Ref. Ares(2018)5002664 - 30/09/2018

software framework for runtime-Adaptive and secure deep Learning On Heterogeneous Architectures

Project Number 780788

Project Acronym ALOHA

D2.1	Report on automated algorithm configuration		
Work Package:	WP2	**Lead Beneficiary:**	ETHZ
Type:	Report	**Dissemination level:**	**Public**
Due Date:	30th September 2018	**Delivery:**	30th September 2018
Version:	D2.1_Report on automated algorithm configuration_v1.0.docx		

Brief description:

Report on the research and development activities concerning the utilities converging in the ALOHA automated algorithm configuration tool. This report serves also as a lightweight form of documentation for D2.3 (due in M12).

Deliverable Author(s):

Name	Beneficiary
Francesco Conti	ETHZ

Deliverable Revision History:

Reviewer Beneficiary	Issue Date	Version	Comments
ETHZ	07/08/2018	V0.0	Draft of ToC
ETHZ	28/08/2018	V0.1	Added contributions on Section 2
ETHZ	19/09/2018	V0.2	Added contributions on Section 2.4
ETHZ	20/09/2018	V0.3	Added UvA, UL and P-ONE contributions
ETHZ	27/09/2018	V0.4	Final draft
ETHZ	28/09/2018	V0.5	Incorporated review from UNICA
ETHZ	30/09/2018	V1.0	Final version

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

Disclaimer

This document may contain material that is copyright of certain ALOHA beneficiaries, and may not be reproduced, copied, or modified in whole or in part for any purpose without written permission from the ALOHA Consortium. The commercial use of any information contained in this document may require a license from the proprietor of that information. The information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

The ALOHA Consortium is the following:

#	Participant Legal Name	Acronym	Country
1	STMICROELECTRONICS SRL	ST-I	Italy
2	UNIVERSITA' DEGLI STUDI DI CAGLIARI	UNICA	Italy
3	UNIVERSITEIT VAN AMSTERDAM	UVA	Netherlands
4	UNIVERSITEIT LEIDEN	UL	Netherlands
5	EIDGENOESSISCHE TECHNISCHE HOCHSCHULE ZUERICH	ETHZ	Switzerland
6	UNIVERSITA' DEGLI STUDI DI SASSARI	UNISS	Italy
7	PKE ELECTRONICS AG	PKE	Austria
8	CA TECHNOLOGIES DEVELOPMENT SPAIN SA	CA	Spain
9	SOFTWARE COMPETENCE CENTER HAGENBERG GMBH	SCCH	Austria
10	SANTER REPLY SPA	REPLY	Italy
11	IBM ISRAEL - SCIENCE AND TECHNOLOGY LTD	IBM	Israel
12	SYSTHMATA YPOLOGISTIKIS ORASHS IRIDA LABS AE	IL	Greece
13	PLURIBUS ONE SRL	P-ONE	Italy
14	MAXQ ARTIFICIAL INTELLIGENCE, LTD (formerly MEDYMATCH TECHNOLOGY, LTD)	MaxQ-AI (formerly MM)	Israel

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

Table of Contents

1 Executive Summary 6
1.1 Acronyms and abbreviations 6
2 Automated algorithm configuration sub-modules 7
2.1 Design-Space Exploration engine 7
2.2 Training Engine 8
2.3 Security evaluation engine 8
2.3.1 Detection of dataset bias 10
2.4 Algorithm configuration refinement for parsimonious inference 12
2.5 Algorithm performance / energy evaluation 13
2.5.1 Model extraction 14
2.5.2 Internal Model Conversion 14
2.5.3 Model Evaluation 16
3 Preliminary integration activities 17
4 References 18

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

Figures

Figure 1: Workflow to prune design space using a Genetic Algorithm in the Explorer module 7
Figure 2: Example instantiation of a genotype 8
Figure 3: Security evaluation of the considered convolutional neural network, and of the robust network trained with adversarial training, under the FGSM attack with increasing perturbation ε. 9
Figure 4: Manipulated MNIST handwritten digits that mislead classification of the undefended network, crafted with the FGSM attack algorithm (for ε = 0.05). Note that, within this setting, the adversarial perturbations are almost imperceptible to the human eye, though still effective to mislead recognition ... 10
Figure 5: Workflow of the tool created to assess bias and its impact measurements given a DL model and the dataset used to train the model 11
Figure 6: Cumulative test loss on CIFAR-10 for VGG-16 fine-tuning 12
Figure 7: Performance/power evaluation component 13
Figure 8: Internal DNN model representation 14
Figure 9: CSDF graph topology generation 15
Figure 10: DNN-to-CSDF conversion example 16

Tables

Table 1: NEURAghe-adapted DNNs 8
Table 2: parameters for the Quantize/QNN transformation of the RPI tool 13

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

1 Executive Summary

This report is related to the activities carried out within Task 2.1, 2.2, 2.3, 2.4 and targets the research and development activities carried out within these tasks. It acts as a lightweight specification of the preliminary version of the algorithm configuration tool.

Since the overall ALOHA integration methodology is built upon an iterative and continuous approach, the interactions among different components of the architecture are subject to possible modifications in future iterations.

ST-I, UNICA, UVA, UL, ETHZ, CA, SCCH, P-ONE have contributed to this deliverable.

