Classification of UML Diagrams to Support Software Engineering Education

José Fernando Tavares, Yandre M.G. Costa and Thelma Elita Colanzi

October 22, 2021

Classification of UML Diagrams to Support Software Engineering Education

José Fernando Tavares*, Yandre M. G. Costa*, Thelma Elita Colanzi*
*State University of Maringá, Maringá, Brazil.
fernando@booknando.com.br,{yandre, thelma}@din.uem.br

***Abstract*—There is a huge necessity for tools that implement accessibility in Software Engineering (SE) education. The use of diagrams to teach software development is a very common practice, and there are a lot of UML diagrams represented as images in didactic materials that need an accessible version for visually impaired or blind students. Machine learning techniques, such as deep learning, can be used to automate this task. The practical application of deep learning in many classification problems in the context of SE is problematic due to the large volumes of labeled data required for training. Transfer learning techniques can help in this type of task by taking advantage of pre-trained models based on Convolutional Neural Networks (CNN), so that better results may be achieved even with few images. In this work, we applied transfer learning and data augmentation for UML diagrams classification on a dataset specially created for the development of this work, containing six types of UML diagrams. The dataset was also made available as a contribution of this work. We experimented three widely-known CNN architectures: VGG16, RestNet50, and InceptionV3. The results demonstrated that the use of transfer learning contributes for achieving good results even using scarce data. However, there is still a room for improvement regarding the successful classification of the UML diagrams addressed in this work.**

***Index Terms*—Software Engineering Education, UML diagrams, Deep Learning, Transfer Learning, Assistive Technologies**

I. Introduction

Science, Technology, Engineering and Maths (STEM) undergraduate courses have often been cited as difficult to learn for people who have some form of visual impairment or blindness [1]. Several aspects of reading are easily accessible for students with vision impairments because screen reading technologies are able to manage the text. However, the problems arise when equations, graphs and diagrams are used.

Diagrams are very common in Software Engineering education. Particularly, the use of UML (Unified Modeling Language) [2] diagrams is of vital importance for teaching object-oriented techniques or more advanced concepts [3]. UML is a visual language designed to model large object-oriented software systems using a diagram notation. Furthermore, this language is the standard modeling language adopted by the Software Engineering industry [4]–[6].

Currently, UML diagrams use to be described and stored in the form of digital images. In addition, we know that the current educational context imposes increasing attention to the aspects of accessibility. So, it would be opportune to describe these diagrams in an alternative language (text, sound or physical devices) which facilitates access to information for blind or low vision people.

The task of image description for accessibility has been carried out with great effort by various content producers, producing high-quality tools that assist in the image description. To the best of our knowledge, the work of Bine et al. [7] is one of the few studies whose objective is to create a model that can be used in the construction of assistive tools for visually impaired students. However, their work only focus on the classification of diagrams related to Theory of Computation, not including UML diagrams.

A systematic literature review [8] summarizes studies whose goal is to turn diagrams accessible for blind or visually impaired from courses in the STEM field. Only a few studies (three among 26) address the UML diagrams, not considering the classification in the context of assistive technologies.

The few studies that automatically classify the images as UML diagrams perform a binary classification to determine if the image is a UML class diagram or a non-UML class diagram [9]–[12]. Despite UML class diagrams are the most used ones, there are other frequently used UML diagrams [4], [5]. So, there is a lack of studies about the classification of a comprehensive set of UML diagrams aiming at providing more specific solutions for the extraction of information from UML diagrams represented in the image format. We infer that the reason for this lack of studies happens because the practical application of modern machine learning techniques, such as deep learning, requires a large volume of labeled data for training [13]. This is a limitation for many classification problems in the context of Software Engineering, including the UML diagrams classification. Transfer learning techniques apply pre-trained models to improve the classification results. Thus, transfer learning can be explored together with machine learning techniques in order to achieve satisfactory results even when there are not so many images available for training.

Our project goal is to support the creation and manipulation of didactic materials and books, which contain UML diagrams, aiming to make them accessible to visually impaired students. Thus, in this work we performed the first step of the project, which consists of an exploratory study to apply Convolutional Neural Networks (CNN) in the task of UML diagrams classification. In addition to CNN, we employed transfer learning and data augmentation aiming to take advantage from pre-trained models. These ML techniques proved to be effective in circumventing issues provoked by the scarcity of data.

