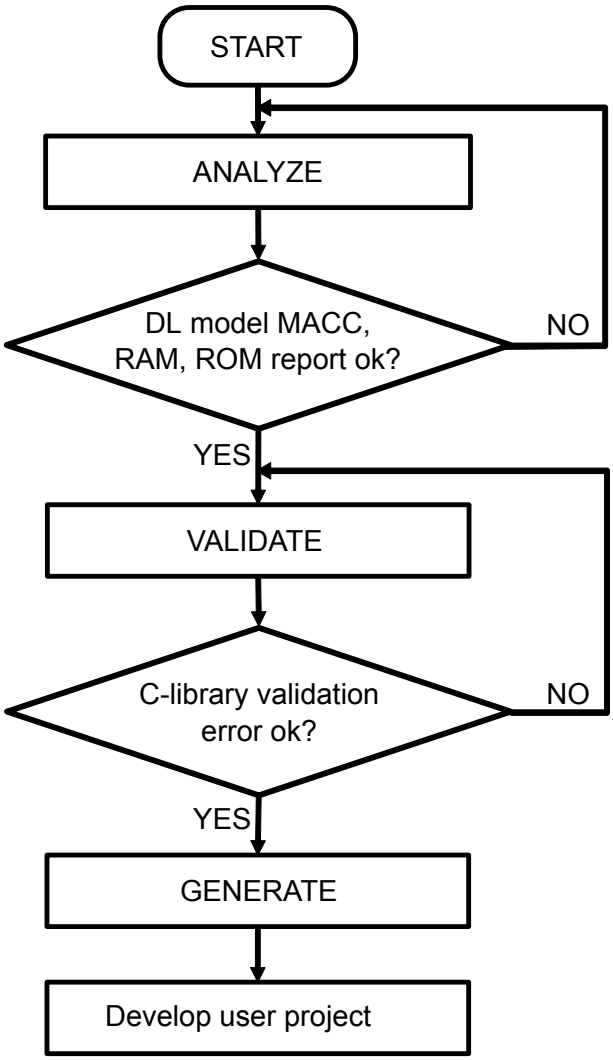


Figure 2 shows a typical workflow for developing a user defined application based on neural networks using the SPC5-AI component.

Figure 2. **SPC5-AI workflow**



3 Network parameters

The network list allows to specify one or more networks on which the SPC5Studio project is based on. Each network defined in the network list can be enabled or disabled. When a network is disabled, it is not processed. In this way the user can temporarily remove one or more networks from the SPC5-AI processing. At least one network in the network list must be enabled any time.

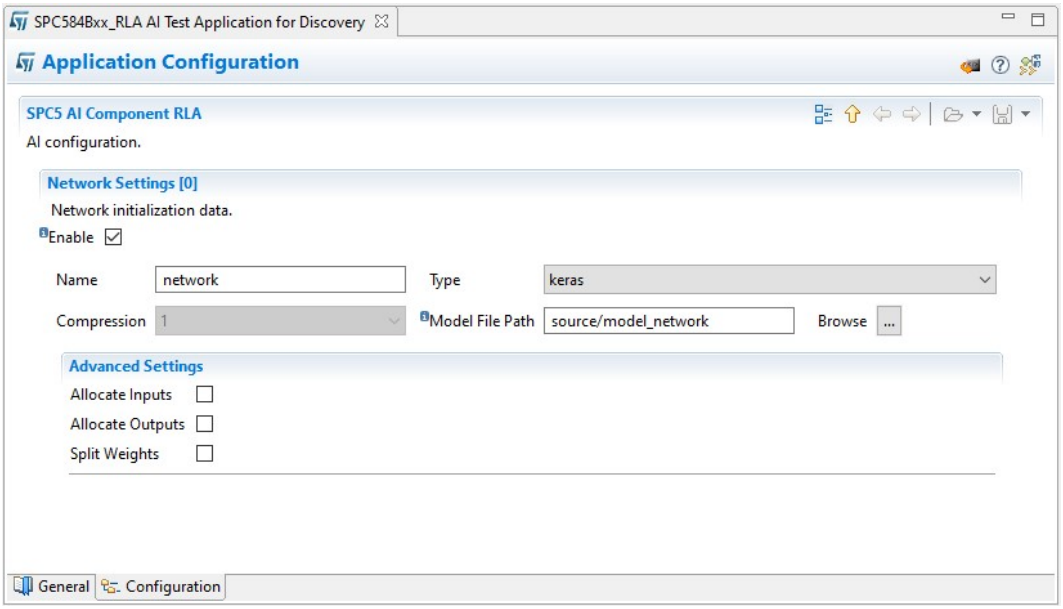
For each new entry in the network list it is possible to specify:

- **Name**: two or more networks with the same name can be defined in the network list, but only one can be enabled. If more networks with the same name are enabled at the same time, an error will occur. Please, note that two names that differ only for the lower case/upper case of one or more characters are considered identical.
- **Type**: it indicates the type of the Deep Learning (DL) framework. The following types are supported in the version SPC5STUDIO.AI v.2.1.2:
  - Keras
  - TensorFlow lite
  - Lasagne
  - Caffe
  - ConvNetJS
  - ONNX

  The list of layers supported is reported into Section 7 Supported deep learning toolboxes and layers.
- **Compression**: it indicates the expected global factor of compression which will be applied to dense layers (if they are present in the neural network topology).
- **Model file path**: it indicates the path in which the model files must be stored. If the model file path is not valid or the folder does not contain any valid model file, the AI processing will be stopped, and an error will be returned.
- **Allocate inputs**: if enabled, this flag indicates that the activations buffer will be also used to handle the input buffers, otherwise they should be allocated separately in the user memory space. Depending on the size of the input data, the activations buffer may be bigger but overall less than the sum of the activation buffer plus the input buffer.
- **Allocate outputs**: if enabled, this flag indicates that the activations buffer will be also used to handle the outputs buffers, otherwise they should be allocated separately in the user memory space.
- **Split weights**: if enabled, this flag indicates that one c-array is generated by weights/bias data tensor instead of having a unique C-array (weights buffer) for the whole.

The Figure 3 shows the network parameters.

Figure 3. **Network parameters**



The Table 2 summarizes the model file extensions for the different network types.

Table 2. **Model file extensions**

DL framework	Type	Model file extensions
Keras	Keras	single file: HDF5 (.h5 or .hdf5) two separate files: HDF5 (.h5 or .hdf5) for the weights and JSON (.json) or YAML (.yml, .yaml) for the model
TensorFlow lite	tflite	single file (.tflite)
Lasagne	lasagne	two separate files: .npz for the weights and Phyton (.py) for the network structure
Caffe	caffe	two separate files: .prototxt and .caffemodel
ConyNetJS	convnetjs	single file: JSON (.json)
ONNX	onnx	single file (.onnx)

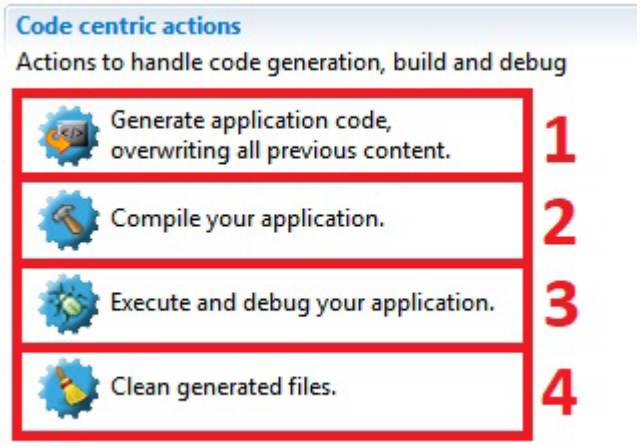
4 SPC-AI command

The SPC5-AI component supports the following commands:

- Analyze
- Generate
- Validate

The generate and validate commands are both automatically run when the SPC5Studio code generation is executed (pushing button 1 in the SPC5Studio Code centric actions panel, see Figure 4). In particular, when the SPC5Studio code generation is executed, if the Validate procedure is enabled in the validate tab of the SPC5-AI component, the validate command is run, otherwise the generate command is run.

Figure 4. SPC5Studio Code centric actions panel



4.1 Analyze

The analyze command allows to check a DL model. For each of the enabled networks in the network list it generates a report that is shown in the SPC5Studio console during the command execution and is also stored in a txt file within the folder ***<prj>/spc5_ai_components/cfg/***. The report allows to check the imported models in terms of supported layers/operators. In order to run the Analyze command, please, select the Analyze tab and click on the Analyze button (see Figure 5).