This report relates to the following future deliverables:

- D2.2, which will update the current content to reflect the latest developments at M18;
- D2.3 and D2.4, which will include a first and a final release of the automated algorithm configuration tool initially outlined in this deliverable.

1.1 Acronyms and abbreviations

Acronym	Meaning
GA	Genetic Algorithms
DNN	Deep Neural Network
SEC	Security Evaluation
FGSM	Fast Gradient Sign Method
RPI	Refinement for Parsimonious Inference
DSE	Design Space Exploration
TE	Training Engine

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

2 Automated algorithm configuration sub-modules

2.1 Design-Space Exploration engine

Error! Reference source not found.1 shows the workflow of the proposed design space exploration methodoly based on Genetic Algorithms (GA). The main strategy of GA is to explore a selective, smaller subset of possible neural network topologies and, towards this end, it presents a few selected best solutions which satisfy pre-defined critera. The module is meant to be initiated with a set of DNNs encoded as *genotypes* of an initial population, evaluated on a fitness score. Iteratively, offspring of this population are created through gene-altering operations (crossovers, mutations changing the topology of the DNN algorithm). The mutations iteratively replace the low-scoring solutions, evolving increasingly better generations.

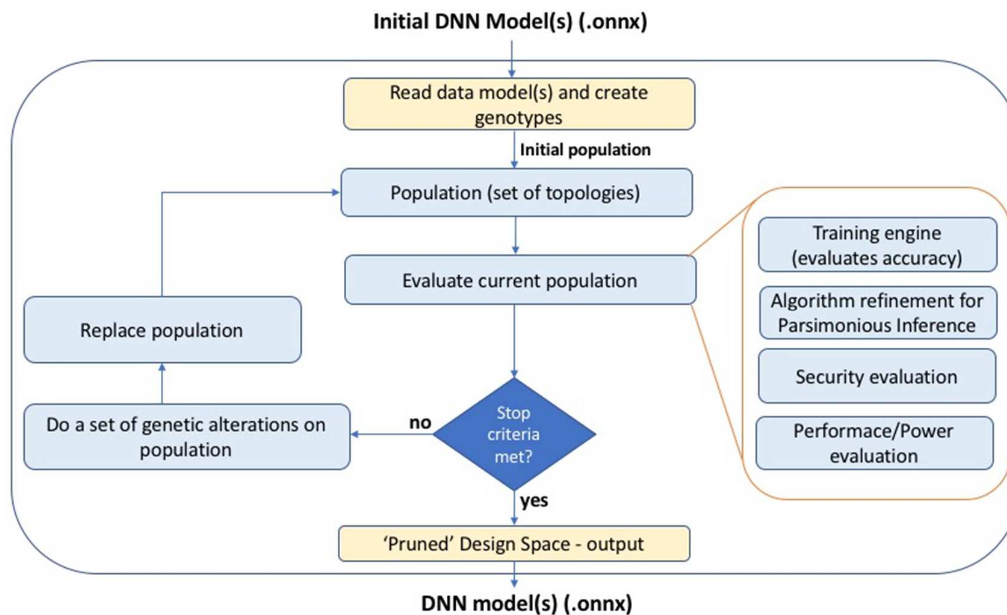

Figure 1: Workflow to prune design space using a Genetic Algorithm in the Explorer module

At this moment, we have a simple GA working, with the loop being completed using the Performance/Power evaluation tool. We start by representing DNNs as genotype, initialization of these genotypes is being done using a basic pattern, where the same type of layers get grouped together with defined constraints. Figure 2 shows a small example of a genotype pattern with Convolution layers followed by pooling layers and then fully connected layers. Input (dataI) and output (dataO) layers are mandatory and there is always only one of them. A convolution layer, in the given example, can each have between 5 and 20 feature maps and there can be 2 to 5 convolutional layers. For VGG, we have another chromosome of "Convolution+Relu layer" which is grouped 2-3 times followed by a maxpool layer. As future work, we will expand the pattern building capacities of our GA tool to achieve other complex topologies of neural networks.

```
Genotype.of(simpleLayerChromosome.of("dataI",1,2),
simpleLayerChromosome.of("Convolution",5,20,IntRange.of(2,5)),
simpleLayerChromosome.of("Pooling",5,20),
simpleLayerChromosome.of("DenseBlock",10,20,IntRange.of(1,5)),
simpleLayerChromosome.of("dataO",1,2) );
```


Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

Figure 2: Example instantiation of a genotype

After running our GA and experimenting with various topology patterns, we found that hardware evaluation (performance and power consumption) is not sufficient to get a reasonable result. With the current setup, where the objective of the GA is set as minimizing power (or maximizing performance), unsurprisingly the GA is geared towards generating the smallest neural network topology with the given criteria. We need to incorporate a conflicting objective to get the GA to expand topologies, which clearly is accuracy in this case.

We are currently exploring the idea where we train the population with a small subset of available data. Complete accuracy calculations can take a long time and make this exploration prohibitive if we fully train every topology under evaluation. To speed up the process of selection of "good" networks for each subsequent iteration, we train all the networks in the population with a small training set, while keeping other hyper parameters such as loss function / learning rate etc. same for all of them.

For this approach to work, we are assuming that accuracy, achieved with a partial training set, is indicative of the "goodness" of any given network. We are not concerned about maximum accuracy achievable by a network, rather only in the prospective potential of the maximum achievable accuracy. This will then allow us to reject those networks that do not show a promise of achieving good accuracy when trained completely. In order to implement this idea, we are currently also working on converting genotype descriptions to the ONNX format, which can then be directly used by python scripts to 'partly' train the network.