Firstly, we create a dataset containing several images of the six most commonly used UML diagrams, namely class diagram, use case diagram, sequence diagram, component diagram, activity diagram and deployment diagram [4]. Then, we apply transfer learning and data augmentation for UML diagrams classification on this dataset. We experimented three widely-known CNN architectures: VGG16, RestNet50, and InceptionV3. The results demonstrated that the use of transfer learning allows achieving good results even using scarce data. However, the obtained results pointed out that the UML diagram classification is a complex problem that needs further studies for improving the classification accuracy, especially considering class diagrams.

The main contributions of our work are twofold: (i) the dataset that encompasses images of six different types of UML diagrams created for the development of this work, which will be made available upon the paper acceptance; and, (ii) the preliminary results of our work and the lessons learned, which shed light on the automatic classification of a more comprehensive set of UML diagrams.

This paper is organized as follows: Section II briefly describes some concepts from machine learning used in this work; Section III addresses related works; Section IV presents the study design; Section V discusses the results obtained; Section VI presents the lessons learned; and finally, Section VII brings some concluding remarks.

II. Machine Learning Techniques

In this section we briefly describe the main machine learning techniques used in this work.

A. *Convolutional Neural Networks (CNN)*

CNN is a specific type of neural network, and currently it is one of the most famous deep models used by the machine learning research community to address image classification tasks. This model is based on ideas originally proposed by LeCun [14], and it is designed such a way that it can naturally deal with data in the form of a matrix. This is one reason why the model is quite suitable for image classification. CNN is composed of several layers which correspond to different levels of representation, and the number of layers determines the depth of the model. The model starts with the raw image to feed the input layer, and after the transformations performed on the following layers, through a backpropagation algorithm, representations for more abstract concepts are obtained. One of the key aspects in this method is that these layers are automatically learned. This is why the term “feature learning” was coined to refer to this process.

Different CNN architectures have been proposed on the last decade aiming to provide models each time better to perform image classification. In this work we selected three of these models, widely-used in the recent literature to address image classification tasks in a wide range of applications: VGG16, Inception V3, and ResNet50.

The VGGNet architecture [15] presents good results on image classification tasks. It is a very deep model based on very small convolution filters (3×3) with a pool layer added after every two or three convolutional layers. The two most used models derived from the original VGG are VGG16 and VGG19, with 16 and 19 weight layers respectively.

Inception V3 [16] is the third version of GoogleLeNet (also known as Inception V1), the winner of the detection challenge on the ILSVRC 2014 contest. In the second and third versions of the architecture, the creators introduced some small changes in the way how the model performs convolutional operations, among others, such a way the computational time decreased and the accuracy of the model increased.

ResNet-50 [17] is a 50 layers depth network derived from the original ResNet, proposed in 2015, which can have up to 152 layers in total. The model is able to learn residual representation functions, not directly learning the signal representation itself. ResNet also introduced the concept of skip connections (i.e. shortcuts to jump over some layers). This model was the winner of the ILSVRC 2015 contest in image classification, detection and localization modality, and it was also the winner of MS COCO 2015 on image detection and segmentation tasks.

B. *Transfer Learning*

Transfer learning is a method widely-used in computer vision that allows building accurate models in a time saving way [18]. With transfer learning, instead of starting the learning process from scratch, we can start from patterns that have been learned when solving a different problem. This way we leverage previous learnings and avoid starting from scratch.

In computer vision, transfer learning is expressed through the use of pre-trained models. A pre-trained model is a model usually trained on a large benchmark dataset to solve a problem similar to the one we want to solve. Accordingly, due to the computational cost of training such models, it is a common practice to import and use models from published literature (e.g. using already established models: VGG, Inception and ResNet), most of the times created from quite huge image datasets. This way, transfer learning can also be seen as a strategy to deal with the lack of data samples, as we can take benefit from knowledge learned from huge datasets. A comprehensive review of pre-trained models’ performance on computer vision problems using data from the ImageNet [19] is presented by Canziani et al. [20].

Marcelino et al. [21] define three strategies to replace the original classifier from the CNN architecture by a pre-trained model. They are defined below and illustrated in Figure 1.