Figure 5. Analyze tab

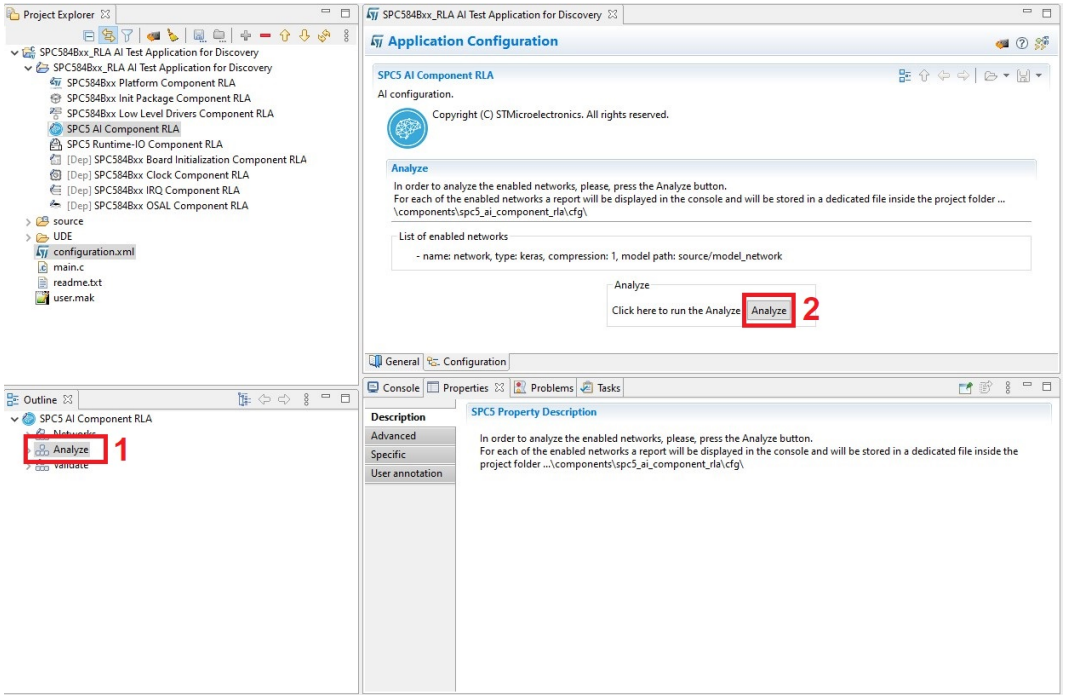


Figure 6 shows an example of analyze report.

Figure 6. **Example of analyze report**

```
model_name         : CNN_LSTM
model_hash         : 240f2f740ba9d67d1fa2fba7b2357b41
input              : input_0 [80 items, 320 B, ai_float, FLOAT32, (20, 1, 4)]
inputs (total)     : 320 B
output             : dense_1 [1 items, 4 B, ai_float, FLOAT32, (1, 1, 1)]
outputs (total)    : 4 B
params #           : 8,961 items (35.00 KiB)
macc               : 95,104
weights (ro)       : 35,972 B (35.13 KiB) (0.36%)
activations (rw)   : 2,304 B (2.25 KiB)
ram (total)        : 2,628 B (2.57 KiB) = 2,304 + 320 + 4

------------------------------------------------------------------------------------------------------------------------
id  layer (type)                    output shape      param #     connected to            macc           rom
------------------------------------------------------------------------------------------------------------------------
0   input_0 (Input)                 (20, 1, 4)
    conv1d_1 (Conv2D)               (20, 1, 32)       544         input_0                 11,552         2,176
------------------------------------------------------------------------------------------------------------------------
1   batch_normalization_1 (ScaleBias) (20, 1, 32)     64          conv1d_1
------------------------------------------------------------------------------------------------------------------------
2   activation_1 (Nonlinearity)     (20, 1, 32)                   batch_normalization_1
------------------------------------------------------------------------------------------------------------------------
3   max_pooling1d_1 (Pool)          (10, 1, 32)                   activation_1
------------------------------------------------------------------------------------------------------------------------
4   lstm_1 (LSTM)                   (1, 1, 32)        8,320       max_pooling1d_1         83,520         33,664
------------------------------------------------------------------------------------------------------------------------
5   dense_1 (Dense)                 (1, 1, 1)         33          lstm_1                  32             132
------------------------------------------------------------------------------------------------------------------------
CNN_LSTM p=8961(35.00 KBytes) macc=95104 rom=35.13 KBytes (0.36%) ram=2.25 KiB io_ram=324 B


Complexity per-layer - macc=95,104 rom=35,972
------------------------------------------------------------------------------------------------------------------------
id      layer (type)                    macc                                    rom
------------------------------------------------------------------------------------------------------------------------
0       conv1d_1 (Conv2D)               |||||                            12.1%  ||                                6.0%
4       lstm_1 (LSTM)                   ||||||||||||||||||||||||||||||||  87.8%  ||||||||||||||||||||||||||||||||  93.6%
5       dense_1 (Dense)                 |                                 0.0%  |                                 0.4%
------------------------------------------------------------------------------------------------------------------------

elapsed time (analyze): 0.70s
```

4.2 Generate

The generate command is used to generate the C-library files for all enabled networks within the network list. Then users can design and develop specific applications based on the API's of these C-libraries. For each of the enabled networks the generate command will create the following files (C-library files):

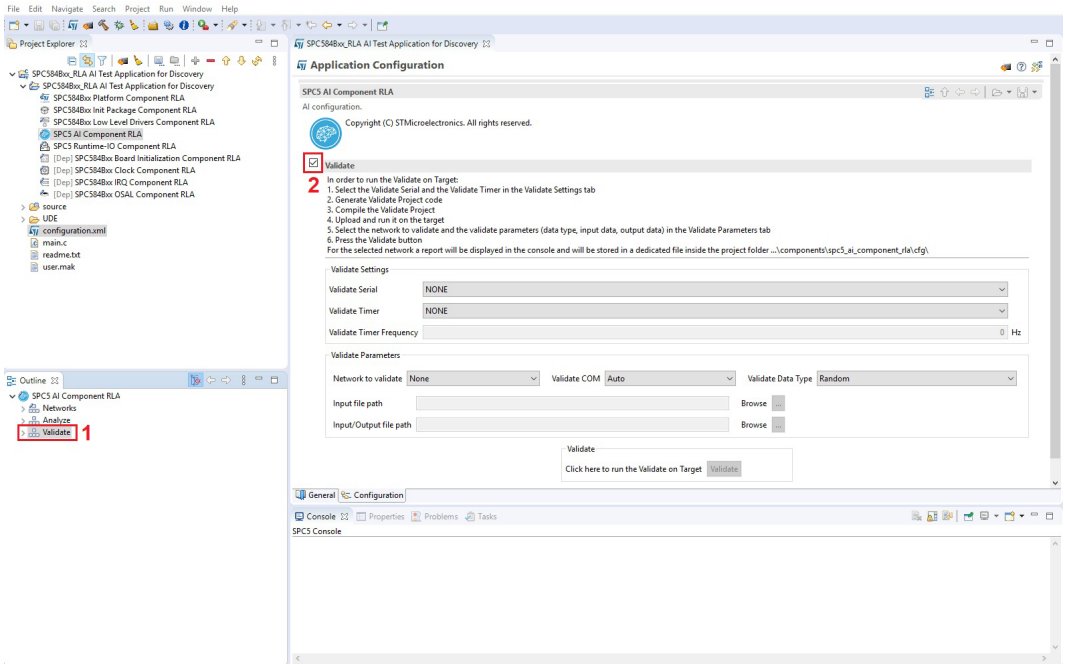
- <network_name>.c
- <network_name>.h
- <network_name>_data.c
- <network_name>_data.h

within the folder ***<prj>/spc5_ai_components/cfg/*** and will generate a report that is shown in the SPC5Studio console during the command execution and is also stored in a txt file within the same folder. If an error occurs during the processing of one of the enabled networks, the generate command will continue to process the other enabled networks. As specified before, the generate command is automatically executed when the SPC5Studio code generation (button 1 of Figure 4) is run and the validate procedure is not enabled in the validate tab of the SPC5-AI component.

4.3 Validate

The validate command allows to import, render and validate the C-libraries related to the enabled networks in the network list. In order to execute the validate command, the validate procedure must be enabled selecting the validate tab and setting the enable flag (see Figure 7).

