Modelling Neural Network Models

Nadia Daoudi[1]✉[0000−0002−1437−667X], Ivan Alfonso[1][0000−0003−3940−0629], and Jordi Cabot[1,2][0000−0003−2418−2489]

[1] Luxembourg Institute of Science and Technology, Luxembourg
{nadia.daoudi,ivan.alfonso,jordi.cabot}@list.lu
[2] University of Luxembourg, Luxembourg

Abstract. Neural networks are now a key element of many complex software systems, typically known as AI-enhanced systems or smart systems. Model-driven engineering (MDE) approaches are often used to model and (semi)automatically generate such complex systems. Nevertheless, for MDE to be used in the creation of smart systems, it is essential to be able to model the implementation of neural networks (NNs) as part of the rest of the system. Unfortunately, modelling support for NNs is rather limited. In particular, there is a lack of complete and expressive neural network metamodels that could facilitate NN development. This poses significant challenges as it creates a disconnect between the development of neural networks and the other components of the system. Overall, this paper introduces a metamodel and its concrete syntax to support neural network design, providing a foundation for future research in smart system development.

Keywords: Metamodel · Model · Neural networks · Grammar

1 Introduction

Neural networks have become a prevalent component in the design of modern software. With the growing demands for more intelligent and efficient software, neural networks empower systems to enhance their functionality and responsiveness. The abundance of data has also significantly increased the reliance on neural networks to perform complex tasks, driving advancements in areas such as predictive analytics and recommender systems.

Offering versatility across different components of a system's architecture, neural networks can be leveraged for both front-end and back-end tasks. On the front-end side, they provide functionalities to support tasks involving direct interactions with users. On the back-end, NNs can be integrated to support critical tasks involving advanced data processing and analysis. These functionalities can be accomplished using various types of neural networks, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

Neural networks are generally developed separately from the other components of the system due to the limited availability of modeling approaches specifically designed for neural networks. Indeed, NN metamodels are a missing piece

in smart software development. In the literature, some studies have attempted to propose metamodels for neural networks [5,12,2]. However, they suffer from limitations, such as being very specific to particular types of neural networks (e.g, a metamodel for convolutional neural networks) or failing to comprehensively address basic neural network types and the different components involved in their development, such as datasets and training setup.

To fill this gap, we contribute a metamodel and its concrete syntax for modelling neural networks. Our NN metamodel covers fundamental components necessary for NN development along with a textual notation for the concrete syntax. Specifically, the metamodel encompasses the elements needed for architecture design, dataset loading, training, and performance evaluation. Our work aims to facilitate neural networks development as it would support future research in enabling NNs to be modelled as seamlessly as any other system component. All the artefacts from our work are available in an open-source repository[3].

The rest of this paper is organised as follows: Section 2 presents basic concepts related to NN development. In Section 3 we describe our metamodel and the concrete syntax. Section 4 presents an evaluation of our metamodel. Finally, we review related research in Section 5 and conclude the paper in Section 6.

2 Background

Neural networks are designed to recognise and identify patterns present in a given dataset in order to perform complex tasks such as classification or regression. The main building blocks of neural networks are layers which represent computational units that perform mathematical operations to transform the input data. Neural networks can also potentially be composed of tensorOps (i.e., short for tensorOperations) and sub-neural networks. TensorOps are low-level operations applied to tensors (i.e., multidimensional arrays of data) within and between layers [6]. Such operations include addition, reshaping, and multiplication to manipulate the data. In some complex architectures, neural networks can include other sub-neural networks as building blocks to create a unified model.

Layers in neural networks can be of different types, such as fully connected layers, convolutional layers and recurrent layers. Fully connected layers perform matrix operations that connect each neuron in the previous layer to every neuron in the subsequent layer. Convolutional layers apply convolutional filters to detect patterns in data, making them particularly effective for tasks such as image processing. Recurrent layers are designed for sequential data processing by maintaining internal memories. Since they can capture temporal dependencies and patterns in the input data, this type of neural networks is effective for tasks such as time series forecasting and natural language understanding. Convolutional and recurrent layers have various sub-types designed to perform specific tasks and handle different data structures [7,14]. We show in Fig. 1 an example of an NN architecture taken from a TensorFlow tutorial[4]. The neural network

[3] `https://github.com/BESSER-PEARL/BESSER`

[4] `https://www.tensorflow.org/tutorials/images/cnn`

consists of various types of layers for image classification, including Conv2D, MaxPool2D, Flatten, and LinearLayer (i.e., fully connected layer).