- **Strategy 1**: *Train the entire model.* In this situation, it is possible to use the architecture of the pre-trained model and train it according to the dataset. It is recommended for large datasets.

- **Strategy 2**: *Train some layers and leave the others frozen.* In a CNN architecture, lower layers refer to general features (problem independent), while higher layers refer to specific features (problem dependent). In this case, we have to adjust the weights of the network. This option is useful when we have a small dataset and a large number of parameters, we need to leave more layers frozen to avoid overfitting. On the other hand, if the dataset is large and the number of parameters

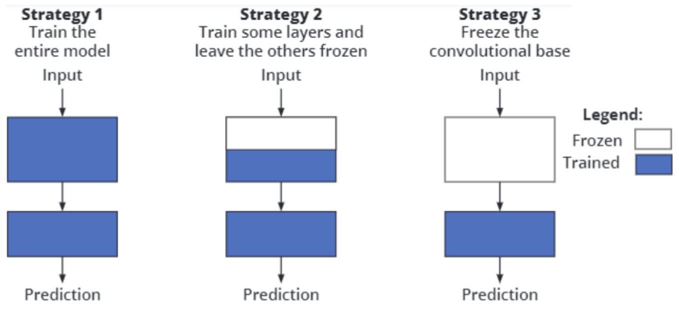

Fig. 1. Strategies for transfer learning [21].

is small, it is possible to improve the model by training more layers to the new task.

- **Strategy 3**: *Freeze the convolutional base.* In this situation, we have an extreme case of the train/freeze trade-off. The rationale behind it is to keep the original form of the convolutional base to use as input for the classifier. By this way, the pre-trained model plays the role of a feature extractor. It can be interesting for small datasets or if the problem solved by the pre-trained model is similar to the one we are working on.

III. Related work

Torres and Barwaldt [8] published a systematic literature review that brings a landscape of the primary studies which aim to turn diagrams accessible for blind people. In this research the authors gather the studies that work with a wide concept of diagram, especially for courses in STEM fields. Only a few studies address the UML diagrams and without approaching the issue of classification in context of assistive technologies. That paper reinforces that there is a lack of studies about UML diagram classification or studies that approach an accessible solution for images of UML diagrams.

Karasneh and Chaudron [22] presented one of the first studies on this subject. The authors proposed a system for extracting information from class diagrams. The proposed system extracts the contents of the class diagram from a given image and transforms it to XMI format. The same authors created a repository composed of class diagram images and XMI representations [23]. The authors also described a protocol to extract information from class diagram using OCR [10]. In another paper, they explore the extraction of UML information from images using the Img2UML software [24].

Hjaltason and Samuelsson [10] performed a binary classification of UML class diagrams using handcrafted features and SVM algorithm. In that work, the authors aimed to facilitate the construction of a dataset composed of UML class diagrams for researchers. Thus, they focused only in class diagrams. Similarly, Moreno et al. [11] created a tool aiming to identify whether an image is a class diagram or not, by means of a binary classifier fed by handcrafted features.

Lastly, transfer learning with VGG16 were employed for class diagram and sequence diagram classification in [12]. This work shows that transfer learning is equally effective to custom deep architectures when a large amount of training data is not

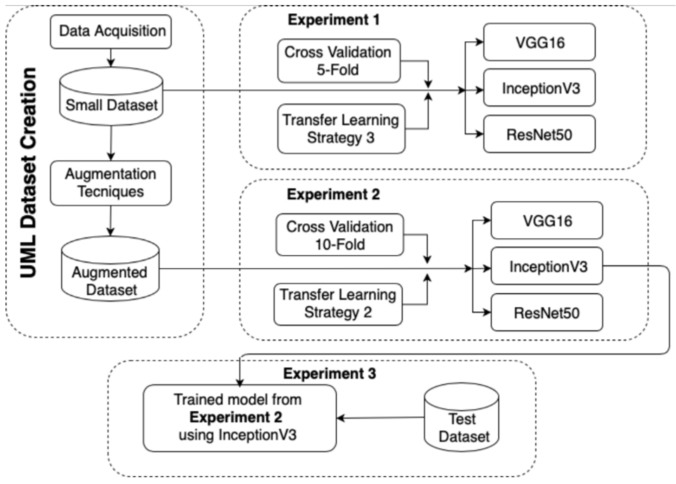

Fig. 2. The method used in this work.

available. The experiments were applied only in two types of UML diagram, using only one type of CNN architecture.