Figure 7. **Validate tab**

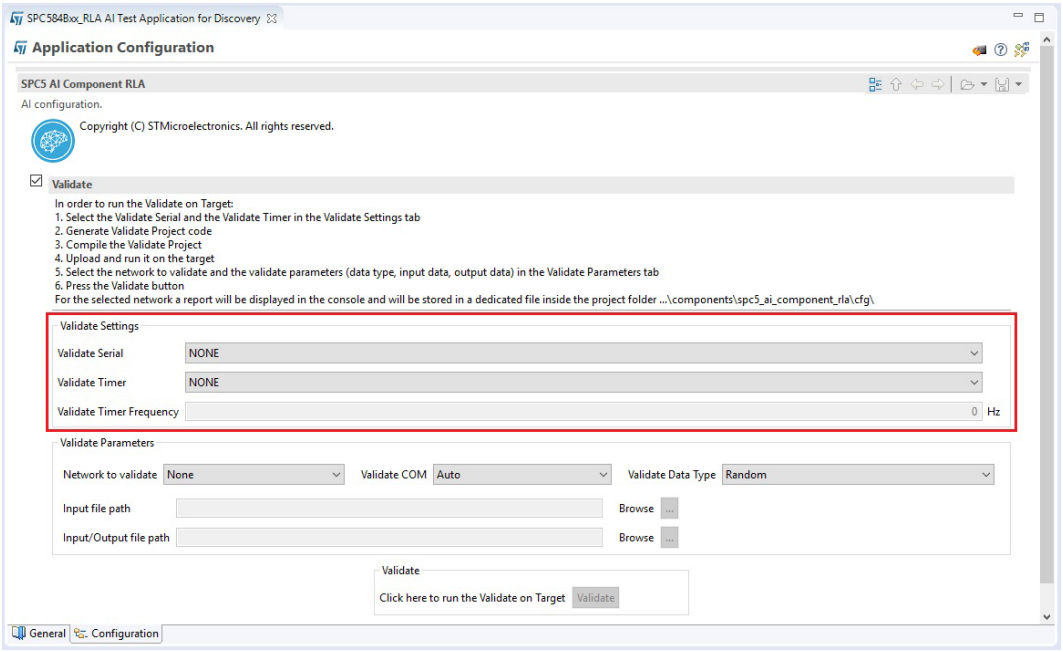


The validate procedure is based on the communication via serial port between the host (that sends the validate data to the target) and the target (that processes the received data). For this reason, within the validate tab of the SPC5-AI component a validate serial must be selected. It is possible to select as validate serial one of the LinFlex available in the selected platform. The LinFlex selected as validate serial in the SPC5-AI component must be enabled in the Low Level Driver component of the SPC5Studio project. If no LinFlex is selected as validate serial, an error will be retuned during the compilation of the validate project.

Moreover, in order to provide a performance estimation at the end of the validate procedure, a Validate Timer should be selected within the validatetabs of the SPC5-AI component. As Validate timer it is possible to select a channel of one of the System Timer Modules (STM) available in the selected platform. The STM channel selected as validateTimer and the related STM module must be enabled and the value of the related prescaler must be configured in the low level driver component of the SPC5Studio project. Within the SPC5-AI components the frequency of the STM channel selected as validate timer is automatically calculated and shown. Since the STM is a timer based on a 32 bit counter, it is recommended to set an STM frequency on the order of 1 MHz, that allows to count with a good resolution (1 µs) for a long period ($(2^{32} – 1)$/ 1 MHz ≈ 72 min) without overflows. If no STM is selected as validate Timer, the validate procedure will return all performance estimation times to zero.

Figure 8 shows the section validate settings of the validate tab of the SPC5-AI component in which it is possible to select the validate Serial and the validate Timer.

Figure 8. **Validate settings**



In order to validate the C-libraries, the following steps are required:

- The C-libraries of all enabled networks must be generated and included in a validate project.
- The validate project must be compiled, downloaded on the target and executed.
- The validate procedure must be run from SPC5Studio.

The first step is exactly the same as a Generate command and it is automatically executed when the validate is enabled in the validate tab of the SPC5-AI component and the SPC5Studio code generation is run (button 1 of Figure 4).

When the generation of C-libraries is completed, the next step is to create a validate project. In order to do this, it is enough to invoke the API ***aiValidateStart()*** within the main of an empty SPC5Studio project. Figure 9 shows the main of the validate project. The C-libraries of all enabled networks will be automatically included in the validate project.

Figure 9. **Validate project**

```
#include "components.h"

/*
 * Application entry point.
 */
int main(void) {

  /* Initialization of all the imported components in the order specified in
     the application wizard. The function is generated automatically.*/
  componentsInit();

  /* Enable Interrupts.*/
  irqIsrEnable();

  /* Run Validate procedure.*/
  aiValidateStart();

  /* Application main loop.*/
  for ( ; ; ) {
  }
}
```

The validate project must be compiled and downloaded on the target. Then it must be run. When the validate project is running on the target, the next step is to select the validate parameters in the section validate parameters of the validate tab (see Figure 10).

Figure 10. **Validate parameters**

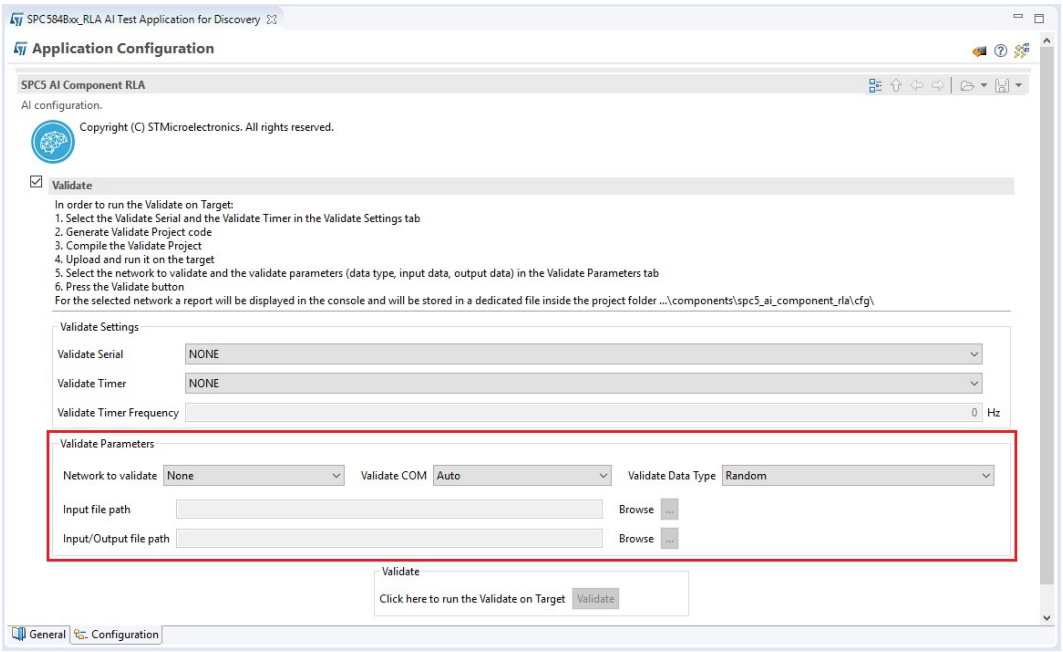
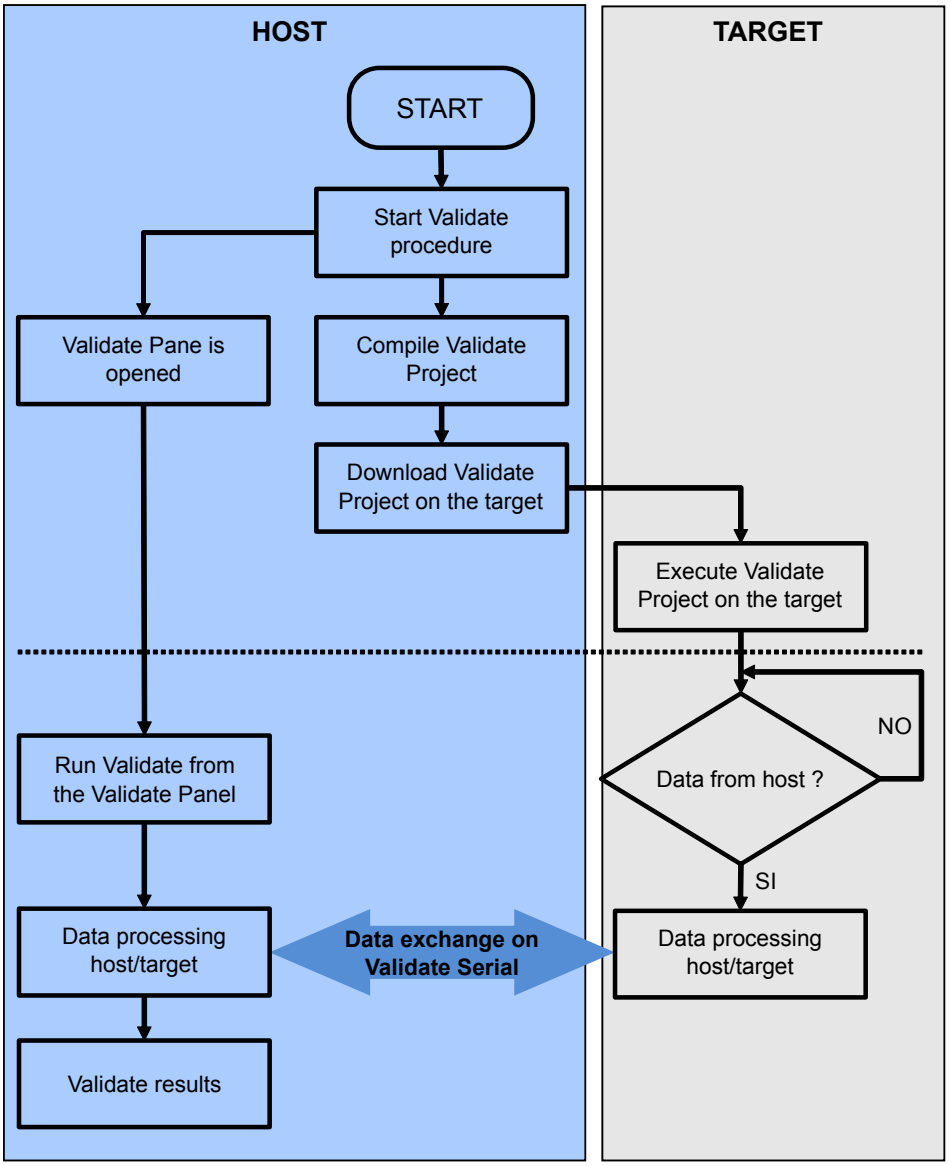