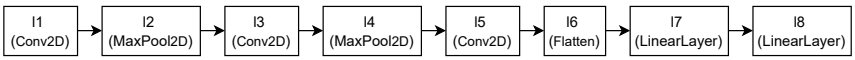


Fig. 1. Example representing an NN model composed of eight layers.

The order of layers in neural network architectures can be of two types:

- Sequential architecture: Layers are structured in a linear, straightforward sequence. The output of a layer is passed to its subsequent layer sequentially.
- Non-sequential architecture: In this type of architecture, layers are not arranged in a simple linear order. Instead, connections can skip layers, allowing a layer to receive input from earlier, non-adjacent layers. This creates a structure where the flow of information does not follow a strict sequence.

To help neural networks learn complex patterns in the data, activation functions can be used to introduce non-linearity to the transformations applied by the layers. After a neural network architecture is designed, hyperparameters need to be defined to guide the learning process and optimise the model's performance. Such hyperparameters can include the optimiser [11], the loss function [19] and the learning rate [4]. Besides, a well curated training dataset also needs to be made available to the neural network so it can learn meaningful patterns. The effectiveness of an NN model is heavily dependent on the quality, diversity, and size of the dataset used in the training. After the neural network is trained, a test dataset is used to assess its learning and validate its performance.

3 NN Modelling Language

In this section, we present our NN metamodel and its concrete syntax.

3.1 The NN metamodel

We provide the class diagram of our NN metamodel in Fig. 2. The key concepts in the NN metamodel are represented using meta-classes and their associations. Our design was heavily inspired by the two popular deep learning frameworks PyTorch[5] and TensorFlow[6]. Specifically, we compared concepts from the two frameworks to come up with a metamodel design that is general to represent neural network components and allows the definition of both sequential and non-sequential architectures. This approach ensures that the metamodel remains versatile and adaptable across different contexts within neural networks development. In the following, we present the main metaclasses that define our metamodel and some of their important attributes.