Bine et al. [7] presented another interesting work, not directly related to UML diagrams, but also in the context of learning support. The authors explored CNN for automata images classification, aiming at supporting visually impaired people in Theory of Computation learning tasks. The experimental protocol included three types of CNN architectures, but transfer learning was not used.

In summary, even though we can find several works applying transfer learning in image classification tasks using CNN architectures, few of them were devoted to UML diagrams classification. Also, differently from [12], it is needed to test other CNN architectures in order to evaluate the CNN accuracy for the referred context. Finally, as aforementioned, different types of UML diagrams are used in practice [4], [5], leading to the need of providing support for a more comprehensive set of UML diagrams than only class diagrams, as done in related works described here.

IV. Study Design and Execution

In this study we performed an exploratory study involving the integration of transfer learning and CNN architectures to classify six types of UML diagrams. In this study, we are interested in answering the following research questions:

RQ1 - Is there a suitable combination of transfer learning strategy and CNN architecture for UML diagram classification? To answer this RQ we executed two different scenarios which integrated different transfer learning strategies using some of the most known models of CNN (see Section II).

RQ2 - Can different types of UML diagrams be predicted by a single classifier? To answer this RQ we executed an experiment with the best performing classifier to predict the UML diagram type on a test dataset, composed by images different from those used to train the model.

Figure 2 shows the steps performed in this study. The first step was the UML Dataset Creation, which is described in Section IV-A. In Experiments 1 and 2, we experimented different transfer learning strategies, datasets and CNN architectures to train the models. Finally, we executed Experiment 3 to test the prediction of the best performing trained model. Results of Experiments 1 and 2 were used to answer **RQ1**, whereas results of Experiment 3 is related to **RQ2**. These steps are detailed in the next subsections.

A. UML Dataset Creation

The basic stuff for the development of a UML diagram classification model is a UML dataset which encompasses images of different types of UML diagrams. The number of public examples of UML diagrams is low. Among the rare datasets available in this context, we can mention the "Online Img2UML Repository", a dataset composed of approximately 800 diagrams [23] which contains only class diagrams.

One of the largest datasets available nowadays is The Lindholmen Dataset [25]. This repository is public and it is composed of links that lead directly to different projects on Github. Authors themselves warn that it has the advantage of having been taken from real projects, but it may contain projects made by students who are still learning the concepts; hence, it may contain repeated and unrepresentative images. In addition, GitHub is dynamic and many of the links may be broken. As reported on the dataset website, many of the diagrams are not in the form of images and are not cataloged.

Aiming at circumventing the scarcity of data available for the development of this work, our choice was to create a repository containing images of six types of UML diagrams. For doing this, we collected images available on the internet using web scrapping scripts on google.com and bing.com. Following, we completed the dataset using images from The Lindholmen Dataset.

Our dataset is composed of six categories of UML diagrams, containing images for training, validation, and testing. We chose the most commonly used UML diagrams to compose our dataset, namely class, use case, sequence, component, activity and deployment diagrams [4].

To train and validate the CNN models we used a dataset composed of 200 images for each category, hereafter referred as *small dataset*. To test the CNN models we used another dataset composed of 50 images for each category, hereafter called as *test dataset*. It is important to remark that the images of the test dataset are different from those ones included in the small dataset.

Next, we describe the adjustments performed on the datasets aiming to make them ready to be used considering the CNN architectures requirements and the purposes of this work.

Images Pre-processing. The images were resized to 900×900 pixels and converted from RGB to grayscale, aiming to make them suitable to be submitted to the CNN architectures used to perform the classification in this work. When resizing, the proportion of the images was maintained, using zero padding when necessary.

Images Labeling. All the images collected were manually labeled by the first author according to its type of UML diagram. Such information is required during the CNN training for the supervised learning. Despite this is an objective task, the image classification was checked by another author to avoid any labeling error.

Data Augmentation. Aiming to evaluate the impact of transfer learning without a critical level of data scarcity, we performed data augmentation on the small dataset. By means of data augmentation, we created a bigger version of the dataset (called *augmented dataset*). We generate four more variations for each image of the small dataset. Hence, the augmented dataset encompasses 6000 images, being 1000 images for each type of UML diagram. The transformations applied to obtain the image variations were horizontal and vertical flip, and rotation of 90 degrees and minus 90 degrees. Other variations were taken into account, but did not provide an effective return for our problem. Both versions of the dataset are publicly available in https://bit.ly/3fTMI6f.