Figure 11 shows the flowchart of the validate procedure.

Figure 11. **Validate flowchart**



The validate parameters are:

- **Network to validate**: it is the name of the network to validate. This parameter can be selected by a list that contains the name of all enabled networks. Only one network at a time can be validated.
- **Validate COM**: it is the host COM where the target is connected. If auto is selected, the validate procedure will autodetect the COM where the target is connected.
- **Validate data type**: it is the type of the data used for the validate procedure. The user can select between:
  - ***Random***: an internal self-generated random data set is used.
  - ***Custom Input Data***: a custom data set is used. In this case the user will have to provide a single file containing the data set. The supported file extensions are:
    - ***.npz***: in this case the file can contain both inputs and expected outputs or the only inputs. If the only inputs are provided, the expected outputs are automatically obtained by the model files using l2r metric.
    - ***.npy*** or ***.csv***: in this case the file contains only the inputs. The expected outputs are automatically obtained by the model files using l2r metric.
  - ***Custom Input/Output Data***: a custom data set is used. In this case the user will have to provide both the custom input data and expected custom output data. The supported file extensions are ***.npz, .npy*** or ***.cvs*** for both custom input and expected output data. Please, note, if an ***.npz*** file containing both input and expected output is provided as custom input data, the custom output data file will be ignored.
    - **Custom Input Data File**: it is the custom input data file (.npz, .npy or .csv).
    - **Custom Output Data File**: it is the custom output data file (.npz, .npy or .csv).

When all the validate parameters are correctly selected, the validate procedure can be started by clicking the validate button in the validate panel. The validate procedure will generate for the selected network a report that is shown in the SPC5Studio console during the command execution and is also stored in a txt file within the folder ***<prj>/spc5_ai_components/cfg/***. Please, before starting the validate procedure, verify that the COM related to the LinFlex selected as validate serial is not busy on another task, otherwise the communication between the host and the target will fail and the Validation procedure will return an error.

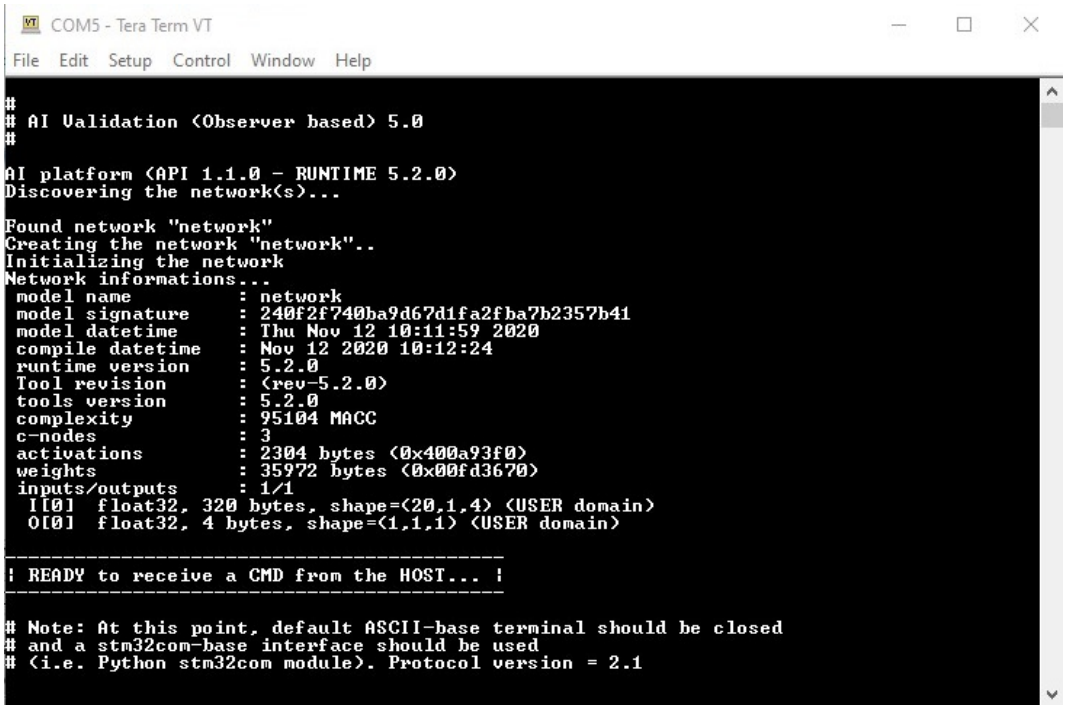
When a validate procedure is completed, a new validate procedure with different validate parameters can be started from the validate panel. But if the user wants to add new networks to the validate project or wants to enable/disable some networks already defined in the network list, it is recommended to restart the validate procedure by running an SPC5Studio code cleaning (button 4 of Figure 4) and then an SPC5Studio code generation (button 1 of Figure 4).

The validate procedure can also be stopped during its execution. Please, note, if the validate procedure is stopped, it could be necessary to disconnect and then reconnect the target to the host before starting a new validate procedure.

If in the validate project the SPC5Studio runtimeIO component is included, when the validate project is executed on the target, just before starting the communication with the host for the validate procedure, the information about the parameters of all networks included in the validate project will be printed on the serial port selected in the RuntimeIO component. So, by opening a console on this serial port it is possible to read all the information about the networks included in the validate project. If the user selects as serial port in the RuntimeIO component the same LinFlex selected as validate serial in the SPC5-AI component, it will be necessary to close the console on which the RuntimeIO component will print the information about the networks before starting the validate procedure from the validate panel in order to avoid that the communication between the host and the target fails.

The Figure 12 shows the information of the networks included in the validate project printed on the serial console when the runtimeIO component is included in the validate project.

Figure 12. **Network information**



5 Embedded Client API

5.1 AI_<NAME>_XXX C-defines

Different C-defines are generated in the `<name>.h` and `<name>_data.h` files. They can be used to retrieve the dimensioning values for C-compile time or dynamic allocation, or for debug purpose.

Table 3. C-compile time or dynamic allocation

C-defines	C-defines
AI_<NAME>_MODEL_NAME	C-string with the C-name of the model
AI_<NAME>_IN/OUT_NUM	indicates the total number of input/output tensors
AI_<NAME>_IN/OUT	C-table ('ai_buffer' type) to describe the input/output tensors (see 'ai_<name>_run()' function)
AI_<NAME>_IN/OUT_SIZE	C-table (integer type) to indicate the number of item by input/output tensors (= H x W x C)
AI_<NAME>_IN/OUT_1_SIZE	indicates the total number of item for the first input/output tensor
AI_<NAME>_IN/OUT_1_SIZE_BYTES	indicates the size in bytes for the first input/output tensor (see 'ai_<name>_run()' function)
AI_<NAME>_DATA_ACTIVATIONS_SIZE	indicates the minimal size in bytes which must provided by a client application layer as working buffer (see 'ai_<name>_init()' function)
AI_<NAME>_DATA__WEIGHTS_SIZE	indicates the size in bytes of the generated weights/bias buffer segment
AI_<NAME>_INPUTS_IN_ACTIVATIONS	indicates that the input buffers can be used from the activations buffer. It is *only* defined if the '--allocate-inputs' option is used.
AI_<NAME>_OUTPUTS_IN_ACTIVATIONS	indicates that the outputs buffers can be used from the activations buffer. It is *only* defined if the '--allocate-outputs' option is used.

