Convolutional neural networks for enhanced classification mechanisms of metamodels

Phuong T. Nguyen[a], Davide Di Ruscio[a,*], Alfonso Pierantonio[a],
Juri Di Rocco[a], Ludovico Iovino[b]

[a] *Università degli studi dell'Aquila, 67100 L'Aquila, Italy*
{phuong.nguyen, davide.diruscio, alfonso.pierantonio, juri.dirocco}@univaq.it
[b] *Gran Sasso Science Institute, Italy*
ludovico.iovino@gssi.it

Abstract

Conventional wisdom on Model-Driven Engineering suggests that metamodels are crucial elements for modeling environments consisting of graphical editors, transformations, code generators, and analysis tools. Software repositories are commonly used in practice for locating existing artifacts provided that a classification procedure is available. However, the manual classification of metamodel in repositories produces results that are influenced by the subjectivity of human perception besides being tedious and prone to errors. Therefore, automated techniques for classifying metamodels stored in repositories are highly desirable and stringent. In this work, we propose MEMOCNN as a novel approach to classification of metamodels. In particular, we consider metamodels as data points and classify them using supervised learning techniques. A convolutional neural network has been built to learn from labeled data, and use the trained weights to group unlabeled metamodels. A comprehensive experimental evaluation proves that the proposal effectively categorizes input data and outperforms a state-of-the-art baseline.

---

*Corresponding author

1. Introduction

Model-Driven Engineering [1] (MDE) helps tame the complexity of nowadays large-scale software development by using models to capture relevant knowledge of the problem domain. Models are typically specified in terms of concepts formalized in metamodels, and they are manipulated via automated transformations to achieve superior automation, whether it be refactoring, simulation, or code generation. As in other knowledge-intensive domains, repositories are large depots of unstructured resources that permit to store experience (in terms of developed artifacts) and to make it available for later use. While the relevance of software repositories is often recognized, the difficulties in building effective, scalable, and useful tools are often underestimated [2]. Leveraging the informative structure within a repository, which comprehends insights, know-how, and generally, accumulated and distributed knowledge, requires the ability to locate and make existing artifacts amenable to reuse, analysis, and documentation (among others) [3]. Over the last decade, several model repositories have been proposed by both academia and industry, e.g., [4, 5, 6, 7], just to mention a few. However, unless repositories permit modelers to easily discover and retrieve the artifacts that satisfy their needs, the potential benefits related to the availability of such resources can be significantly compromised. Clustering and classification approaches have already been proposed to categorize the metamodel repositories automatically [8, 9]. In contrast, manual procedures to classify artifacts are typically time-consuming, prone to errors, and might lead to accuracy problems. Clustering techniques have been employed in order to group *similar* metamodels in an unsupervised manner [8]. Unfortunately, the performance of clustering techniques heavily relies on the chosen similarity measure, and two main problems that hinder its effectiveness are namely: *(i)* timing performance; and *(ii)* the identification of the appropriate number of clusters [9].

Recently, Machine Learning (ML) gained an interest in the scientific community as it promises to yield unprecedented results and performance compared with conventional approaches [10]. The core principle of an ML algorithm is

2

that it attempts to simulate the learning activities of human beings [11]. As a result, ML systems are capable of autonomously extracting meaningful patterns from real-world examples, like humans [12, 13]. Recently, the AURORA tool [9] has been proposed to classify metamodel repositories using neural networks automatically. By learning from a training set consisting of labeled metamodels, it classifies incoming unlabeled ones based on well-established techniques. Even though the approach has been validated on a dataset consisting of more than 500 metamodels, it has some performance issues as detailed later in this paper.

In this work, metamodels are considered as data points and they are classified by means of supervised learning techniques. In particular, we exploit convolutional neural networks (CNNs) [14] to learn from labeled data, and use the trained weights to group unlabeled metamodels. Such networks have been successfully employed in various domains, such as imaging applications [15, 16, 17], and natural language processing [18]. By means of supervised learning, we overcome the main limitation of different unsupervised clustering techniques where it is necessary to specify the number of clusters in advance. Meanwhile, with supervised learning the number of clusters depends on the training data, and it has been specified ex-ante in the labeled data. In our experiment, we used a dataset with nine categories, however such a number can easily be changed if there are more categories previously classified. In this respect, there is no need to dictate any number in advance with our approach.

The rationale behind the selection of a CNN is as follows. To choose a suitable network for classifying metamodels, we performed an investigation into various techniques, namely recurrent neural networks (RNNs), long short-term memory (LSTM) neural networks [19], and CNNs. As a matter of fact, RNNs and LSTMs are suitable for working with time series data such as weather forecast or pedestrian trajectory [15]. We anticipate that they can be used to solve other tasks in MDE, e.g., recommendation of model repairs, or for assisting the specification of metamodels. For classification, we select CNNs as they have been designed to successfully work with images. The recent development of disruptive deep neural networks which are based on a CNN has enabled a large

3

number of applications in different domains. Thus, the deployment of a CNN to classify metamodels allows us to take advantage of a well-founded background

65 related to many technological breakthroughs, which have been proven through a series of work in image classification.

By mimicking the human visual system [20], the approach (named MEM-OCNN hereafter) considerably improves the accuracy of AURORA by outperforming the baseline for different quality metrics as demonstrated by the

70 experiments. Moreover, the analysis reveals that MEMOCNN becomes more effective and efficient when there is more data available for training. In this respect, the contributions of the work can be summarized as follows: *(i)* a representation scheme to transform metamodels into a CNN-processable format; *(ii)* a convolution neural network for automatic classification of metamodel repos-

75 itories; *(iii)* an empirical evaluation with different experimental configurations to study the proposed approach's performance; and finally *(iv)* a MEMOCNN prototype together with the metadata used in our evaluation is released to the benefit of future research.[1]

Outline. The structure of the paper is as follows. Section 2 presents a back-

80 ground for our work. The proposed approach is illustrated in Sect. 3. The next section explains the evaluation settings used to study the approach's performance. The experimental results are reported and analyzed in Sect. 5. In Sect. 6, we discuss the outcomes of the experiments and highlight the probable threats to validity. Section 7 reviews the related work and associates them with

85 our approach. Finally, Sect. 8 outlines possible future work and draws some conclusions.

2. Background

Section 2.1 provides an introduction to metamodels. Afterward, Sect. 2.2 describes the problem of classification as a machine learning task. We introduce
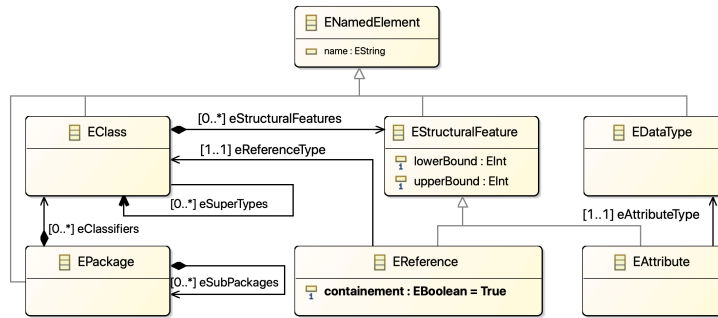
---

[1]https://github.com/MDEGroup/memoCNN/

4

Figure 1: An excerpt of the *ecore* metamodel.

classification of metamodels in Sect. 2.3. Finally, a brief introduction to CNNs
is provided in Sect. 2.4, highlighting the most relevant aspects.

2.1. A brief introduction to metamodels

A metamodel defines the abstract concepts of a domain where concepts, as
well as the relationships among them, are expressed by modeling infrastruc-
ture [1]. The Eclipse Modeling Framework (EMF)[2] provides modelers with
an infrastructure to design, define and edit models. EMF includes *Ecore* as a
meta-metamodel for describing metamodels. Figure 2.1 depicts a simplification
of the *Ecore* metamodel. All the represented meta classes are named elements.
EPackage is composed of subpackages as well as classes, while EClass consists
of EStructuralFeatures, i.e., EAttributes and EReferences. Moreover, EClass
can inherit structural features from other classes. EAttribute and EReference
inherit the lowerbound and upperbound attributes in order to define the car-
dinalities of structural features. EReference links the container to EClass by
eReferenceType reference. Moreover, containment is a boolean attribute, which
allows one to specify if the linked EClass is contained, e.g., to trigger cascade-
deletion, or not in the container being modeled. EAttributes are typed as
EDataType instance by eAttributeType reference. The encoding schema that
we are going to present in Sect. 3.2 relies on the *Ecore* simplification depicted
in Fig. 2.1.

---

[2]https://www.eclipse.org/modeling/emf/

5

2.2. Classification as a supervised learning task

Classification belongs to supervised learning where training data is used to identify recurrent patterns, which in turn are used to classify new instances. In other words, classification relies on the existence of predefined classes of objects and aims to assign a new item to a class. In this respect, labeled training data is needed to guide the learning process which is conducted to refine the constituent weights and biases, by introducing all the training instances. Meanwhile, testing is to predict a label for an input instance by means of the weights and biases previously obtained from learning. Several machine learning algorithms can be exploited for classification, e.g., Decision Trees, Random Forrest, or Support Vector Machines [21]. Among others, neural networks have been widely used for classification [22], due to their effectiveness.

Classification offers a wide range of applications, covering different domains. The most prominent use cases include speech recognition [23], biometric identification [24], document classification [25], to name but a few. In image processing, the extraction of low-level features from images and videos have been extensively investigated [26], and recently the focus has been shifted to automatically classify images and videos into semantically meaningful categories. For instance, HTML documents have been classified into a hierarchy of categories [27] to support Web documents sharing in a groups of users with common interests. Classification is also useful when the intent is to automatically and efficiently group images as static diagrams, e.g., UML diagrams. The automatic classification of images has been also successfully and extensively applied on medical images [28] to automate a process of pattern recognition related to some diseases. In recent years, the research community has been increasingly interested in categorizing large volumes of unstructured text, given the proliferation of social networks [29].

2.3. Classification of metamodels

Open-source software platforms typically provide developers with guidance to locate software artifacts to be reused through various built-in facilities. Unfortunately, finding relevant information in a repository presents technical dif-

ficulties as the stored artifacts need to be accurately classified before they can be detected. Until recently, such classification has been operated manually by annotating the artifacts with metadata whose granularity is adapted to the

¹⁴⁵ level of accuracy dictated by the problem at hand. Like other hand-operated activities, the manual classification of items stored in repositories is tedious, time-consuming, and prone to error. To this end, clustering has demonstrated to be a workable solution to the problem [8]. However, clustering techniques suffer from a low efficiency as well as the difficulty in selecting an appropriate

¹⁵⁰ number of clusters [9].

AURORA represents the first approach in the domain of automated classification of metamodels with machine learning techniques [9]. The system works on the premise that the prescribed information related to metamodels and their categories can be used to train a supervised classifier. The approach is based

¹⁵⁵ on a feed-forward neural network, and it learns from a training data consisting of labeled metamodels and classifies unknown ones. An empirical evaluation [9] on a dataset of manually classified metamodels showed that the system obtains a considerably high prediction performance.

Nevertheless, AURORA has its own drawbacks. Though it performs well
¹⁶⁰ on the given dataset, which consists of a small number of metamodels, i.e., 555 (see Sect. 4.2), it is less efficient on larger datasets as it is the case with feed-forward neural networks [30]. As a consequence, with larger metamodels, the network performs slowly, and the learning process could take longer to converge, i.e., the accuracy can only be increased after several epochs. In
¹⁶⁵ this respect, convolutional neural networks (CNN) may be a natural choice to obtain a scalable classification procedure. Indeed, such kind of networks is characterized by a degree of accuracy that improves with larger dataset populations. In the next subsection, we introduce the technical background of CNNs as a base for further discussions.