2.2 Training Engine

Work on the preliminary version of the Training Engine (TE) has been focused on the definition of the integration with other tools, focusing in particular on synchronizing with reference data sets provided by the ALOHA use-cases, by the coordinated definition of data parsers that can be used by the TE and support the use-case data.

Moreover, we performed some initial experiments with the training of architecture-friendly DNNs. Specifically, we modified the topology of VGG-16 so that it would be better adapted to the architecture of NEURAghe, one of the deployment platforms targeted within the project. To do so, we slightly increased (*VGG16-over*) or decreased (*VGG16-under*) the number of neurons in each layer so that it is perfectly matched to NEURAghe's computation units. Table 1 shows the outcome of this procedure in terms of performance and accuracy, showing how in particular the *VGG16-over* model was able to achieve both higher performance and accuracy.

Table 1: NEURAghe-adapted DNNs

Parameter	Performance	Top-1 accuracy
VGG16	173 Gop/s	88.4%
VGG16-over	182 Gop/s	89.6%
VGG16-under	184 Gop/s	79.7%

2.3 Security evaluation engine

The preliminary version of the Security Evaluation (SEC) engine is built on the open-source PyTorch framework. It will be then deployed in a Docker instance to be fully integrated in the ALOHA toolflow.

The SEC engine allows one to assess the security of deep networks to adversarial examples, i.e., carefully-crafted input data perturbations aimed to mislead detection. This is achieved through the notion of security evaluation curves, i.e., showing how the performance of a trained deep learning model decreases under input data perturbations crafted with an increasing amount of noise (Biggio and Roli, 2018).

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

The SEC engine takes as input: the dataset, the trained network model, and the parameters required to run the evaluation (i.e., the kind and maximum level of perturbation to be applied on the input data). It runs the analysis and then returns the complete security evaluation curve as well as summary statistics (e.g., the average value of the curve) representing a qualitative level of security (to be associated to three categorical levels, namely, low, medium or high security). Notably, the input data modified by the SEC engine can also be used to improve the robustness of the learning model to the considered perturbations, via adversarial training, i.e., by re-training the neural network including such attacks as part of the training data (Goodfellow et al., 2015). To this end, the SEC engine may be required to additionally generate a number of perturbed data points (i.e., adversarial examples) which may be subsequently used by the training engine to augment the training data and improve the security (or robustness) of the network under training.

We report here a clarifying example of application of the SEC engine and of the impact of adversarial training on the network security/robustness. We consider a well-known deep network used for the recognition of MNIST handwritten digits. The architecture of this convolutional neural network consists of different convolutional layers with pooling and ReLU activations and a fully-connected output layer.[1] We trained it on the MNIST training set (consisting of 60, 000 images), after normalizing all images in [0, 1] by dividing the pixel values by 255, and manipulated 10, 000 test samples with the Fast Gradient Sign Method (FGSM) attack algorithm (Goodfellow et al., 2015). This attack bounds the max-norm distance between the source image x and its adversarial counterpart x′ as $\|x - x'\|_\infty \leq \varepsilon$. This means that every pixel p in the image x′ can be manipulated independently in the interval [p−ε, p+ε].

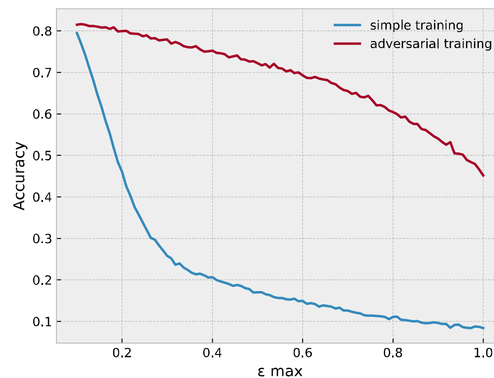

Figure 3: Security evaluation of the considered convolutional neural network, and of the robust network trained with adversarial training, under the FGSM attack with increasing perturbation ε.

The corresponding security evaluation curve is reported in Figure 3 , showing how classification accuracy degrades under attacks characterized by an increasing perturbation ε. Note how the security of the considered network has been substantially increased in this case via adversarial training (the curve corresponding to the network learned with adversarial training decreases more gracefully). In Figure 4, we show some examples of manipulated MNIST handwritten digits, able to mislead classification of the undefended network, along with their corresponding adversarial perturbations (magnified to improve visibility). Adversarial training shows that adding such data points to the training data used to learn the

[1] For further details on the network architecture, see https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

undefended network can improve its security against such kind of perturbations.

It is finally worth remarking that the ALOHA toolflow may also be easily extended to include other state-of-the-art defenses against adversarial examples, such as defenses based on the explicit detection or rejection of such samples, or defenses based on the use of specific hyperparameter configurations, including approaches for properly regularizing the loss function optimized during training (Biggio and Roli, 2018).