[5] `https://pytorch.org`
[6] `https://tensorflow.org`

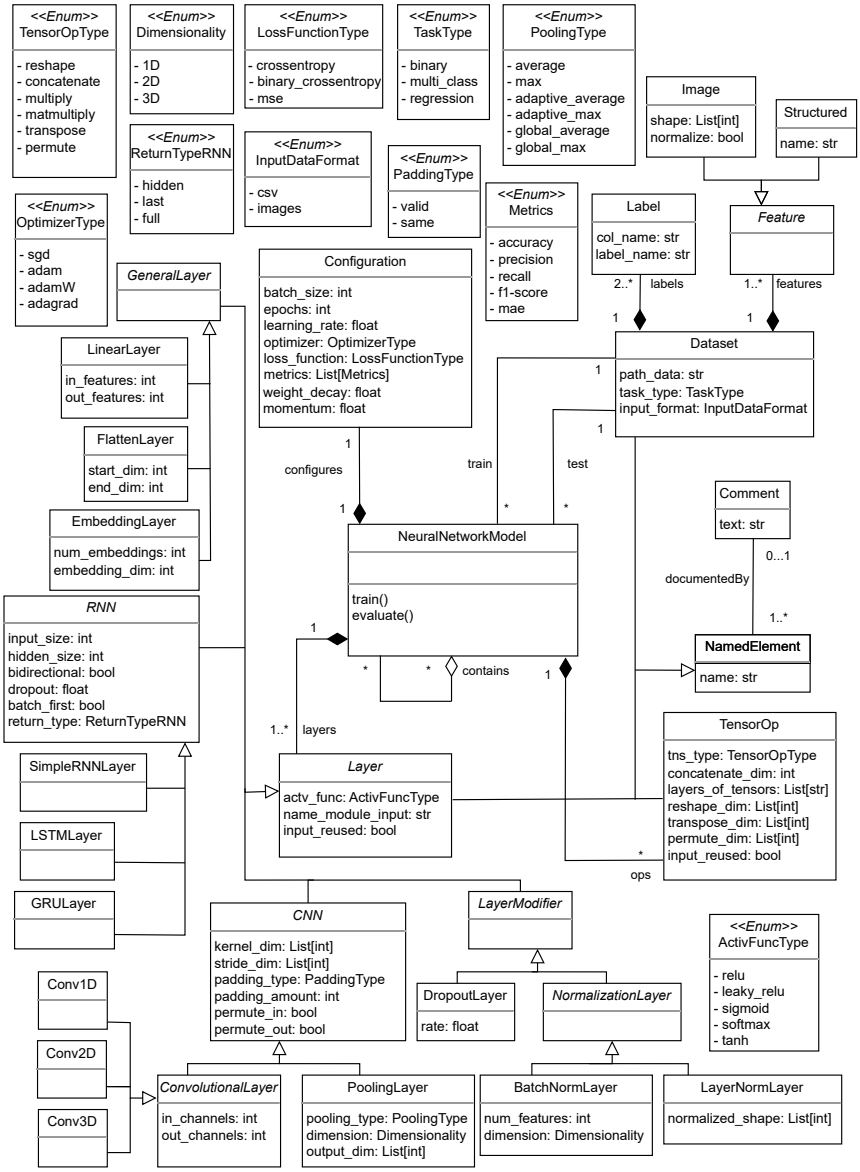


Fig. 2. The NN metamodel

NeuralNetworkModel At its core, the metamodel contains the *NeuralNetworkModel* metaclass that comprises properties and behaviours of an NN model. A *NeuralNetworkModel* can contain three types of components, referred to as *modules*: (sub)-*NeuralNetworkModel*s, *Layer*s and *TensorOp*s. Specifically, a *NeuralNetworkModel* can incorporate another *NeuralNetworkModel* as a component (*contains* relationship in the metamodel). This is common in NNs, where a model can include other models as part of its architecture. The other two modules (i.e., *Layer* and *TensorOp*) are described in the following paragraphs.

Layer A *Layer* is an abstract metaclass that encapsulates properties of NN layers. It defines the activation function (i.e., *actv_func* attribute) that is detailed through an enumeration listing the possible types of activation function, such as relu, sigmoid, and softmax. As it is the case for all the enumerations in the metamodel, the list of activation functions can be extended as needed. In our metamodel, a layer can be of four types, represented by the following abstract metaclasses: *GeneralLayer*, *LayerModifier*, *CNN*, and *RNN*.

TensorOp This metaclass is used to represent an operation or function applied to tensors. In our metamodel, six types of tensorOps are defined in the *TensorOpType* enumeration. The *type* attribute of the *TensorOp* metaclass gets its value from this enumeration. Beside *type*, the *TensorOp* metaclass includes additional attributes that serve as parameters for the tensor operations.

Configuration This metaclass contains attributes representing several hyperparameters essential for training neural networks, and an attribute *metrics* to store the metrics used for evaluating the model's performance.

Dataset Neural networks rely on datasets to learn and generalise based on examples. In our metamodel, it is represented using the metaclass *Dataset* that can serve either as the training or the test dataset.

Comment Neural networks components are generally adopted for different reasons (i.e., a sub-NN has shown promising results in another architecture, a type of layer is well suited to the task, a layer has given better results based on trial and error, ...). To keep track of these design decisions, we use the *Comment* metaclass that adds relevant documentation to any *NamedElement*. Note that several concepts inherit from *NamedElement* in our metamodel, facilitating the reuse and sharing of NN models.

3.2 Concrete syntax

Neural networks (NNs) can be described using various notations. In this work, we propose a textual notation for their definition, supported by a grammar we developed to instantiate the concepts of our metamodel.

Listing 1.1 presents an excerpt of our grammar, developed using ANTLR[7]. The primary rule, *neuralNetwork*, allows for specifying a name (ID), layers, sub-neural networks, tensor operations, modules to represent the sequential architecture of the NN, parameters (defined in the subsequent parser rule in line 12), along with datasets for training and testing. We employ specific operators in ANTLR to enforce cardinalities in our metamodel: the '+' operator indicates one or more occurrences, '?' denotes zero or one, and '*' represents zero or more.