7

2.4. Convolutional neural networks

Ordinary neural networks connect all the neurons of a layer to the next layer. In this way, the number of weights and biases increases exponentially with the number of layers. Such a structure is not suitable for working with images since it adds up to a huge number of parameters. CNNs [20] have been specifically designed for working on images, and they attempt to capture the spatial and temporal structures, which are intrinsic features of an image. Furthermore, CNNs also minimize the number of parameters by using convolutional filters. A *filter* or *kernel* is a small square matrix used to capture specific features from an input image, e.g., nodes and edges. In practice, several kernels can be used to extract various types of features from images. Given a convolutional layer, each perceptron gets input from a certain part from the previous layer, which is called a *receptive field*. The convolution operation is performed by sliding the filter along the width and height of a *feature map*, which is either the input image, or the result of the convolution operation. An output feature map of one layer becomes the input feature map of the succeeding layer. A CNN consists of three types of layers, i.e., convolutional, pooling, and fully-connected layers, whose functionalities are explained below:

- *Convolution layer*: it performs most of the computation within a CNN to extract important features of an input image;

- *Pooling layer*: it downsamples a feature map by taking the maximum value within a window, preferably a square one, to reduce the number of parameters [20]. The layer also extracts rotational and positional invariant features from a feature map;

- *Fully-connected layer*: the layer works as a conventional perceptron, each of its neurons is fully connected to the previous layer.

For illustration purpose only, we take an example in Fig. 2 where there is an excerpt of a CNN. A tensor of size $96 \times 96 \times 3$ represents an input image, and the number 3 corresponds to the three color channels in images,
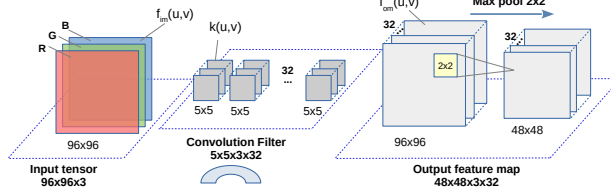
8

Figure 2: Filtering and pooling.

i.e., Red (R), Green (G), and Blue (B). A 4D filter of size $5 \times 5 \times 3 \times 32$ is convolved with the input feature map to produce an output feature map of size $96 \times 96 \times 1 \times 32$. To elaborate on the computation, we consider only a slice of the 4D filter, corresponding to a matrix $k(u,v)$. Given an input feature map $f_{im}(u,v)$, the output feature map $f_{om}(u,v)$ is created by convolving $k(u,v)$ with $f_{im}(u,v)$:

$$f_{om}(u,v) = (f_{im} * k)(u,v) = \sum_i \sum_j f_{im}(i,j).k(u-i,v-j) \tag{1}$$

Pooling is performed by means of Maxpool 2×2 to reduce the resulted feature map's width and height to a half, yielding a new feature map of size $48 \times 48 \times 1 \times 32$.

Furthermore, in a CNN there are the following features:

- *Dropout*: it is a parameter used to distribute the learned representation across all the neurons. It is an effective measure to minimize overfitting [31];

- *Softmax*: the last fully-connected layer of a CNN normally uses Softmax as the activation function and it converts a set of real numbers to probabilities which sum to 1.0 [20]. Given C classes, and y_k is the output of the kth neuron, the final prediction is the class that gets the maximum probability, i.e., $\hat{y} = argmax\, p_k$, $k \in 1..C$, where p_k is computed below.

$$p_k = \frac{exp(y_k)}{\sum_{k=1}^{C} exp(y_k)} \tag{2}$$

- *Rectified Linear Units (ReLU)*: convolutional layers use ReLU as the activation function, which returns 0 given a negative input, and returns the

9

input itself if it is larger than 0, $f(x) = max(0, x)$.

3. MEMOCNN: A convolutional neural network to classify metamodels

In an attempt to simulate humans' cognition towards the relationship be-
tween metamodels and their categories, we conceptualized the usage of a well-
defined deep learning technique, i.e., convolutional neural networks. In this
section, we present MEMOCNN, an approach to classification of **me**ta**mo**dels
exploiting a **C**onvolutional **N**eural **N**etwork. Section 3.1 presents the con-
ceived architecture to realize MEMOCNN. An explanatory example showing
how metamodels are encoded is presented in Sect. 3.2. Finally, Sect. 3.3 illus-
trates a concrete configuration of a MEMOCNN network.

3.1. Architecture

Figure 3.1 shows the main components of the MEMOCNN architecture sup-
porting two main phases, i.e., *training* and *deployment*. Metamodels can be
collected from different sources, and in the scope of this paper we make use
of GITHUB as the main data source, since the platform offers a large number
of metamodels repositories. Afterward, expert modelers are asked to inspect
the metamodels and manually classify them into independent groups based on
their experience. By means of labeled data including metamodels and labels
assigned by humans, the training process is in charge of identifying the right
weights to configure the CNN CLASSIFIER component. Finally, the weights
obtained by CNN CLASSIFIER are then utilized to classify metamodels given
as input to the deployment process.

The TRAINING phase starts with collecting metamodels from public repos-
itories like GITHUB by means of the CRAWLER component ①. NLP PROCES-
SOR ② applies Natural Language Processing steps (i.e., stop words removal,
stemming, and lemmatization) on all the named elements, which are contained
in the collected metamodels. DATA ENCODER ③ takes as input the manipu-
lated metamodels and encodes them as matrixes enabling the subsequent phases
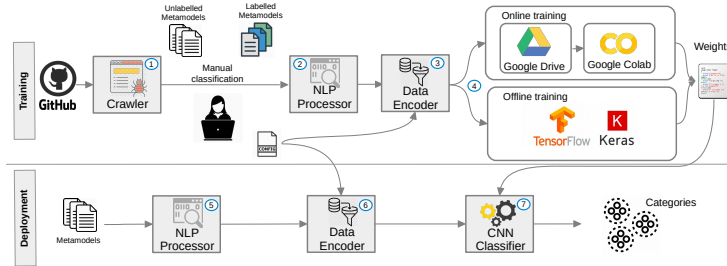
10

Figure 3: MEMoCNN's system architecture.

²⁵⁰ of the approach. To identify the properties to be included in the encoding, we followed a *feature engineering* process that started with considering only terms of the metamodels (uni-gram), we subsequently added containment information (bi-gram), up to including all the missing metamodel properties, including relation cardinalities, typing information, etc. (n-gram). In particular, with ²⁵⁵ the uni-gram encoding scheme, only the terms contained in the metamodels being processed are considered. All the aspects related to metamodel structure are discarded. With the bi-gram scheme, information about containment relations between named elements is also encoded. The n-gram scheme fully encodes the input metamodels by enriching the bi-gram scheme by including ²⁶⁰ all the properties of the represented structural features (e.g., typing information, relations cardinalities, etc.). The whole approach has been evaluated by considering the different encoding schemes individually.

The proposed architecture allows for both offline and online training ④. By the former, one can train the network on a PC or a laptop installed with ²⁶⁵ TensorFlow and Keras. By the latter, the code is executed directly on Google Colaboratory[3] connecting to metadata stored into Google Drive.[4] The final outcome of the training process is a set of weights and biases, which then can be used to deploy the network.

The DEPLOYMENT phase shown in Fig. 3.1 refers to the adoption of a previ-²⁷⁰ ously trained and deployed CNN CLASSIFIER ⑦ to automatically classify input

---

[3]https://colab.research.google.com/
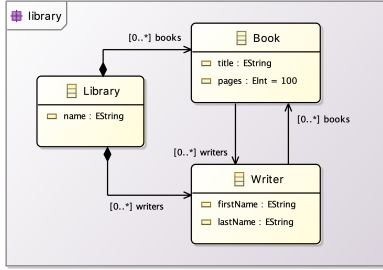
[4]https://drive.google.com/

11

Figure 4: The *Library* metamodel.

metamodels. To this end, as also done in the training phase, the metamodels of interest have to be manipulated by means of NLP steps ⑤ and properly represented by means of one of the encoding scheme previously discussed ⑥. MEMOCNN has been implemented on top of TensorFlow and Keras,[5] two well-founded Machine Learning frameworks. To facilitate future research, we made available online the MEMOCNN tool and the metadata used in the evaluation.

3.2. Metamodel encoding

We propose a method to transform a metamodel into a picture by making use of its modeling elements such as metaclasses, packages, and attributes. In other words, we attempt to *picturize* a metamodel, exploiting its constituent features. Starting from a set of metamodels, we construct a set of terms and use it to convert all the metamodels into vectors.

To illustrate how the data processing works, we consider the *Library* metamodel in Fig. 4. This metamodel can be used for defining models, where a *Library* can be composed of a set of *Book*s and *Writer*s. A writer can edit books, and then a book can be written by a set of writers. A writer is defined in terms of its *firstname* and *lastname*, whereas a book is characterized by its *title* and the number of *pages*.

---

[5]Keras has been integrated into TensorFlow and provides to developers a higher abstract level, and thus a more user-friendly programming interface. Meanwhile, TensorFlow offers a greater flexibility in managing a network's settings. We exploited them both to build MEM-OCNN, thereby earning different degrees of granularity for the experimental configurations.

The lexical terms in the metamodel are extracted and encoded as prescribed by the above encoding schemes. Table 1 summarizes the outcome by presenting all lexical terms in the metamodel, and the corresponding encodings for both the raw and normalized feature vectors. For the sake of presentation, each metamodel element name is given with an identifier and its metamodel type. For instance, the package name *library* is *(i)* individually encoded in uni-grams; *(ii)* in bi-grams as part of the terms of the metaclasses #2, #6 and #10; and *(iii)* in n-grams as part of the terms of the structural features #3, #4, #5, #7, #8, #9, #11, #12 and #13. In addition, the encodings related to normalized feature vectors require the term to be subject to stemming, lemmatization, and stop words removal. Thus, the term books is translated into *book* and comprises both the metaclass *Book* and the feature *books*. Moreover, additional information about cardinalities, e.g., 0.1 and whether the node is subject to containment, e.g., TRUE, is appended.

With the aim of giving a flavour of the NLP steps that are applied by NLP PROCESSOR depicted in Fig. 3.1 (see ② and ⑤), the *normalized feature vectors* shown in Table 1 represent the output of stemming, lemmatization, and stop words removal tasks applied on the *raw feature vectors*. For Step ⑤, we make use of the *nltk* python library[6] to perform the required NLP steps. First, a term is tokenized into subterms by using as delimiters any non-alphabetical character and blank spaces. After that, the *nltk Wordnet Lemmatizer* module is invoked to squeeze the inflected subterms to the root English term. Then, the *Porter stemmer* algorithm [32] reduces the subterms by removing the common English endings from them. Afterward, English stop words which form the list of subterms are also removed. Finally, the resulting subterms are concatenated to create features.

---

[6] https://www.nltk.org/

Medamodel element			Raw feature vector			Normalized feature vector		
Type	**id**	**Name**	**uni-gram**	**bi-gram**	**n-gram**	**uni-gram**	**bi-gram**	**n-gram**
package	#1	library	library	-	-	library*	-	-
metaclass	#2	Library	Library	library.Library	-	library*	librari.librari	-
attribute	#3	name	name	Library.name	Library.name.EString.0.1	name	librari.nam	librari.name.estring.0.1
reference	#4	writers	writers	Library.writers	Library.writers.Writer.false.0.-1	writ**	librari.writ	librari.writ.writ.false.0.-1
reference	#5	books	books	Library.books	Library.books.Book.false.0.-1	book***	librari.book	librari.book.book.false.0.-1
metaclass	#6	Writer	Writer	library.Writer	-	writ**	librari.writ	-
attribute	#7	firstName	firstName	Writer.firstName	Writer.firstName.EString.0.1	firstnam	writ.firstnam	writ.firstnam.estring.0.1
attribute	#8	lastName	lastName	Writer.lastName	Writer.lastName.EString.0.1	lastnam	writ.lastnam	writ.lastnam.estring.0.1
reference	#9	books	books	Writer.books	Writer.books.Book.false.0.-1		writ.book	writ.book.book.false.0.-1
metaclass	#10	Book	Book	library.Book	-	book***	library.book	-
attribute	#11	title	title	Book.title	Book.title.EString.0.1	titl	book.titl	book.titl.estring.0.1
attribute	#12	pages	pages	Book.pages	Book.pages.EInt.0.1	page	book.pag	book.pag.eint.0.1
reference	#13	writers	writers	Book.writers	Book.writers.Writer.false.0.-1	writ**	book.writ	book.pag.eint.0.1

Table 1: The extraction of the *Library* metamodel.