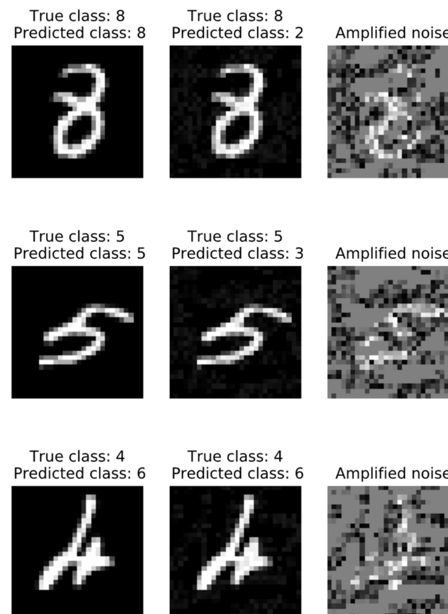

Figure 4: Manipulated MNIST handwritten digits that mislead classification of the undefended network, crafted with the FGSM attack algorithm (for ε = 0.05). Note that, within this setting, the adversarial perturbations are almost imperceptible to the human eye, though still effective to mislead recognition

2.3.1 Detection of dataset bias

Biases might come from different sources and are affected by all the components of any data pipeline in any organization and usually are reflected into data unbalancing, directly affecting those datasets used to extract insights from a given domain.

While adding Machine Learning algorithms to the picture, bias affecting this type of automatic systems is defined as machine bias. Machine bias is created by two main sources: biased learning due to biases present in the training data and biased learning caused by algorithms giving attention to irrelevant or undesired features from the training data. One of the most dangerous risks of using machine biased algorithms in production is discrimination, prohibited by laws such as GPDR (https://eugdpr.org/). Machine biases might also be intentionally created by individuals with malicious purposes, as happens in the case of poisoning attacks, where information is modified or added such as certain types of biases are learned by the algorithm.

In this context, the system created by CA Technologies in the task of security evaluation studies biases in the input data to avoid replicating those biases into machine biases without the knowledge of developers, as it might cause undesired effects if used in production. The tool receives as input a model specified in .ONNX format and the dataset utilized to train the model.

To perform the analysis, classical statistical tests used to detect biases are combined with the notion of features and samples importance in machine learning.

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

As has been widely studied in the machine learning field, some features are more relevant than others for predicting a given target variable. To obtain an approximation of the importance of each feature, a RandomForest model is trained over the same dataset used to train the deep learning model. Once the RandomForest model is trained, a post-processing function is then executed to measure the relative relevance of every feature inside of the model and those values are then aggregated and normalized to obtain a importance score for each feature (Breiman, Friedman, 1984).

Recent state of the art works propose different mechanisms to measure the sample importance for the training of deep learning models. Those techniques allow to obtain a value directly correlated to the impact of each sample during the learning phase. To measure this importance, techniques such as influence functions will be leveraged (Wei P, et al. 2017), (Data A. et al, 2016).

With the idea that unbalances might cause are more dangerous impact when affecting highly relevant features and samples, the proposed tool will combine the scores for data bias, feature importance and data importance to calculate a bias impact score for a given pair [Dataset, Model].

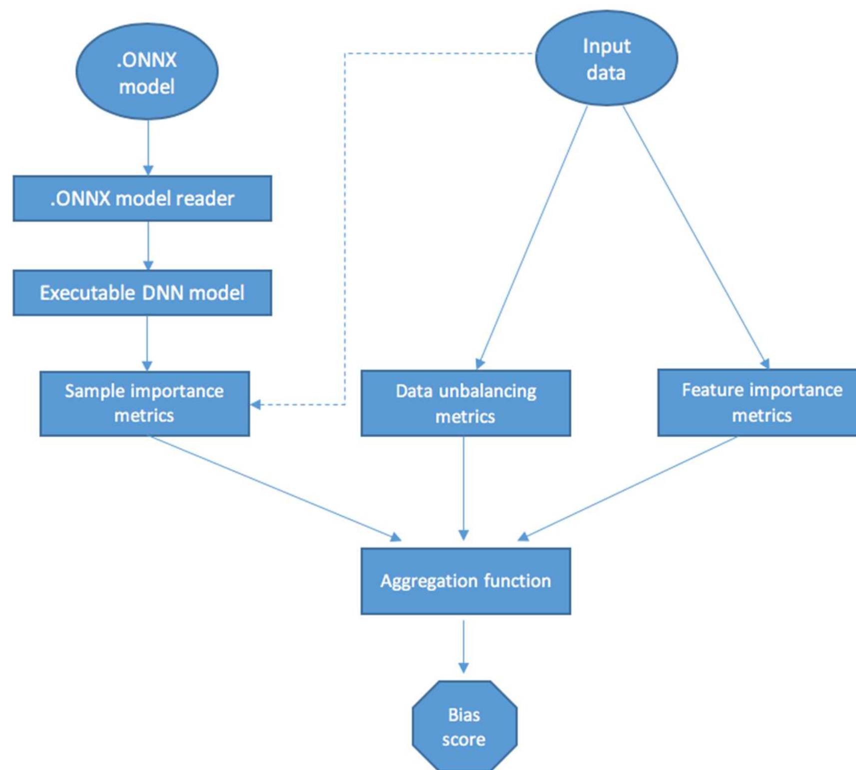

Figure 5: Workflow of the tool created to assess bias and its impact measurements given a DL model and the dataset used to train the model

The system has been developed in Python, taking advantage of its well-known statistical packages, and encapsulated in a Docker image, exposing a REST API to interact with other components of the ALOHA Toolflow. In future developments, it will be integrated more closely with other pieces of the security evaluation component. For this first release, it was decided to keep the components isolated until the development of the complete project reaches a more mature state. The workflow of the created tool is depicted in Figure 5.