A key rule in the grammar is *layer* (lines 21-23), where each layer must have a defined name (ID) and a set of parameters, which vary depending on the layer type. For every layer type, we define a dedicated parser rule. For instance, the *linear* rule (lines 31-34) contains general layer parameters (defined by the *layerParams* rule on line 25), such as the activation function, along with specific parameters for linear layers, such as *in_features* and *out_features*.

Our language supports the definition of NNs using the *modules* rule (line 41), which enables the user to specify the order of layers, sub-neural networks, and tensor operations for the NN.

Finally, we define in lines 43-48 the rule for specifying the training dataset, including the name (ID), *path_data*, *task_type*, and other parameters as defined in the metamodel. The complete grammar is available in the project repository.

Listing 1.1. DSL grammar excerpt

```
1  grammar NN;
2
3  neuralNetwork  : ID ':'
4                   'layers' ':' layer+
5                   ('sub_nn' ':' sub_nn+)*
6                   ('tensor_ops' ':' tensorOp+)*
7                   'modules' ':' modules
8                   'config' ':' configuration
9                   'trainingDataset' ':' dataset
10                  'testingDataset' ':' dataset ;
11
12 configuration  : 'batch_size' '=' INT
13                  'epochs' '=' INT
14                  'learning_rate' '=' DOUBLE
15                  'optimiser' '=' STRING
16                  'metrics' '=' strList
17                  'loss_function' '=' lossFunction
18                  'weight_decay' '=' DOUBLE ;
19                  'momentum' '=' DOUBLE ;
20
21 layer          : '-' ID ':'
22                  (generalLayer | rnn
23                  | cnn | layerModifier) ;
24
25 layerParams    : ('actv_func' '=' activityFuncType)?
26                  ('name_layer_input' '=' STRING)?
27                  ('input_reused' '=' BOOL)? ;
28
29 generalLayer   : linear | flatten ;
30
31 linear         : 'type' '=' 'Linear'
32                  layerParams
33                  'in_features' '=' INT
34                  'out_features' '=' INT ;
35
```

[7] https://www.antlr.org/

```
flatten         : 'type' '=' 'Flatten'
                  layerParams
                  ('start_dim' '=' INT)?
                  ('end_dim' '=' INT)? ;

modules         : ('-' ID)+ ;

dataset         : 'name' '=' ID
                  'path_data' '=' STRING
                  'task_type' '=' taskType
                  'input_format' '=' inputFormat
                  'image' '=' intList
                  'labels' '=' '{' label ',' label (',' label)? '}' ;

//Other rules omitted for brevity purposes ...
```

Listing 1.2 provides a textual example of the neural network (NN) model shown in Fig. 1. The model definition begins by specifying the NN's name (my_model). Next, the layers are defined (lines 2-19) outlining three layers (l1, l2, and l3), with l1 and l3 being 2D Convolutional layers, and l2 as a Pooling layer. Then, the modules definition (lines 21-22) specifies the order of the layers. Finally, hyperparameters are defined (lines 23-29), such as the "adam" optimiser. The full textual model can be accessed in the project repository.

Listing 1.2. NN textual model example

```
my_model:
    layers:
        - l1:
            type=Conv2D
            actv_func=relu
            in_channels=3
            out_channels=32
            kernel_dim=[3, 3]
        - l2:
            type=Pooling
            pooling_type=max
            dimension=2D
            kernel_dim=[2, 2]
        - l3:
            type=Conv2D
            actv_func=relu
            in_channels=32
            out_channels=64
            kernel_dim=[3, 3]
        //Other layer definitions omitted for brevity.
    modules:
        - l1 - l2 - l3 - l4 - l5 - l6 - l7 - l8
    config:
        batch_size=32
        epochs=10
        learning_rate=0.001
        optimiser="adam"
        metrics=["f1-score"]
        loss_function=crossentropy
//Dataset definitions ommited for brevity.
```

4 Evaluation

In this section, we aim to evaluate the expressiveness of our metamodel. Specifically, we select four state-of-the-art NN architectures from the literature and rely on our approach to model them:

- AlexNet [8], which is composed of various layers including Conv2D, Pooling, Dropout and Linear layers. It contains two sub-NNs: one to extract the features and the other to conduct the classification.
- VGG16 [15], which incorporates Conv2D, Pooling, Dropout and Linear layers. It is composed of two sub-NNs for feature extraction and classification.
- LSTM [21], which is an RNN-based neural network since it contains LSTM layers in addition to Linear, Dropout and Embedding layers.
- CNN-RNN [20], which includes a mix of CNN and RNN layers such as Conv1D, GRU, Linear, Dropout and Embedding layers. It also uses the concatenate tensorOp and implements a non-sequential architecture.

Our attempt to model these NNs helped us refine our metamodel to include some concepts that we overlooked in our initial design. For instance, the LSTM model uses a bidirectional LSTM layer that processes data in both forward and backward directions to capture information from past and future contexts within a sequence. Our metamodel was missing this "bidirectional" concept and was then updated after our attempt to model the LSTM neural network. We have successfully modelled the four NNs using our approach, which demonstrate its effectiveness in expressing state-of-the-art neural networks. The four modelled neural networks are available in our repository.

5 Related Work

Neural networks are becoming an integral component of intelligent systems, driving advancements in various domains. An NN metamodel has been proposed [5] to improve the verification and versioning of AI components. While addressing some NN concepts, this metamodel does not cover fundamental layer types, limiting its overall applicability. Similarly, an NN markup language has been proposed [12]; however, it remains basic as it defines networks in terms of connected processing units, and does not address the specific structures and connections found in complex architectures like CNNs and RNNs. Another NN metamodel [2] has been proposed, only covering multi-layer perceptrons. An UML-based design flow has been proposed to streamline NN deployment [17]. Nonetheless, this design has limited coverage, as it does not encompass basic layer types.

To support the design of convolutional neural networks (CNNs) in autonomous vehicle perception (AVP), a metamodel [13] has been proposed. While the metamodel has similarities to our work, its application domain is narrow as it only covers CNNs which makes it very specific and not appropriate to model other well-known NN types such as RNNs. An NN metamodel [9] has been proposed to incorporate predictive tasks into manufacturing. It represents NNs using basic neuron and edge abstractions, omitting other fundamental NN types and potentially limiting its versatility. To automate AI-based software development processes [10], a metamodel has been proposed addressing the procedural aspects of AI development. However, it does not specifically target ML or NN concepts such as algorithms and hyperparameters.

Several research works have also proposed domain-specific languages (DSLs) to enhance computational efficiency in ML. DeepDSL [22] enhances the portability and efficiency of deep learning models. TensorFlow Eager [1] provides an interactive model to improve development flexibility. Diesel [3] optimises GPU performance for linear algebra and NN computations. FastML [18] facilitates the parallelisation of ML algorithms on mult-icore processors. OptiML [16] simplifies and optimises ML tasks execution through implicit parallelism.

While the above works develop metamodels or DSLs for neural networks, their applicability and focus is limited. With our work, we aim to propose a comprehensive metamodel that covers the fundamental types of NN layers along with the other concepts needed for NN training and evaluation.

6 Conclusion

To support smart component development, we designed an NN metamodel that covers fundamental concepts to neural network development. Our metamodel captures the elements necessary for architecture design (layers, tensorOps, sub-NNs), dataset loading, training and performance evaluation. Additionally, we contributed a textual notation for the concrete syntax to facilitate NN definition and conducted an evaluation to assess the expressiveness of our approach. All the artefacts from our work are available in an open-source repository.

As further work, we will work on the development of a useful ecosystem of tools around this metamodel such as importers, code-generators and model transformations and even model-based migration processes from/to neural networks implemented with different libraries.

Acknowledgments. This project is supported by the Luxembourg National Research Fund (FNR) PEARL program, grant agreement 16544475.

