DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Département
Architecture, Conception et Logiciels Embarqués

Exploration of Neural Networks & Benchmarking of Computing Solutions with the N2D2 Platform

Olivier Bichler, David Briand, Benjamin Bertelone

Wednesday 1st March, 2017

Laboratoire d'Intégration des Systèmes et des Technologies

Laboratoire d'Electronique et de Technologie de l'Information

Commissariat à l'Energie Atomique et aux Energies Alternatives
Institut Carnot CEA LIST
Centre de Saclay | Nano-Innov Bât 862 | PC 172
91191 Gif sur Yvette Cedex
Tel. : +33 (0)1.69.08.49.67 | Fax : +33(0)1.69.08.83.95
thierry.collette@cea.fr

Direction de la Recherche Technologique
Département Architecture Conception et Logiciels Embarqués

Contents

1 Presentation

The N2D2 platform is a comprehensive solution for fast and accurate Deep Neural Network (DNN) simulation and full and automated DNN-based applications building. The platform integrates database construction, data pre-processing, network building, benchmarking and hardware export to various targets. It is particularly useful for DNN design and exploration, allowing simple and fast prototyping of DNN with different topologies. It is possible to define and learn multiple network topology variations and compare the performances (in terms of recognition rate and computationnal cost) automatically. Export targets include CPU, DSP and GPU with OpenMP, OpenCL, Cuda and CuDNN programming models as well as custom hardware IP code generation with High-Level Synthesis for FPGA and dedicated configurable DNN accelerator IP[1].

In the following, the first section describes the database handling capabilities of the tool, which can automatically generate learning, validation and testing data sets from any hand made database (for example from simple files directories). The second section briefly describes the data pre-processing capabilites built-in the tool, which does not require any external pre-processing step and can handle many data transformation, normalization and augmentation (for example using elastic distortion to improve the learning). The third section show an example of DNN building using a simple INI text configuration file. The fourth section show some examples of metrics obtained after the learning and testing to evaluate the performances of the learned DNN. Next, the fifth section introduces the DNN hardware export capabilities of the toolflow, which can automatically generate ready to use code for various targets such as embedded GPUs or full custom dedicated FPGA IP. Finally, we conclude by summarising the main features of the tool.

1.1 Database handling

The tool integrates everything needed to handle custom or hand made databases:

- Genericity: load image and sound, 1D, 2D or 3D data;
- Associate a label for each data point (useful for scene labeling for example) or a single label to each data file (one object/class per image for example), 1D or 2D labels;
- Advanced Region of Interest (ROI) handling:
  Support arbitrary ROI shapes (circular, rectangular, polygonal or pixelwise defined);
  Convert ROIs to data point (pixelwise) labels;
  Extract one or multiple ROIs from an initial dataset to create as many corresponding additional data to feed the DNN;
- Native support of file directory-based databases, where each sub-directory represents a different label. Most used image file formats are supported (JPEG, PNG, PGM...);
- Possibility to add custom datafile format in the tool without any change in the code base;
- Automatic random partitionning of the database into learning, validation and testing sets.

1.2 Data pre-processing

Data pre-processing, such as image rescaling, normalization, filtering... is directly integrated into the toolflow, with no need for external tool or pre-processing. Each pre-processing step is called a *transformation*.

The full sequence of transformations can be specified easily in a INI text configuration file. For example:

```
; First step: convert the image to grayscale
[env.Transformation-1]
Type=ChannelExtractionTransformation
CSChannel=Gray
```

[1]Ongoing work

```
; Second step: rescale the image to a 29x29 size
[env.Transformation-2]
Type=RescaleTransformation
Width=29
Height=29

; Third step: apply histogram equalization to the image
[env.Transformation-3]
Type=EqualizeTransformation

; Fourth step (only during learning): apply random elastic distortions to the images to extent the
     learning set
[env.OnTheFlyTransformation]
Type=DistortionTransformation
ApplyTo=LearnOnly
ElasticGaussianSize=21
ElasticSigma=6.0
ElasticScaling=20.0
Scaling=15.0
Rotation=15.0
```

Example of pre-processing transformations built-in in the tool are:

- Image color space change and color channel extraction;
- Elastic distortion;
- Histogram equalization (including CLAHE);
- Convolutional filtering of the image with custom or pre-defined kernels (Gaussian, Gabor...);
- (Random) image flipping;
- (Random) extraction of fixed-size slices in a given label (for multi-label images)
- Normalization;
- Rescaling, padding/cropping, triming;
- Image data range clipping;
- (Random) extraction of fixed-size slices.

1.3 Deep network building

The building of a deep network is straightforward and can be done withing the same INI configuration file. Several layer types are available: convolutional, pooling, fully connected, Radial-basis function (RBF) and softmax. The tool is highly modular and new layer types can be added without any change in the code base. Parameters of each layer type are modifiable, for example for the convolutional layer, one can specify the size of the convolution kernels, the stride, the number of kernels per input map and the learning parameters (learning rate, initial weights value...). For the learning, the data dynamic can be chosen between 16 bits (with NVIDIA® CuDNN[2]), 32 bit and 64 bit floating point numbers.

The following example, which will serve as the use case for the rest of this presentation, shows how to build a DNN with 5 layers: one convolution layer, followed by one MAX pooling layer, followed by two fully connected layers and a softmax output layer.

```
; Specify the input data format
[env]
SizeX=24
SizeY=24
BatchSize=12

; First layer: convolutional with 3x3 kernels
[conv1]
Input=env
Type=Conv
```

[2] On future GPUs

```
KernelWidth=3
KernelHeight=3
NbChannels=32
Stride=1

; Second layer: MAX pooling with pooling area 2x2
[pool1]
Input=conv1
Type=Pool
Pooling=Max
PoolWidth=2
PoolHeight=2
NbChannels=32
Stride=2
Mapping.Size=1 ; one to one connection between convolution output maps and pooling input maps

; Third layer: fully connected layer with 60 neurons
[fc1]
Input=pool1
Type=Fc
NbOutputs=60

; Fourth layer: fully connected with 10 neurons
[fc2]
Input=fc1
Type=Fc
NbOutputs=10

; Final layer: softmax
[softmax]
Input=fc2
Type=Softmax
NbOutputs=10
WithLoss=1

[softmax.Target]
TargetValue=1.0
DefaultValue=0.0
```

The resulting DNN is shown in figure 1.

The learning is accelerated in GPU using the NVIDIA® CuDNN framework, integrated into the toolflow. Using GPU acceleration, learning times can be reduced typically by two orders of magnitude, enabling the learning of large databases within tens of minutes to a few hours instead of several days or weeks for non-GPU accelerated learning.

1.4 Performances evaluation

The software automatically outputs all the information needed for the network applicative performances analysis, such as the recognition rate and the validation score during the learning; the confusion matrix during learning, validation and test; the memory and computation requirements of the network; the output maps activity for each layer, and so on, as shown in figure 2.

1.5 Hardware exports

Once the learned DNN recognition rate performances are satisfying, an optimized version of the network can be automatically exported for various embedded targets. An automated network computation performances benchmarking can also be performed among different targets.

The following targets are currently supported by the toolflow:

- Plain C code (no dynamic memory allocation, no floating point processing);

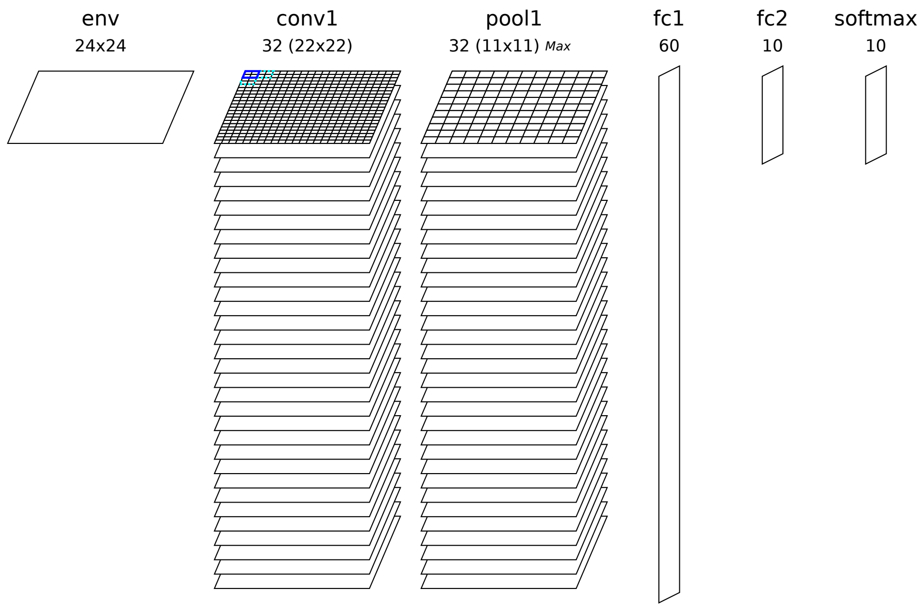
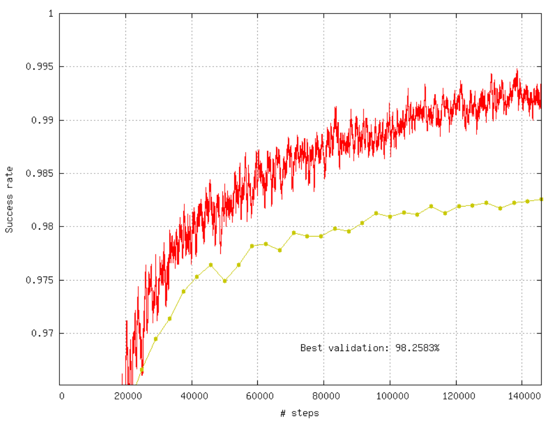
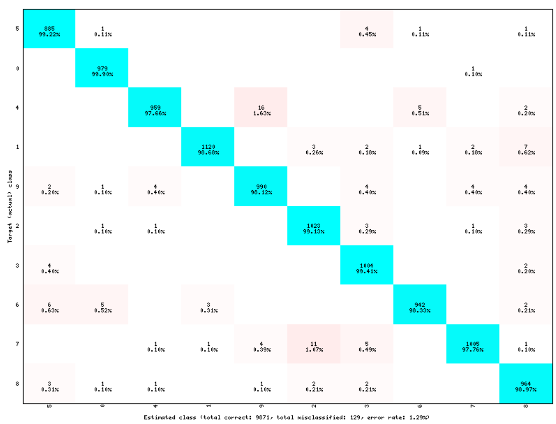


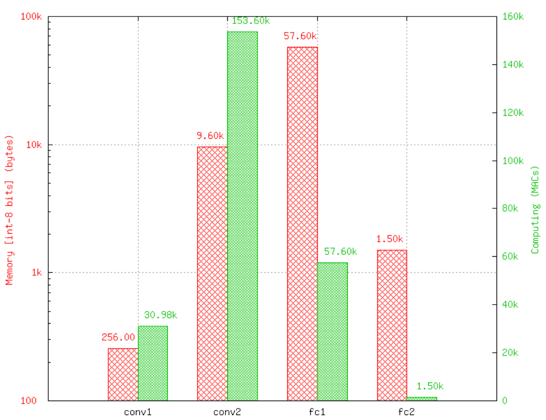
Figure 1: Automatically generated and ready to learn DNN from the INI configuration file example.



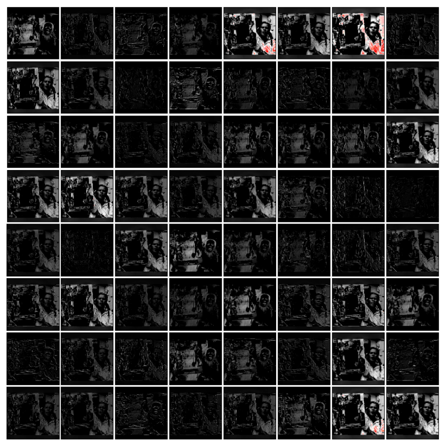
Recognition rate and validation score



Confusion matrix



Memory and computation requirements



Output maps activity

Figure 2: Example of information automatically generated by the software during and after learning.

- C code accelerated with OpenMP;
- C code tailored for High-Level Synthesis (HLS) with Xilinx® Vivado® HLS;
  Direct synthesis to FPGA, with timing and utilization after routing;

Possibility to constrain the maximum number of clock cycles desired to compute the whole network;

FPGA utilization vs number of clock cycle trade-off analysis;

- OpenCL code optimized for either CPU/DSP or GPU;
- Cuda kernels and CuDNN code optimized for NVIDIA® GPUs.

Different automated optimizations are embedded in the exports:

- DNN weights and signal data precision reduction (down to 8 bit integers or less for custom FPGA IPs);
- Non-linear network activation functions approximations;
- Different weights discretization methods.

The exports are generated automatically and come with a Makefile and a working testbench, including the pre-processed testing dataset. Once generated, the testbench is ready to be compiled and executed on the target platform. The applicative performance (recognition rate) as well as the computing time per input data can then be directly mesured by the testbench.

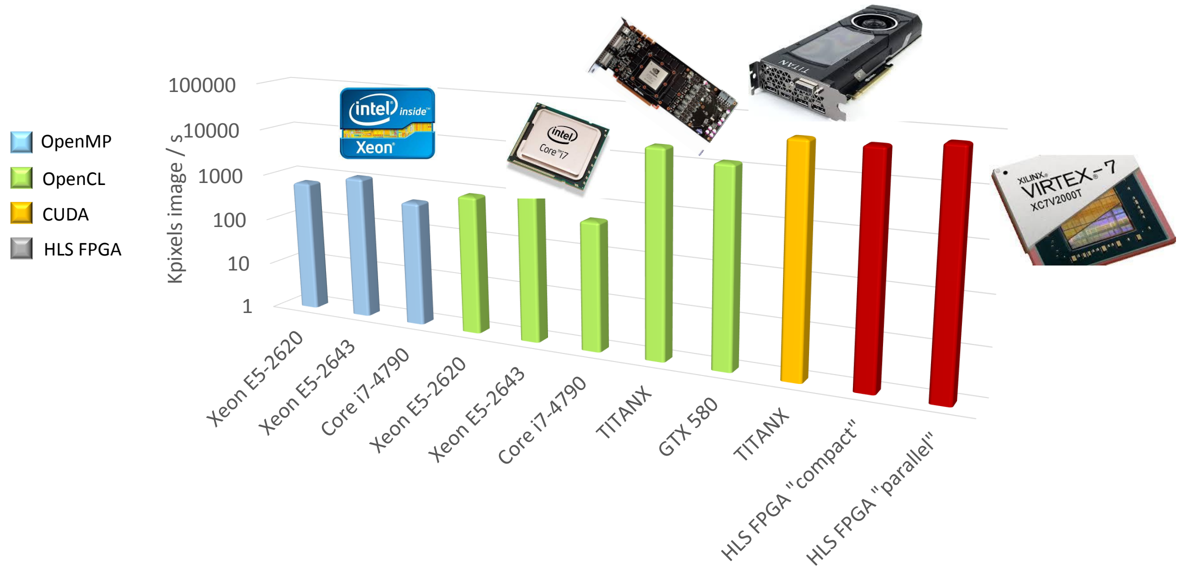


Figure 3: Example of network benchmarking on different hardware targets.

The figure 3 shows an example of benchmarking results of the previous DNN on different targets (in log scale). Compared to desktop CPUs, the number of input image pixels processed per second is more than one order of magnitude higher with GPUsand at least two orders of magnitude better with synthesized DNN on FPGA.