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

2.4 Algorithm configuration refinement for parsimonious inference

The preliminary version of the Refinement for Parsimonious Inference (RPI) tool is built on top of the open-source *PyTorch* framework; we used the latest stable version available (0.4) for our initial evaluation, which has focused on the QUANTIZE transformation as described in D1.1 at M6. The RPI tool will be deployed in a Docker instance and finally integrated with the rest of the ALOHA tool flow.

In the ALOHA toolflow, we are defining heuristics that enable quantization to be applied to arbitrary CNN topologies by performing a "local search" for an optimally quantized DNN in the context of the larger automated configuration flow. Only minimal, quantization-related changes to the topology are performed: for example, in order to ensure that activations are typically well-representable by a fixed-point format, we systematically introduce batch normalization after all convolutional layers (even ones that are not originally followed by BN), followed by the activation quantization operator.

As first target, we focused primarily on Quantized Neural Networks (QNNs) as proposed in Hubara *et al.*[2] as a methodology to trade off energy with accuracy. The standard convolutional layer is approximated by representing the weight tensors $\boldsymbol{W}$ and activation tensors $\boldsymbol{X}$ by means of Q-bit integers ($\boldsymbol{W_q}$ and $\boldsymbol{X_q}$ respectively), representing a fixed-point value in the range $[-1;\ +1)$ (closed on the lower side, open on the upper side) with precision $\varepsilon = 2^{-(Q-1)}$.

To reduce arithmetic precision while striving for high final accuracy, the QUANTIZE transformation applies a "relaxation" heuristic which progressively relaxes the value of the precision ε when the total loss / accuracy satisfies a set of external parameters. Figure 6 shows an example of relaxation procedure applied on a VGG-16 topology trained on the CIFAR-10 dataset. The following table reports the main external parameters of the relaxation strategy as well as their value in the example of Figure 6.

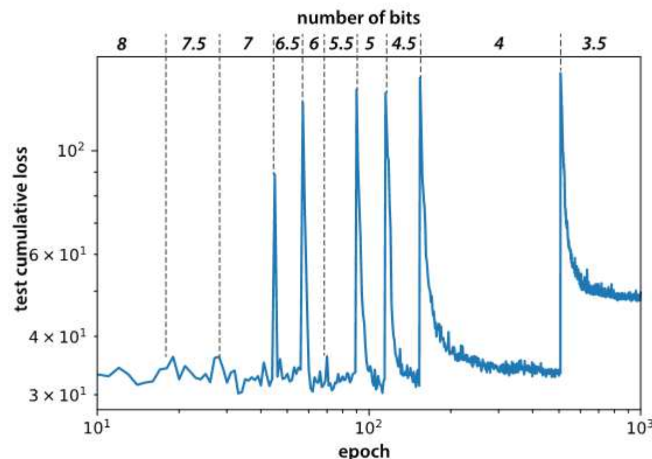

Figure 6: Cumulative test loss on CIFAR-10 for VGG-16 fine-tuning

[2] [1] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," arXiv:1609.07061 [cs], Sep. 2016.

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

Table 2: parameters for the Quantize/QNN transformation of the RPI tool

Parameter	Value	Description
relaxation_factor	$\sqrt{2}$	Factor by which the precision ε is relaxed. ***relaxation_factor*** = $\sqrt{2}$ is equivalent to a precision drop of 0.5 bits.
delta_loss_window	20 epochs	Window used for computing mean and standard deviation of the variation of the total loss (*delta_loss*) between consecutive epochs.
delta_loss_avg_thresh	5	If both the average and the standard deviation of the *delta_loss* between consecutive epochs are less than these thresholds, and absolute total loss is less than ***loss_target***, then the precision is relaxed by ***relaxation_factor*** bits.
delta_loss_stdev_thresh	5	
loss_target	100	Value of total loss below which the precision can be relaxed.

2.5 Algorithm performance / energy evaluation

The performance/power evaluation component evaluates the performance and the power consumption associated with the execution of the inference of a candidate design point on the target architecture. The general structure of the performance/power evaluation component is shown in Figure 7.