5.2 ai_name_create()

```
ai_error ai_<name>_create(ai_handle* network, const ai_buffer* network_config);

ai_handle ai_<name>_destroy(ai_handle network);
```

This **mandatory** function is the *early* function which must be called by the application to create an instance of the neural network. Provided `'ai_handle'` is updated and it refers to a context (opaque object) which should be passed to the other functions.

- The `'network_config'` parameter is a specific network configuration buffer (opaque structure) coded as a `'ai_buffer'`. It is normally generated by the code generator and should be *not modified* by the application. For the current supported STM32 series and model, this object is always empty and `NULL` can be passed but it is preferable to pass `'AI_NETWORK_DATA_CONFIG'` (see `'<name>.h'` file).

When the instance is no more used by the application, `'ai_<name>_destroy()'` function should be called to release the possible allocated resources.

5.3 ai_name_init()

```
ai_bool ai_<name>_init(ai_handle network, const ai_network_params* params);
```

This **mandatory** function is used by the application to initialize the internal run-time data structures and to set the activations buffer and weights buffer.

- `params` parameter is a structure (`ai_network_params` type) which allows to pass the references of the generated weights (`params` field) and the activation/crash memory buffer (`activations` field)
- `network` handle should be a valid handle, see 'ai_<name>_create()' function

```
/* @file: ai_platform.h */

typedef struct ai_network_params_ {

  ai_buffer   params;         /*! info about params buffer(required!) */

  ai_buffer   activations;    /*! info about activations buffer (required!) */

} ai_network_params;
```

- `params` attribute handles the weights/bias memory buffer
- `activations` attribute handles the activations buffer which is used by the inference engine.
- size of associated memory blocks are respectively defined by the following C-defines (see `<name>_data.h` file)
  - `AI_<NAME>_DATA_WEIGHTS_SIZE`
  - `AI_<NAME>_DATA_ACTIVATIONS_SIZE`

`AI_NETWORK_DATA_WEIGHTS()` and `AI_NETWORK_DATA_ACTIVATIONS()` helper macros should be used to populate the requested `params` structure. Note that the `ai_network_data_weights_get()` functions allows to retrieve the base address of the weights buffer (see `'<network>_data.h'` file).

```
AI_ALIGNED(4)

static ai_u8 activations[AI_NETWORK_DATA_ACTIVATIONS_SIZE];

const ai_network_params params = {

  AI_NETWORK_DATA_WEIGHTS(ai_network_data_weights_get()),

  AI_NETWORK_DATA_ACTIVATIONS(activations) };

ai_network_init(network, &params);
```

5.4 ai_name_run()

```
ai_i32 ai_<name>_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) allow to provide the input tensors and to store the predicted output tensors respectively.

- Returned value is the number of the input tensors processed when n_batches >= 1. If <=0 ,'ai_network_get_error()' function should be used to know the error

Note: two separate lists of inputs and outputs 'ai_buffer' can be passed. This permits to support a neural network model with multiple inputs or/and outputs. 'AI_NETWORK_IN_NUM' and respectively 'AI_NETWORK_OUT_NUM' helper macro can be used to know at compile-time the number of inputs and outputs. These values are also returned by the "struct ai_network_report" (see 'ai_<name>_get_info()' function).

5.5 ai_name_get_error()

```
ai_error ai_<name>_get_error(ai_handle network);
```

This function can be used by the client application to retrieve the 1st error reported during the execution of a 'ai_<name>_xxx()' function.

- See ai_platform.h file to have the list of the returned error type ('ai_error_type') and associated code ('ai_error_code').

5.6 ai_name_get_info()