1.6 Summary

The N2D2 platform is today a complete and production ready neural network building tool, which does not require advanced knownledges in deep learning to be used. It is tailored for fast neural network applications generation and porting with minimum overhead in terms of database creation and management, data pre-processing, networks configuration and optimized code generation, which can save months of manual porting and verification effort to a single automated step in the tool.

2 Performing simulations

2.1 Obtaining the latest version of this manual

Before going further, please make sure you are reading the latest version of this manual. It is located in the `manual` sub-directory. To compile the manual in PDF, just run the following command:

```
cd manual && make
```

2.2 Obtaining N2D2

2.2.1 Prerequisites

First, make sure you have the following packages installed:

- gnuplot
- opencv
- opencv-devel (on RHEL 6, requires rhel-x86_64-workstation-optional-6 repository channel)

Plus, to be able to use GPU acceleration:

- cuda

  To install cuda, add the cuda repository for RHEL 6:

  ```
  rpm -Uhv http://developer.download.nvidia.com/compute/cuda/repos/rhel6/x86_64/cuda-repo-
      rhel6-7.5-18.x86_64.rpm
  yum clean expire-cache
  yum install cuda
  ```

  Make sure the cuda library path (by default: `/usr/local/cuda/lib64`) is added to the `LD_LIBRARY_PATH` environment variable.
- cudnn

  Manual installation required: register to NVIDIA Developer and download the latest version of CuDNN. Simply copy the header and library files from the CuDNN archive to the corresponding directories in the cuda installation path (by default: `/usr/local/cuda/include` and `/usr/local/cuda/lib64`, respectively).

2.2.2 Getting the sources

Use the following command:

```
git clone git@github.com:CEA-LIST/N2D2.git
```

2.2.3 Compilation

To compile the program:

```
mkdir build
cd build
cmake .. && make
```

2.3 Downloading training datasets

A python script located in the repository root directory allows you to automatically download some well-known datasets, like MNIST and GTSRB:

```
./tools/install_stimuli.py
```

By default, the datasets are downloaded in the path specified in the `N2D2_DATA` environment variable, which is the root path used by the N2D2 tool to locate the databases. If the `N2D2_DATA` variable is not set, the default value used is `/local/$USER/n2d2_data/` (or `/local/n2d2_data/` if the `USER` environment variable is not set) on Linux and `C:\n2d2_data\` on Windows.

2.4 Run the learning

The following command will run the learning for 600,000 image presentations/steps and log the performances of the network every 10,000 steps:

```
./n2d2 "mnist24_16c4s2_24c5s2_150_10.ini" -learn 600000 -log 10000
```

Note: you may want to check the gradient computation using the `-check` option. Note that it can be extremely long and can occasionally fail if the required precision is too high.

2.5 Test a learned network

After the learning is completed, this command evaluate the network performances on the test data set:

```
./n2d2 "mnist24_16c4s2_24c5s2_150_10.ini" -test
```

2.5.1 Interpreting the results

Recognition rate The recognition rate and the validation score are reported during the learning in the *TargetScore_*/Success_validation.png* file, as shown in figure 4.

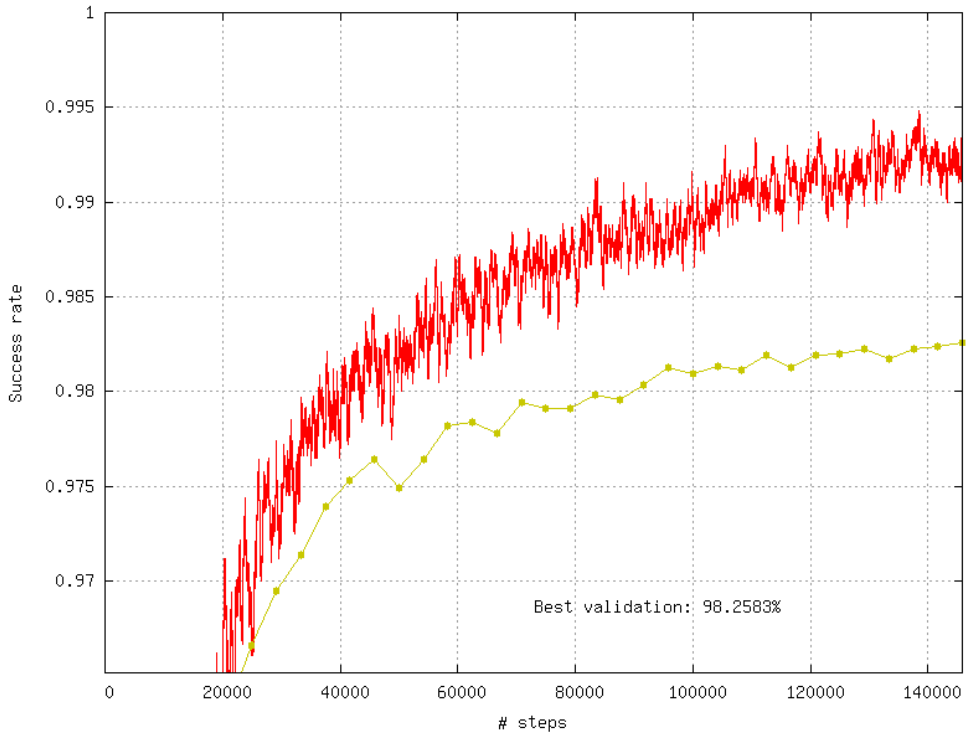


Figure 4: Recognition rate and validation score during learning.

Confusion matrix The software automatically outputs the confusion matrix during learning, validation and test, with an example shown in figure 5. Each row of the matrix contains the number of occurrences estimated by the network for each label, for all the data corresponding to a single actual, target label. Or equivalently, each column of the matrix contains the number of actual, target label occurrences, corresponding to the same estimated label. Idealy, the matrix should be diagonal, with no occurrence of an estimated label for a different actual label (network mistake).

The confusion matrix reports can be found in the simulation directory:

- *TargetScore_*/ConfusionMatrix_learning.png*;

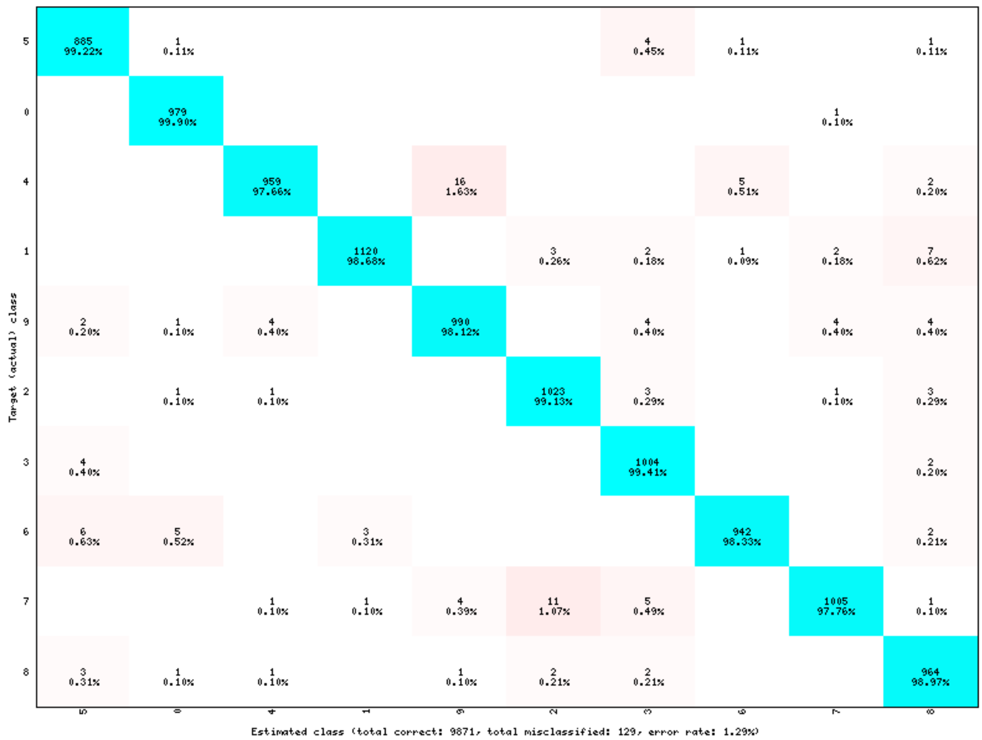


Figure 5: Example of confusion matrix obtained after the learning.

- *TargetScore_*/ConfusionMatrix_validation.png*;
- *TargetScore_*/ConfusionMatrix_test.png*.

Memory and computation requirements The software also report the memory and computation requirements of the network, as shown in figure 6. The corresponding report can be found in the *stats* sub-directory of the simulation.

Kernels and weights distribution The synaptic weights obtained during and after the learning can be analyzed, in terms of distribution (*weights* sub-directory of the simulation) or in terms of kernels (*kernels* sub-directory of the simulation), as shown in 7.

Output maps activity The initial output maps activity for each layer can be visualized in the *outputs_init* sub-directory of the simulation, as shown in figure 8.

2.6 Export a learned network

```
./n2d2 "mnist24_16c4s2_24c5s2_150_10.ini" -export CPP_OpenCL
```

Export types:

- `C` C export using OpenMP;
- `C_HLS` C export tailored for HLS with Vivado HLS;
- `CPP_OpenCL` C++ export using OpenCL;
- `CPP_Cuda` C++ export using Cuda;
- `CPP_cuDNN` C++ export using CuDNN;
- `SC_Spike` SystemC spike export.

Other program options related to the exports:

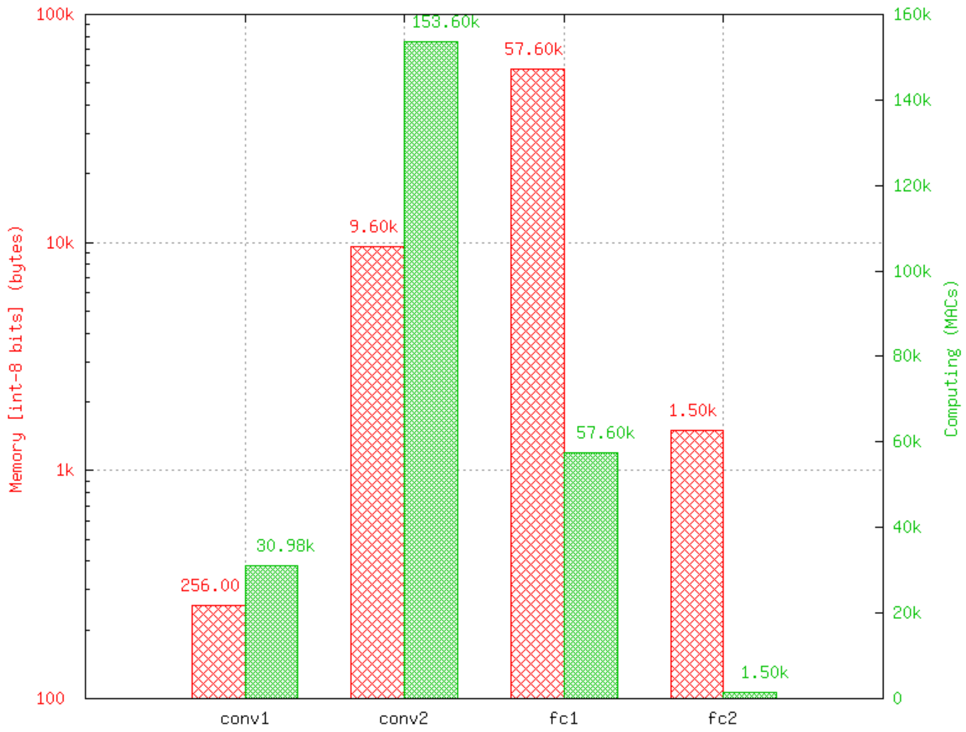


Figure 6: Example of memory and computation requirements of the network.

Option [default value]	Description
`-uenv`	If present, treat the input stimuli data as unsigned
`-nbbits` [8]	Number of bits for the weights and signals. Must be 8, 16, 32 or 64 for integer export, or -32, -64 for floating point export. The number of bits can be arbitrary for the `C_HLS` export (for example, 6 bits)

N2D2 IP *only*

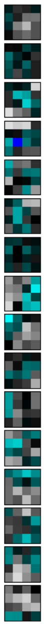
2.6.1 C export

Test the exported network:

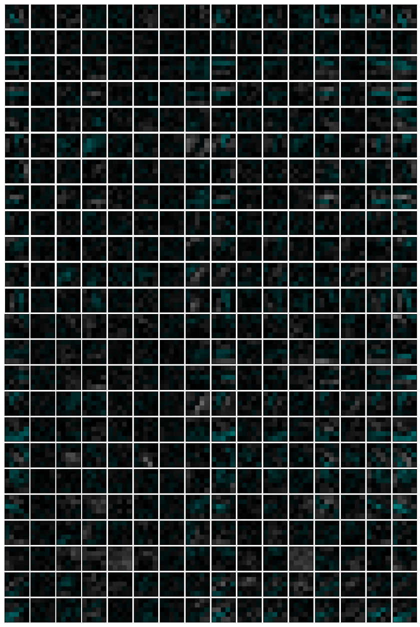
```
cd export_C_int8
make
./bin/n2d2_test
```

The result should look like:

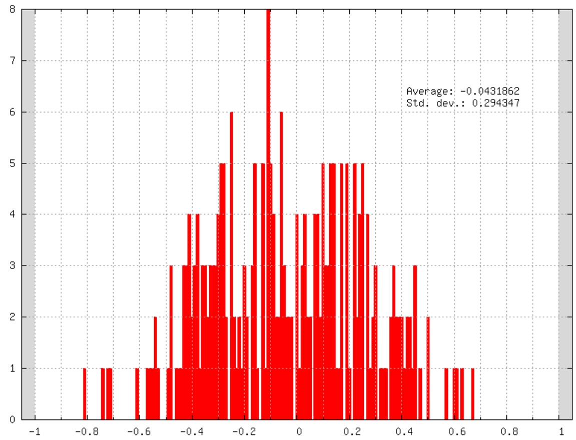
```
...
1652.00/1762    (avg = 93.757094%)
1653.00/1763    (avg = 93.760635%)
1654.00/1764    (avg = 93.764172%)
Tested 1764 stimuli
Success rate = 93.764172%
Process time per stimulus = 187.548186 us (12 threads)

Confusion matrix:
-------------------------------------------------
| T \ E |       0 |       1 |       2 |       3 |
-------------------------------------------------
|     0 |     329 |       1 |       5 |       2 |
|       |  97.63% |   0.30% |   1.48% |   0.59% |
|     1 |       0 |     692 |       2 |       6 |
|       |   0.00% |  98.86% |   0.29% |   0.86% |
|     2 |      11 |      27 |     609 |      55 |
```



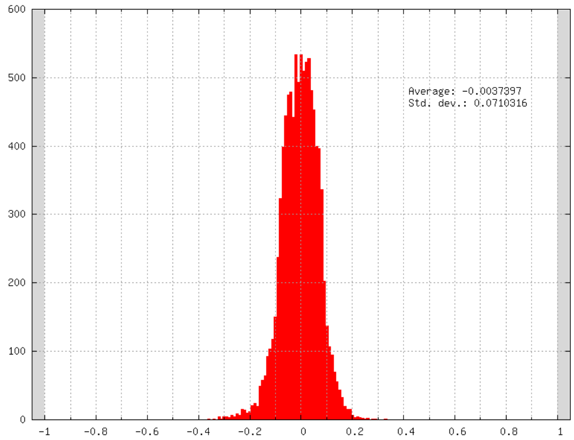
conv1 kernels



conv2 kernels



conv1 weights distribution



conv2 weights distribution

Figure 7: Example of kernels and weights distribution analysis for two convolutional layers.

```
|          |    1.57%  |    3.85%  |   86.75%  |    7.83%  |
|        3 |        0  |        0  |        1  |       24  |
|          |    0.00%  |    0.00%  |    4.00%  |   96.00%  |
------------------------------------------------------------
T: Target    E: Estimated
```