References

1. Agrawal, A., Modi, A., Passos, A., Lavoie, A., Agarwal, A., Shankar, A., Ganichev, I., Levenberg, J., Hong, M., Monga, R., Cai, S.: Tensorflow eager: A multi-stage, python-embedded dsl for machine learning. In: Proceedings of Machine Learning and Systems. vol. 1, pp. 178–189 (2019)
2. Al-Azzoni, I.: Model Driven Approach for Neural Networks. In: International Conference on Intelligent Data Science Technologies and Applications. pp. 87–94 (2020)
3. Elango, V., Rubin, N., Ravishankar, M., Sandanagobalane, H., Grover, V.: Diesel: DSL for linear algebra and neural net computations on GPUs. In: Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages. p. 42–51. New York, NY, USA (2018)
4. He, F., Liu, T., Tao, D.: Control Batch Size and Learning Rate to Generalize Well: Theoretical and Empirical Evidence. In: Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)
5. van den Heuvel, W.J., Tamburri, D.A.: Model-Driven ML-Ops for Intelligent Enterprise Applications: Vision, Approaches and Challenges. In: Shishkov, B. (ed.) Business Modeling and Software Design. pp. 169–181. Cham (2020)

6. Ji, Y., Wang, Q., Li, X., Liu, J.: A Survey on Tensor Techniques and Applications in Machine Learning. IEEE Access **7**, 162950–162990 (2019)
7. Ketkar, N., Moolayil, J.: Convolutional Neural Networks, pp. 197–242. Apress, Berkeley, CA (2021)
8. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. In: Advances in Neural Information Processing Systems. vol. 25. Curran Associates, Inc. (2012)
9. Lechevalier, D., Hudak, S., Ak, R., Lee, Y.T., Foufou, S.: A neural network metamodel and its application for manufacturing. In: 2015 IEEE International Conference on Big Data (Big Data). pp. 1428–1435 (2015)
10. Morales, S., Clarisó, R., Cabot, J.: Towards a DSL for AI Engineering Process Modeling. In: International Conference on Product-Focused Software Process Improvement. pp. 53–60. Springer (2022)
11. Nwankpa, C.: Advances in optimisation algorithms and techniques for deep learning **5**(5), 563 – 577 (Oct 2020)
12. Rubtsov, D., Butakov, S.: A unified format for trained neural network description. In: IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222). vol. 4, pp. 2367–2372 vol.4 (2001)
13. Safdar, A., Azam, F., Anwar, M.W., Akram, U., Rasheed, Y.: MoDLF: a model-driven deep learning framework for autonomous vehicle perception (AVP). In: the International Conference on Model Driven Engineering Languages and Systems. p. 187–198. MODELS '22, New York, NY, USA (2022)
14. Schmidt, R.M.: Recurrent neural networks (rnns): A gentle introduction and overview. arXiv preprint arXiv:1912.05911 (2019)
15. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
16. Sujeeth, A.K., Lee, H., Brown, K.J., Chafi, H., Wu, M., Atreya, A.R., Olukotun, K., Rompf, T., Odersky, M.: OptiML: an implicitly parallel domain-specific language for machine learning. In: Proceedings of the 28th International Conference on International Conference on Machine Learning. p. 609–616. ICML'11 (2011)
17. Suárez, D., Posadas, H., Fernández, V.: UML-Based Design Flow for Systems with Neural Networks. In: 2023 38th Conference on Design of Circuits and Integrated Systems. pp. 1–6 (2023)
18. Tsopgny, N.G.N., Nguélé, T.M., Kouakam, E.: DSL for parallelizing Machine Learning algorithms on multicore architecture. In: CARI 2022 (2022)
19. Wang, Q., Ma, Y., Zhao, K., Tian, Y.: A comprehensive survey of loss functions in machine learning. Annals of Data Science pp. 1–26 (2020)
20. Wang, X., Jiang, W., Luo, Z.: Combination of Convolutional and Recurrent Neural Network for Sentiment Analysis of Short Texts. In: Proceedings of the 26th International Conference on Computational Linguistics. pp. 2428–2437 (Dec 2016)
21. Yang, S., Yu, X., Zhou, Y.: LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example. In: International Workshop on Electronic Communication and Artificial Intelligence. pp. 98–101 (2020)
22. Zhao, T., Huang, X.: Design and implementation of DeepDSL: A DSL for deep learning. Computer Languages, Systems & Structures **54**, 39–70 (2018)