(b) The final matrix

Row_0
Row_1
Row_2
Row_3

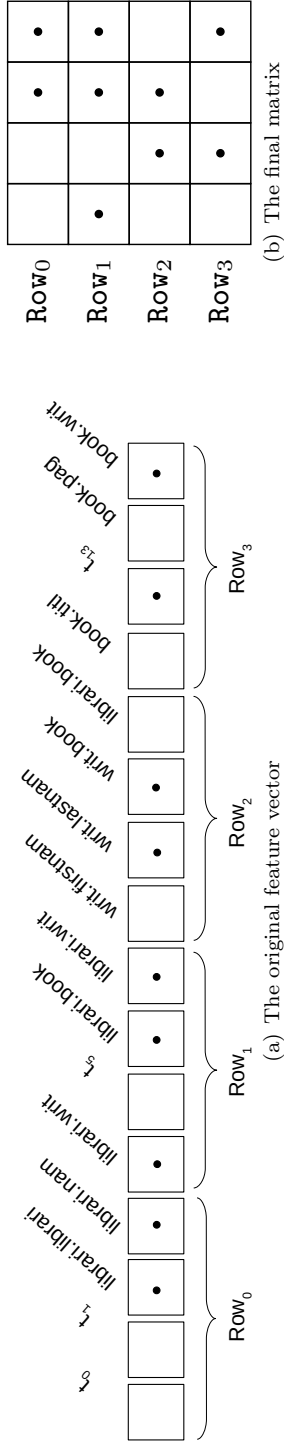(a) The original feature vector

Figure 5: The encoding of the *Library* metamodel.
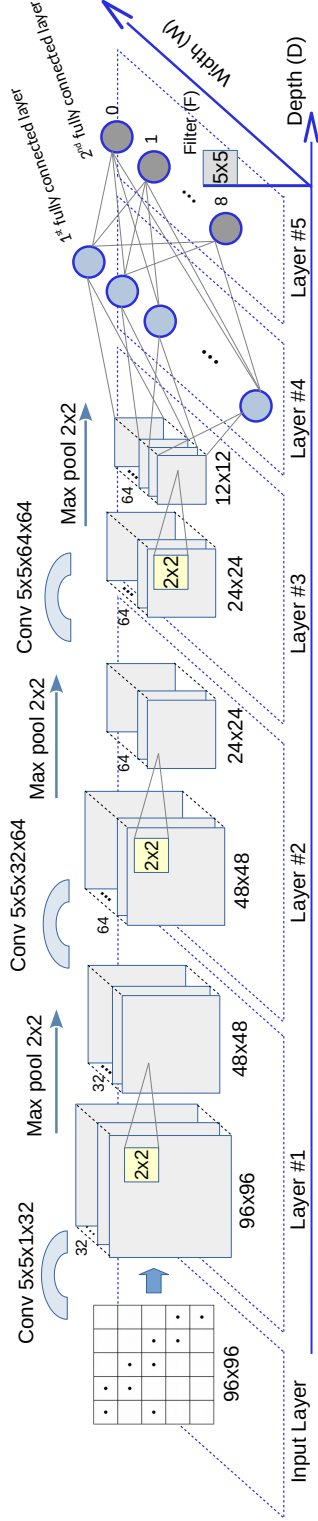

Figure 6: A memoCNN network with five layers.

₃₁₅ By analyzing all the input metamodels, the corresponding terms are collected and used as indexes of vectors each encoding a corresponding metamodel. To illustrate how the encoding paradigm works, we take an explanatory example in Fig. 5(a). For the sake of presentation, we consider a corpus with a total of 16 terms.[7] In the considered example, there are 13 of them related to the

₃₂₀ bi-gram encoding of the *Library* metamodel previously discussed. In Fig. 5(a), the blank cells represent no occurrences of the related terms, or in other words the corresponding value is equal to 0. Nevertheless, we refrain from using the one-hot vector encoding scheme, since such a representation neglects the importance of each constituent term. Thus, in our encoding scheme, the cells with a

₃₂₅ dot contain real numbers which quantify the occurrences of the corresponding terms in the encoded metamodel using the TF-IDF function. In particular, we normalized such values with respect to the most occurred terms in the dataset by *min-max feature scaling* [33]. This allows us to keep a notionally common scale among the metamodel features representations.

₃₃₀ Starting from the vector in Fig. 5(a), we serialize it into a matrix, preferably a square one as shown in Fig. 5(b). By traversing a vector from its beginning to the end, each time we select four consecutive entries to populate a row for the matrix. In this sense, the size of the vector needs to be a square number. So if the size is lower than the closest square number, then we pad the missing

₃₃₅ elements with zeros to constitute a square matrix. With this representation, we attempt to capture features by employing a unified structure across all the metamodels: each cell in the matrix represents a specific term, and adjacent cells with a dot form patterns like nodes and edges, as we find in images. In this respect, we hypothesize that metamodels within a same categories would

₃₄₀ share common pictorial patterns.

In real usage, even when the dataset becomes bigger, we can always apply the same process to parse and represent the data. A set of training metamodels is parsed using all the aforementioned steps, resulting in a corpus of

---

[7]In practice, the number of terms is much larger.

16

(a) Class 0   (b) Class 1   (c) Class 2

(d) Class 3   (e) Class 4   (f) Class 5
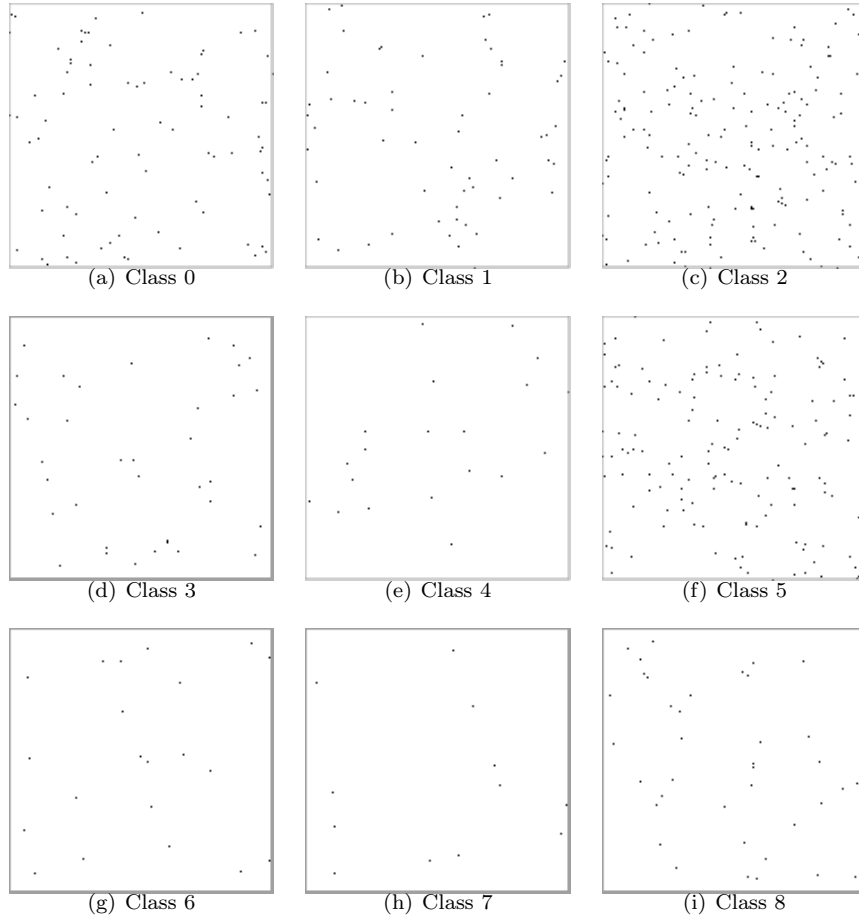
(g) Class 6   (h) Class 7   (i) Class 8

Figure 7: Examples of the metamodels after being transformed into images.

the terms that appear in the dataset. Different from the previous work about

345 AURORA [9], in this work we do not use any cut-off value but make use of all the parsed terms. Given a big dataset, each metamodel is then serialized into a vector, albeit with several entries. The final matrix will be populated accordingly, which is much bigger compared to the example one in Fig. 5(b). In this way, we have a matrix of M×M size, where M can be hundreds, or

350 thousands, like in image classification with high resolution images.

To demonstrate how the resulting pictures look like in practice, we exploit a real dataset whose characteristics are described in detail in Sect. 4.2. We

represent them in the bitmap format following the proposed representation (cf. Fig. 5(a)) and Fig. 5(b)). We select for each class a metamodel and show the corresponding images, which are all of size 148×148, in Fig. 7(a)–7(i). From the figures, it is evident that each class has a different distribution, and thus the density of dots across an image. For instance, Class 2 and Class 5 are much denser compared to Class 6 or Class 7. As a matter of fact, CNNs are highly suitable for capturing this type of features, e.g., nodes or edges [14], since they use the convolution operation to filter and retain features from input images. In this respect, we suppose that our proposed representation scheme will facilitate the recognition of metamodels. We are going to validate this hypothesis with concrete experiments in Sect. 5.

3.3. An example of MEMOCNN network

Figure 6 illustrates an example of a MEMOCNN network. The input layer corresponds to an input metamodel which is represented as a 2D image of size 96×96. There are three convolutional layers, i.e., Layer #1, #2, #3 and two fully connected (FC) layers, i.e., Layer #4 and Layer #5. While the number of neurons of the 1st FC layer can be experimentally configured, the number of neurons of the 2nd FC layer is set to the number of categories. For each transition from one layer to the next one, a convolution filter is applied on the input feature map, resulting in an output feature map. For example, by Layer #3, a filter of size $5 \times 5 \times 64 \times 64$ is convolved with the input feature map, and this yields an output feature map of size $24 \times 24 \times 1 \times 64$. Within the layer, pooling is conducted to reduce the dimension, producing an output feature map of size $12 \times 12 \times 1 \times 64$. From the left to the right side of Fig. 6, the size of the feature maps decreases, whilst the number of filters increases. The network was the result of a series of experiments on the given dataset. We suppose that such a configuration is subject to change in practice, depending on the input data. For a larger dataset, we expect a deeper and/or wider network, which is generally the case with image datasets [34].

18

During the experiments conducted to evaluate MEMOCNN, we changed the number of layers as well as the number of filters to create different configurations and they are going to be clarified in the next section.

385 4. Evaluation

This section presents the experimental evaluation conducted to analyze the performance of MEMOCNN. Section 4.1 introduces the research questions being addressed in this paper. The dataset and the metrics used to evaluate the proposed approach are illustrated in Sect. 4.2 and Sect. 4.3, respectively.
390 Afterward, Sect. 4.4 highlights the experimental configurations.

4.1. Research questions

By using a series of experiments on a dataset that has been manually classified, we aim to answer the following research questions:

- RQ_1: *How effective is* MEMOCNN *at classifying the considered cate-*
395 *gories?* It is of great importance to know if MEMOCNN can accurately classify metamodels into their correct categories.

- RQ_2: *Which network configuration contributes to a better performance?* We investigate whether the approach yields scalable performance by analyzing how a more complex network helps enhance the prediction accuracy
400 on the given dataset.

- RQ_3: *How does* MEMOCNN *compare with AURORA?* Finally, using a common dataset, we evaluate MEMOCNN against AURORA to ascertain whether the proposed approach gives a better prediction performance compared to the baseline.

405 4.2. Dataset

In the evaluation, we made use of the same dataset that was exploited to evaluate AURORA [35]. The dataset consists of metamodels related to domain specific languages (DSLs) collected from GITHUB. All the metamodels were

19

inspected and manually classified by an expert modeler based on his experi-

ence [35]. The final dataset consists of 555 metamodels distributed in nine independent categories. A summary of the categories and their cardinality is provided in Fig. 8(a). The most populous category is *State machine DSLs* with 159 metamodels, accounting for 29% of the total amount. Meanwhile, the smallest category is *Issue Tracker DSLs* with only seven metamodels. Such a category may render the classification difficult since there is a limited amount of data for training. The distribution of metamodels with respect to the number of metaclasses, attributes, and references is depicted in Fig. 8(b), where most of the metamodels contain a low number of metaclasses as well as attributes and references. The metamodels are concentrated on the bottom left corner of the tensor. Only those of the category *Review system DSLs* contain a large number of metaclasses, and they are scattered across the 3D space. In summary, the dataset exhibits a variety of categories with different characteristics and cardinalities. To identify possible cloned metamodels in the dataset, we exploited the *match-based* similarity function, which has been devised in one of our previous work [8]. In particular, the function takes two metamodels as input, i.e., mm_1 and mm_2, and it returns 1 if every element of mm_1 exactly matches with one element of mm_2, and vice versa. By computing the similarity among all metamodels pairs, we detected 60 clones, accounting for 10.8% of the dataset. In practice, modeling repositories commonly include many model clones as it has been pointed out in a recent work [36], where the authors conducted an empirical study on 4.5 millions of GitHub repositories to identify code file duplicates. The study reveals that more than 40% of the code files are duplicated. Due to this reason, we believe that maintaining a controlled number of metamodel clones in the dataset helps resemble a real repository scenario. As a result, we decided to keep cloned metamodels in the dataset.