`N2D2 IP` *only*

2.6.2 `CPP_OpenCL` export

The OpenCL export can run the generated program in GPU or CPU architectures. Compilation features:

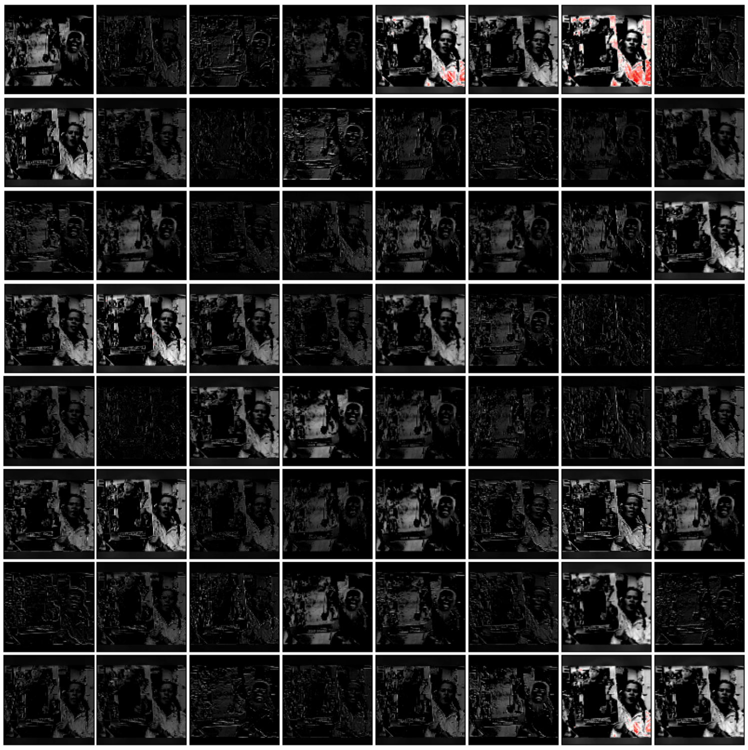


Figure 8: Output maps activity example of the first convolutional layer of the network.

Preprocessor command [default value]	Description
`PROFILING` [0]	Compile the binary with a synchronization between each layers and return the mean execution time of each layer. This preprocessor option can decrease performances.
`GENERATE_KBIN` [0]	Generate the binary output of the OpenCL kernel .cl file use. The binary is store in the /bin folder.
`LOAD_KBIN` [0]	Indicate to the program to load an OpenCL kernel as a binary from the /bin folder instead of a .cl file.
`CUDA` [0]	Use the CUDA OpenCL SDK locate at $/usr/local/cuda$
`MALI` [0]	Use the MALI OpenCL SDK locate at $/usr/Mali_OpenCL_SDK_vXXX$
`INTEL` [0]	Use the INTEL OpenCL SDK locate at $/opt/intel/opencl$
`AMD` [1]	Use the AMD OpenCL SDK locate at $/opt/AMDAPPSDK-XXX$

Program options related to the OpenCL export:

Option [default value]	Description
`-cpu`	If present, force to use a CPU architecture to run the program
`-gpu`	If present, force to use a GPU architecture to run the program
`-batch` [1]	Size of the batch to use
`-stimulus` [NULL]	Path to a specific input stimulus to test. For example: -stimulus $/stimulus/env0000.pgm$ command will test the file env0000.pgm of the stimulus folder.

Test the exported network:

```
cd export_CPP_OpenCL_float32
make
./bin/n2d2_opencl_test -gpu
```

2.6.3 CPP_cuDNN export

The cuDNN export can run the generated program in NVIDIA GPU architecture. It use CUDA and cuDNN library. Compilation features:

Preprocessor command [default value]	Description
`PROFILING` [0]	Compile the binary with a synchronization between each layers and return the mean execution time of each layer. This preprocessor option can decrease performances.
`ARCH32` [0]	Compile the binary with the 32-bits architecture compatibility.

Program options related to the cuDNN export:

Option [default value]	Description
`-batch` [1]	Size of the batch to use
`-dev` [0]	CUDA Device ID selection
`-stimulus` [NULL]	Path to a specific input stimulus to test. For example: -stimulus */stimulus/env*0000*.pgm* command will test the file env0000.pgm of the stimulus folder.

Test the exported network:

```
cd export_CPP_cuDNN_float32
make
./bin/n2d2_cudnn_test
```

N2D2 IP *only*

2.6.4 C_HLS export

Test the exported network:

```
cd export_C_HLS_int8
make
./bin/n2d2_test
```

Run the High-Level Synthesis (HLS) with Xilinx® Vivado® HLS:

```
vivado_hls -f run_hls.tcl
```

3 INI file interface

3.1 Global parameters

Option [default value]	Description
`DefaultModel` [`Transcode`]	Default layers model. Can be `Frame`, `Frame_CUDA`, `Transcode` or `Spike`
`SignalsDiscretization` [0]	Number of levels for signal discretization
`FreeParametersDiscretization` [0]	Number of levels for weights discretization

3.2 Databases

The tool integrates pre-defined modules for several well-known database used in the deep learning community, such as MNIST, GTSRB, CIFAR10 and so on. That way, no extra step is necessary to be able to directly build a network and learn it on these database.

3.2.1 MNIST

MNIST (LeCun et al., 1998) is already fractionned into a learning set and a testing set, with:

- 60,000 digits in the learning set;
- 10,000 digits in the testing set.

Example:

```
[database]
Type=MNIST_IDX_Database
Validation=0.2 ; Fraction of learning stimuli used for the validation [default: 0.0]
```

Option [default value]	Description
`Validation` [0.0]	Fraction of the learning set used for validation
`DataPath` [`$N2D2_DATA`/mnist]	Path to the database

3.2.2 GTSRB

GTSRB (Stallkamp et al., 2012) is already fractionned into a learning set and a testing set, with:

- 39,209 digits in the learning set;
- 12,630 digits in the testing set.

Example:

```
[database]
Type=GTSRB_DIR_Database
Validation=0.2 ; Fraction of learning stimuli used for the validation [default: 0.0]
```

Option [default value]	Description
`Validation` [0.0]	Fraction of the learning set used for validation
`DataPath` [`$N2D2_DATA`/GTSRB]	Path to the database

3.2.3 Directory

Hand made database stored in files directories are directly supported with the `DIR_Database` module. For example, suppose your database is organized as following (in the path specified in the `N2D2_DATA` environment variable):

- `GST/airplanes`: 800 images
- `GST/car_side`: 123 images
- `GST/Faces`: 435 images
- `GST/Motorbikes`: 798 images

You can then instanciate this database as input of your neural network using the following parameters:

```
[database]
Type=DIR_Database
DataPath=${N2D2_DATA}/GST
Learn=0.4 ; 40% of images of the smallest category = 49 (0.4x123) images for each category will be
    used for learning
Validation=0.2 ; 20% of images of the smallest category = 25 (0.2x123) images for each category
    will be used for validation
; the remaining images will be used for testing
```

Each subdirectory will be treated as a different label, so there will be 4 different labels, named after the directory name.

The stimuli are equi-partitioned for the learning set and the validation set, meaning that the same number of stimuli for each category is used. If the learn fraction is 0.4 and the validation fraction is 0.2, as in the example above, the partitioning will be the following:

Label ID	Label name	Learn set	Validation set	Test set
0	`airplanes`	49	25	726
1	`car_side`	49	25	49
2	`Faces`	49	25	361
3	`Motorbikes`	49	25	724
	Total:	196	100	1860

Mandatory option

Option [default value]	Description
`DataPath`	Path to the root stimuli directory
`Learn`	If `PerLabelPartitioning` is true, fraction of images used for the learning; else, number of images used for the learning, regardless of their labels
`LoadInMemory` [0]	Load the whole database into memory
`Depth` [1]	Number of sub-directory levels to include. Examples: `Depth` = 0: load stimuli only from the current directory (`DataPath`) `Depth` = 1: load stimuli from `DataPath` and stimuli contained in the sub-directories of `DataPath` `Depth` < 0: load stimuli recursively from `DataPath` and all its sub-directories
`LabelName` []	Base stimuli label name
`LabelDepth` [1]	Number of sub-directory name levels used to form the stimuli labels. Examples: `LabelDepth` = -1: no label for all stimuli (label ID = -1)

	LabelDepth = 0: uses LabelName for all stimuli LabelDepth = 1: uses LabelName for stimuli in the current directory (DataPath) and LabelName/*sub-directory name* for stimuli in the sub-directories
PerLabelPartitioning [1]	If true, the stimuli are equi-partitioned for the learn/validation/test sets, meaning that the same number of stimuli for each label is used
Validation [0.0]	If PerLabelPartitioning is true, fraction of images used for the validation; else, number of images used for the validation, regardless of their labels
Test [1.0-Learn-Validation]	If PerLabelPartitioning is true, fraction of images used for the test; else, number of images used for the test, regardless of their labels
ROIFile []	File containing the stimuli ROIs. If a ROI file is specified, LabelDepth should be set to -1
DefaultLabel []	Label name for pixels outside any ROI (default is no label, pixels are ignored)
ROIsMargin [0]	Number of pixels around ROIs that are ignored (and not considered as DefaultLabel pixels)

3.2.4 Other built-in databases

CIFAR10_Database CIFAR10 database (Krizhevsky, 2009).

Option [default value]	Description
Validation [0.0]	Fraction of the learning set used for validation
DataPath [$N2D2_DATA/cifar-10-batches-bin]	Path to the database

CIFAR100_Database CIFAR100 database (Krizhevsky, 2009).

Option [default value]	Description
Validation [0.0]	Fraction of the learning set used for validation
UseCoarse [0]	If true, use the coarse labeling (10 labels instead of 100)
DataPath [$N2D2_DATA/cifar-100-binary]	Path to the database

CKP_Database The Extended Cohn-Kanade (CK+) database for expression recognition (Lucey et al., 2010).

Option [default value]	Description
Learn	Fraction of images used for the learning
Validation [0.0]	Fraction of images used for the validation
DataPath [$N2D2_DATA/cohn-kanade-images]	Path to the database

Caltech101_DIR_Database Caltech 101 database (Fei-Fei et al., 2004).

Option [default value]	Description
Learn	Fraction of images used for the learning
Validation [0.0]	Fraction of images used for the validation
IncClutter [0]	If true, includes the BACKGROUND_Google directory of the database
DataPath [$N2D2_DATA/ 101_ObjectCategories]	Path to the database

Caltech256_DIR_Database Caltech 256 database (Griffin et al., 2007).

Option [default value]	Description
Learn	Fraction of images used for the learning
Validation [0.0]	Fraction of images used for the validation
IncClutter [0]	If true, includes the BACKGROUND_Google directory of the database
DataPath [$N2D2_DATA/ 256_ObjectCategories]	Path to the database

CaltechPedestrian_Database Caltech Pedestrian database (Dollár et al., 2009).

Note that the images and annotations must first be extracted from the seq video data located in the *videos* directory using the `dbExtract.m` Matlab tool provided in the "Matlab evaluation/labeling code" downloadable on the dataset website.

Assuming the following directory structure (in the path specified in the `N2D2_DATA` environment variable):

- `CaltechPedestrians/data-USA/videos/...` (from the *setxx.tar* files)
- `CaltechPedestrians/data-USA/annotations/...` (from the *setxx.tar* files)
- `CaltechPedestrians/tools/piotr_toolbox/toolbox` (from the Piotr's Matlab Toolbox archive)
- `CaltechPedestrians/*.m` including `dbExtract.m` (from the Matlab evaluation/labeling code)

Use the following command in Matlab to generate the images and annotations:

```
cd([getenv('N2D2_DATA') '/CaltechPedestrians'])
addpath(genpath('tools/piotr_toolbox/toolbox')) % add the Piotr's Matlab Toolbox in the Matlab
    path
dbInfo('USA')
dbExtract()
```

Option [default value]	Description
Validation [0.0]	Fraction of the learning set used for validation
SingleLabel [1]	Use the same label for "person" and "people" bounding box
IncAmbiguous [0]	Include ambiguous bounding box labeled "person?" using the same label as "person"
DataPath [$N2D2_DATA/ CaltechPedestrians/data-USA/images]	Path to the database images
LabelPath [$N2D2_DATA/ CaltechPedestrians/data-USA/annotations]	Path to the database annotations

Daimler_Database Daimler Monocular Pedestrian Detection Benchmark (Daimler Pedestrian).

Option [default value]	Description
Learn [1.0]	Fraction of images used for the learning
Validation [0.0]	Fraction of images used for the validation
Test [0.0]	Fraction of images used for the test
Fully [0]	When activate it use the test dataset to learn. Use only on fully-cnn mode

FDDB_Database Face Detection Data Set and Benchmark (FDDB) (Jain and Learned-Miller, 2010).

Option [default value]	Description
Learn	Fraction of images used for the learning
Validation [0.0]	Fraction of images used for the validation
DataPath [$N2D2_DATA/FDDB]	Path to the images (decompressed originalPics.tar.gz)
LabelPath [$N2D2_DATA/FDDB]	Path to the annotations (decompressed FDDB-folds.tgz)

GTSDB_DIR_Database GTSDB database (Houben et al., 2013).

Option [default value]	Description
Learn	Fraction of images used for the learning
Validation [0.0]	Fraction of images used for the validation
DataPath [$N2D2_DATA/FullIJCNN2013]	Path to the database

ILSVRC2012_Database ILSVRC2012 database (Russakovsky et al., 2015).

Option [default value]	Description
Learn	Fraction of images used for the learning
DataPath [$N2D2_DATA/ILSVRC2012]	Path to the database
LabelPath [$N2D2_DATA /ILSVRC2012/synsets.txt]	Path to the database labels list file

KITTY_Database KITTY Database.

Option [default value]	Description
Learn [0.8]	Fraction of images used for the learning
Validation [0.2]	Fraction of images used for the validation

KITTY_Road_Database KITTY Road Database. The KITTY Road Database provide ROI which can be used to road segmentation.

Option [default value]	Description
Learn [0.8]	Fraction of images used for the learning
Validation [0.2]	Fraction of images used for the validation

LITISRouen_Database LITIS Rouen audio scene dataset (Rakotomamonjy and Gasso, 2014).

Option [default value]	Description
`Learn` [0.4]	Fraction of images used for the learning
`Validation` [0.4]	Fraction of images used for the validation
`DataPath` [`$N2D2_DATA`/data_rouen]	Path to the database

3.2.5 Dataset images slicing

It is possible to automatically slice images from a dataset, with a given slice size and stride, using the `.slicing` attribute. This effectively increases the number of stimuli in the set.

```
[database.slicing]
ApplyTo=NoLearn
Width=2048
Height=1024
StrideX=2048
StrideY=1024
```

3.3 Stimuli data analysis

You can enable stimuli data reporting with the following section (the name of the section must start with `env.StimuliData`):

```
[env.StimuliData-raw]
ApplyTo=LearnOnly
LogSizeRange=1
LogValueRange=1
```

The stimuli data reported for the full MNIST learning set will look like:

```
env.StimuliData-raw data:
Number of stimuli: 60000
Data width range: [28, 28]
Data height range: [28, 28]
Data channels range: [1, 1]
Value range: [0, 255]
Value mean: 33.3184
Value std. dev.: 78.5675
```

3.3.1 Zero-mean and unity standard deviation normalization

It it possible to normalize the whole database to have zero mean and unity standard deviation on the learning set using a `RangeAffineTransformation` transformation:

```
; Stimuli normalization based on learning set global mean and std.dev.
[env.Transformation-normalize]
Type=RangeAffineTransformation
FirstOperator=Minus
FirstValue=[env.StimuliData-raw]_GlobalValue.mean
SecondOperator=Divides
SecondValue=[env.StimuliData-raw]_GlobalValue.stdDev
```

The variables `_GlobalValue.mean` and `_GlobalValue.stdDev` are automatically generated in the `[env.StimuliData-raw]` block. Thanks to this facility, unknown and arbitrary database can be analysed and normalized in one single step without requiring any external data manipulation.