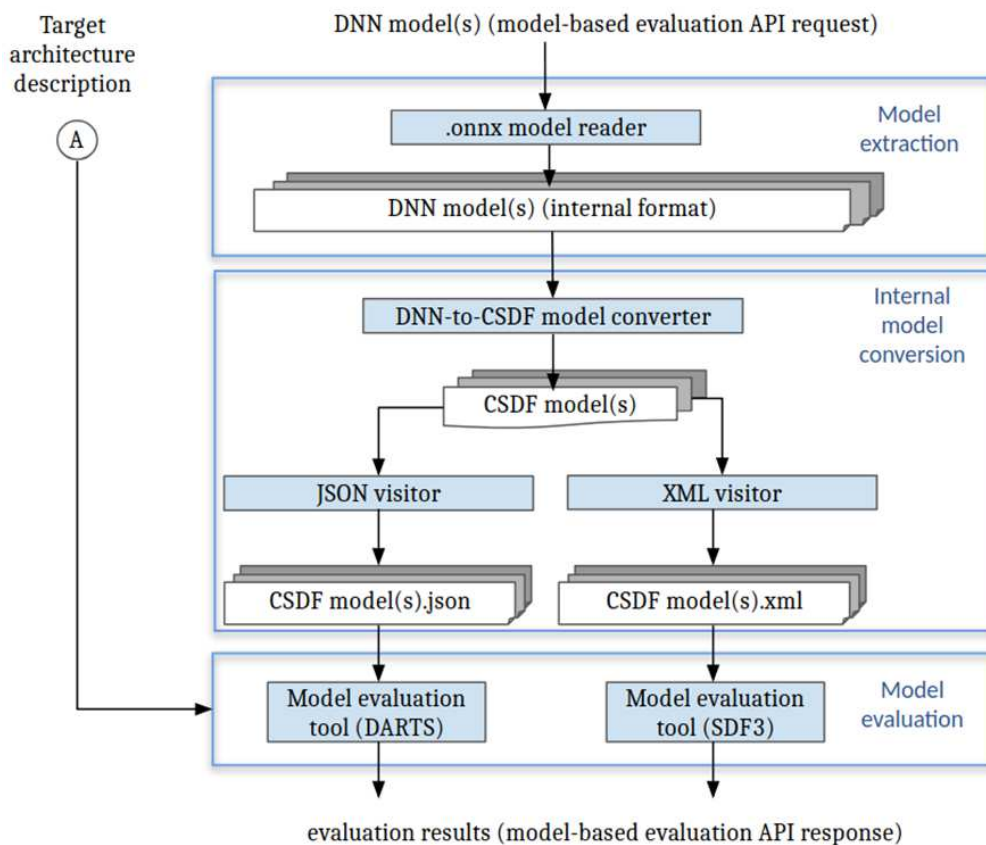

Figure 7: Performance/power evaluation component

The component takes as an input one or several DNN models and the target architecture description. The DNN models come from the DSE engine and are communicated via the model-based evaluation API. The output of the component provides, for every DNN model, a set of evaluated performance/power

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

parameters. The evaluation of a DNN model is performed in three main steps described in more details below: model extraction, internal model conversion and model evaluation. Finally, the performance/power evaluation component use the model-based evaluation API to return the evaluated parameters in '.json' format.

2.5.1 Model extraction

Model extraction step is implemented in a tool called '.onnx model reader'. During the model extraction, an internal DNN model representation is extracted from a specification of the input DNN provided in '.onnx' exchange format. Unlike the input DNN specification, where deep neural networks are repserented as graphic models in which one graph node corresponds to one DNN layer, the internal model use block-based models representation. In a block-based model representation, each node of a DNN graph corresponds to a block, describing the subgraph of the original neural network model. In addition, the internal model use transformations, that allow to switch between different representations of functionally equivalent models and avoid the re-evaluation of similar models. An example of an internal model repsentation and split/merge transformations is shown in Figure 8. It should also be noted, that the internal model does not store real weights, but only their descriptions, sufficient for the power/performance evaluation.

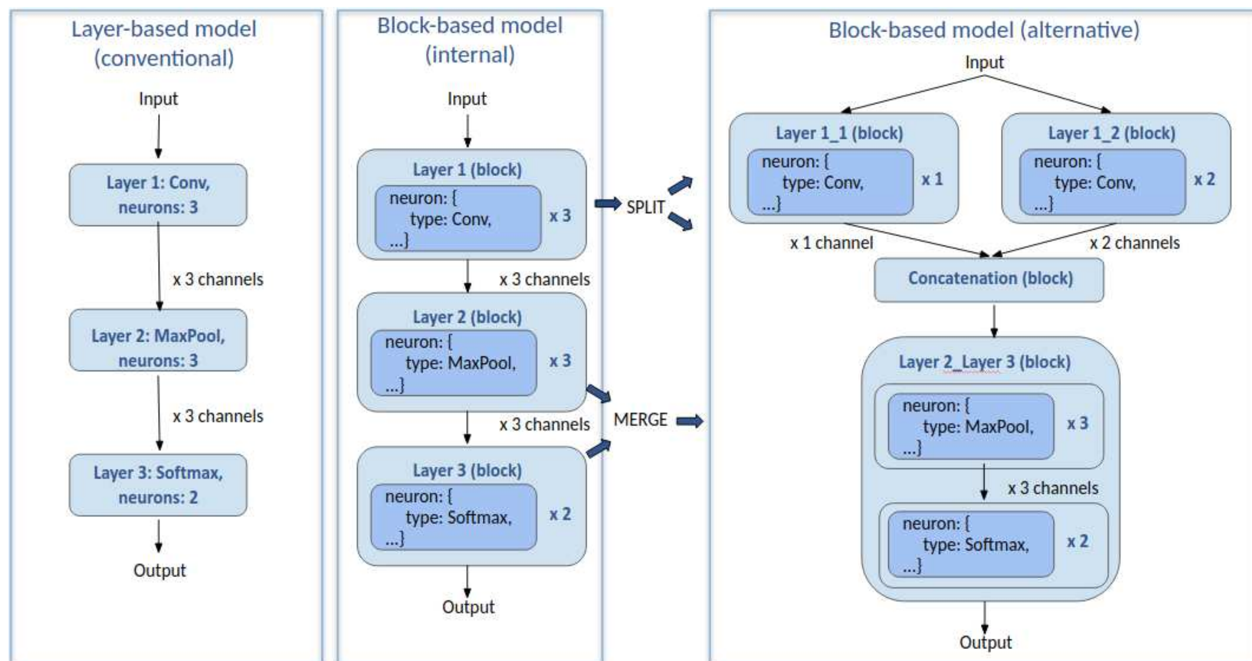

Figure 8: Internal DNN model representation

2.5.2 Internal Model Conversion

In this step, the extracted internal DNN model is converted to a functionally equivalent data-flow model called Cyclo-static DataFlow (CSDF)[3]. This step is implemented in a tool called 'DNN-to-CSDF model converter' that generates the CSDF model (in '.json' format) as a graph of concurrent tasks communicating data via FIFOs.