4.3. Metrics

As seen in the dataset (Sect. 4.2), there are nine categories in total, and in the scope of this work we deal with multi-class (or multinomial) classification [37], i.e., each instance of the data needs to be classified into one of the
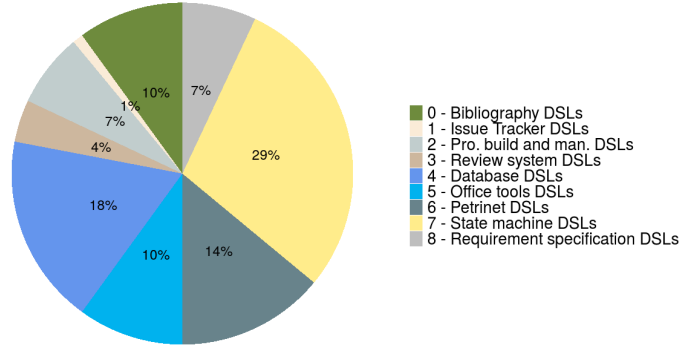
(a) The categories and their cardinality
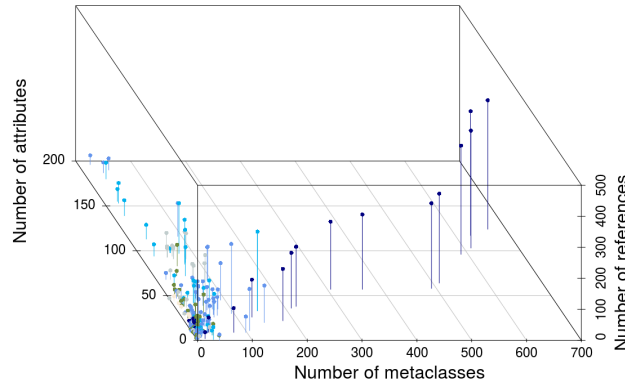
(b) Distribution of metamodels

Figure 8: A summary of the dataset.

⁴⁴⁰ nine categories. To thoroughly study the performance of our approach, we employed cross-validation [38] for evaluating the classification outcomes. For each of the categories mentioned in Sect. 4.2, we measured the relevance of the ground-truth data, i.e., the metamodels and their corresponding labels, with the classified results obtained by running the systems, i.e., AURORA and

⁴⁴⁵ MEMOCNN. We are interested in understanding how well the produced classes match with the ground-truth data, considering the fact that the dataset is imbalance. Thus, *blanced accuracy*, *precision*, *recall*, F_1-*score*, *ROC*, and *AUC*

21

are utilized to study the systems' performance as detailed below. First, given a category, the following definitions are considered:

- **450** • *True positive* (TP): the metamodels being correctly classified;

- • *False positive* (FP): the classified metamodels which actually do not match with the ground-truth data;

- • *False negative* (FN): the metamodels that should have been classified since they belong to the ground-truth data, but they are not;

- **455** • *True negative* (TN): the metamodels that are not classified and they also do not belong to the ground-truth data.

Then, the metrics are defined as follows:

4.3.1. Precision and Recall

Precision and recall are used to quantify the level of relevance of the clas-
460 sified items to the ground-truth data as given below. Precision measures the fraction of correctly classified items to the total number of items, while Recall (or *True positive rate*, TPR) is defined as the fraction of actual positive cases that are correctly classified.

$$P = \frac{TP}{TP + FP} \qquad (3) \qquad R = TPR = \frac{TP}{TP + FN} \qquad (4)$$

465 ### 4.3.2. F_1-score

It is a harmonic average of precision and recall.

$$F_1 = \frac{2 \cdot P \cdot R}{P + R} \qquad (5)$$

In the context of multi-class classification, macro-averaged F_1-score (macro-F_1 for short) is computed as the average of the F_1-scores of all categories.

4.3.3. False positive rate (FPR)

470 It is the ratio of the falsely classified items to the total number of items.

$$FPR = \frac{FP}{TN + FP} \qquad (6)$$

22

4.3.4. Balanced accuracy

As seen in Sect. 4.2, the dataset is imbalanced and thus we use balance accuracy to measure the performance as defined below:

$$Accuracy = \frac{TPR + FPR}{2} \times 100\% \qquad (7)$$

4.3.5. ROC curve and AUC

A receiver operating characteristic (ROC) curve represents the relationship between FPR (false positive rate) and TPR (true positive rate, Recall) [39]. Such a curve is sketched in a 2D space where the x-Axis and the y-Axis correspond to FRP and TPR, respectively. In this respect, an ROC curve spans from (0,0) to (1,1) in the space. Furthermore, the area under the ROC curve (AUC) is used as an explicit indication of how good a classifier can be. A *dummy* classifier, i.e., the one that randomly assigns a label to metamodels, would have an AUC value of 0.5. In contrast, a perfect classifier would have an AUC value of 1.0. In this sense, an ROC curve being closer to the upper left corner accounts for a better prediction performance, compared to another one that is closer to the lower right corner in the 2D space.

Since ROC curves are dedicated for binary classification, in our work, we transform the multi-class problem into binary classification using the following method. Given one among the nine considered categories, i.e., C_i, we consider it as the positive class, while all the remaining eight categories are considered as a single class, or negative. That means, if an instance is correctly classified into its category, we count it as a true positive. In contrast, if it is classified into any of the negative classes, then it is counted as a false negative. In this way, we are able to derive an ROC as like for binary classification.

23

Configuration		C$_1$	C$_2$	C$_3$	C$_4$	C$_5$	C$_6$
ML framework		Keras	Keras	Keras	Keras	Keras	TensorFlow
Encoding		uni-gram	bi-gram	n-gram	uni-gram	uni-gram	uni-gram
Layer #1	Input tensor	$94 \times 94 \times 1$	$148 \times 148 \times 1$	$184 \times 184 \times 1$	$96 \times 96 \times 1$	$96 \times 96 \times 1$	$96 \times 96 \times 1$
	Conv. kernel	$3 \times 3 \times 1 \times 16$	$3 \times 3 \times 1 \times 16$	$3 \times 3 \times 1 \times 16$	$3 \times 3 \times 1 \times 16$	$3 \times 3 \times 1 \times 16$	$5 \times 5 \times 1 \times 32$
	Pooling	Maxpool 2×2	Maxpool 2×2	Maxpool 2×2	Maxpool 2×2	Maxpool 2×2	Maxpool 2×2
	Dropout	0.5	0.5	0.5	0.5	0.5	0.5
	Act. Function	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
Layer #2	Input tensor				$48 \times 48 \times 1 \times 16$	$48 \times 48 \times 1 \times 16$	$48 \times 48 \times 1 \times 64$
	Conv. kernel				$3 \times 3 \times 1 \times 32$	$3 \times 3 \times 1 \times 32$	$5 \times 5 \times 32 \times 64$
	Pooling				Maxpool 2×2	Maxpool 2×2	Maxpool 2×2
	Dropout				0.5	0.5	0.5
	Act. Function				ReLU	ReLU	ReLU
Layer #3	Input tensor					$24 \times 24 \times 1 \times 32$	$24 \times 24 \times 1 \times 64$
	Conv. kernel					$3 \times 3 \times 1 \times 64$	$5 \times 5 \times 64 \times 64$
	Pooling					Maxpool 2×2	Maxpool 2×2
	Dropout					0.6	0.6
	Act. Function					ReLU	ReLU
1st FC layer	# of nodes	32	32	32	128	128	256
	Act. Function	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
2nd FC layer	# of nodes	9	9	9	9	9	9
	Act. Function	Softmax	Softmax	Softmax	Softmax	Softmax	Softmax
Number of epochs		15/30	15/30	15/30	15	15	15
Loss function [40]		Cross Entropy	Cross Entropy	Cross Entropy	Cross Entropy	Cross Entropy	Cross Entropy
Optimizer [41]		Adam	Adam	Adam	Adam	Adam	Adam

Table 2: Experimental configurations.

4.4. Configurations

In order to thoroughly study the performance of MEMOCNN, we considered different experimental settings as given in Table 2. The configurations are built by customizing various parameters, i.e., encoding schemes, the number of layers (depth), the number of filters (width), and kernel's size. Each change in the parameters aims to investigate possible effects on the final classification. By each configuration, the first layer's input tensor corresponds to the input *image*, and its size may vary depending on the encoding scheme. Three configurations namely C_1, C_2, C_3 are used to address $\mathbf{RQ}_1$ and $\mathbf{RQ}_3$. Four configurations, namely C_1, C_4, C_5, and C_6, are used to answer $\mathbf{RQ}_2$. To study the performance of MEMOCNN in comparison to AURORA, we set up only one layer for MEMOCNN, and this is the reason why there is no information for Layer #2 and Layer #3 in Table 2, i.e., the corresponding cells are left blank. The number of nodes for the 2nd fully-connected layer corresponds to the number of categories, as shown in Fig. 8(a), and this explains why the number is always 9 throughout the table.

All the configurations have been built using Keras except C_6 that has been made with TensorFlow since the framework allows for more control over the network. It is worth noting that the Dropout value has been empirically set to 0.5 or 0.6 [31]. By all, except the 2nd fully-connected layer, the activation function ReLU is used. The Softmax function is selected for the output layer as it is suitable for classifying inputs into multiple categories (see Sect. 2.4). Moreover, we used the cross entropy loss function [40] and the Adam optimizer [41] throughout the experiments as these two functions have been widely exploited in various classification scenarios. We are going to give more details for each configuration in Sect. 5.

5. Experimental Results

Although MEMOCNN supports both offline and online training (see Fig. 3.1), in the scope of this paper, we opt for the latter since Google offers a robust platform that helps accelerate the computation with MEMOCNN. In particular,

25

⁵²⁵ we uploaded the related metadata to Google Drive to feed the network, which
is then executed on Google Colab. This section reports and analyzes the results
in detail by answering the research questions in Sect. 4.1.

5.1. RQ_1: How effective is MEMOCNN at classifying the considered categories?

First, we ran experiments following the ten-fold cross-validation technique,
⁵³⁰ using three configurations C_1, C_2, C_3, and the number of epochs was set to 15
(cf. Table 2). We computed class-wise average precision, recall, and F_1-score
and show them in Table 5.1. The results in Table 5.1 demonstrate that all
the considered encoding schemes help gain a considerably high precision. For
example, the highest precision is 1.000, obtained for Class 1 by both C_1 and
⁵³⁵ C_2. The lowest precision is 0.736 by Class 4. Concerning recall, we can see that
uni-gram brings the best prediction for Class 1 and 2. With respect to macro-
F_1 and balanced accuracy, we also see that C_1 yields the best performance.
Altogether, it is evident that using uni-gram as the encoding scheme (C_1)
helps obtain the best classification performance compared to bi-gram (C_2) and
⁵⁴⁰ n-gram (C_3).