After normalization, the stimuli data reported is:

```
env.StimuliData-normalized data:
Number of stimuli: 60000
Data width range: [28, 28]
Data height range: [28, 28]
Data channels range: [1, 1]
Value range: [-0.424074, 2.82154]
Value mean: 2.64796e-07
Value std. dev.: 1
```

Where we can check that the global mean is close to 0 and the standard deviation is 1 on the whole dataset. The result of the transformation on the first images of the set can be checked in the generated *frames* folder, as shown in figure 9.

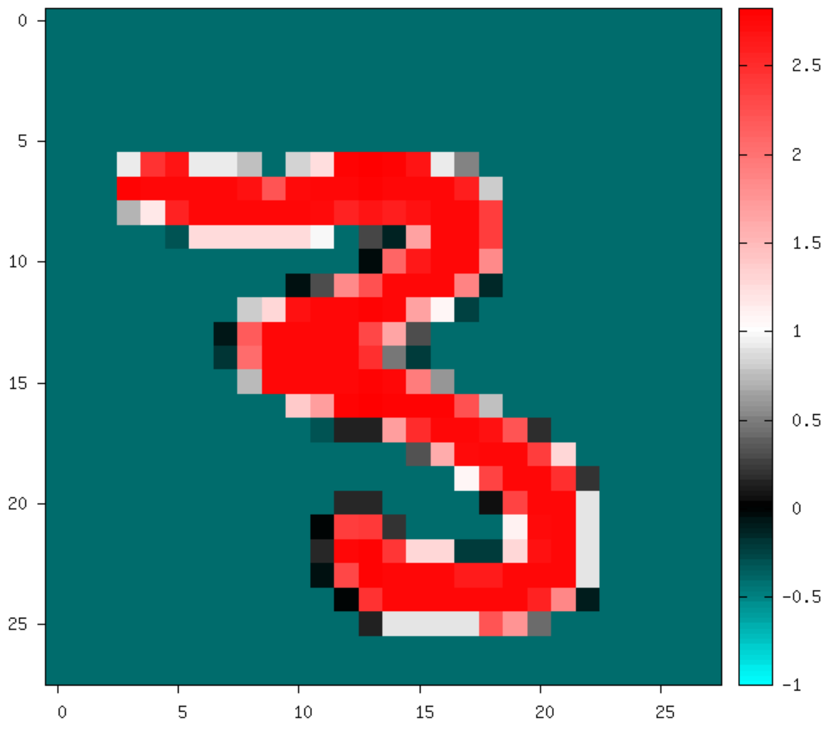


Figure 9: Image of the set after normalization.

3.3.2 Substracting the mean image of the set

Using the `StimuliData` object followed with an `AffineTransformation`, it is also possible to use the mean image of the dataset to normalize the data:

```
[env.StimuliData-meanData]
ApplyTo=LearnOnly
MeanData=1 ; Provides the _MeanData parameter used in the transformation

[env.Transformation]
Type=AffineTransformation
FirstOperator=Minus
FirstValue=[env.StimuliData-meanData]_MeanData
```

The resulting global mean image can be visualized in *env.StimuliData-meanData/meanData.bin.png* an is shown in figure 10.

After this transformation, the reported stimuli data becomes:

```
env.StimuliData-processed data:
Number of stimuli: 60000
Data width range: [28, 28]
```

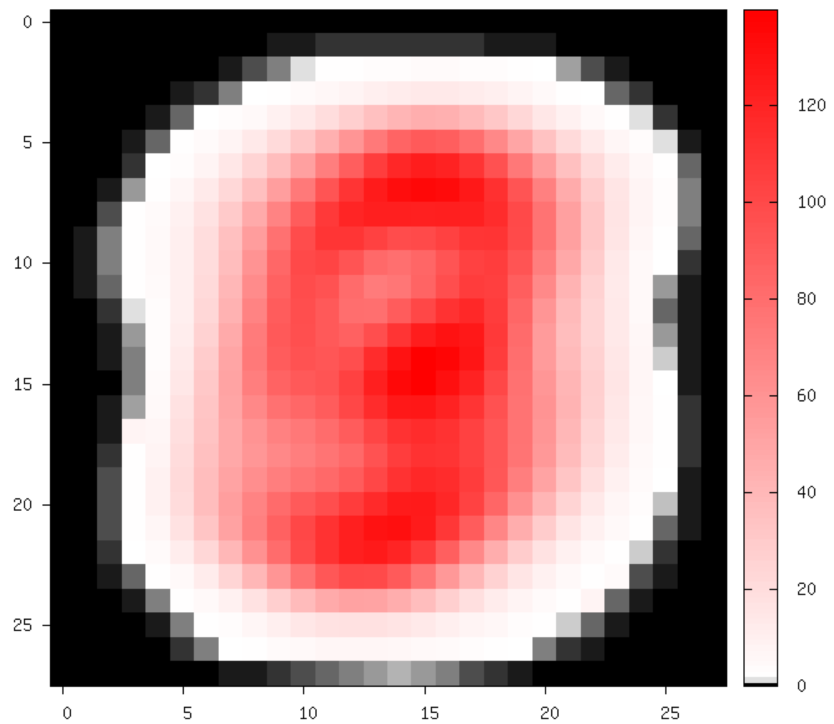


Figure 10: Global mean image generated by `StimuliData` with the `MeanData` parameter enabled.

```
Data height range: [28, 28]
Data channels range: [1, 1]
Value range: [-139.554, 254.979]
Value mean: -3.45583e-08
Value std. dev.: 66.1288
```

The result of the transformation on the first images of the set can be checked in the generated *frames* folder, as shown in figure 11.

3.4 Environment

The environment simply specify the input data format of the network (width, height and batch size). Example:

```
[env]
SizeX=24
SizeY=24
BatchSize=12 ; [default: 1]
```

Option [default value]	Description
`SizeX`	Environment width
`SizeY`	Environment height
`NbChannels` [1]	Number of channels (applicable only if there is no `env.ChannelTransformation[...]`)
`BatchSize` [1]	Batch size
`CompositeStimuli` [0]	If true, use pixel-wise stimuli labels
`CachePath` []	Stimuli cache path (no cache if left empty)
`StimulusType` [`SingleBurst`]	Method for converting stimuli into spike trains. Can be any of `SingleBurst`, `Periodic`, `JitteredPeriodic` or `Poissonian`

`DiscardedLateStimuli` [1.0]	The pixels in the pre-processed stimuli with a value above this limit never generate spiking events
`PeriodMeanMin` [50 `TimeMs`]	Mean minimum period $\overline{T_{min}}$, used for periodic temporal codings, corresponding to pixels in the pre-processed stimuli with a value of 0 (which are supposed to be the most significant pixels)
`PeriodMeanMax` [12 `TimeS`]	Mean maximum period $\overline{T_{max}}$, used for periodic temporal codings, corresponding to pixels in the pre-processed stimuli with a value of 1 (which are supposed to be the least significant pixels). This maximum period may be never reached if `DiscardedLateStimuli` is lower than 1.0
`PeriodRelStdDev` [0.1]	Relative standard deviation, used for periodic temporal codings, applied to the spiking period of a pixel
`PeriodMin` [11 `TimeMs`]	Absolute minimum period, or spiking interval, used for periodic temporal codings, for any pixel

3.4.1 Built-in transformations

There are 6 possible categories of transformations:

- `env.Transformation[...]` Transformations applied to the input images before channels creation;
- `env.OnTheFlyTransformation[...]` On-the-fly transformations applied to the input images before channels creation;
- `env.ChannelTransformation[...]` Create or add transformation for a specific channel;
- `env.ChannelOnTheFlyTransformation[...]` Create or add on-the-fly transformation for a specific channel;
- `env.ChannelsTransformation[...]` Transformations applied to all the channels of the input images;
- `env.ChannelsOnTheFlyTransformation[...]` On-the-fly transformations applied to all the channels of the input images.

Example:

```
[env.Transformation]
Type=PadCropTransformation
Width=24
Height=24
```

Several transformations can applied successively. In this case, to be able to apply multiple transformations of the same category, a different suffix (`[...]`) must be added to each transformation.

The transformations will be processed in the order of appearance in the INI file regardless of their suffix.

Common set of parameters for any kind of transformation:

Option [default value]	Description
`ApplyTo` [`All`]	Apply the transformation only to the specified stimuli sets. Can be: `LearnOnly`: learning set only `ValidationOnly`: validation set only `TestOnly`: testing set only `NoLearn`: validation and testing sets only `NoValidation`: learning and testing sets only `NoTest`: learning and validation sets only `All`: all sets (default)

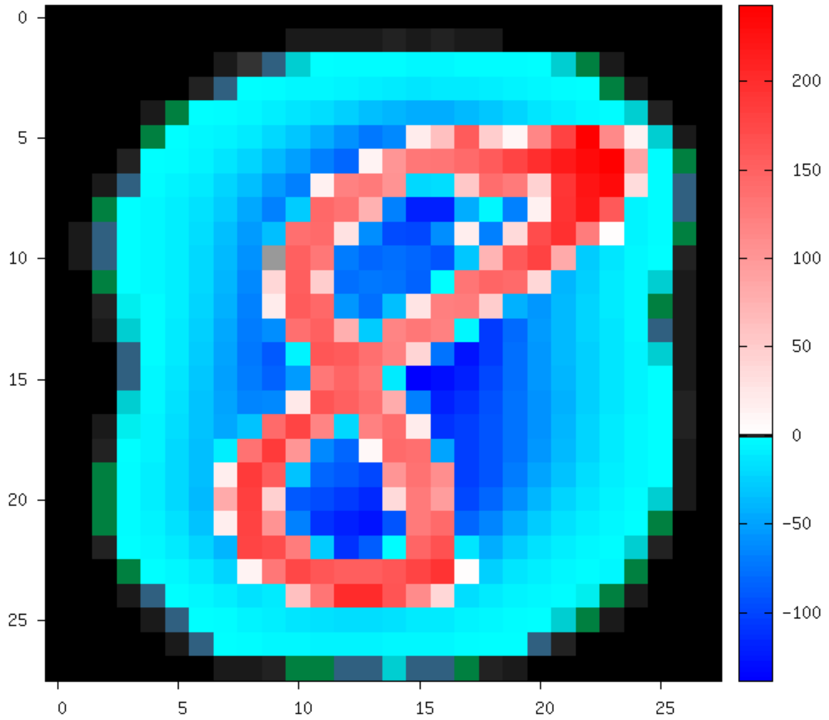


Figure 11: Image of the set after the `AffineTransformation` substracting the global mean image (keep in mind that the original image value range is [0, 255]).

Example:

```
[env.Transformation-1]
Type=ChannelExtractionTransformation
CSChannel=Gray

[env.Transformation-2]
Type=RescaleTransformation
Width=29
Height=29

[env.Transformation-3]
Type=EqualizeTransformation

[env.OnTheFlyTransformation]
Type=DistortionTransformation
ApplyTo=LearnOnly ; Apply this transformation for the Learning set only
ElasticGaussianSize=21
ElasticSigma=6.0
ElasticScaling=20.0
Scaling=15.0
Rotation=15.0
```

List of available transformations:

`AffineTransformation` Apply an element-wise affine transformation to the image with matrixes of the same size.

Option [default value]	Description
`FirstOperator`	First element-wise operator, can be `Plus`, `Minus`, `Multiplies`, `Divides`
`FirstValue`	First matrix file name
`SecondOperator` [`Plus`]	Second element-wise operator, can be `Plus`, `Minus`, `Multiplies`, `Divides`
`SecondValue` []	Second matrix file name

The final operation is the following, with A the image matrix, B_{1st}, B_{2nd} the matrixes to add/substract/multiply/divide and $\odot$ the element-wise operator :

$$f(A) = (A \underset{op1st}{\odot} B_{1st}) \underset{op2nd}{\odot} B_{2nd}$$

`ApodizationTransformation` Apply an apodization window to each data row.

Option [default value]	Description
`Size`	Window total size (must match the number of data columns)
`WindowName` [`Rectangular`]	Window name. Possible values are: `Rectangular`: Rectangular `Hann`: Hann `Hamming`: Hamming `Cosine`: Cosine `Gaussian`: Gaussian `Blackman`: Blackman `Kaiser`: Kaiser

`Gaussian window` Gaussian window.

Option [default value]	Description
WindowName`.Sigma` [0.4]	Sigma

`Blackman window` Blackman window.

Option [default value]	Description
WindowName`.Alpha` [0.16]	Alpha

`Kaiser window` Kaiser window.

Option [default value]	Description
WindowName`.Beta` [5.0]	Beta

`ChannelExtractionTransformation` Extract an image channel.

Option	Description
`CSChannel`	`Blue`: blue channel in the BGR colorspace, or first channel of any colorspace `Green`: green channel in the BGR colorspace, or second channel of any colorspace `Red`: red channel in the BGR colorspace, or third channel of any colorspace `Hue`: hue channel in the HSV colorspace `Saturation`: saturation channel in the HSV colorspace `Value`: value channel in the HSV colorspace `Gray`: gray conversion `Y`: Y channel in the YCbCr colorspace `Cb`: Cb channel in the YCbCr colorspace `Cr`: Cr channel in the YCbCr colorspace

`ColorSpaceTransformation` Change the current image colorspace.

Option	Description

`ColorSpace`	`BGR`: if the image is in grayscale, convert it in BGR
	`HSV`
	`HLS`
	`YCrCb`
	`CIELab`
	`CIELuv`

`DFTTransformation` Apply a DFT to the data. The input data must be single channel, the resulting data is two channels, the first for the real part and the second for the imaginary part.

Option [default value]	Description
`TwoDimensional` [1]	If true, compute a 2D image DFT. Otherwise, compute the 1D DFT of each data row

Note that this transformation can add zero-padding if required by the underlying FFT implementation.

`N2D2 IP` *only* **`DistortionTransformation`** Apply elastic distortion to the image. This transformation is generally used on-the-fly (so that a different distortion is performed for each image), and for the learning only.

Option [default value]	Description
`ElasticGaussianSize` [15]	Size of the gaussian for elastic distortion (in pixels)
`ElasticSigma` [6.0]	Sigma of the gaussian for elastic distortion
`ElasticScaling` [0.0]	Scaling of the gaussian for elastic distortion
`Scaling` [0.0]	Maximum random scaling amplitude (+/-, in percentage)
`Rotation` [0.0]	Maximum random rotation amplitude (+/-, in °)

`N2D2 IP` *only* **`EqualizeTransformation`** Image histogram equalization.

Option [default value]	Description
`Method` [`Standard`]	`Standard`: standard histogram equalization `CLAHE`: contrast limited adaptive histogram equalization
`CLAHE_ClipLimit` [40.0]	Threshold for contrast limiting (for `CLAHE` only)
`CLAHE_GridSize` [8]	Size of grid for histogram equalization (for `CLAHE` only). Input image will be divided into equally sized rectangular tiles. This parameter defines the number of tiles in row and column.