[3] G. Bilsen, M. Engels, R. Lauwereins. Cycle-static data flow. *IEEE Transactions on Signal Processing*, 1996.

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

The internal model conversion is performed in four main steps and begins with CSDF topology generation, shown in Figure 9. The bold red lines show the convertion from DNN internal model highlighted in blue to CSDF model highlighted in red. Each block of the internal DNN model is converted to a node of the CSDF graph. Each connection between neural network blocks is converted to a CSDF FIFO communication channel. The communication channels are linked to CSDF graph nodes via ports.

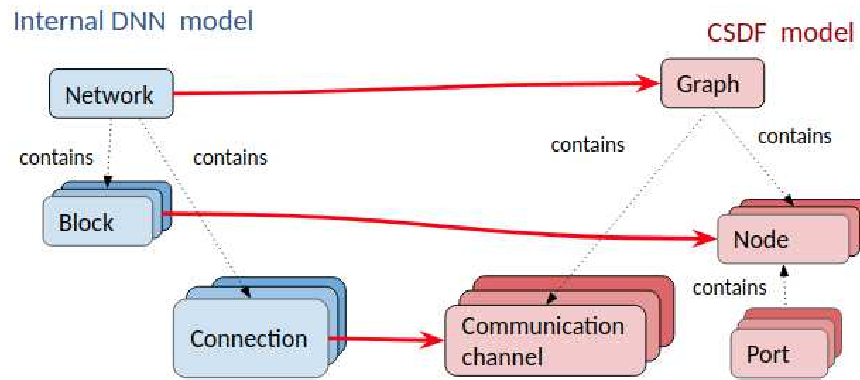

Figure 9: CSDF graph topology generation

After the CSDF topology is generated, the CSDF model parameters are calculated. The CSDF model parameters calculation is shown in Figure 10, describing conversion a convolutional block of two neurons into a CSDF node.

The 'exec_template' field of CSDF node shown in the top right box in Figure 10, reflects the functionality inherited from the DNN block shown in the top left box of Figure 10. The task set for the 'exec_template' field is stored in 'function' field of the node. The 'wcet' field of the CSDF node contains tasks time in seconds. This field should be set up from the target hardware architecture description.

The data flow, entering and leaving a DNN block in the top left box of Figure 10 is represented as two arrows labeled by typed tensors. For each arrow the CNN-to-CSDF converter generates a CSDF FIFO buffered channel, that store data tokens. The DNN-to-CSDF converter assigns one token to one element of the corresponding tensor. Therefore, in the simplest case, the maximum number of tokens stored in an CSDF FIFO channel is calculated as total number of elements in the corresponding tensor. However, it can be optimized. An optimization example is shown in the bootom left box in Figure 10, where a Convolution operation, performed by a CSDF node from top right box of Figure 10 is unfolded as a sequence of partial convolutions over the input local areas.