```
ai_bool ai_<name>_get_info(ai_handle network, ai_network_report* report);
```

This function allows to retrieve the run-time data attributes of an instantiated model. Refer to 'ai_platform.h' file to show the detail of the returned 'ai_network_report' C-struct. It should be called after the call of 'ai_<name>_init()'.

6 Compilers

The SPC5-AI component is compatible with the following compilers:

- FreeGCC
- Hightec Free
- Hightec Pro
- Green Hills

In order to optimize the performances, it is recommended:

- in the tab **Build Settings** of the **Platform Component** of the SPC5Studio project:
  - Set **Optimization Level** to -O2
  - Set flag **Use FPU** for FreeGCC and Hightec
  - Set **Common Options** to
    - ***-gdwarf-2 -fomit-frame-pointer -falign-functions=16 -fno-gcse -std=gnu99*** for FreeGCC
    - ***-gdwarf-2 -fomit-frame-pointer -falign-functions=16 -std=gnu99*** for Hightec Free/Pro
    - ***--ee -ansi --no_coverage_analysis -check=nomemory -inline_prologue -std=c99 -G -dual_debug --gnu_asm -gnu99*** for Green Hills
- in the tab **Runtime Settings** of the **Platform component** of the SPC5Studio project:
  - Enable **ICache**/**DCache** if supported.

7 Supported deep learning toolboxes and layers

7.1 Introduction

SPC5-AI component 2.1.2 currently supports the following deep learning toolboxes:

- Keras (refer to *keras.io*)
- TensorFlow Lite (refer to *tensorflow.org/lite*)
- Lasagne (refer to *lasagne.readthedocs.io/en/latest*)
- Caffe (refer to *caffe.berkeleyvision.org*)
- ConvNetJs (refer to *cs.stanford.edu/people/karpathy/convnetjs*)
- ONNX (refer to *onnx.ai*)

For each toolbox only a subset of all the possible layers and layer parameters are supported, depending on the expressive power of the network C API and on the parser for the specific toolbox. Moreover, the layers are supported in floating point exploiting the capabilities of the SPC58 cores.

The configurations supported by the current version of the tool for each toolbox in terms of layers are described below (using the toolbox naming conventions) and the attributes (parameters) supported for each layer. When the same functionality is supported by multiple layers, they are listed together.

7.2 Lasagne

Lasagne, based on the computational library Theano, was one of the first user-friendly deep learning toolboxes. Theano now is on end-of-life support, but Lasagne is supported for historical reasons as several projects are using it. Only the last version for GitHub (0.2-dev) is supported, while the stable 0.1 version from pip is not supported.

7.2.1 Model format

Lasagne doesn't have a file format for the network structure, while the weights can be saved and loaded from numpy files (.npz). For the network structure, the tool expects a specifically crafted Python module building the network.

The module must have the following elements:

- INPUT: input of the network declared as a Theano tensor.
- INPUT_SHAPE: shape of the input tensor; the first dimension is treated as batch size and it is ignored.
- network: function building the network; the single input parameter must match the INPUT variable declared above.

If no element is defined or the weights in the numpy arrays don't match the model, loading fails.

7.2.2 Layer support

The following layers and attributes are supported:

- Conv2DLayer: convolutional layer, only 2D convolutions are supported. The following attributes are supported:
  - pad: only explicit (list of integers) padding supported.
  - filter_size: arbitrary filter kernel sizes, provided they are smaller than the input size
  - stride: arbitrary strides, provided they are smaller than the input size.
  - num_filters: number of output channels.
  - use_bias: don't use bias if set to 'False'.
- MaxPool2DLayer: max pooling layer, only the 2D version is supported. The following attributes are supported:
  - pad: only explicit (list of integers) padding supported.
  - pool_size: arbitrary pool sizes, provided they are smaller than the input size.
  - stride: arbitrary strides, provided they are smaller than the input size.
  - ignore_border: if 'True', ignores partial pooling regions in computing the output.
- AvgPool2DLayer: average pooling layer, same attributes as the max pooling layer.
- GlobalPoolLayer: global max and average pooling layers, supported by setting pool_size and strides to the input size of the layer.
- Dense: dense (fully connected) layer. The following attributes are supported:
  - num_units: number of output features'
  - use_bias: don't use bias if set to 'False'.
- NonlinearityLayer: nonlinear activation layer, decoded also when part of Conv2D and Dense. The following attribute is supported:
  - nonlinearity: type of nonlinear activation; the following functions are supported: identity, linear, rectify, softmax, tanh, sigmoid
- BatchNormLayer: batch normalization layer. The following attribute is supported:
  - axes: input dimensions on which the normalization is performed. Only normalization on the channel axis is supported.
- LocalResponseNormalization2DLayer: local response normalization layer within channels. The following attribute is supported:
  - alpha, k, beta, n: parameters of the normalization equation.
- ReshapeLayer: changes the shape of the input tensor without changing the number of elements. The following attribute is supported:
  - shape: target shape. Infer (-1) values are also supported.
- DimshuffleLayer: dimension permutation layer; the following attribute is supported:
  - pattern: target dimension order given the input dimension order, only for non-batch dimensions.
- InputLayer: optional placeholder for the network's input, dropped during conversion.
- DropoutLayer: training only layer, ignored in the conversion.

7.3 Keras

In Keras the Tensorflow backend with channels-last dimension ordering is supported. Keras 2.0 up to version 2.3.1 is supported, while networks defined in Keras 1.x are not supported.

Model may be loaded from a single file with model and weights (.h5, .hdf5) or from the model configuration and weight in separate files. In the latter case, the weights are loaded from a HDF5 file (.h5, .hdf5) and model configuration is loaded from a text file, either JSON (.json) or YAML (.yml , .yaml).

The following layers and attributes is supported:

- Dense: dense (fully connected) layer. The following attributes are supported:
  - units: number of output features.
  - use_bias: don't use bias if set to 'False'.
- Activation: nonlinear activation layer, decoded also when part of Conv2D, DepthwiseConv2D, SeparableConv2D and Dense. The following attributes is supported:
  - nonlinearity: type of nonlinear activation; all the Keras functions are supported including softmax, elu, selu, softplus, softsign, relu, tanh, sigmoid, hard_sigmoid, exponential and linear. relu6 is supported only as a custom function (see note below)
- Flatten: flattens the non-batch input dimensions to a vector; mapped as a special reshape layer.
- InputLayer: optional placeholder for the network's input, dropped during conversion.
- Reshape: changes the shape of the input tensor without changing the number of elements. The following attributes is supported:
  - shape: target shape. Infer (-1) and keep same (0) values are also supported.
- Permute: dimension permutation layer, only 3D tensors are supported. The following attributes is supported:
  - dims: target dimension order given the input dimension order, only for non-batch dimensions.
- RepeatVector: repeats a vector along a spatial dimension. The following attributes is supported:
  - n: number of repetitions.
- Conv1D, Conv2D: convolutional layers; 1D convolutions are supported by adding a singleton dimension on x. The following attributes are supported:
  - padding: 'SAME' and 'VALID' strategies; arbitrary padding is supported using the ZeroPadding layers (see below). In this case the input size of the network is computed from the input of the ZeroPadding layer.
  - kernel_size: arbitrary filter kernel sizes, provided they are smaller than the input size.
  - stride: arbitrary strides, provided they are smaller than the input size.
  - filters: number of output channels.
  - use_bias: don't use bias if set to 'False'.
  - dilatation_rate: specify the dilation rate to use for dilated convolution.
- SeparableConv1D, SeparableConv2D: concatenation of a depthwise convolution without bias and a 1x1 convolution. The attributes of DepthwiseConv2D and Conv2D are supported.
- DepthwiseConv1D, DepthwiseConv2D: depthwise convolutional layer; supported the same attributes as Convolution2D plus the additional attributes:
  - depth_multiplier: number of channels per feature group; the number of output groups and channels is automatically inferred.
- Conv2DTranspose: transposed convolutional layer; the attributes of Conv2D are supported, plus the additional attribute:
  - output_padding: padding along the output tensor, all values are supported.
- Cropping1D, Cropping2D: cropping layer, all the parameters are supported.
- Upsampling1D, Upsampling2D: image upsampling layer, all the parameters are supported.
- ZeroPadding1D, ZeroPadding2D: padding layer. The following attribute is supported:
  - padding: list of padding values. Symmetric and asymmetric paddings is supported.
- MaxPooling1D, MaxPooling2D: max pooling layer, the 1D versions are supported by adding a singleton dimension on x. The following attributes are supported:
  - padding: 'SAME' and 'VALID' strategies.
  - pool_size: arbitrary pool sizes, provided they are smaller than the input size.
  - strides: arbitrary strides, provided they are smaller than the input size.

- AveragePooling1D, AveragePooling2D: average pooling layers, same attributes as the max pooling layers.
- GlobalMaxPooling1D, GlobalMaxPooling2D, GlobalAveragePooling1D, GlobalAveragePooling2D: global max and average pooling layers, supported by setting pool_size and strides to the input size of the layer.
- LSTM: Long-Short Term Memory layer. The following attributes are supported:
  - units: number of cell / hidden units.
  - activation: hidden-to-output activation, the following functions are supported: linear, relu, tanh, sigmoid, hard_sigmoid.
  - recurrent_activation: hidden-to-hidden activation, the functions from activation are supported.
  - return_sequences: if 'True', return all the hidden states instead of the last one only.
  - use_bias: don't use bias if set to 'False'.
- GRU: Gated Recurrent Unit layer; supports all the LSTM attributes and additionally support:
  - reset_after: GRU only, uses a slightly different formula to align to the CuDNN implementation. Available only in Keras >= 2.1.0.
- Add, Subtract, Multiply, Maximum, Minimum: layers applying an operation element by element; broadcasting of input dimensions is supported.
- Concatenate: concatenate two input layers, the following attribute is supported:
  - axis: concatenation axis.
- ReLU, ThresholdedReLU, LeakyReLU: advanced ReLU layers, all the parameters are supported.
- PReLU: parametric ReLU, the shared_axes attribute is supported only if the axes are a set of contiguous leading axes (e.g. 1, 2).
- ELU, Softmax: parametric nonlinear layers, all the parameters are supported.
- BatchNormalization: batch normalization layer. The following attribute is supported:
  - axis: input dimension on which the normalization is performed. Only normalization on the last axis (channels) is supported.
- Bidirectional: wrapper for bidirectional RNNs, only graphs with a single layer are handled; the following attribute is supported:
  - merge_mode: only the concat, mul and sum modes are supported.
- Dropout, ActivityRegularization, SpatialDropout1D, SpatialDropout2D, GaussianNoise, GaussianDropout, AlphaDropout: training only layers, ignored in the conversion.

Note: the relu6 function is required by the MobileNet architectures, but it is not defined among the standard Keras activations. For Keras up to 2.2.2 you may use the definition in keras_applications.mobilenet; later versions use the supported builtin ReLU layer.

To load or save models using relu6 you need to pass the function as custom object, i.e.:

- model = load_model(model_file, custom_objects={'relu6': relu6})

where the function is imported from the mobilenet package or from Tensorflow (tf.nn.relu6).

7.4 Caffe

In Caffe only the 1.0 official distribution is supported. The following layers are supported:

- Convolution: convolutional layer. The following attributes are supported:
  - pad: arbitrary pad values, provided they are smaller than the filter kernel size.
  - kernel_size: arbitrary filter kernel sizes, provided they are smaller than the input size.
  - stride: arbitrary strides, provided they are smaller than the input size.
  - num_output: number of output channels.
  - group: number of feature groups in the output (for e.g. AlexNet architectures).
  - bias_term: don't use bias if set to 'False'.
- Deconvolution: transposed convolution layer. All the Convolution attributes are supported and no layer-specific attributes are present in Caffe.
- InnerProduct: dense (inner product) layer.

- Pooling: pooling layer. The following attributes are supported:
  - kernel_size: arbitrary pool sizes, provided they are smaller than the input size.
  - stride: arbitrary strides, provided they are smaller than the input size.
  - pad: arbitrary pad values, provided they are smaller than the pool kernel size.
  - pool_function: function used for pooling. MAX (max pooling) and AVE (average pooling) are supported.
  - global_pooling: if 'True', use global pooling; supported by setting kernel_size and stride to the input size of the layer.
  - num_units: number of output features.
  - use_bias: don't use bias if set to 'False'.
- Scale: dimension-wise scaling layer, it can be used as an inference-only batch normalization layer. The following attributes are supported:
  - axis: dimension to scale.
  - use_bias: also use a bias term if set to 'True'.
- Bias: layer adding a dimension-wise bias term; only bias terms applied on the channel dimension are supported. The following attribute is supported:
  - axis: dimension to which add a bias.
- BatchNorm: batch normalization layer, global statistics and moving average parameters are not supported. The following attribute is supported:
  - eps: correction factor to avoid division by zero.
- LRN: local response normalization layer, only normalization within channels is supported. The following attributes are supported:
  - alpha, k, beta, local_size: parameters of the normalization equation.
  - region: type of normalization region. Only 'ACROSS_CHANNELS' is supported.
- ReLU: ReLU nonlinear activation layer. The following attributes is supported:
  - negative_slope: slope in the negative x if used as leaky ReLU.
- PReLU: parametric ReLU nonlinear activation. The following attribute is supported:
  - channel_shared: if the slope parameter is shared across channels.
- ELU: Exponential Linerar Unit layer. The following attribute is supported:
  - alpha: scale of the exponential part.
- Sigmoid, TanH, BNLL: nonlinearities with no parameters.
- Softmax: softmax nonlinear activation layer. The following attributes is supported:
  - axis: input dimension on which softmax is computed. Only normalization on the channel axis is supported.
- Concat: concatenate two or more layers, the following attribute is supported:
  - axis: concatenation axis.
- Eltwise: layer applying an operation element-wise. The following attribute is supported:
  - operation: element-wise operation, support for SUM, MUL, MAX.
- Reduction: layer applying an operation along an axis. The following attributes are supported:
  - axis: axis along which the operation is performed.
  - operation: reduction operation; only SUM is supported.
- Split: sends a layer output to multiple layers.
- Flatten: flattens input dimensions to a vector; mapped as a special Reshape layer. The following attribute is supported:
  - axis, end_axis: range of dimensions to flatten.
- Reshape: changes the shape of the input tensor without changing the number of elements. The following attribute is supported:
  - shape: target shape. Keep same (None) values are also supported.
- Input: optional placeholder for the network's input, dropped during conversion.
- Dropout: training only layer, ignored in the conversion.

7.5 ConvNetJs

ConvNetJs is a simple deep learning toolbox written in Javascript; only the 0.3 version from npm is supported. The following layer is supported:

- conv: 2D convolutional layer. The following attributes are supported:
  - sx, sy: filter size, arbitrary values are supported, provided they are smaller than the input size.
  - stride: arbitrary strides, provided they are smaller than the input size.
  - pad: arbitrary pad values, provided they are smaller than the filter size.
- pool: max pooling layer. The following attributes are supported:
  - sx, sy: pooling size, arbitrary values are supported, provided they are smaller than the input size.
  - stride: arbitrary strides, provided they are smaller than the input size.
  - pad: arbitrary pad values, provided they are smaller than the pool size.
- fc: dense (fully connected) layer; optional bias is supported. The following attribute is also supported:
  - out_depth: number of output features.
- relu, softmax, sigmoid, tanh: nonlinear activation layers; softmax is applied only on the channel dimension.
- Input: optional placeholder for the network's input, dropped during conversion.

7.6 Tensorflow Lite

Tensorflow Lite is the format used to deploy neural network models on mobile platforms. SPC5-AI converts the bytestream (.tflite files) to C code; a number of operators from the supported operator list are handled but quantized models are not supported (refer to *tensorflow.org/lite/guide/ops_compatibility*). The following operators are supported:

- ADD, DIV, FLOOR_DIV, FLOOR_MOD, MINIMUM, MAXIMUM, MUL, POW, SUB: element-wise operators, the optional fused activation is supported.
- AVERAGE_POOL_2D, MAX_POOL_2D: pooling operators, all parameters are supported.
- CONCATENATION: concatenate tensors, axis and fused activation are supported.
- CONV_2D, TRANSPOSE_CONV, DEPTHWISE_CONV_2D: convolutional layers, all parameters are supported.
- ABS, CEIL, COS, ELU, EXP, FLOOR, LEAKY_RELU, LOG, LOGISTIC, NEG, RELU, RELU_N1_TO_1, RELU6, ROUND, RSQRT, SIN, SQRT, TANH: nonlinear functions supported.
- FULLY_CONNECTED: dense layer, fused activation is supported.
- LOCAL_RESPONSE_NORMALIZATION: local response normalization along the channel dimension; all parameters are supported.
- PAD, PADV2: padding operator, only zero padding on the spatial dimensions is supported.
- PRELU: parametric ReLU, axis sharing is supported only on the leading dimensions.
- MEAN: compute the mean along one or more axes; only reduction along spatial dimensions is supported and mapped as global average pooling. keep_dims (leave dimension 1 on the reduce axes) is supported.
- QUANTIZE, DEQUANTIZE: format conversion layers, integer to integer (8 bits only) and integer to float conversions are supported.
- REDUCE_MAX: compute the maximum along one or more axes; reduction along spatial dimensions is mapped as global max pooling. keep_dims is also supported.
- REDUCE_MIN, REDUCE_PROD, SUM: axis reduction operators using the minimum, product and sum respectively. keep_dims is also supported.
- RESHAPE, SQUEEZE: tensor reshape operators, all parameters are supported.
- RESIZE_NEAREST_NEIGHBOR, RESIZE_BILINEAR: interpolation operators, only upsampling along spatial dimensions is supported. The align_corner attribute is supported.
- SLICE: dimension slicing, all parameters are supported.
- LOG_SOFTMAX, SOFTMAX: softmax nonlinearity, the beta parameter is not supported.
- SPLIT: tensor splitting operator, only the case where num_splits is 1 is supported.
- STRIDED_SLICE: dimension slicing with strides and optional squeeze; only unit strides are supported and shrink_axis_mask is not handled.
- TRANSPOSE: traspose a tensors; permutations involving the batch dimension are not supported.

7.7 ONNX

In ONNX a subset of operators from Opset 7, 8, 9 and 10 of ONNX 1.6 is supported. Model may be loaded from a single file with model and weights (.onnx). The following layers and attributes are supported:

- Abs, Acos, Acosh, Asin, Asinh, Atan, Atanh, Ceil, Clip, Cos, Cosh, Erf, Exp, Floor, Log, Neg, Reciprocal, Relu, Round, Rsqrt, Sigmoid, Sign, Sin, Sinh, Softplus, Softsign, Sqrt, Tan, Tanh: nonlinear operators supported.
- Add: element-wise binary addition, supporting multi-directional broadcasting.
- AveragePool: average pooling layer; only 1D and 2D pooling are supported. The following attributes are supported:
  - auto_pad: 'NOTSET' (default),'SAME_UPPER', 'SAME-LOWER' and 'VALID' strategies.
  - ceil_mode: whether to use ceil or floor (default) to compute the output shape.
  - count_include_pad: whether to include pad pixels when calculating values at the edges; the default is 0 (false).