`N2D2 IP` *only* **`ExpandLabelTransformation`** Expand single image label (1x1 pixel) to full frame label.

`FilterTransformation` Apply a convolution filter to the image.

Option [default value]	Description
`Kernel`	Convolution kernel. Possible values are: `*`: custom kernel `Gaussian`: Gaussian kernel `LoG`: Laplacian Of Gaussian kernel `DoG`: Difference Of Gaussian kernel `Gabor`: Gabor kernel

`*` kernel Custom kernel.

Option	Description
`Kernel.SizeX` [0]	Width of the kernel (numer of columns)
`Kernel.SizeY` [0]	Height of the kernel (number of rows)
`Kernel.Mat`	List of row-major ordered coefficients of the kernel

If both `Kernel.SizeX` and `Kernel.SizeY` are 0, the kernel is assumed to be square.

`Gaussian` kernel Gaussian kernel.

Option [default value]	Description
`Kernel.SizeX`	Width of the kernel (numer of columns)
`Kernel.SizeY`	Height of the kernel (number of rows)
`Kernel.Positive` [1]	If true, the center of the kernel is positive
`Kernel.Sigma` [$\sqrt{2.0}$]	Sigma of the kernel

`LoG` kernel Laplacian Of Gaussian kernel.

Option [default value]	Description
`Kernel.SizeX`	Width of the kernel (numer of columns)
`Kernel.SizeY`	Height of the kernel (number of rows)
`Kernel.Positive` [1]	If true, the center of the kernel is positive
`Kernel.Sigma` [$\sqrt{2.0}$]	Sigma of the kernel

`DoG` kernel Difference Of Gaussian kernel kernel.

Option [default value]	Description
`Kernel.SizeX`	Width of the kernel (numer of columns)
`Kernel.SizeY`	Height of the kernel (number of rows)
`Kernel.Positive` [1]	If true, the center of the kernel is positive
`Kernel.Sigma1` [2.0]	Sigma1 of the kernel
`Kernel.Sigma2` [1.0]	Sigma2 of the kernel

Gabor kernel Gabor kernel.

Option [default value]	Description
`Kernel.SizeX`	Width of the kernel (numer of columns)
`Kernel.SizeY`	Height of the kernel (number of rows)
`Kernel.Theta`	Theta of the kernel
`Kernel.Sigma` [$\sqrt{2.0}$]	Sigma of the kernel
`Kernel.Lambda` [10.0]	Lambda of the kernel
`Kernel.Psi` [$\pi/2.0$]	Psi of the kernel
`Kernel.Gamma` [0.5]	Gamma of the kernel

FlipTransformation Image flip transformation.

Option [default value]	Description
`HorizontalFlip` [0]	If true, flip the image horizontally
`VerticalFlip` [0]	If true, flip the image vertically
`RandomHorizontalFlip` [0]	If true, randomly flip the image horizontally
`RandomVerticalFlip` [0]	If true, randomly flip the image vertically

N2D2 IP *only* **GradientFilterTransformation** Compute image gradient.

Option [default value]	Description
`Scale` [1.0]	Scale to apply to the computed gradient
`Delta` [0.0]	Bias to add to the computed gradient
`GradientFilter` [`Sobel`]	Filter type to use for computing the gradient. Possible options are: `Sobel`, `Scharr` and `Laplacian`
`KernelSize` [3]	Size of the filter kernel (has no effect when using the `Scharr` filter, which kernel size is always 3x3)
`ApplyToLabels` [0]	If true, use the computed gradient to filter the image label and ignore pixel areas where the gradient is below the `Threshold`. In this case, only the labels are modified, not the image
`InvThreshold` [0]	If true, ignored label pixels will be the ones with a low gradient (low contrasted areas)
`Threshold` [0.5]	Threshold applied on the image gradient
`Label` []	List of labels to filter (space-separated)
`GradientScale` [1.0]	Rescale the image by this factor before applying the gradient and the threshold, then scale it back to filter the labels

N2D2 IP *only* **LabelSliceExtractionTransformation** Extract a slice from an image belonging to a given label.

Option [default value]	Description
`Width`	Width of the slice to extract
`Height`	Height of the slice to extract
`Label` [-1]	Slice should belong to this label ID. If -1, the label ID is random

MagnitudePhaseTransformation Compute the magnitude and phase of a complex two channels input data, with the first channel x being the real part and the second channel y the imaginary part. The resulting data is two channels, the first one with the magnitude and the second one with the phase.

Option [default value]	Description
`LogScale` [0]	If true, compute the magnitude in log scale

The magnitude is:

$$M_{i,j} = \sqrt{x_{i,j}^2 + x_{i,j}^2}$$

If `LogScale` $= 1$, compute $M'_{i,j} = log(1 + M_{i,j})$.
The phase is:

$$\theta_{i,j} = atan2(y_{i,j}, x_{i,j})$$

N2D2 IP *only* **`MorphologicalReconstructionTransformation`** Apply a morphological reconstruction transformation to the image. This transformation is also useful for post-processing.

Option [default value]	Description
`Operation`	Morphological operation to apply. Can be: `ReconstructionByErosion`: reconstruction by erosion operation `ReconstructionByDilation`: reconstruction by dilation operation `OpeningByReconstruction`: opening by reconstruction operation `ClosingByReconstruction`: closing by reconstruction operation
`Size`	Size of the structuring element
`ApplyToLabels` [0]	If true, apply the transformation to the labels instead of the image
`Shape` [Rectangular]	Shape of the structuring element used for morphology operations. Can be `Rectangular`, `Elliptic` or `Cross`.
`NbIterations` [1]	Number of times erosion and dilation are applied for opening and closing reconstructions

N2D2 IP *only* **`MorphologyTransformation`** Apply a morphology transformation to the image. This transformation is also useful for post-processing.

Option [default value]	Description
`Operation`	Morphological operation to apply. Can be: `Erode`: erode operation ($= erode(src)$) `Dilate`: dilate operation ($= dilate(src)$) `Opening`: opening operation ($open(src) = dilate(erode(src))$) `Closing`: closing operation ($close(src) = erode(dilate(src))$) `Gradient`: morphological gradient ($= dilate(src) - erode(src)$) `TopHat`: top hat ($= src - open(src)$) `BlackHat`: black hat ($= close(src) - src$)
`Size`	Size of the structuring element
`ApplyToLabels` [0]	If true, apply the transformation to the labels instead of the image
`Shape` [Rectangular]	Shape of the structuring element used for morphology operations. Can be `Rectangular`, `Elliptic` or `Cross`.
`NbIterations` [1]	Number of times erosion and dilation are applied

`NormalizeTransformation` Normalize the image.

Option [default value]	Description
`Norm` [`MinMax`]	Norm type, can be: `L1`: L1 normalization `L2`: L2 normalization `Linf`: Linf normalization `MinMax`: min-max normalization
`NormValue` [1.0]	Norm value (for `L1`, `L2` and `Linf`) Such that $\|\|data\|\|_{L_p} = NormValue$
`NormMin` [0.0]	Min value (for `MinMax` only) Such that $min(data) = NormMin$
`NormMax` [1.0]	Max value (for `MinMax` only) Such that $max(data) = NormMax$
`PerChannel` [0]	If true, normalize each channel individually

`PadCropTransformation` Pad/crop the image to a specified size.

Option [default value]	Description
`Width`	Width of the padded/cropped image
`Height`	Height of the padded/cropped image
`PaddingBackground` [`MeanColor`]	Background color used when padding. Possible values: `MeanColor`: pad with the mean color of the image `BlackColor`: pad with black

N2D2 IP *only* **`RandomAffineTransformation`** Apply a global random affine transformation to the values of the image.

Option [default value]	Description
`GainVar`	Random gain is in range ±`GainVar`
`BiasVar` [0.0]	Random bias is in range ±`BiasVar`

`RangeAffineTransformation` Apply an affine transformation to the values of the image.

Option [default value]	Description
`FirstOperator`	First operator, can be `Plus`, `Minus`, `Multiplies`, `Divides`
`FirstValue`	First value
`SecondOperator` [`Plus`]	Second operator, can be `Plus`, `Minus`, `Multiplies`, `Divides`
`SecondValue` [0.0]	Second value

The final operation is the following:

$$f(x) = (x \overset{o}{_{op_{1st}}} val_{1st}) \overset{o}{_{op_{2nd}}} val_{2nd}$$

N2D2 IP *only* **`RangeClippingTransformation`** Clip the value range of the image.

Option [default value]	Description
`RangeMin` [$min(data)$]	Image values below `RangeMin` are clipped to 0
`RangeMax` [$max(data)$]	Image values above `RangeMax` are clipped to 1 (or the maximum integer value of the data type)

`RescaleTransformation` Rescale the image to a specified size.

Option [default value]	Description
`Width`	Width of the rescaled image
`Height`	Height of the rescaled image
`KeepAspectRatio` [0]	If true, keeps the aspect ratio of the image
`ResizeToFit` [1]	If true, resize along the longest dimension when `KeepAspectRatio` is true

`ReshapeTransformation` Reshape the data to a specified size.

Option [default value]	Description
`NbRows`	New number of rows
`NbCols` [0]	New number of cols (0 = no check)
`NbChannels` [0]	New number of channels (0 = no change)

`N2D2 IP` *only* **`SliceExtractionTransformation`** Extract a slice from an image.

Option [default value]	Description
`Width`	Width of the slice to extract
`Height`	Height of the slice to extract
`OffsetX` [0]	X offset of the slice to extract
`OffsetY` [0]	Y offset of the slice to extract
`RandomOffsetX` [0]	If true, the X offset is chosen randomly
`RandomOffsetY` [0]	If true, the Y offset is chosen randomly
`AllowPadding` [0]	If true, zero-padding is allowed if the image is smaller than the slice to extract

`ThresholdTransformation` Apply a thresholding transformation to the image. This transformation is also useful for post-processing.

Option [default value]	Description
`Threshold`	Threshold value
`OtsuMethod` [0]	Use Otsu's method to determine the optimal threshold (if true, the `Threshold` value is ignored)

`Operation` [Binary]	Thresholding operation to apply. Can be:
	`Binary`
	`BinaryInverted`
	`Truncate`
	`ToZero`
	`ToZeroInverted`

`MaxValue` [1.0]	Max. value to use with `Binary` and `BinaryInverted` operations

`TrimTransformation` Trim the image.

Option [default value]	Description
`NbLevels`	Number of levels for the color discretization of the image
`Method` [`Discretize`]	Possible values are: `Reduce`: discretization using K-means `Discretize`: simple discretization

N2D2 IP *only* **`WallisFilterTransformation`** Apply Wallis filter to the image.

Option [default value]	Description
`Size`	Size of the filter
`Mean` [0.0]	Target mean value
`StdDev` [1.0]	Target standard deviation
`PerChannel` [0]	If true, apply Wallis filter to each channel individually (this parameter is meaningful only if `Size` is 0)

3.5 Network layers

3.5.1 Layer definition

Common set of parameters for any kind of layer.

Option [default value]	Description
`Input`	Name of the section(s) for the input layer(s). Comma separated
`Type`	Type of the layer. Can be any of the type described below
`Model` [`DefaultModel`]	Layer model to use
`ConfigSection` []	Name of the configuration section for layer

To specify that the back-propagated error must be computed at the output of a given layer (generally the last layer, or output layer), one must add a target section named *LayerName*.`Target`:

```
...
[LayerName.Target]
TargetValue=1.0 ; default: 1.0
DefaultValue=0.0 ; default: -1.0
```

3.5.2 Weight fillers

Fillers to initialize weights and biases in the different type of layer.

Usage example:

```
[conv1]
...
WeightsFiller=NormalFiller
WeightsFiller.Mean=0.0
WeightsFiller.StdDev=0.05
...
```

The initial weights distribution for each layer can be checked in the *weights_init* folder, with an example shown in figure 12.

`ConstantFiller` Fill with a constant value.

Option	Description
FillerName.`Value`	Value for the filling

`NormalFiller` Fill with a normal distribution.

Option [default value]	Description
FillerName.`Mean` [0.0]	Mean value of the distribution
FillerName.`StdDev` [1.0]	Standard deviation of the distribution

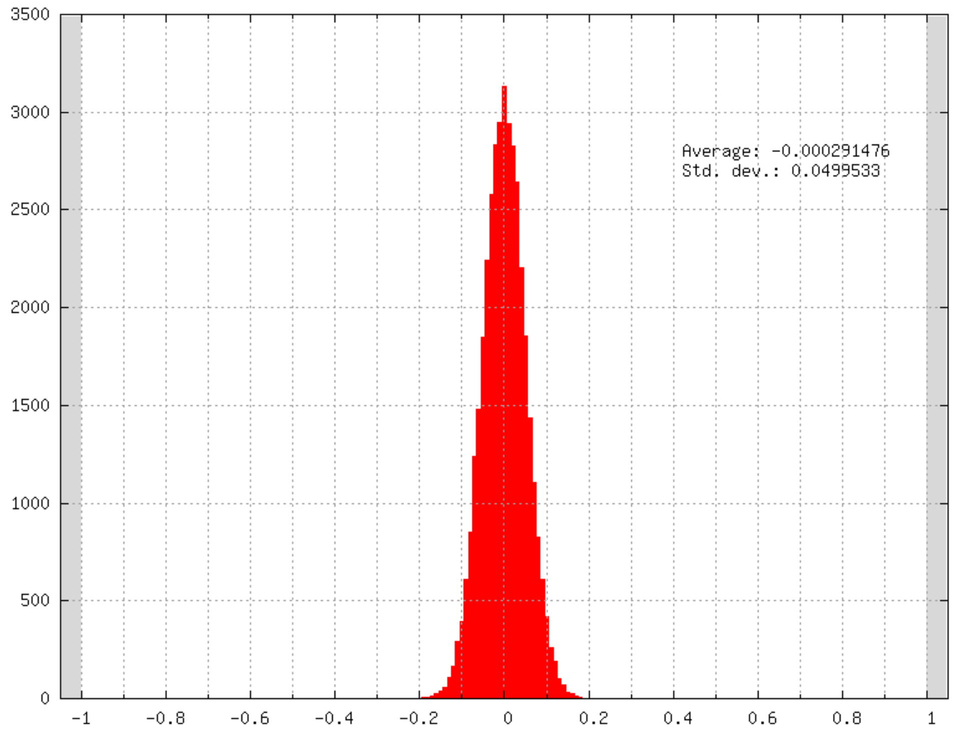


Figure 12: Initial weights distribution of a layer using a normal distribution (`NormalFiller`) with a 0 mean and a 0.05 standard deviation.

`UniformFiller` Fill with an uniform distribution.

Option [default value]	Description
FillerName.`Min` [0.0]	Min. value
FillerName.`Max` [1.0]	Max. value

`XavierFiller` Fill with an uniform distribution with normalized variance (Glorot and Bengio, 2010).

Option [default value]	Description
FillerName.`VarianceNorm` [`FanIn`]	Normalization, can be `FanIn`, `Average` or `FanOut`
FillerName.`Distribution` [`Uniform`]	Distribution, can be `Uniform` or `Normal`

Use an uniform distribution with interval $[-scale, scale]$, with $scale = \sqrt{\frac{3.0}{n}}$.