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

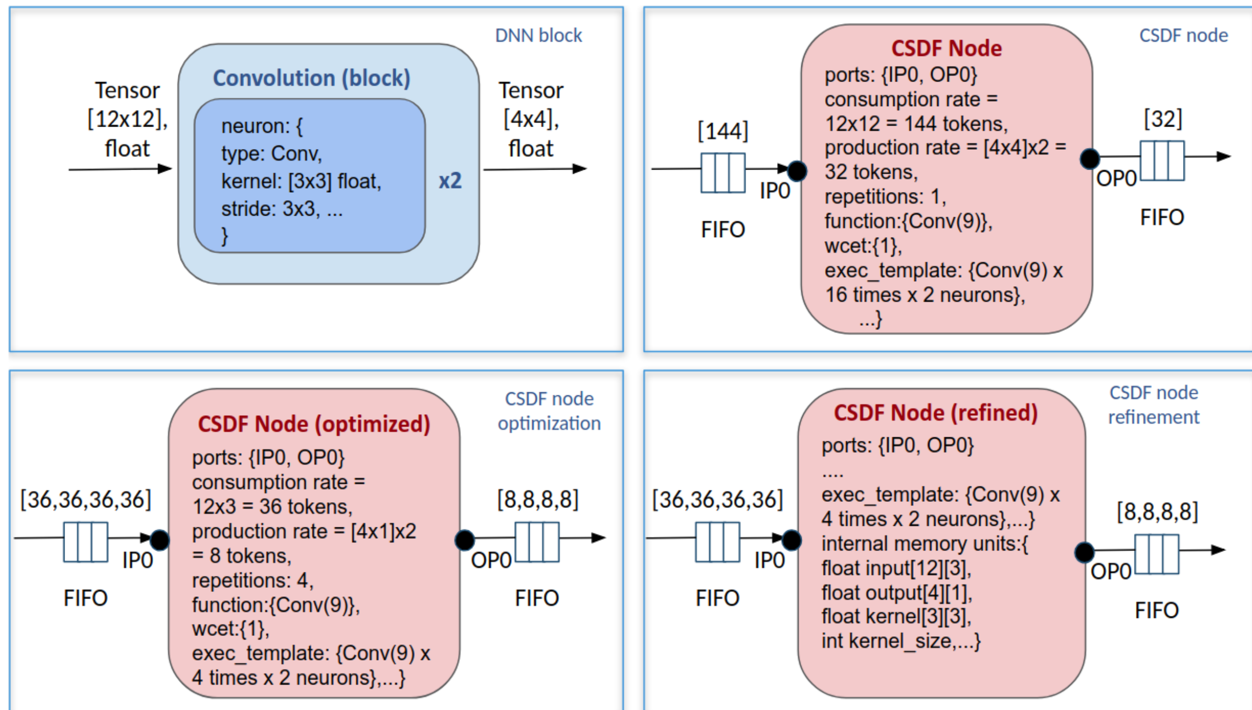

Figure 10: DNN-to-CSDF conversion example

Theoretically, the smallest input local area is an area, covered by the convolutional kernel. But for a real application such fine-grained unfolding can be tricky. The fact is that data read/write operations are usually performed by lines. Thus, operating over 2D local areas needs in additional synchronization which leads to performance overheads. An accepted trade-off is to simulate reading of data by lines (as in real hardware systems), but to limit the number of read lines to the minimum required. This reduces the number of tokens stored in the CSDF channel buffers, while avoids unnecessary synchronization. In the example shown in Figure 10, an optimized CSDF node unfolds convolutional operation over 12x12 matrix as 4 operations over the 12x3 local input areas, where 12 is a full length of a line and 3 is minimal required height of local area. It provides the same result as the original node, but requires smaller buffer sizes and therefore reduces the total amount of memory, required by the model. For current moment buffer sizes are reduced as much is possible. However, the minimal buffer sizes are not always increase the model performance, because the latter strongly dependens on the target hardware architecture. Thus, the buffers sizes are planned as a controlled parameter.

To take into account the constant values, stored in the CSDF nodes, such as weights and constant parameters, each CSDF node is associated with a number of internal memory units, where each memory unit describes typed constant value in C++ - like notation. Internal memory units also describe input and output data arrays, which are used to accumulate incoming and outgoing data of the CSDF node and preserving the tokens order. An example of the CSDF node refinement by data units is shown in the bottom right box of Figure 10.

2.5.3 Model Evaluation

In this step, the target architecture is taken into account and generated CSDF model is evaluated according to the following parameters

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

- *Performance:* the DNN inference execution time in seconds (s);
- *Energy:* the DNN inference energy consumption in joules (J);
- *Processors:* the number of processors required for DNN inference;
- *Memory:* the memory required for DNN inference in bytes (B).

For CSDF model evaluation, the power/performance evaluation tool supports two alternative open-source frameworks: DARTS[4] and SDF3[5]. Both frameworks are able to compute the execution time (throughput and latency) of the CSDF graph and memory to execute the generated CSDF graph. The DARTS tool additionally provides evaluation of the number of required processors. By default, DARTS tool is used for evaluation.

For performance and processors evaluation, DARTS tool constructs a real-time schedule for the tasks in generated CSDF graph model. Taking into account execution templates of each CSDF graph node, tasks times, obtained from target architecture description and constructed schedule, DARTS evaluates the total time of the graph firing and optimal number of processors for it.

For memory evaluation, the total buffer sizes calculated by DARTS are added to the memory estimates, obtained from the typed internal memory units descriptions. The result of addition is refined by typed specifications (e.g. int type of the description is refined to int32 type, used by target platform).

The current research is related to the extension of the DARTS tool with techniques for estimating the energy consumption of the SDF graph when executed on the target architecture.

SDF3 allows the use of alternative algorithms for analyzing the data flow model and is introduced to provide ALOHA project more freedom in the choice of analysis and modeling tools.

3 Preliminary integration activities

In deliverable D1.1 initial interfaces had been defined. Continuation of the work on tool's REST container, testing of the interfaces between the DSE, TE, SEC, RPI are planned after the face to face meeting organized by CA in M10 to understand Docker containers and their communication. The face to face meeting will be also used to evaluate integration strategies between the automated algorithm configuration tools.

[4] http://daedalus.liacs.nl/daedalus-rt.html
[5] http://www.es.ele.tue.nl/sdf3/

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.

4 References

B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition, 84:317–331, 2018.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In International Conference on Learning Representations, 2015.

Anupam Datta, S. Sen, Y. Sick, Algorithmic transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems, 2016.

Wei Koh, Pang & Liang, Percy. Understanding Black-box Predictions via Influence Functions, 2017.

Breiman, Friedman, Classification and regression trees, 1984.

European General Data Protection Regulation, GDPR, 2018. https://eugdpr.org/, last access Sept 2018

Copyright - This document has been produced and funded under the ALOHA H2020 Grant Agreement 780788.
Unless officially marked PUBLIC, this document and its contents remain the property of the beneficiaries of the ALOHA Consortium and may not be distributed or reproduced without the express written approval of the project Consortium.