		Class (see Fig. 8 for greater detail)								
	Metric	0	1	2	3	4	5	6	7	8
C_1 (uni-gram)	Precision	0.924	1.000	0.950	0.875	0.904	0.968	0.988	0.937	0.868
	Recall	1.000	1.000	0.780	0.843	0.990	0.857	0.969	0.969	0.901
	F_1	0.960	1.000	0.860	0.857	0.945	0.909	0.978	0.952	0.884
	macro-F_1	0.927								
	Accuracy	0.897								
C_2 (bi-gram)	Precision	0.883	1.000	0.923	0.983	0.736	0.936	0.963	0.867	0.885
	Recall	0.852	0.682	0.670	0.921	0.913	0.738	0.827	0.899	0.676
	F_1	0.867	0.810	0.748	0.951	0.815	0.805	0.889	0.882	0.767
	macro-F_1	0.842								
	Accuracy	0.782								
C_3 (n-gram)	Precision	0.900	0.958	0.885	0.851	0.837	0.936	0.964	0.891	0.852
	Recall	0.859	0.817	0.791	0.901	0.892	0.822	0.890	0.722	0.663
	F_1	0.879	0.882	0.835	0.875	0.836	0.875	0.925	0.797	0.740
	macro-F_1	0.853								
	Accuracy	0.821								

Table 3: Precision, recall, and F_1-score.
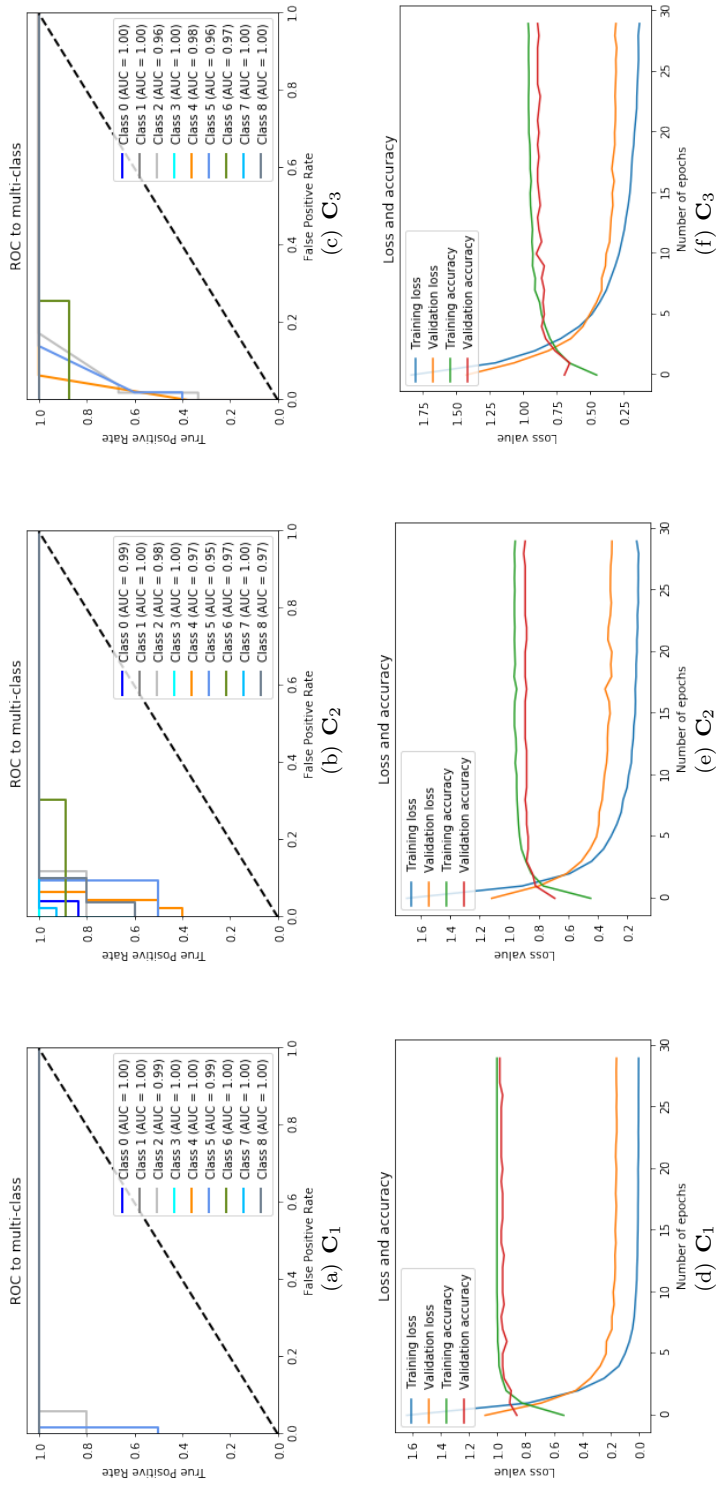
(a) C_1 (b) C_2 (c) C_3

(d) C_1 (e) C_2 (f) C_3

Figure 9: Receiver operating characteristic, loss and accuracy curves (training set = 80%, validation set = 20%).
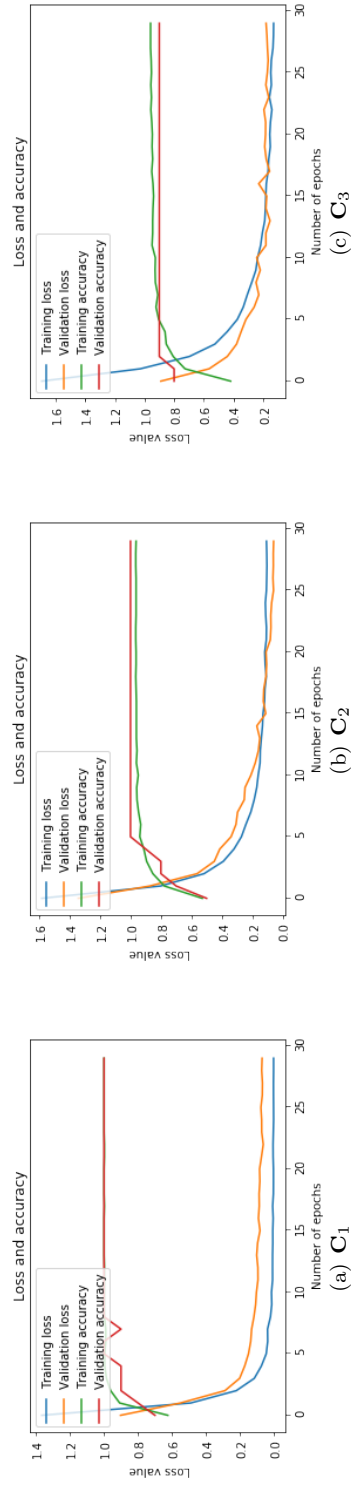
(a) C_1 (b) C_2 (c) C_3

Figure 10: Accuracy and loss (training set = 95%, validation set = 5%).

Afterward, we performed additional experiments to study how well MEMO-CNN classifies the metamodels into the classes presented in Fig. 8. Furthermore, to better observe the change in performance, the number of epochs was increased to 30.

545   For this evaluation, ROC and AUC are employed to measure the outcomes as they give a greater insight into each classified category [42]. Figures 9(a), 9(b), and 9(c) depict the ROC curves together with their corresponding AUC values obtained from the experiments for all the considered categories. For uni-gram, we see that most of the ROC curves are close to the
550   upper left corner (0,1), implying that the tool can properly classify the metamodels into their original categories. Similarly, it is evident that using bi-gram and n-gram to encode the metamodels also brings benefits to the classification, though a bit worse compared to using uni-gram. In particular, MEMoCNN gets AUC values ranging from 0.95 to 1.0 for all the categories, indicating a
555   high match rate.

Given an ML model, it is essential to understand whether it is either *underfitting* or *overfitting*, since none of them is useful for the final deployment. A model is considered to be underfitting if it fails to learn from data. In contrast, if it learns too well on the training data but performs poorly on the validation
560   data, then it becomes overfitting. To combat overfitting, the common practice is to introduce regularization, for example by adding L1 and L2. However, in most cases, we realized that these methods did not seem to work as expected, e.g., we still witnessed overfitting here and there. We anticipate that this happened due to the small dataset used in our evaluation. It is our assumption
565   that the countermeasures to mitigate overfitting take effect only when there is a larger number of metamodels.

To investigate if MEMoCNN can be *good fit*, i.e., it is neither underfitting nor overfitting, starting from the data divided by the ten-fold cross-validation technique, we further split each training fold into two parts: 80% for training
570   and 20% for validation. The accuracy and loss recorded during the experiments are shown in Fig. 9(d), 9(e), and 9(f). We see that while the training accuracy

28

and validation accuracy are almost concordant with each other, the training loss and validation loss are not. A common pattern for the validation plots is that after a dramatic fall, they start to decrease slowly once a certain threshold has been reached, e.g., after three epochs. Furthermore there is always a gap between the plot of validation loss and that of the training loss. In other words, the model appears to be slightly overfitting by the current configuration. We assume that this happens, again, due to the limited amount of data used for training [20]. Thus, to further examine the model, we increased the amount of data for training by specifying the following ratio: 95% of the data is used for training and only 5% is used for validation. The resulted learning curves for this setting are depicted in Fig. 10(a), 10(b), and 10(c). In these figures, a clear pattern can be seen: The curves for validation loss and training loss stay close and they are nearly identical. In this respect, we come to the conclusion that with more data used for training, the model becomes less overfitting and almost good fit.

The dataset used in the present paper is considerably small and imbalanced. While we see that some types of neural networks perform well on small dataset, e.g., feed-forward neural networks, we believe that the advantages of a CNN for the classification of metamodels can be better showcased, only if there is a fairly large as well as balanced dataset. In this sense, it is important to experiment on a dataset with more metamodels for each category, and we consider the issue as our future work.

> **Answer to RQ$_1$:** MEMOCNN effectively classifies the testing metamodels into their original categories. When it comes to a larger dataset for training, the system turns to be good fit.

5.2. RQ$_2$: Which network configuration contributes to a better performance?

A CNN can be characterized with different dimensions, as shown in Fig. 6: Depth (D) is the number of layers, including the fully connected ones; Width (W) corresponds to the number of filters (kernels) for each convolutional layer.

In recent years, several attempts have been made to increase the prediction performance of a CNN by extending it in different directions [43].

The research question is particularly relevant for investigating the influence of the network configurations. In particular, we study whether an increase in the number of layers, as well as the number of filters, helps improve the overall accuracy on the given dataset. To this end, four configurations are considered, namely C_1, C_4, C_5, and C_6, as shown in Table 2. For the sake of simplicity, in these configurations, we used uni-gram as the encoding scheme, as it has demonstrated itself, through $\mathbf{RQ}_1$, to obtain the best performance among others. As mentioned before, C_1 consists of only one convolutional layer. By C_4, there are two convolutional layers, and the width of the first and the second layer is 16 and 32, respectively. Compared to C_4, C_5 has one more layer with width 64. Finally, C_6 is one layer deeper than C_5, and it is also wider by all the convolutional layers. By the first layer, there are 32 filters, while the second and the third layer consist of 64 filters. For the 2nd fully-connected layer, we used twice the number of nodes compared to C_5, i.e., 256 instead of 128 nodes. Furthermore, the size of the filters is also larger than that of the other configurations: C_6 employs filters of size 5×5, instead of size 3×3 as by C_4 and C_5. Altogether, it is evident that the network by C_6 is not only *wider* and *deeper*, but also *bigger* (in filters' size) in comparison with the other experimental settings. To further study $\mathbf{RQ}_1$, different amounts of data are considered. First, we used 80% of the original training data for training and 20% for validation. Second, 100% of each training fold is used for training, no data is exploited for validation.

The final accuracies we got for the considered configurations are shown in Fig. 11. Overall, the figure suggests that using a more complex network causes a reduction in accuracy. For example, with C_4, the accuracy is 90.29%, which is higher than that of C_5 and C_6, i.e., 89.20%, and 85.17%, respectively. This is interesting since we expect that a broader and deeper network would bring in a performance gain [43]. However, the results in this experiment clearly show
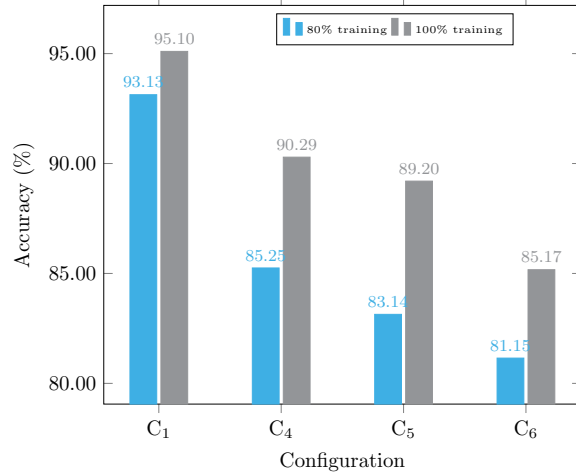
30

Figure 11: Accuracy for C_1, C_4, C_5, and C_6.

that the converse is true. We are going to give some explanations for such an outcome in Sect. 6.1.

The results reported in Fig. 11 further support our findings in $\mathbf{RQ}_1$: MEM-OCNN is more effective when it is fed with more training data. In Fig. 11, given each configuration among C_1, C_4, C_5, and C_6, when we compare the first column with the second one, we see that using more data to feed MEMOCNN induces a higher accuracy. Take as an example, for C_5, the accuracy increases from 83.14% to 89.20% when more metamodels are used to train MEMOCNN. Furthermore, we spot a remarkable pattern: complex network structures favor more data. In particular, when we move from the left to the right side of Fig. 11, the gap between the performance of the two bars representing different levels of input data becomes more significant. For instance, by C_1, the accuracies are 93.13% and 95.10%, meanwhile by C_6, the corresponding scores are 81.15% and 85.17%. This indicates that deeper and broader networks suffer a setback when they are fed with a small amount of data. Altogether, we conclude that complex networks are supposed to achieve a better performance if we incorporate more data for the training process.