- $n = \mathit{fan\text{-}in}$ with `FanIn`, resulting in $Var(W) = \frac{1}{\mathit{fan\text{-}in}}$
- $n = \frac{(\mathit{fan\text{-}in}+\mathit{fan\text{-}out})}{2}$ with `Average`, resulting in $Var(W) = \frac{2}{\mathit{fan\text{-}in}+\mathit{fan\text{-}out}}$
- $n = \mathit{fan\text{-}out}$ with `FanOut`, resulting in $Var(W) = \frac{1}{\mathit{fan\text{-}out}}$

3.5.3 Weight solvers

`SGDSolver_Frame` SGD Solver for `Frame` models.

Option [default value]	Description
SolverName.LearningRate [0.01]	Learning rate
SolverName.Momentum [0.0]	Momentum
SolverName.Decay [0.0]	Decay
SolverName.LearningRatePolicy [None]	Learning rate decay policy. Can be any of None, StepDecay, ExponentialDecay, InvTDecay, PolyDecay
SolverName.LearningRateStepSize [1]	Learning rate step size (in number of stimuli)
SolverName.LearningRateDecay [0.1]	Learning rate decay
SolverName.Clamping [0]	If true, clamp the weights and bias between -1 and 1
SolverName.Power [0.0]	Polynomial learning rule power parameter
SolverName.MaxIterations [0.0]	Polynomial learning rule maximum number of iterations

The learning rate decay policies are the following:

- StepDecay: every *SolverName*.LearningRateStepSize stimuli, the learning rate is reduced by a factor *SolverName*.LearningRateDecay;
- ExponentialDecay: the learning rate is $\alpha = \alpha_0 \exp(-kt)$, with α_0 the initial learning rate *SolverName*.LearningRate, k the rate decay *SolverName* .LearningRateDecay and t the step number (one step every *SolverName*.LearningRateStepSize stimuli);
- InvTDecay: the learning rate is $\alpha = \alpha_0/(1 + kt)$, with α_0 the initial learning rate *SolverName*.LearningRate, k the rate decay *SolverName*.LearningRateDecay and t the step number (one step every *SolverName*.LearningRateStepSize stimuli).
- InvDecay: the learning rate is $\alpha = \alpha_0 * (1 + kt)^{-n}$, with α_0 the initial learning rate *SolverName*.LearningRate, k the rate decay *SolverName*.LearningRateDecay, t the current iteration and n the power parameter *SolverName*.Power
- PolyDecay: the learning rate is $\alpha = \alpha_0 * (1 - \frac{k}{t})^n$, with α_0 the initial learning rate *SolverName*.LearningRate, k the current iteration, t the maximum number of iteration *SolverName*.MaxIterations and n the power parameter *SolverName*.Power

SGDSolver_Frame_CUDA SGD Solver for Frame_CUDA models.

Option [default value]	Description
SolverName.LearningRate [0.01]	Learning rate
SolverName.Momentum [0.0]	Momentum
SolverName.Decay [0.0]	Decay
SolverName.LearningRatePolicy [None]	Learning rate decay policy. Can be any of None, StepDecay, ExponentialDecay, InvTDecay
SolverName.LearningRateStepSize [1]	Learning rate step size (in number of stimuli)
SolverName.LearningRateDecay [0.1]	Learning rate decay
SolverName.Clamping [0]	If true, clamp the weights and bias between -1 and 1

The learning rate decay policies are identical to the ones in the `SGDSolver\_Frame` solver.

3.5.4 Activation functions

Activation function to be used at the output of layers.

Usage example:

```
[conv1]
...
ActivationFunction=Rectifier
ActivationFunction.LeakSlope=0.01
ActivationFunction.Clipping=20
...
```

`Logistic` Logistic activation function.

`LogisticWithLoss` Logistic with loss activation function.

`Rectifier` Rectifier or ReLU activation function.

Option [default value]	Description
`ActivationFunction.LeakSlope` [0.0]	Leak slope for negative inputs
`ActivationFunction.Clipping` [0.0]	Clipping value for positive outputs

`Saturation` Saturation activation function.

`Softplus` Softplus activation function.

`Tanh` Tanh activation function.

Computes $y = tanh(\alpha x)$.

Option [default value]	Description
`ActivationFunction.Alpha` [1.0]	α parameter

`TanhLeCun` Tanh activation function with an α parameter of $1.7159 \times (2.0/3.0)$.

3.5.5 `Conv`

Convolutional layer.

Option [default value]	Description
`KernelWidth`	Width of the kernel
`KernelHeight`	Height of the kernel
`NbChannels`	Number of output channels
`SubSampleX` [1]	X-axis subsampling factor of the output feature maps
`SubSampleY` [1]	Y-axis subsampling factor of the output feature maps
`SubSample` [1]	Subsampling factor of the output feature maps (mutually exclusive with `SubSampleX` and `SubSampleY`)
`StrideX` [1]	X-axis stride of the kernels
`StrideY` [1]	Y-axis stride of the kernels

`Stride` [1]	Stride of the kernels (mutually exclusive with `StrideX` and `StrideY`)
`PaddingX` [0]	X-axis input padding
`PaddingY` [0]	Y-axis input padding
`Padding` [0]	Input padding (mutually exclusive with `PaddingX` and `PaddingY`)
`ActivationFunction` [`Tanh`]	Activation function. Can be any of `Logistic`, `LogisticWithLoss`, `Rectifier`, `Softplus`, `TanhLeCun`, `Linear`, `Saturation` or `Tanh`
`WeightsFiller` [`NormalFiller(0.0, 0.05)`]	Weights initial values filler
`BiasFiller` [`NormalFiller(0.0, 0.05)`]	Biases initial values filler
`Mapping.SizeX` [1]	Mapping canvas pattern default width
`Mapping.SizeY` [1]	Mapping canvas pattern default height
`Mapping.Size` [1]	Mapping canvas pattern default size (mutually exclusive with `Mapping.SizeX` and `Mapping.SizeY`)
`Mapping.StrideX` [1]	Mapping canvas default X-axis step
`Mapping.StrideY` [1]	Mapping canvas default Y-axis step
`Mapping.Stride` [1]	Mapping canvas default step (mutually exclusive with `Mapping.StrideX` and `Mapping.StrideY`)
`Mapping.OffsetX` [0]	Mapping canvas default X-axis offset
`Mapping.OffsetY` [0]	Mapping canvas default Y-axis offset
`Mapping.Offset` [0]	Mapping canvas default offset (mutually exclusive with `Mapping.OffsetX` and `Mapping.OffsetY`)
`Mapping.NbIterations` [0]	Mapping canvas pattern default number of iterations (0 means no limit)
`Mapping(in).SizeX` [1]	Mapping canvas pattern default width for input layer `in`
`Mapping(in).SizeY` [1]	Mapping canvas pattern default height for input layer `in`
`Mapping(in).Size` [1]	Mapping canvas pattern default size for input layer `in` (mutually exclusive with `Mapping(in).SizeX` and `Mapping(in).SizeY`)
`Mapping(in).StrideX` [1]	Mapping canvas default X-axis step for input layer `in`
`Mapping(in).StrideY` [1]	Mapping canvas default Y-axis step for input layer `in`
`Mapping(in).Stride` [1]	Mapping canvas default step for input layer `in` (mutually exclusive with `Mapping(in).StrideX` and `Mapping(in).StrideY`)
`Mapping(in).OffsetX` [0]	Mapping canvas default X-axis offset for input layer `in`
`Mapping(in).OffsetY` [0]	Mapping canvas default Y-axis offset for input layer `in`
`Mapping(in).Offset` [0]	Mapping canvas default offset for input layer `in` (mutually exclusive with `Mapping(in).OffsetX` and `Mapping(in).OffsetY`)
`Mapping(in).NbIterations` [0]	Mapping canvas pattern default number of iterations for input layer `in` (0 means no limit)

Configuration parameters (*Frame* models)

Option [default value]	Model(s)	Description
`NoBias` [1]	*all Frame*	If true, don't use bias
`Solvers.*`	*all Frame*	Any solver parameters

WeightsSolver.*	*all Frame*	Weights solver parameters, take precedence over the Solvers.* parameters
BiasSolver.*	*all Frame*	Bias solver parameters, take precedence over the Solvers.* parameters

Configuration parameters (*Spike* models)

Experimental option (implementation may be wrong or susceptible to change)

Option [default value]	Model(s)	Description
IncomingDelay [1 TimePs ;100 TimeFs]	*all Spike*	Synaptic incoming delay w_{delay}
Threshold [1.0]	Spike, Spike_RRAM	Threshold of the neuron I_{thres}
BipolarThreshold [1]	Spike, Spike_RRAM	If true, the threshold is also applied to the absolute value of negative values (generating negative spikes)
Leak [0.0]	Spike, Spike_RRAM	Neural leak time constant τ_{leak} (if 0, no leak)
Refractory [0.0]	Spike, Spike_RRAM	Neural refractory period T_{refrac}
WeightsRelInit [0.0;0.05]	Spike	Relative initial synaptic weight w_{init}
WeightsMinMean [1;0.1]	Spike_RRAM	Mean minimum synaptic weight w_{min}
WeightsMaxMean [100;10.0]	Spike_RRAM	Mean maximum synaptic weight w_{max}
WeightsMinVarSlope [0.0]	Spike_RRAM	OXRAM specific parameter
WeightsMinVarOrigin [0.0]	Spike_RRAM	OXRAM specific parameter
WeightsMaxVarSlope [0.0]	Spike_RRAM	OXRAM specific parameter
WeightsMaxVarOrigin [0.0]	Spike_RRAM	OXRAM specific parameter
WeightsSetProba [1.0]	Spike_RRAM	Intrinsic SET switching probability P_{SET} (upon receiving a SET programming pulse). Assuming uniform statistical distribution (not well supported by experiments on RRAM)
WeightsResetProba [1.0]	Spike_RRAM	Intrinsic RESET switching probability P_{RESET} (upon receiving a RESET programming pulse). Assuming uniform statistical distribution (not well supported by experiments on RRAM)
SynapticRedundancy [1]	Spike_RRAM	Synaptic redundancy (number of RRAM device per synapse)
BipolarWeights [0]	Spike_RRAM	Bipolar weights
BipolarIntegration [0]	Spike_RRAM	Bipolar integration
LtpProba [0.2]	Spike_RRAM	Extrinsic STDP LTP probability (cumulative with intrinsic SET switching probability P_{SET})
LtdProba [0.1]	Spike_RRAM	Extrinsic STDP LTD probability (cumulative with intrinsic RESET switching probability P_{RESET})
StdpLtp [1000 TimePs]	Spike_RRAM	STDP LTP time window T_{LTP}
InhibitRefractory [0 TimePs]	Spike_RRAM	Neural lateral inhibition period $T_{inhibit}$
EnableStdp [1]	Spike_RRAM	If false, STDP is disabled (no synaptic weight change)
RefractoryIntegration [1]	Spike_RRAM	If true, reset the integration to 0 during the refractory period
DigitalIntegration [0]	Spike_RRAM	If false, the analog value of the devices is integrated, instead of their binary value

3.5.6 Deconv

Deconvolutionlayer.

Option [default value]	Description
KernelWidth	Width of the kernel
KernelHeight	Height of the kernel
NbChannels	Number of output channels
StrideX [1]	X-axis stride of the kernels
StrideY [1]	Y-axis stride of the kernels
Stride [1]	Stride of the kernels (mutually exclusive with StrideX and StrideY)
PaddingX [0]	X-axis input padding
PaddingY [0]	Y-axis input padding
Padding [0]	Input padding (mutually exclusive with PaddingX and PaddingY)
ActivationFunction [Tanh]	Activation function. Can be any of Logistic, LogisticWithLoss, Rectifier, Softplus, TanhLeCun, Linear, Saturation or Tanh
WeightsFiller [NormalFiller(0.0, 0.05)]	Weights initial values filler
BiasFiller [NormalFiller(0.0, 0.05)]	Biases initial values filler
Mapping.SizeX [1]	Mapping canvas pattern default width
Mapping.SizeY [1]	Mapping canvas pattern default height
Mapping.Size [1]	Mapping canvas pattern default size (mutually exclusive with Mapping.SizeX and Mapping.SizeY)
Mapping.StrideX [1]	Mapping canvas default X-axis step
Mapping.StrideY [1]	Mapping canvas default Y-axis step
Mapping.Stride [1]	Mapping canvas default step (mutually exclusive with Mapping.StrideX and Mapping.StrideY)
Mapping.OffsetX [0]	Mapping canvas default X-axis offset
Mapping.OffsetY [0]	Mapping canvas default Y-axis offset
Mapping.Offset [0]	Mapping canvas default offset (mutually exclusive with Mapping.OffsetX and Mapping.OffsetY)
Mapping.NbIterations [0]	Mapping canvas pattern default number of iterations (0 means no limit)
Mapping(in).SizeX [1]	Mapping canvas pattern default width for input layer in
Mapping(in).SizeY [1]	Mapping canvas pattern default height for input layer in
Mapping(in).Size [1]	Mapping canvas pattern default size for input layer in (mutually exclusive with Mapping(in).SizeX and Mapping(in).SizeY)
Mapping(in).StrideX [1]	Mapping canvas default X-axis step for input layer in
Mapping(in).StrideY [1]	Mapping canvas default Y-axis step for input layer in
Mapping(in).Stride [1]	Mapping canvas default step for input layer in (mutually exclusive with Mapping(in).StrideX and Mapping(in).StrideY)
Mapping(in).OffsetX [0]	Mapping canvas default X-axis offset for input layer in
Mapping(in).OffsetY [0]	Mapping canvas default Y-axis offset for input layer in
Mapping(in).Offset [0]	Mapping canvas default offset for input layer in (mutually exclusive with Mapping(in).OffsetX and

	Mapping(in).OffsetY)
Mapping(in).NbIterations [0]	Mapping canvas pattern default number of iterations for input layer in (0 means no limit)

Configuration parameters (*Frame* models)

Option [default value]	Model(s)	Description
NoBias [1]	*all Frame*	If true, don't use bias
BackPropagate [1]	*all Frame*	If true, enable backpropogation
Solvers.*	*all Frame*	Any solver parameters
WeightsSolver.*	*all Frame*	Weights solver parameters, take precedence over the Solvers.* parameters
BiasSolver.*	*all Frame*	Bias solver parameters, take precedence over the Solvers.* parameters

3.5.7 Pool

Pooling layer.

Option [default value]	Description
PoolWidth	Width of the pooling area
PoolHeight	Height of the pooling area
NbChannels	Number of output channels
StrideX [1]	X-axis stride of the pooling areas
StrideY [1]	Y-axis stride of the pooling areas
Stride [1]	Stride of the pooling areas (mutually exclusive with StrideX and StrideY)
PaddingX [0]	X-axis input padding
PaddingY [0]	Y-axis input padding
Padding [0]	Input padding
ActivationFunction [Linear]	Activation function. Can be any of Logistic, LogisticWithLoss, Rectifier, Softplus, TanhLeCun, Linear, Saturation or Tanh
Mapping.SizeX [1]	Mapping canvas pattern default width
Mapping.SizeY [1]	Mapping canvas pattern default height
Mapping.Size [1]	Mapping canvas pattern default size (mutually exclusive with Mapping.SizeX and Mapping.SizeY)
Mapping.StrideX [1]	Mapping canvas default X-axis step
Mapping.StrideY [1]	Mapping canvas default Y-axis step
Mapping.Stride [1]	Mapping canvas default step (mutually exclusive with Mapping.StrideX and Mapping.StrideY)
Mapping.OffsetX [0]	Mapping canvas default X-axis offset
Mapping.OffsetY [0]	Mapping canvas default Y-axis offset
Mapping.Offset [0]	Mapping canvas default offset (mutually exclusive with Mapping.OffsetX and Mapping.OffsetY)
Mapping.NbIterations [0]	Mapping canvas pattern default number of iterations (0 means no limit)
Mapping(in).SizeX [1]	Mapping canvas pattern default width for input layer in

`Mapping(in).SizeY` [1]	Mapping canvas pattern default height for input layer `in`
`Mapping(in).Size` [1]	Mapping canvas pattern default size for input layer `in` (mutually exclusive with `Mapping(in).SizeX` and `Mapping(in).SizeY`)
`Mapping(in).StrideX` [1]	Mapping canvas default X-axis step for input layer `in`
`Mapping(in).StrideY` [1]	Mapping canvas default Y-axis step for input layer `in`
`Mapping(in).Stride` [1]	Mapping canvas default step for input layer `in` (mutually exclusive with `Mapping(in).StrideX` and `Mapping(in).StrideY`)
`Mapping(in).OffsetX` [0]	Mapping canvas default X-axis offset for input layer `in`
`Mapping(in).OffsetY` [0]	Mapping canvas default Y-axis offset for input layer `in`
`Mapping(in).Offset` [0]	Mapping canvas default offset for input layer `in` (mutually exclusive with `Mapping(in).OffsetX` and `Mapping(in).OffsetY`)
`Mapping(in).NbIterations` [0]	Mapping canvas pattern default number of iterations for input layer `in` (0 means no limit)

Configuration parameters (*Spike* models)

Option [default value]	Model(s)	Description
`IncomingDelay` [1 `TimePs` ;100 `TimeFs`] value	*all Spike*	Synaptic incoming delay w_{delay}

3.5.8 `FMP`

Fractional max pooling layer (Graham, 2014).