  - kernel_shape: the size of the pooling kernel along each axis.
  - pads: padding at the beginning and at the end of each spatial axis.
  - strides: arbitrary strides, provided they are smaller than the input size.
- BatchNormalization: batch normalization layer. Only one output (Y) is supported. The following attribute is supported:
  - epsilon: the epsilon value to use to avoid division by zero.
- Concat: concatenate a list of tensors into a single tensor. Concatenation along the batch dimension is not supported. The following attribute is supported:
  - axis: concatenation axis. The accepted range is [1, r-1] where r is rank(inputs).
- Constant: a constant tensor. The following attribute is supported:
  - value: the constant value for the elements of the output tensor.
- Conv: convolutional layers; 1D and 2D convolutions are supported. The following attributes are supported:
  - auto_pad: 'NOTSET' (default),'SAME_UPPER', 'SAME-LOWER' and 'VALID' strategies.
  - dilations: dilation value along each spatial axis of the filter.
  - group: number of feature groups in the output channels.
  - kernel_shape: arbitrary filter kernel sizes, provided that they are smaller than the input size. If not present, it is inferred from the weights.
  - pads: padding at the beginning and the end of each spatial axis.
  - strides: arbitrary strides along each axis, provided that they are smaller than the input size.
- ConvTranspose: transposed convolutional layers; 1D and 2D convolutions are supported. The following attributes are supported:
  - auto_pad: 'NOTSET' (default),'SAME_UPPER', 'SAME-LOWER' and 'VALID' strategies.
  - dilations: dilation value along each spatial axis of the filter.
  - group: number of feature groups in the output channels.
  - kernel_shape: arbitrary filter kernel sizes, provided that they are smaller than the input size. If not present, it is inferred from the weights.
  - output_padding: additional padding on the bottom / left of the output.
  - output_shape: the shape of the output can be explicitly set; if output_padding is set, the pads amount will be inferred.
  - pads: padding at the beginning and the end of each spatial axis.
  - strides: arbitrary strides along each axis, provided that they are smaller than the input size.
- Div: float division operator.
- Elu: exponential linear unit operator. The following attribute is supported:
  - alpha: coefficient of Elu.
- Flatten: flattens the non-batch input dimensions to a vector; mapped as a special Reshape layer. The following attribute is supported:
  - axis: up to which input dimensions (exclusive) the input should be flattened; axis=0 is not supported.

- Gemm: general Matrix multiplication: Y = alpha * A' * B' + beta * C, The following attributes are supported:
  - alpha: scalar multiplier for the product of the input tensors: A * B.
  - beta: scalar multiplier for input tensor C.
  - transA: whether A should be transposed.
  - transB: whether B should be transposed.
- Hardmax: hardmax non-linearity. It is supported only for 1D tensors and only on the channel dimension.
- HardSigmoid: hardsigmoid non-linearity. Supported only for 1D tensors, on the channel dimension. Supported only with the default parameters.
- GlobalAveragePool, GlobalMaxPool: global max and average pooling layers, supported by setting pool_size and strides to the input size of the layer.
- InstanceNormalization: instance normalization layer. The following attribute is supported:
  - epsilon: the epsilon value to use to avoid division by zero.
- LeakyReLU: advanced ReLU layers. The following attribute is supported:
  - alpha: leakage coefficient.
- LogSoftmax: log-softmax non-linearity. It is supported only for 1D tensors and only on the channel dimension.
- LpNormalization: Lp-normalization operator. The following attributes are supported:
  - axis: axis on which to apply normalization.
  - p: order of the normalization (1 and 2 supported).
- LRN: local response normalization layer within channels. The following attributes are supported:
  - alpha, beta, bias: parameters of the normalization equation.
  - size: the number of channels to sum over.
- MatMul: Matrix product.
- MaxPool: max pooling layer. Only 1D and 2D pooling are supported. The following attributes are supported:
  - auto_pad: 'NOTSET' (default),'SAME_UPPER', 'SAME-LOWER' and 'VALID' strategies.
  - ceil_mode: whether to use ceil or floor (default) to compute the output shape.
  - kernel_shape: the size of the kernel along each axis.
  - pads: padding at the beginning and the end of each spatial axis.
  - strides: arbitrary strides, provided they are smaller than the input size.
- Max, Min, Sub, Sum: max, min, sub and sum element-wise operators; the single input case is not supported.
- Mul, Pow: mul, pow element-wise operators.
- Pad: pads the input tensor with the number of start and end pad values for axis. The following attributes are supported:
  - mode: constant (default), reflect, edge.
  - pads: padding at the beginning and the end of each spatial axis. Only positive values are supported.
  - value: scalar float value indicating the padding value to use (default 0.0f).
- PRelu: PRelu non-linearity. Shared axes in PReLU are supported only for the leading dimensions.
- ReduceMax, ReduceMin, ReduceMean, ReduceProd, ReduceSum: axis reduction operators using the maximum, minimum, mean, product and sum operations respectively. The following attributes are supported:
  - axes: axis indices along which the tensor will be reduced. Reduction is not supported on the batch dimension.
  - keep_dims: whether to keep the reduced dimension or not; the default is 1 (true).
- Reshape: reshape the input tensor.

- Resize: resize the input tensor. The following attributes are supported:
  - coordinate_transformation_mode: describes how to transform the coordinate in the resized tensor to the coordinate in the original tensor
  - cubic_coeff_a: the coefficient 'a' used in cubic interpolation
  - exclude_outside
  - extrapolation_value
  - mode: two interpolation modes are supported: nearest (default) and linear.
  - nearest_mode: four modes: round_prefer_floor (default, as known as round half down), round_prefer_ceil (as known as round half up), floor, ceil.
- Selu: scaled ELU operator. The following attributes are supported:
  - alpha: leakage coefficient.
  - gamma: scale coefficient.
- Slice: returns a slice of the input tensor along multiple axes. The following attributes are supported:
  - ends: end indices.
  - starts: start indices.
  - axes: axes on which starts and ends apply; only positive values are supported.
- Squeeze: removes singleton dimensions from the input tensor. The following attribute is supported:
  - axes: indices of the dimensions to be removed; if empty, all the singleton dimensions are removed.
- Softmax: softmax non-linearity. It is supported only for 1D tensors and only on the channel dimension.
- Tile: constructs a tensor by tiling the input tensor; tiling on the batch dimension is not supported.
- ThresholdedRelu: advanced Relu layer. The following attribute is supported:
  - alpha: threshold value.
- Transpose: changes the order of the axes in the input tensor; transposing the batch dimension is not supported. The following attribute is supported:
  - perm: list of integer mapping the output axes to the input axes.
- Unsqueeze: Insert single-dimensional entries in the input tensor shape. The following attribute is supported:
  - axes: list of integers indicating the dimensions to be inserted.
- Upsample: Upsample the input tensor. Only the Upsample-7 version is supported. The following attributes are supported:
  - mode: nearest (default) and bilinear are supported.
  - scales: list of scales along each dimension; it takes values greater than or equal to 1.

7.8 Tool versions

The **SPC-AI component 2.1.2** is based on **X-CUBE-AI 5.2.0**, that is based on the following tools:

- **Python** version 3.5.7
- **Numpy** version 1.17.2
- **TF** version 2.3.0
- **TF** Keras version 2.4.0
- **Caffe** version 1.0.0
- **Lasagne** version 0.2.dev1
- **ONNX** version 1.6.0
- **ONNX RT** version 1.1.2

8 Key metrics

8.1 Computational complexity: MACC and cycles/MACC

When a model is analyzed, a global logical computational complexity is reported. MACC indicates the number of multiply-and-accumulate operations which are requested to perform an inference. Reported value is computed independently of the floating-point data format or the underlying C-implementation or/and possible optimization.

Applications allow to report the average number of clocks requested for the whole C-model allowing to compute the number of requested cycles by MACC. This indicator highlights the global efficiency of the underlying C-implementation (including the HW platform aspects).

$$cycle/MACC = \frac{averagenumberofclockcyclesbyinterface}{MACC} \quad (1)$$

8.2 Memory occupancy

When a model is rendered, two memory-related metrics ('weights (ro)' or 'ROM' and 'activations (rw)' or 'RAM') are reported to know the size of the memory which will be requested to integrate the generated C-files. Dynamically, the application allows to know the stack size and the heap sizes used by the C-model.

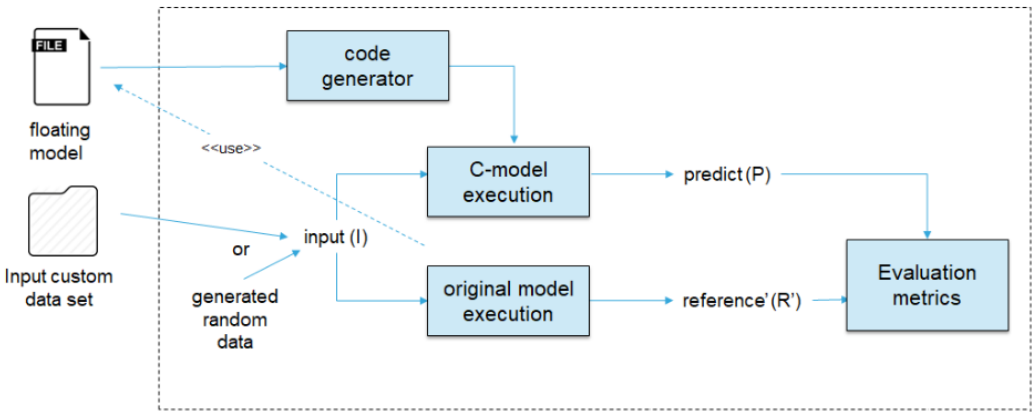
8.3 L2 relative error (l2r)

L2r is the scalar value of the relative 2-norm or Euclidean distance between the generated values of the original model (R') and the C-model (P). This metric is only used with the predicted outputs (P and R') of the C-model and associated floating-point model.

$$l2r = \frac{\| R'j - Pj \|}{\| Pj \|} = \frac{\sqrt{\Sigma_{j=0}^{N} (R'j - Pj)^2}}{\Sigma_{j=0}^{N} (Pj)^2} \quad (2)$$

The Figure 13 shows the data flow for the computation of the metrics.

Figure 13. Computing data flow of the metrics



Revision history

Table 4. **Document revision history**

Date	Version	Changes
22-Jul-2020	1	Initial release.
15-Sep-2020	2	Confidentiality level changed from restricted to public. No content change.
22-Oct-2020	3	Updated: • Updated: • Table 2. Model file extensions; • Figure 6. Example of Analyze report; • Section 4.3 Validate; • Section 6.3 Keras; • Section 6.6 Tensorflow Lite; • Section 6.8 Tool versions. Added Section 6.7 ONNX. Minor text changes.
13-Jan-2021	4	Updated Section 3 Network parameters, Figure 5. Analyze tab, Figure 6. Example of Analyze report, Figure 12. Network information and Section 6.8 Tool versions. Minor text changes.
24-Sep-2021	5	Added section Section 5 .

Contents

List of tables

List of figures