31

> **Answer to RQ$_2$**: An increase either in the network's width or depth reduces MEMOCNN's prediction accuracy on the given dataset. Given that a small amount of data is used to train complex networks, such a reduction becomes more evident.

5.3. RQ$_3$: How does MEMOCNN compare with AURORA?

To aim for a reliable comparison with AURORA, we exploited the replication package which is already available online.[8] Furthermore, we ran the experiments with AURORA using the settings presented in the original paper [9] that yield the best performance. For MEMOCNN, three configurations have been employed, i.e., C$_1$, C$_2$, and C$_3$ as specified in Table 2. Moreover, the ten-fold cross-validation technique is applied on both systems: the dataset is divided into ten equal parts, and the validation is done ten times: each time one fold is used for testing, while the other nine folds are used for training.

The accuracy as well as the precision, recall, and macro-F$_1$ scores obtained by both systems for different encoding schemes, i.e., uni-gram, bi-gram, and n-gram are shown in Table 4. The table demonstrates that the baseline achieves a good performance as it has been previously claimed [9]. In particular, AURORA has an accuracy being larger than 82% for all encoding schemes. Furthermore, our experiments also confirm that using uni-gram to feed AURORA yields the best accuracy, compared to using bi-gram and n-gram. In comparison with the baseline, MEMOCNN achieves a better accuracy with respect to all three encoding schemes. When encoding with uni-gram, our proposed approach gets an accuracy of 93.72% while the corresponding score of AURORA is 92.25%. Similarly, for other encodings, MEMOCNN outperforms AURORA in terms of accuracy.

We consider the precision, recall, macro-F$_1$ scores by both AURORA and MEMOCNN as shown in Table 4. With C$_1$, MEMOCNN can provide better precision and recall, for example MEMOCNN gets a recall of 0.95 compared to

---

[8]https://github.com/MDEGroup/AURORA

32

0.89 by AURORA. This also applies to the outcomes obtained by using bi-gram to provide input. As a consequence, our proposed approach gets a better sensitivity than AURORA does. The final macro-F_1 scores demonstrate that MEMOCNN achieves superior performance than that of the baseline. By C_3, MEMOCNN outperforms AURORA by all metrics. For example, MEMOCNN gets 0.87, 0.84, and 0.86 as precision, recall, and macro-F_1, respectively; while the corresponding scores by AURORA are 0.75, 0.62, and 0.69, respectively.

The results obtained by our approach appear to be encouraging at first sight. However, by carefully examining the outcomes presented in Table 4, we noticed that the difference in performance between the two approaches is not significantly big. For instance, with accuracy, in comparison with AURORA, MEMOCNN gets more than 1% and less than 2% performance gain typically, e.g., 82.23% compared to 81.20% (with n-gram), or 85.11% compared to 83.12% (with bi-gram). The same trend can also be witnessed with other quality indexes, i.e., precision, recall, and macro-F_1. It is necessary to ascertain the cause of this outcome, and thus we came across the following question: "*Is* MEMOCNN *only capable of narrowly surpassing AURORA, or is it because of the dataset?*" If the former were the case, then the novelty of MEMOCNN would become questionable, as no-one is willing to sacrifice the simplicity and compactness of AURORA, as shown in Sect. 2.3, to get a moderate gain in performance. However, our intuition is that the outcome happens due to the

| | System | Configuration | | |
		C_1	C_2	C_3
Precision	AURORA	0.92	0.84	0.75
	MEMOCNN	**0.95**	**0.88**	**0.87**
Recall	AURORA	0.89	0.75	0.62
	MEMOCNN	**0.95**	**0.85**	**0.84**
macro-F_1	AURORA	0.92	0.81	0.69
	MEMOCNN	**0.95**	**0.86**	**0.86**
Accuracy (%)	AURORA	92.25	83.12	81.20
	MEMOCNN	**93.72**	**85.11**	**82.23**

Table 4: Precision, recall, F_1-score, and accuracy.

33

considerably small dataset, and thus there is limited data available for training. In fact, a CNN generally consumes a large amount of data to tune its internal parameters [30, 44]. Furthermore, as we have shown in $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$, MEMOCNN favors a bigger dataset for obtaining a better performance.

System	Testing-Training				
	50%-50%	40%-60%	30%-70%	20%-80%	10%-90%
AURORA	85.20	84.23	89.25	89.14	92.52
MEMOCNN	**89.32**	**91.38**	**92.15**	**94.50**	**96.10**

Table 5: Accuracy for various hold-out settings (%).

Thus, to validate our hypothesis, we conducted additional experiments by specifying different ratios of testing to training data through hold-out validation [45]. Moreover, to simplify the comparison, we exploited only uni-gram to provide input for both systems since the encoding facilitates the best performance among the other encoding schemes as we already showed by means of $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$. In particular, we increased by 10% for training data per each experiment, resulting in five different settings, as given in Table 5 which reports the results we got for different hold-out settings. While in general, AURORA gains an increase in performance, albeit small, when we change the amount of training data, it is clear that MEMOCNN outperforms AURORA by all the considered configurations. The most significant difference in accuracy is seen when we set 40% of the dataset as testing and the remaining 60% as training, i.e., AURORA gets 84.23%, and MEMOCNN gets 91.38% correctly classified metamodels. Furthermore, MEMOCNN obtains better accuracy if there is more data available for training. Starting from 89.22%, the accuracy gradually grows up to 96.10% when 10% of the data is used for testing, and 90% is used for training.

By measuring the execution time for both systems, we realized that MEMOCNN is more efficient in timing compared AURORA, i.e., 82 seconds to 90 seconds for the same unit of input data. However, this is only a marginal difference, and this happens due to the fact that we have a limited amount of data.

34

Generally, feed-forward neural networks will be more time consuming if we run them on a larger dataset, i.e., containing more metamodels. This is the reason why no subtle difference between the timing performance of AURORA and MEMOCNN is seen. Meanwhile, CNNs can maintain a linear increase in timing for a denser dataset, thanks to its convolution operations, which attempt to reduce the number of pairwise multiplications. Thus, we suppose that the difference in timing will be more evident given a dataset with a large number of metamodels.

> **Answer to RQ_3**: MEMOCNN outperforms AURORA with respect to accuracy, precision, recall, macro F_1, and timing. Furthermore, MEMOCNN improves its prediction performance when being trained with more data.

6. Discussions

The lessons learned from the experiments are discussed together with future developments in Sect. 6.1. The next section discusses the threats to validity.

6.1. Lessons learned and provisions

The experience of preparing the experimental evaluation and the corresponding results provided us with invaluable insights. The most remarkable lessons include:

▷ *Data is the cure.* By considering RQ_1 and RQ_2 together, we realized that data augmentation helps tackle two different issues: low accuracy and overfitting. MEMOCNN is less overfitting and more accurate if being fed with more training data. Our work also confirms that adding data is an effective measure to combat overfitting [44].

▷ *Deeper and wider networks can be useful down the road.* The results in RQ_2 show that on the considered dataset (see Sect. 4.2), MEMOCNN configurations related to complex networks do not bring any improvements in performance. This lack of progress is understandable at the light of the following arguments. Deepening allows a network to approximate the target function with increased nonlinearity, thereby getting better feature representations. Nevertheless, it

35

₇₄₅ also adds up to the complexity of the network, which makes the network more difficult to optimize and susceptible to overfitting. For example, given a training metamodel, the model memorizes well the target class, and thus it fails to generalize from data. So a question arises at the moment: "*Are deeper and wider networks not useful at all?*" The experimental results in Fig. 11 give

₇₅₀ evidence that complex networks suffer a setback given a small amount of data used for training. Moreover, as it has been found out by related studies [30, 44], CNNs favor more data due to the massive number of parameters. This capability essentially means that deeper/wider network structures can be beneficial under certain circumstances. In summary, the answer to the above question

₇₅₅ is: "*Yes, deeper and wider networks are useful to a big dataset for enhancing prediction accuracy.*"

▷ *Combination of AURORA and* MEMOCNN. To the best of our knowledge, the proposed approach is the first attempt that represents metamodels using matrices and exploits a convolutional neural network to classify them. Though

₇₆₀ MEMOCNN obtains a better performance compared to AURORA, we do not undermine the elegance of the baseline. The tool is suitable for working with a small dataset, and it obtains an excellent performance. In this respect, we suppose that a combination of both AURORA and MEMOCNN would be a complementary system.

₇₆₅ Apart from the lessons, based on the current results, we envisage that there are the following future developments.

▷ *Application to classify documents.* The approach proposed by Kim [46] works on sentences; however it may not apply to long documents since the representation matrix becomes enormous, which in turn places a burden on the compu-

₇₇₀ tational performance. The fact that MEMOCNN performs well on metamodels implies that we can use it to classify documents. First, uni-gram, bi-gram, and n-gram are used to parse documents, as it is done with metamodels, then MEMOCNN can be used to classify the metadata. In a broader sense, our proposed approach can also be applied to group Stack Overflow posts into categories.

785 ▷ *Incorporating more input dimensions.* During the evaluation, we made use of one-channel input tensors. However, we could think of extending it to more channels by incorporating various features. Referring to the Computer Vision domain, images are typically represented using three color channels, namely R, G, B (see Fig. 2). We anticipate the usage of a 3D input tensor by making use

780 of the three encoding schemes together, i.e., uni-gram, bi-gram, and n-gram. The final aim is to better capture the features of a metamodel by taking into consideration different representation schemes.

▷ *Transfer learning.* For CNNs, it is crucial to collect enough data for training. However, labeled data can only be obtained through a manual process, which is

785 both time-consuming and prone to error. To this end, transfer learning has been devised as a practical solution to export the knowledge extracted from a mature source domain to a novice target domain [47]. Transfer learning facilitates the re-use of convolution weights from a model trained on large datasets for training new models with a limited amount of labeled items. Compared to using only

790 random weights (as done with MEMOCNN in this paper), transfer learning is still useful even when the target domain is quite different from the original one where the weights are obtained [48]. We suppose that the application of transfer learning to MEMOCNN is beneficial to the final classification since our representation scheme allows for the transformation of a metamodel into

795 a picture-like format. This hypothesis needs to be properly validated, and we consider it as future work. Furthermore, once we have a larger dataset, we would be interested in embedding an auto-encoder, as this would possibly enhance the classification performance.

6.2. Threats to validity

800 In this section a discussion of validity threats, which might harm the experimental evaluation is given.

Internal validity. Such threats are the internal factors that may affect the outcomes. A probable threat that could be seen through the results obtained for the categories with a considerably low number of items for training and

805 testing. While the threat is partly mitigated by some of the categories in the

37

given dataset as they contain a significant number of metamodels, we believe that the threat can be completely eliminated only if there is a fairly large as well as balanced dataset. In this sense, it is necessary to consider a bigger dataset with more metamodels for each category, and we consider the issue as our future work. Another threat is related to the coverage of metamodel constraints. The definition of a metamodel is often endowed with constraints that impose additional properties to the conforming models. In order to consider constraints in the proposed classification technique, the metamodel encoding should encompass constraints as well. However, while it should not pose too many difficulties, it must be investigated whether *(i)* it is possible to distinguish, for instance, two versions of the same metamodel, each associated with a different constraint, and *(ii)* it makes sense at all because a constraint might not have enough informative content to the end of the classification.

External validity. The main threat to *external validity* pertains to the generalizability of our findings, i.e., if they are still valid outside the scope of this study. We confronted the issue by evaluating AURORA and memoCNN using different validation scenarios, aiming to simulate an actual usage of the systems. Moreover, we also performed different trials of the same experiments to examine if there is any subtle difference among the outcomes obtained by each trial. Overall, we have seen that the obtained results are consistent with those presented in this paper.

Construct validity. The threats are related to the simulated setting used to evaluate the tools. The evaluation has been conducted on a training set and a test set, modeling a real application scenario. Depending on the purpose, we used two different settings, namely hold-out validation and ten-fold cross-validation. Furthermore, to aim for a reliable comparison of memoCNN with AURORA, we made use of the original implementation of AURORA without touching its internal design.