Option [default value]	Description
`NbChannels`	Number of output channels
`ScalingRatio`	Scaling ratio. The output size is *round* $\left(\frac{\text{input size}}{\text{scaling ratio}}\right)$.
`ActivationFunction` [`Linear`]	Activation function. Can be any of `Logistic`, `LogisticWithLoss`, `Rectifier`, `Softplus`, `TanhLeCun`, `Linear`, `Saturation` or `Tanh`

Configuration parameters (*Frame* models)

Option [default value]	Model(s)	Description
`Overlapping` [1]	*all Frame*	If true, use overlapping regions, else use disjoint regions
`PseudoRandom` [1]	*all Frame*	If true, use pseudorandom sequences, else use random sequences

3.5.9 Fc

Fully connected layer.

Option [default value]	Description
NbOutputs	Number of output neurons
WeightsFiller [NormalFiller(0.0, 0.05)]	Weights initial values filler
BiasFiller [NormalFiller(0.0, 0.05)]	Biases initial values filler

Configuration parameters (*Frame* models)

Option [default value]	Model(s)	Description
NoBias [1]	*all Frame*	If true, don't use bias
BackPropagate [1]	*all Frame*	If true, enable backpropogation
Solvers.*	*all Frame*	Any solver parameters
WeightsSolver.*	*all Frame*	Weights solver parameters, take precedence over the Solvers.* parameters
BiasSolver.*	*all Frame*	Bias solver parameters, take precedence over the Solvers.* parameters
DropConnect [1.0]	Frame	If below 1.0, fraction of synapses that are disabled with drop connect

Configuration parameters (*Spike* models)

Option [default value]	Model(s)	Description
IncomingDelay [1 TimePs ;100 TimeFs]	*all Spike*	Synaptic incoming delay w_{delay}
Threshold [1.0]	Spike, Spike_RRAM	Threshold of the neuron I_{thres}
BipolarThreshold [1]	Spike, Spike_RRAM	If true, the threshold is also applied to the absolute value of negative values (generating negative spikes)
Leak [0.0]	Spike, Spike_RRAM	Neural leak time constant τ_{leak} (if 0, no leak)
Refractory [0.0]	Spike, Spike_RRAM	Neural refractory period T_{refrac}
TerminateDelta [0]	Spike, Spike_RRAM	Terminate delta
WeightsRelInit [0.0;0.05]	Spike	Relative initial synaptic weight w_{init}
WeightsMinMean [1;0.1]	Spike_RRAM	Mean minimum synaptic weight w_{min}
WeightsMaxMean [100;10.0]	Spike_RRAM	Mean maximum synaptic weight w_{max}
WeightsMinVarSlope [0.0]	Spike_RRAM	OXRAM specific parameter
WeightsMinVarOrigin [0.0]	Spike_RRAM	OXRAM specific parameter
WeightsMaxVarSlope [0.0]	Spike_RRAM	OXRAM specific parameter
WeightsMaxVarOrigin [0.0]	Spike_RRAM	OXRAM specific parameter
WeightsSetProba [1.0]	Spike_RRAM	Intrinsic SET switching probability P_{SET} (upon receiving a SET programming pulse). Assuming uniform statistical distribution (not well supported by experiments on RRAM)

WeightsResetProba [1.0]	Spike_RRAM	Intrinsic RESET switching probability P_{RESET} (upon receiving a RESET programming pulse). Assuming uniform statistical distribution (not well supported by experiments on RRAM)
SynapticRedundancy [1]	Spike_RRAM	Synaptic redundancy (number of RRAM device per synapse)
BipolarWeights [0]	Spike_RRAM	Bipolar weights
BipolarIntegration [0]	Spike_RRAM	Bipolar integration
LtpProba [0.2]	Spike_RRAM	Extrinsic STDP LTP probability (cumulative with intrinsic SET switching probability P_{SET})
LtdProba [0.1]	Spike_RRAM	Extrinsic STDP LTD probability (cumulative with intrinsic RESET switching probability P_{RESET})
StdpLtp [1000 TimePs]	Spike_RRAM	STDP LTP time window T_{LTP}
InhibitRefractory [0 TimePs]	Spike_RRAM	Neural lateral inhibition period $T_{inhibit}$
EnableStdp [1]	Spike_RRAM	If false, STDP is disabled (no synaptic weight change)
RefractoryIntegration [1]	Spike_RRAM	If true, reset the integration to 0 during the refractory period
DigitalIntegration [0]	Spike_RRAM	If false, the analog value of the devices is integrated, instead of their binary value

N2D2 IP *only*

3.5.10 Rbf

Radial basis function fully connected layer.

Option [default value]	Description
NbOutputs	Number of output neurons
CentersFiller [NormalFiller(0.5, 0.05)]	Centers initial values filler
ScalingFiller [NormalFiller(10.0, 0.05)]	Scaling initial values filler

Configuration parameters (*Frame* models)

Option [default value]	Model(s)	Description
Solvers.*	*all Frame*	Any solver parameters
CentersSolver.*	*all Frame*	Centers solver parameters, take precedence over the Solvers.* parameters
ScalingSolver.*	*all Frame*	Scaling solver parameters, take precedence over the Solvers.* parameters
RbfApprox [None]	Frame	Approximation for the Gaussian function, can be any of: None, Rectangular or SemiLinear

3.5.11 `Softmax`

Softmax layer.

Option [default value]	Description
`NbOutputs`	Number of output neurons
`WithLoss` [0]	Softmax followed with a multinomial logistic layer

The softmax function performs the following operation, with $a^i_{x,y}$ and $b^i_{x,y}$ the input and the output respectively at position (x, y) on channel i:

$$b^i_{x,y} = \frac{\exp(a^i_{x,y})}{\sum\limits_{j=0}^{N} \exp(a^j_{x,y})}$$

and

$$\mathrm{d}a^i_{x,y} = \sum_{j=0}^{N} \left(\delta_{ij} - a^i_{x,y}\right) a^j_{x,y} \mathrm{d}b^j_{x,y}$$

When the `WithLoss` option is enabled, compute the gradient directly in respect of the cross-entropy loss:

$$L_{x,y} = \sum_{j=0}^{N} t^j_{x,y} \log(b^j_{x,y})$$

In this case, the gradient output becomes:

$$\mathrm{d}a^i_{x,y} = \mathrm{d}b^i_{x,y}$$

with

$$\mathrm{d}b^i_{x,y} = t^i_{x,y} - b^i_{x,y}$$

3.5.12 `LRN`

Local Response Normalization (LRN) layer.

Option [default value]	Description
`NbOutputs`	Number of output neurons

The response-normalized activity $b^i_{x,y}$ is given by the expression:

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum\limits_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} \left(a^j_{x,y}\right)^2\right)^{\beta}}$$

Configuration parameters (*Frame* models)

Option [default value]	Model(s)	Description
`N` [5]	`all Frame`	Normalization window width in elements
`Alpha` [1.0e-4]	`all Frame`	Value of the alpha variance scaling parameter in the normalization formula
`Beta` [0.75]	`all Frame`	Value of the beta power parameter in the normalization formula
`K` [2.0]	`all Frame`	Value of the k parameter in normalization formula

3.5.13 `Dropout`

Dropout layer (Srivastava et al., 2012).

Option [default value]	Description
`NbOutputs`	Number of output neurons

Configuration parameters (*Frame* models)

Option [default value]	Model(s)	Description
`Dropout` [0.5]	`Frame_CUDA`	The probability with which the value from input would be dropped

3.5.14 `BatchNorm`

Batch Normalization layer (Ioffe and Szegedy, 2015).

Option [default value]	Description
`NbOutputs`	Number of output neurons
`ActivationFunction` [`Tanh`]	Activation function. Can be any of `Logistic`, `LogisticWithLoss`, `Rectifier`, `Softplus`, `TanhLeCun`, `Linear`, `Saturation` or `Tanh`

Configuration parameters (*Frame* models)

Option [default value]	Model(s)	Description
`Solvers.`*	*all Frame*	Any solver parameters
`ScaleSolver.`*	*all Frame*	Scale solver parameters, take precedence over the `Solvers.`* parameters
`BiasSolver.`*	*all Frame*	Bias solver parameters, take precedence over the `Solvers.`* parameters
`Epsilon` [0.0]	*all Frame*	Epsilon value used in the batch normalization formula. If 0.0, automatically choose the minimum possible value.

3.5.15 Transformation

Transformation layer, which can apply any transformation described in 3.4.1. Useful for fully CNN post-processing for example.

Option [default value]	Description
NbOutputs	Number of outputs
Transformation	Name of the transformation to apply

The Transformation options must be placed in the same section.

Usage example for fully CNNs:

```
[post.Transformation-thres]
Input=... ; for example, network's logistic of softmax output layer
NbOutputs=1
Type=Transformation
Transformation=ThresholdTransformation
Operation=ToZero
Threshold=0.75

[post.Transformation-morpho]
Input=post.Transformation-thres
NbOutputs=1
Type=Transformation
Transformation=MorphologyTransformation
Operation=Opening
Size=3
```

4 Tutorials

4.1 Building a classifier neural network

For this tutorial, we will use the classical MNIST handwritten digit dataset. A driver module already exists for this dataset, named `MNIST_IDX_Database`.

To instantiate it, just add the following lines in a new INI file:

```
[database]
Type=MNIST_IDX_Database
Validation=0.2 ; Use 20% of the dataset for validation
```

In order to create a neural network, we first need to define its input, which is declared with a `[sp]` section (*sp* for *StimuliProvider*). In this section, we configure the size of the input and the batch size:

```
[sp]
SizeX=32
SizeY=32
BatchSize=128
```

We can also add pre-processing transformations to the *StimuliProvider*, knowing that the final data size after transformations must match the size declared in the `[sp]` section. Here, we must rescale the MNIST 28x28 images to match the 32x32 network input size.

```
[sp.Transformation_1]
Type=RescaleTransformation
Width=[sp]SizeX
Height=[sp]SizeY
```

Next, we declare the neural network layers. In this example, we reproduced the well-known LeNet network. The first layer is a 5x5 convolutional layer, with 6 channels. Since there is only one input channel, there will be only 6 convolution kernels in this layer.

```
[conv1]
Input=sp
Type=Conv
KernelWidth=5
KernelHeight=5
NbChannels=6
```

The next layer is a 2x2 MAX pooling layer, with a stride of 2 (non-overlapping MAX pooling).

```
[pool1]
Input=conv1
Type=Pool
PoolWidth=2
PoolHeight=2
NbChannels=[conv1]NbChannels
Stride=2
Pooling=Max
Mapping.Size=1 ; One to one connection between input and output channels
```

The next layer is a 5x5 convolutional layer with 16 channels.

```
[conv2]
Input=pool1
Type=Conv
KernelWidth=5
KernelHeight=5
NbChannels=16
```

Note that in LeNet, the `[conv2]` layer is not fully connected to the pooling layer. In N2D2, a custom mapping can be defined for each input connection. The connection of n-th output map to the inputs is defined by the n-th column of the matrix below, where the rows correspond to the inputs.

```
Map(pool1)=\
1 0 0 0 1 1 1 0 0 1 1 1 1 0 1 1 \
1 1 0 0 0 1 1 1 0 0 1 1 1 1 0 1 \
1 1 1 0 0 0 1 1 1 0 0 1 0 1 1 1 \
0 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 \
0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 \
0 0 0 1 1 1 0 0 1 1 1 1 0 1 1 1
```

Another MAX pooling and convolution layer follow:

```
[pool2]
Input=conv2
Type=Pool
PoolWidth=2
PoolHeight=2
NbChannels=[conv2]NbChannels
Stride=2
Pooling=Max
Mapping.Size=1

[conv3]
Input=pool2
Type=Conv
KernelWidth=5
KernelHeight=5
NbChannels=120
```

The network is composed of two fully-connected layers of 84 and 10 neurons respectively:

```
[fc1]
Input=conv3
Type=Fc
NbOutputs=84

[fc2]
Input=fc1
Type=Fc
NbOutputs=10
```

Finally, we use a softmax layer to obtain output classification probabilities and compute the loss function.

```
[softmax]
Input=fc2
Type=Softmax
NbOutputs=[fc2]NbOutputs
WithLoss=1
```

In order to tell N2D2 to compute the error and the classification score on this softmax layer, one must attach a N2D2 *Target* to this layer, with a section with the same name suffixed with `.Target`:

```
[softmax.Target]
```

By default, the activation function for the convolution and the fully-connected layers is the hyperbolic tangent. Because the `[fc2]` layer is fed to a softmax, it should not have any activation function. We can specify it by adding the following line in the `[fc2]` section:

```
[fc2]
...
ActivationFunction=Linear
```

In order to improve further the networks performances, several things can be done:

- **Use ReLU activation functions.** In order to do so, just add the following in the `[conv1]`, `[conv2]`, `[conv3]` and `[fc1]` layer sections:

  ```
  ActivationFunction=Rectifier
  ```

For the ReLU activation function to be effective, the weights must be initialized carefully, in order to avoid dead units that would be stuck in the $]-\infty, 0]$ output range before the ReLU function. In N2D2, one can use a custom `WeightsFiller` for the weights initialization. For the ReLU activation function, a popular and efficient filler is the so-called `XavierFiller` (see the 3.5.2 section for more information):

```
WeightsFiller=XavierFiller
```

- **Use dropout layers.** Dropout is highly effective to improve the network generalization capacity. Here is an example of a dropout layer inserted between the `[fc1]` and `[fc2]` layers:

```
[fc1]
...

[fc1.drop]
Input=fc1
Type=Dropout
NbOutputs=[fc1]NbOutputs

[fc2]
Input=fc1.drop ; Replaces "Input=fc1"
...
```

- **Tune the learning parameters.** You may want to tune the learning rate and other learning parameters depending on the learning problem at hand. In order to do so, you can add a configuration section that can be common (or not) to all the layers. Here is an example of configuration section:

```
[conv1]
...
ConfigSection=common.config

[...]
...

[common.config]
NoBias=1
WeightsSolver.LearningRate=0.05
WeightsSolver.Decay=0.0005
Solvers.LearningRatePolicy=StepDecay
Solvers.LearningRateStepSize=[sp]_EpochSize
Solvers.LearningRateDecay=0.993
Solvers.Clamping=1
```

For more details on the configuration parameters for the `Solver`, see section 3.5.3.

- **Add input distortion.** See for example the `DistortionTransformation` (section 3.4.1).

The complete INI model corresponding to this tutorial can be found in *models/LeNet.ini*.

In order to use CUDA/GPU accelerated learning, the default layer model should be switched to `Frame_CUDA`. You can enable this model by adding the following line at the top of the INI file (before the first section):

```
DefaultModel=Frame_CUDA
```

4.2 Building a segmentation neural network

In this tutorial, we will learn how to do image segmentation with N2D2. As an example, we will implement a face detection and gender recognition neural network, using the IMDB-WIKI dataset.

First, we need to instanciate the IMDB-WIKI dataset built-in N2D2 driver:

```
[database]
Type=IMDBWIKI_Database
```

```
WikiSet=1 ; Use the WIKI part of the dataset
IMDBSet=0 ; Don't use the IMDB part (less accurate annotation)
Learn=0.90
Validation=0.05
DefaultLabel=background ; Label for pixels outside any ROI (default is no label, pixels are
    ignored)
```

We must specify a default label for the background, because we want to learn to differenciate faces from the background (and not simply ignore the background for the learning).

The network input is then declared:

```
[sp]
SizeX=480
SizeY=360
BatchSize=48
CompositeStimuli=1
```

In order to work with segmented data, i.e. data with bounding box annotations or pixel-wise annotations (as opposed to a single label per data), one must enable the `CompositeStimuli` option in the `[sp]` section.

We can then perform various operations on the data before feeding it to the network, like for example converting the 3-channels RGB input images to single-channel gray images:

```
[sp.Transformation-1]
Type=ChannelExtractionTransformation
CSChannel=Gray
```

We must only rescale the images to match the networks input size. This can be done using a `RescaleTransformation`, followed by a `PadCropTransformation` if one want to keep the images aspect ratio.

```
[sp.Transformation-2]
Type=RescaleTransformation
Width=[sp]SizeX
Height=[sp]SizeY
KeepAspectRatio=1 ; Keep images aspect ratio

; Required to ensure all the images are the same size
[sp.Transformation-3]
Type=PadCropTransformation
Width=[sp]SizeX
Height=[sp]SizeY
```

A common additional operation to extend the learning set is to apply random horizontal mirror to images. This can be achieved with the following `FlipTransformation`:

```
[sp.OnTheFlyTransformation-4]
Type=FlipTransformation
RandomHorizontalFlip=1
ApplyTo=LearnOnly ; Apply this transformation only on the learning set
```

Note that this is an *on-the-fly* transformation, meaning it cannot be cached and is re-executed every time even for the same stimuli. We also apply this transformation only on the learning set, with the `ApplyTo` option.

Next, the neural network can be described:

```
[conv1.1]
Input=sp
Type=Conv
...

[pool1]
...

[...]
```

```
...

[fc2]
Input=drop1
Type=Conv
...

[drop2]
Input=fc2
Type=Dropout
NbOutputs=[fc2]NbChannels
```

A full network description can be found in the *IMDBWIKI.ini* file in the *models* directory of N2D2. It is a fully-CNN network.

Here we will focus on the output layers required to detect the faces and classify their gender. We start from the `[drop2]` layer, which has 128 channels of size 60x45.

4.2.1 Faces detection

We want to first add an output stage for the faces detection. It is a 1x1 convolutional layer with a single 60x45 output map. For each output pixel, this layer outputs the probability that the pixel belongs to a face.

```
[fc3.face]
Input=drop2
Type=Conv
KernelWidth=1
KernelHeight=1
NbChannels=1
Stride=1
ActivationFunction=LogisticWithLoss
WeightsFiller=XavierFiller
ConfigSection=common.config ; Same solver options that the other layers
```

In order to do so, the activation function of this layer must be of type `LogisticWithLoss`.

We must also tell N2D2 to compute the error and the classification score on this softmax layer, by attaching a N2D2 *Target* to this layer, with a section with the same name suffixed with `.Target`:

```
[fc3.face.Target]
LabelsMapping=${N2D2_MODELS}/IMDBWIKI_target_face.dat
; Visualization parameters
NoDisplayLabel=0
LabelsHueOffset=90
```

In this *Target*, we must specify how the dataset annotations are mapped to the layer's output. This can be done in a separate file using the `LabelsMapping` parameter. Here, since the output layer has a single output per pixel, the target value can only be 0 or 1. A target value of -1 means that this output is ignored (no error back-propagated). Since the only annotations in the IMDB-WIKI dataset are faces, the mapping described in the *IMDBWIKI_target_face.dat* file is easy:

```
# background
background 0

# padding (*) is ignored (-1)
* -1

# not background = face
default 1
```

4.2.2 Gender recognition

We can also add a second output stage for gender recognition. Like before, it would be a 1x1 convolutional layer with a single 60x45 output map. But here, for each output pixel, this layer would output the probability that the pixel represents a female face.

```
[fc3.gender]
Input=drop2
Type=Conv
KernelWidth=1
KernelHeight=1
NbChannels=1
Stride=1
ActivationFunction=LogisticWithLoss
WeightsFiller=XavierFiller
ConfigSection=common.config
```

The output layer is therefore identical to the face's output layer, but the target mapping is different. For the target mapping, the idea is simply to ignore all pixels not belonging to a face and affect the target 0 to male pixels and the target 1 to female pixels.

```
[fc3.gender.Target]
LabelsMapping=${N2D2_MODELS}/IMDBWIKI_target_gender.dat
; Only display gender probability for pixels detected as face pixels
MaskLabelTarget=fc3.face.Target
MaskedLabel=1
```

The content of the *IMDBWIKI_target_gender.dat* file would therefore look like:

```
# background
# ?-* (unknown gender)
# padding
default -1

# male gender
M-? 0 # unknown age
M-0 0
M-1 0
M-2 0
...
M-98 0
M-99 0

# female gender
F-? 1 # unknown age
F-0 1
F-1 1
F-2 1
...
F-98 1
F-99 1
```

4.2.3 ROIs extraction

The next step would be to extract detected face ROIs and assign for each ROI the most probable gender. To this end, we can first set a detection threshold, in terms of probability, to select face pixels. In the following, the threshold is fixed to 75% face probability:

```
[post.Transformation-thres]
Input=fc3.face
Type=Transformation
NbOutputs=1
Transformation=ThresholdTransformation
Operation=ToZero
Threshold=0.75
```

We can then assign a target of type `TargetROIs` to this layer that will automatically create the bounding box using a segmentation algorithm.

```
[post.Transformation-thres.Target-face]
Type=TargetROIs
MinOverlap=0.33 ; Min. overlap fraction to match the ROI to an annotation
FilterMinWidth=5 ; Min. ROI width
FilterMinHeight=5 ; Min. ROI height
FilterMinAspectRatio=0.5 ; Min. ROI aspect ratio
FilterMaxAspectRatio=1.5 ; Max. ROI aspect ratio
LabelsMapping=${N2D2_MODELS}/IMDBWIKI_target_face.dat
```

In order to assign a gender to the extracted ROIs, the above target must be modified to:

```
[post.Transformation-thres.Target-gender]
Type=TargetROIs
ROIsLabelTarget=fc3.gender.Target
MinOverlap=0.33
FilterMinWidth=5
FilterMinHeight=5
FilterMinAspectRatio=0.5
FilterMaxAspectRatio=1.5
LabelsMapping=${N2D2_MODELS}/IMDBWIKI_target_gender.dat
```

Here, we use the `fc3.gender.Target` target to determine the most probable gender of the ROI.

4.2.4 Data visualization

For each *Target* in the network, a corresponding folder is created in the simulation directory, which contains learning, validation and test confusion matrixes. The output estimation of the network for each stimulus is also generated automatically for the test dataset and can be visualized with the *./test.py* helper tool. An example is shown in figure 13.

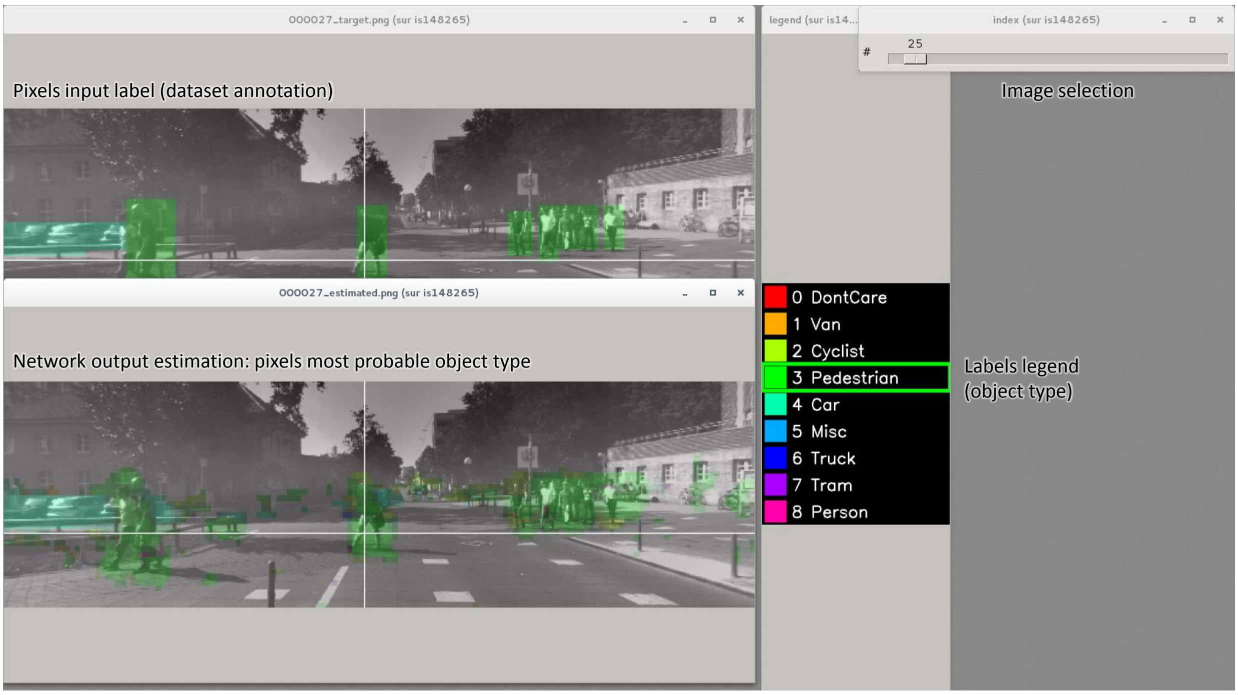


Figure 13: Example of the target visualization helper tool.

4.3 Transcoding a learned network in spike-coding

N2D2 embeds an event-based simulator (historically known as 'Xnet') and allows to transcode a whole DNN in a spike-coding version and evaluate the resulting spiking neural network performances. In this tutorial, we will transcode the LeNet network described in section 4.1.

4.3.1 Render the network compatible with spike simulations

The first step is to specify that we want to use a transcode model (allowing both formal and spike simulation of the same network), by changing the `DefaultModel` to:

```
DefaultModel=Transcode_CUDA
```

In order to perform spike simulations, the input of the network must be of type *Environment*, which is a derived class of *StimuliProvider* that adds spike coding support. In the INI model file, it is therefore necessary to replace the `[sp]` section by an `[env]` section and replace all references of `sp` to `env`.

Note that these changes have at this point no impact at all on the formal coding simulations. The beginning of the INI file should be:

```
DefaultModel=Transcode_CUDA

; Database
[database]
Type=MNIST_IDX_Database
Validation=0.2 ; Use 20% of the dataset for validation

; Environment
[env]
SizeX=32
SizeY=32
BatchSize=128

[env.Transformation_1]
Type=RescaleTransformation
Width=[env]SizeX
Height=[env]SizeY

[conv1]
Input=env
...
```

The dropout layer has no equivalence in spike-coding inference and must be removed:

```
...
[fc1.drop]
Input=fc1
Type=Dropout
NbOutputs=[fc1]NbOutputs

[fc2]
Input=fc1.drop
...
```

The softmax layer has no equivalence in spike-coding inference and must be removed as well. The *Target* must therefore be attached to `[fc2]`:

```
...
[softmax]
Input=fc2
Type=Softmax
NbOutputs=[fc2]NbOutputs
WithLoss=1
```

```
~~[softmax.Target]~~

[fc2.Target]
...
```

The network is now compatible with spike-coding simulations. However, we did not specify at this point how to translate the input stimuli data into spikes, nor the spiking neuron parameters (threshold value, leak time constant...).

4.3.2 Configure spike-coding parameters

The first step is to configure how the input stimuli data must be coded into spikes. To this end, we must attach a configuration section to the *Environment*. Here, we specify a periodic coding with random initial jitter with a minimum period of 10 ns and a maximum period of 100 us:

```
[env]
...
ConfigSection=env.config

[env.config]
; Spike-based computing
StimulusType=JitteredPeriodic
PeriodMin=1,000,000 ; unit = fs
PeriodMeanMin=10,000,000 ; unit = fs
PeriodMeanMax=100,000,000,000 ; unit = fs
PeriodRelStdDev=0.0
```

The next step is to specify the neurons parameters, that will be common to all layers and can therefore be specified in the `[common.config]` section. In N2D2, the base spike-coding layers use a Leaky Integrate-and-Fire (LIF) neuron model. By default, the leak time constant is zero, resulting to simple Integrate-and-Fire (IF) neurons.

Here we simply specify that the neurons threshold must be the unity, that the threshold is only positive and that there is no incoming synaptic delay:

```
[common.config]
...
; Spike-based computing
Threshold=1.0
BipolarThreshold=0
IncomingDelay=0
```

Finally, we can limit the number of spikes required for the computation of each stimulus by adding a decision delta threshold at the output layer:

```
[fc2]
...
ConfigSection=common.config,fc2.config

[fc2.Target]

[fc2.config]
; Spike-based computing
TerminateDelta=4
BipolarThreshold=1
```

The complete INI model corresponding to this tutorial can be found in *models/LeNet_Spike.ini*.

Here is a summary of the steps required to reproduce the whole experiment:

```
./n2d2 "$N2D2_MODELS/LeNet.ini" -learn 6000000 -log 100000
./n2d2 "$N2D2_MODELS/LeNet_Spike.ini" -test
```

The final recognition rate reported at the end of the spike inference should be almost identical to the formal coding network (around 99% for the LeNet network).

Various statistics are available at the end of the spike-coding simulation in the *stats_spike* folder and the *stats_spike.log* file. Looking in the *stats_spike.log* file, one can read the following line towards the end of the file:

```
Read events per virtual synapse per pattern (average): 0.654124
```

This line reports the average number of accumulation operations per synapse per input stimulus in the network. If this number if below 1.0, it means that the spiking version of the network is more efficient than its formal counterpart in terms of total number of operations!

References

P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark. In *CVPR*, 2009.

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *IEEE. CVPR 2004, Workshop on Generative-Model Based Vision*, 2004.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, page 249–256, 2010.

B. Graham. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014.

G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset, 2007.

S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

V. Jain and E. Learned-Miller. FDDB: A benchmark for face detection in unconstrained settings, 2010.

A. Krizhevsky. Learning multiple layers of features from tiny images, 2009.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews. The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression. 2010.

A. Rakotomamonjy and G. Gasso. Histogram of gradients of time-frequency representations for audio scene detection, 2014.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from voverfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2012.

J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 2012. ISSN 0893-6080. doi: 10.1016/j.neunet.2012.02.016.