Conclusion validity. Validity threats may be related to whether the experiment methodology is related to the outcome, or there may also be other

38

parameters that may have influenced it. The evaluation metrics, i.e., accuracy, precision, recall, F_1-score, ROC, and AUC might pose a threat to conclusion validity. The same metrics were employed to evaluate both systems as a mitigation measure.

7. Related Work

The primary technical advantages claimed by the approach presented in this paper are related to the adoption of ML techniques in MDE. Thus, we review some of the most notable related work on the adoption of such cognitive approaches in the context of MDE, and the categorization of models. Afterward, we discuss some prominent applications of CNNs, as well as the attempts made to improve CNNs' prediction performance by dimension extension and transfer learning.

7.1. Machine Learning in MDE

The MDE community has made considerable progress in recent years as regards adopting various ML algorithms to solve different issues [49, 50]. Kusmenko *et al.* [51] implement *MontiAnna*, a holistic deep learning modeling framework where deep neural networks can be designed, trained, and integrated into Component & Connector systems. Afterward, *MontiAnna* has been extended using Reinforcement Learning (RL) [52]. By exploiting RL algorithms, two systems [53, 54] can repair errors in a set of broken models, which have been introduced by conflicting changes. Various studies [55, 56] use ML techniques to automatically infer model transformation rules from sets of source and target models. Meta-learning is a technique that aims at using ML itself to automatically learn the most appropriate algorithms and parameters for an ML problem. Hartmann *et al.* [57] demonstrate the parallelism between metamodeling and meta-learning, and they envision a notation for describing meta-learning processes.

Learning algorithms have been successfully applied to solve different problems in the MDE scenario. Nevertheless, the adoption of cognitive techniques

39

⁸⁶⁵ in this domain does not seem to have received attention being commensurate with its potential. In this respect, MEMOCNN proved to advance the existing approaches in the classification of metamodels.

7.2. Categorization of metamodels

Classification and clustering techniques have been employed to categorize ⁸⁷⁰ metamodels for a diversity of purposes, not excluded reuse [8, 58, 59]. In particular, an agglomerative hierarchical clustering technique has been employed to organize metamodels in repositories [8] automatically. Hierarchical clustering techniques are the main engine to perform metamodel comparison, analysis, and visualization [58, 59]. In these approaches, metamodel named elements ⁸⁷⁵ are converted into lexical terms and represented as a vector space model. This encoding enables the usage of hierarchical clustering algorithms to group similar objects and represent it as a dendrogram. The approaches mentioned above [58, 59] share the same intent of this paper even though they are based on unsupervised learning techniques, whereas our work follows the supervised learning ⁸⁸⁰ strategy. Lopez *et al.* [60] present a domain-oriented approach for software requirements reuse. First, requirements captured from semiformal diagrams are injected into models, which are then analyzed to check for any probable quality issues. Finally, a mechanism to cluster the requirements is applied to allow the domain requirements patterns to be identified and reused. *Reuser* [61] is a tool ⁸⁸⁵ used to automatically retrieve related UML artifacts and propose them to the modeler. A search algorithm is applied to find similar artifacts by classifying them into a concept lattice. Then it compares the input artifact query against the concept lattice, to retrieve the closest set of matching artifacts.

Jiang *et al.* [62] discuss different characteristics of UML metamodel exten-⁸⁹⁰sion mechanisms according to a four-level classification. Each level of metamodel extension has different features in (contrasting) aspects such as readability, expression capability, use scope, and tool support. The paper aims at providing modelers with a reliable theoretical base for selecting the right level to extend the metamodel to find the right trade-off of the above aspects. A

40

classification model for UML stereotypes has been developed [63], where the artifacts are analyzed according to their potential to alter the syntax and semantics of the base language to be able to control their application practice. Feature-based criteria for classifying approaches according to the type of spatial relation involved have been widely discussed [64].

UML class diagrams are commonly used to specify the designs of systems, which then can be used to guide the construction of software [65]. There are two main types of UML diagrams: *(i)* FwCD diagrams are hand-made as part of the forward-looking development process; *(ii)* RECD diagrams are reverse engineered from the source code. Osman *et al.* [65] propose a supervised machine learning algorithm to build an automated classifier for classifying UML digrams. The approach can be used to classify a diagram into either an FwCD or an RECD, and it has been evaluated using a test set of 999 class diagrams obtained from open source projects. By running experiments with different machine learning algorithms, the authors concluded that Random Forest is the most suitable algorithm for their classification tasks.

To our knowledge, AURORA [9] is the first effort to classify metamodels using a feed-forward neural network. Despite its simplicity, the tool is efficient, and on a considerably small dataset, it classifies the metamodels, obtaining high accuracy. In this work, we improved AURORA by employing a CNN, which is supposed to learn better on a large dataset.

7.3. Application and extension of CNNs

CNNs were originally designed to work with images, and thus they have been widely used to solve various recognition tasks. To name but a few, CNNs have been applied to detect fruits [34, 66], to classify images [14, 20], allowing for the transfer of well pre-trained hyper-parameters to a new application. The success of CNNs in Computer Vision is a source of inspiration for applications in other domains. For instance, CNNs have been exploited to classify DNA sequences [67], sentences [46], or to process natural language [18]. To the best of our knowledge MEMOCNN is the first attempt to bring CNNs into a new

₉₂₅ domain, i.e., MDE [1]. By developing MEMOCNN on top of a convolutional neural network, we inherit a well-founded background of knowledge to solve the problem of metamodel classification. This also paves the way for deploying transfer learning, which allows for increasing both efficiency and effectiveness.

As seen in Fig. 6, a CNN can be characterized with width, depth, resolu-₉₃₀ tion, and filter size. So far, various approaches have been proposed to boost up the performance of a CNN by extending it in these dimensions. Two of the most recent studies of this type are EfficientNet and MixNet. The EfficientNet family [43] improves performance by extending the original network in depth, width, and resolution. It decreases the learning rate of the network, given that ₉₃₅ the number of layers reaches a high value. By relying on width scaling, it is possible to capture features more definitely, easing the training process. However, the scalability of the approach is questionable because of reduced precision over high-level features. The memory cost represents the main hindrance as the network achieves a higher accuracy with a high-quality resolution. Like ₉₄₀ this, the compound scaling method does not expose the limitations of traditional techniques that tend to act on a single dimension at once. MixNet [68] is a family of neural networks that combines multiple kernel sizes in a single convolution, thus allowing for recognizing various types of patterns from input images. MixNet has demonstrated itself to outperform some state-of-the-art ₉₄₅ CNNs significantly. As we already pointed out in $\mathbf{RQ}_3$, and Sect. 6.1, the expansion of MEMOCNN in different dimensions may bring benefits. However, the performance depends much on the input data: given a small amount of training data, enlarging the network turns out to be counterproductive.

7.4. Transfer learning

₉₅₀ The technique has been widely used by imaging applications to improve the learning performance of CNNs [47, 48]. Several studies adopt weights pretrained using the ImageNet dataset, which consists of a huge number of images covering various categories [69]. Some notable studies that make use of ImageNet are: the work by Razavian *et al.* to classify images [70], or the one by

Karpathy *et al.* [71] to generate natural language descriptions for images. The accuracy obtained by employing ImageNet is better than that by using random learning. We assume that transfer learning would be beneficial to the training of MEMOCNN. Compared to using only random weights, as we have done with MEMOCNN in this paper, transfer learning is still helpful, also when the target domain is quite different from the original one where the weights have been obtained [48].

8. Conclusions and future work

MEMOCNN has been proposed as a novel approach to the classification of metamodels by adapting a deep learning technique that has gained considerable traction in other domains. The comparison with AURORA, a state-of-the-art baseline, on a manually classified dataset demonstrates that MEMOCNN obtains a better performance with respect to different quality metrics. Our future plan is to investigate whether transfer learning is beneficial to MEMOCNN by adopting pre-trained weights using the ImageNet dataset. Moreover, we believe that the performance of memoCNN can be better judged when there is more data for training. Thus, we will invest time and efforts to populate a larger dataset, aiming to empower the classification capability. We plan to collect metamodels from GitHub using the dedicated API, and then manually classify them by involving expert modelers to build a training set. Last but not least, we are going to deploy the framework to other domains, such as the classification of documents and Stack Overflow posts to support software development activities.

Acknowledgments

[1] D. C. Schmidt, Guest editor's introduction: Model-driven engineering, Computer 39 (2) (2006) 25–31. doi:10.1109/MC.2006.58.

[2] M. White, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, Toward deep learning software repositories, in: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, IEEE, 2015, pp. 334–345.

[3] J. Di Rocco, D. Di Ruscio, L. Iovino, A. Pierantonio, Collaborative repositories in model-driven engineering [software technology], IEEE Software 32 (3) (2015) 28–34.

[4] C. Hein, T. Ritter, M. Wagner, Model-driven tool integration with modelbus, in: Workshop Future Trends of Model-Driven Development, 2009, pp. 50–52.

[5] B. Karasneh, M. R. Chaudron, Online img2uml repository: An online repository for uml models., in: EESSMOD@ MoDELS, 2013, pp. 61–66.

[6] M. Koegel, J. Helming, EMFStore: a model repository for EMF models, in: 2010 ACM/IEEE 32nd International Conference on Software Engineering, Vol. 2, IEEE, 2010, pp. 307–308.

[7] R. Kutsche, N. Milanovic, G. Bauhoff, T. Baum, M. Cartsburg, D. Kumpe, J. Widiker, Bizycle: Model-based interoperability platform for software and data integration, Proceedings of the MDTPI at ECMDA 430.

[8] F. Basciani, J. Di Rocco, D. Di Ruscio, L. Iovino, A. Pierantonio, Automated clustering of metamodel repositories, in: S. Nurcan, P. Soffer, M. Bajec, J. Eder (Eds.), Advanced Information Systems Engineering, Springer International Publishing, Cham, 2016, pp. 342–358.

[9] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, A. Pierantonio, L. Iovino, Automated classification of metamodel repositories: A machine learning approach, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), 2019, pp. 272–282. doi:10.1109/MODELS.2019.00011.

44

[10] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, The MIT Press, 2016.

[11] I. Portugal, P. S. C. Alencar, D. D. Cowan, The use of machine learning algorithms in recommender systems: A systematic review, CoRR abs/1511.05263. `arXiv:1511.05263`.

[12] P. Domingos, A few useful things to know about machine learning, Commun. ACM 55 (10) (2012) 78–87. `doi:10.1145/2347736.2347755`.

[13] T. Wuest, D. Weimer, C. Irgens, K.-D. Thoben, Machine learning in manufacturing: advantages, challenges, and applications, Production & Manufacturing Research 4 (1) (2016) 23–45. `doi:10.1080/21693277.2016.1192517`.

[14] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[15] Z. Zhao, P. Zheng, S. Xu, X. Wu, Object detection with deep learning: A review, IEEE Transactions on Neural Networks and Learning Systems 30 (11) (2019) 3212–3232. `doi:10.1109/TNNLS.2018.2876865`.

[16] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015) 770–778.

[17] A. Nguyen, D. Kanoulas, D. G. Caldwell, N. G. Tsagarakis, Detecting object affordances with Convolutional Neural Networks, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016, pp. 2765–2770. `doi:10.1109/IROS.2016.7759429`.

[18] Y. Belinkov, J. Glass, Analysis methods in neural language processing: A survey, Transactions of the Association for Computational Linguistics 7

(2019) 49–72. doi:10.1162/tacl_a_00254.

1040   URL https://www.aclweb.org/anthology/Q19-1004

[19] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.
URL https://doi.org/10.1162/neco.1997.9.8.1735

[20] W. Rawat, Z. Wang, Deep convolutional neural networks for image classification: A comprehensive review, Neural Comput. 29 (9) (2017) 2352–2449. doi:10.1162/neco_a_00990.
URL https://doi.org/10.1162/neco_a_00990

[21] S. B. Kotsiantis, I. D. Zaharakis, P. E. Pintelas, Machine learning: a review of classification and combining techniques, Artif. Intell. Rev. 26 (3) (2006) 159–190. doi:10.1007/s10462-007-9052-3.
URL https://doi.org/10.1007/s10462-007-9052-3

[22] G. P. Zhang, Neural networks for classification: a survey, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 30 (4) (2000) 451–462.

[23] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, K. Shaalan, Speech recognition using deep neural networks: A systematic review, IEEE Access 7 (2019) 19143–19165.

[24] K. Sundararajan, D. L. Woodard, Deep learning for biometrics: A survey, ACM Comput. Surv. 51 (3). doi:10.1145/3190618.
URL https://doi.org/10.1145/3190618

[25] F. Sebastiani, Machine learning in automated text categorization, ACM Comput. Surv. 34 (1) (2002) 1–47. doi:10.1145/505282.505283.
URL https://doi.org/10.1145/505282.505283

[26] A. Hartmann, R. Lienhart, Automatic classification of images on the web, Vol. 4676, 2002, pp. 31–40. doi:10.1117/12.451108.

[27] M. Ceci, F. Esposito, M. Lapi, D. Malerba, Automated classification of web documents into a hierarchy of categories, in: Intelligent Information Processing and Web Mining, Springer, 2003, pp. 59–68.

[28] Z. Shi, L. He, K. Suzuki, T. Nakamura, H. Itoh, Survey on neural networks used for medical image processing, International journal of computational science 3 (1) (2009) 86.

[29] J. Hartmann, J. Huppertz, C. Schamp, M. Heitmann, Comparing automated text classification methods, International Journal of Research in Marketing 36 (1) (2019) 20–38.

[30] S. B. Driss, M. Soua, R. Kachouri, M. Akil, A comparison study between MLP and convolutional neural network models for character recognition, in: N. Kehtarnavaz, M. F. Carlsohn (Eds.), Real-Time Image and Video Processing 2017, Vol. 10223, International Society for Optics and Photonics, SPIE, 2017, pp. 32 – 42. doi:10.1117/12.2262589.
URL https://doi.org/10.1117/12.2262589

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.

[32] M. F. Porter, et al., An algorithm for suffix stripping., Program 14 (3) (1980) 130–137.

[33] P. Juszczak, D. Tax, R. P. Duin, Feature scaling in support vector data description, in: Proc. asci, Citeseer, 2002, pp. 95–102.

[34] L. T. Duong, P. T. Nguyen, C. Di Sipio, D. Di Ruscio, Automated fruit recognition using EfficientNet and MixNet, Computers and Electronics in Agriculture 171 (2020) 105326. doi:https://doi.org/10.1016/j.compag.2020.105326.
URL http://www.sciencedirect.com/science/article/pii/S0168169919319787

[35] Önder Babur, A labeled Ecore metamodel dataset for domain clustering (Mar. 2019). doi:10.5281/zenodo.2585456.
URL https://doi.org/10.5281/zenodo.2585456

[36] C. V. Lopes, P. Maj, P. Martins, V. Saini, D. Yang, J. Zitny, H. Sajnani, J. Vitek, Déjàvu: a map of code duplicates on github, Proceedings of the ACM on Programming Languages 1 (OOPSLA) (2017) 1–28.

[37] M. Aly, Survey on multiclass classification methods, Neural networks (2005) 1–9.

[38] R. Kohavi, A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection, in: 14th International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., San Francisco, 1995, pp. 1137–1143.

[39] T. Fawcett, An introduction to roc analysis, Pattern Recogn. Lett. 27 (8) (2006) 861–874. doi:10.1016/j.patrec.2005.10.010.
URL https://doi.org/10.1016/j.patrec.2005.10.010

[40] Z. Zhang, M. Sabuncu, Generalized cross entropy loss for training deep neural networks with noisy labels, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems 31, Curran Associates, Inc., 2018, pp. 8778–8788.

[41] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.
URL http://arxiv.org/abs/1412.6980

[42] J. Davis, M. Goadrich, The Relationship Between Precision-Recall and ROC Curves, in: ICML '06: Proceedings of the 23rd international conference on Machine learning, ACM, New York, NY, USA, 2006, pp. 233–240.

48

doi:http://doi.acm.org/10.1145/1143844.1143874.

URL http://portal.acm.org/citation.cfm?id=1143874

[43] M. Tan, Q. Le, EfficientNet: Rethinking model scaling for convolutional neural networks, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, Vol. 97 of Proceedings of Machine Learning Research, PMLR, Long Beach, California, USA, 2019, pp. 6105–6114.

URL http://proceedings.mlr.press/v97/tan19a.html

[44] R. Yamashita, M. Nishio, R. K. G. Do, K. Togashi, Convolutional neural networks: an overview and application in radiology, Insights into Imaging 9 (4) (2018) 611–629. doi:10.1007/s13244-018-0639-9.

URL https://doi.org/10.1007/s13244-018-0639-9

[45] T. Borovicka, M. Jirina Jr, M. Jirina, P. Kordik, Selecting representative data sets, INTECH Open Access Publisher, 2012.

[46] Y. Kim, Convolutional neural networks for sentence classification, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1746–1751. doi:10.3115/v1/D14-1181.

URL https://www.aclweb.org/anthology/D14-1181

[47] K. Weiss, T. Khoshgoftaar, D. Wang, A survey of transfer learning, Journal of Big Data 3. doi:10.1186/s40537-016-0043-6.

[48] Z. Huang, Z. Pan, B. Lei, Transfer Learning with Deep Convolutional Neural Network for SAR Target Classification with Limited Labeled Data, Remote Sensing 9 (9). doi:10.3390/rs9090907.

URL https://www.mdpi.com/2072-4292/9/9/907

[49] Ö. Babur, M. R. Chaudron, L. Cleophas, D. Di Ruscio, D. Kolovos, Ammore 2018: First international workshop on analytics and mining of model repositories, in: CEUR Workshop Proceedings, Vol. 2245, 2018, pp. 778–779.

49

[50] L. Burgueño, A. Burdusel, S. Gérard, M. Wimmer, Mde intelligence19: First international workshop on artificial intelligence and model-driven engineering, 2019.

[51] E. Kusmenko, S. Nickels, S. Pavlitskaya, B. Rumpe, T. Timmermanns, Modeling and training of neural processing systems, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, 2019, pp. 283–293.

[52] N. Gatto, E. Kusmenko, B. Rumpe, Modeling deep reinforcement learning based architectures for cyber-physical systems, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2019, pp. 196–202.

[53] A. Barriga, A. Rutle, R. Heldal, Automatic model repair using reinforcement learning, in: Proceedings of Workshops co-located with MODELS 2018, Copenhagen, Denmark, October, 14, 2018., 2018, pp. 781–786.

[54] A. Barriga, A. Rutle, R. Heldal, Personalized and automatic model repairing using reinforcement learning, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2019, pp. 175–181.

[55] X. Dolques, M. Huchard, C. Nebut, P. Reitz, Learning Transformation Rules from Transformation Examples: An Approach Based on Relational Concept Analysis, in: 2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops, 2010, pp. 27–32.

[56] L. Burgueño, J. Cabot, S. Gérard, An lstm-based neural network architecture for model transformations, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), 2019, pp. 294–299. doi:10.1109/MODELS.2019.00013.

[57] T. Hartmann, A. Moawad, C. Schockaert, F. Fouquet, Y. Le Traon, Meta-modelling meta-learning, in: 2019 ACM/IEEE 22nd International Confer-

[50] L. Burgueño, A. Burdusel, S. Gérard, M. Wimmer, Mde intelligence19: First international workshop on artificial intelligence and model-driven engineering, 2019.

[51] E. Kusmenko, S. Nickels, S. Pavlitskaya, B. Rumpe, T. Timmermanns, Modeling and training of neural processing systems, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, 2019, pp. 283–293.

[52] N. Gatto, E. Kusmenko, B. Rumpe, Modeling deep reinforcement learning based architectures for cyber-physical systems, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2019, pp. 196–202.

[53] A. Barriga, A. Rutle, R. Heldal, Automatic model repair using reinforcement learning, in: Proceedings of Workshops co-located with MODELS 2018, Copenhagen, Denmark, October, 14, 2018., 2018, pp. 781–786.

[54] A. Barriga, A. Rutle, R. Heldal, Personalized and automatic model repairing using reinforcement learning, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2019, pp. 175–181.

[55] X. Dolques, M. Huchard, C. Nebut, P. Reitz, Learning Transformation Rules from Transformation Examples: An Approach Based on Relational Concept Analysis, in: 2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops, 2010, pp. 27–32.

[56] L. Burgueño, J. Cabot, S. Gérard, An lstm-based neural network architecture for model transformations, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), 2019, pp. 294–299. doi:10.1109/MODELS.2019.00013.

[57] T. Hartmann, A. Moawad, C. Schockaert, F. Fouquet, Y. Le Traon, Meta-modelling meta-learning, in: 2019 ACM/IEEE 22nd International Confer-

ence on Model Driven Engineering Languages and Systems (MODELS), 2019, pp. 300–305. doi:10.1109/MODELS.2019.00014.

[58] Ö. Babur, L. Cleophas, M. van den Brand, Hierarchical clustering of metamodels for comparative analysis and visualization, in: A. Wąsowski, H. Lönn (Eds.), Modelling Foundations and Applications, Springer International Publishing, Cham, 2016, pp. 3–18.

[59] O. Babur, L. Cleophas, Using n-grams for the Automated Clustering of Structural Models, in: B. Steffen, C. Baier, M. van den Brand, J. Eder, M. Hinchey, T. Margaria (Eds.), SOFSEM 2017: Theory and Practice of Computer Science, Springer International Publishing, Cham, 2017, pp. 510–524.

[60] O. Lopez, M. A. Laguna, F. J. García, Reuse based analysis and clustering of requirements diagrams, in: Pre-Proceedings of the Eighth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'02), 2002, pp. 71–82.

[61] W. N. Robinson, H. G. Woo, Finding reusable uml sequence diagrams automatically, IEEE Software 21 (5) (2004) 60–67. doi:10.1109/MS.2004.1331304.

[62] Y. Jiang, W. Shao, L. Zhang, Z. Ma, X. Meng, H. Ma, On the classification of uml's meta model extension mechanism, in: T. Baar, A. Strohmeier, A. Moreira, S. J. Mellor (Eds.), The Unified Modeling Language. Modeling Languages and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 54–68.

[63] S. Berner, M. Glinz, S. Joos, A classification of stereotypes for object-oriented modeling languages, in: R. France, B. Rumpe (Eds.), UML'99 — The Unified Modeling Language, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 249–264.

[64] P. Bottoni, A. Grau, A suite of metamodels as a basis for a classification of visual languages, in: 2004 IEEE Symposium on Visual Languages - Human Centric Computing, 2004, pp. 83–90. doi:10.1109/VLHCC.2004.5.

[65] M. H. Osman, T. Ho-Quang, M. Chaudron, An automated approach for classifying reverse-engineered and forward-engineered uml class diagrams, in: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2018, pp. 396–399.

[66] G. Gatica, S. Best, J. Ceroni, G. Lefranc, Olive fruits recognition using neural networks, Procedia Computer Science 17 (2013) 412–419, first International Conference on Information Technology and Quantitative Management. doi:https://doi.org/10.1016/j.procs.2013.05.053.
URL http://www.sciencedirect.com/science/article/pii/S1877050913001865

[67] G. Aoki, Y. Sakakibara, Convolutional neural networks for classification of alignments of non-coding RNA sequences, Bioinformatics 34 (13) (2018) i237–i244. arXiv:https://academic.oup.com/bioinformatics/article-pdf/34/13/i237/25098204/bty228.pdf, doi:10.1093/bioinformatics/bty228.
URL https://doi.org/10.1093/bioinformatics/bty228

[68] M. Tan, Q. V. Le, MixConv: Mixed Depthwise Convolutional Kernels, CoRR abs/1907.09595. arXiv:1907.09595.
URL http://arxiv.org/abs/1907.09595

[69] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, Int. J. Comput. Vision 115 (3) (2015) 211–252. doi:10.1007/s11263-015-0816-y.
URL http://dx.doi.org/10.1007/s11263-015-0816-y

[70] A. S. Razavian, H. Azizpour, J. Sullivan, S. Carlsson, Cnn features off-the-shelf: an astounding baseline for recognition., CoRR abs/1403.6382.

1235   URL   http://dblp.uni-trier.de/db/journals/corr/corr1403.
          html#RazavianASC14

    [71] A. Karpathy, L. Fei-Fei, Deep visual-semantic alignments for generating
         image descriptions, IEEE Trans. Pattern Anal. Mach. Intell. 39 (4) (2017)
         664–676. doi:10.1109/TPAMI.2016.2598339.
1240   URL https://doi.org/10.1109/TPAMI.2016.2598339