B. Experimental protocol

We propose two different scenarios aiming to evaluate the impacts of using the dataset with and without performing data augmentation. Thus, in the first scenario, we use the *small dataset*, and in the second one, we use the *augmented dataset*. In both scenarios we used cross validation to prevent overfitting. Cross validation splits the dataset into different folds (splits of data) to train/test the model.

As shown in Figure 2, Experiment 1 was performed on the *small dataset*, while Experiment 2 was performed on the *augmented dataset*. In Experiment 1, five-fold cross validation was used, as the number of images of the dataset is small. For Experiment 2 we used ten-fold cross validation.

Transfer learning was used in both experiments. To use a pre-trained model, we remove the original classifier from the CNN architecture. Then, we add a new classifier that fits our purposes, and perform a fine-tuning on the model following one of three strategies defined in Section II. In Experiment 1 we applied the Strategy 3, while in Experiment 2 we applied the Strategy 2.

As described in Section II, VGG16, Inception V3 and ResNet50 are the CNN architectures chosen to the accomplishment of the classification of the images in this work. The implementation was carried out using the Tensorflow and Keras framework within the Google Colab environment. This environment offered a more robust GPU and consequently the possibility to perform more tests in less time. The datasets were uploaded to Google Drive allowing easy access through the Google Colab notebook API. All codes used were made publicly available in https://bityli.com/sc9o5.

For every CNN architecture we follow the next steps:

- Creation of K-fold for cross-validation, with $K = 5$ for small dataset and $K = 10$ for augmented dataset;
- Load of model with weights pre-trained on ImageNet removing the classification layer;
- Freeze/unfreeze n layers in original architecture. For Strategy 3 all layers were frozen, using the CNN architecture as feature extracting. For Strategy 2 we experimented different amount of unfrozen layers for every

TABLE I
ACCURACY AND STANDARD DEVIATION (STD. DEV.) FOR CNN MODELS.

Experiment	**CNN Architecture**	**Accuracy (%) and Std. Dev.**
Experiment 1	VGG 16	80.6±2.6
	Inception V3	**87.8±1.4**
	ResNet 50	84.8±2.1
Experiment 2	VGG 16	96.9±0.8
	Inception V3	**98.7±1.0**
	ResNet 50	98.2±0.6

CNN architecture. The best results were obtained using 175 unfrozen layers for InceptionV3, 143 for ResNet50 and 11 for VGG16;

- Add a GlobalAveragePooling2D layer;
- Add a Dense layer (6) with softmax activation;
- Compile the model using Adam Optimizer and accuracy metrics;
- Fit the model for 10 epochs. Different amounts of epochs were tested, but the model did not improve its results with more than 10 epochs;
- Use of callbacks: reduce learning rate and model checkpoint to save the best model;
- Saving accuracy and loss;
- Plot results by folds.

After discovering the best configuration to create the UML diagram classifier, we performed the Experiment 3 with the *test dataset* in order to answer RQ2. Details of the results for the three experimental protocols are presented next.

V. RESULTS AND DISCUSSION

In this section we present the results obtained using the three different CNN architectures evaluated in this work. The results for Experiments 1 and 2 are presented in Table I.

As previously described, for *Experiment 1* we used the small dataset and trained the 3 CNN models using the Strategy 3 (described in the Section II) for transfer learning, with five-fold cross-validation, the results were collected in terms of accuracy and standard deviation between the folds. Thus, the accuracy percentage presented in Table I is the average obtained between the folds. As we can see, Inception V3 performed better than ResNet 50, which, in turn, performed better than VGG 16 for Experiment 1.

Experiment 2 was carried out using the augmented dataset, with ten-fold cross-validation and using the Strategy 2 for transfer learning. As we can see, VGG16 underperformed again when compared to the other two models.

At a first glance, we can see that the Experiment 2, using the augmented dataset, presents better results in all the CNN architectures, confirming that it is interesting to provide more examples for every architecture evaluated, even when transfer learning is used. Besides that, we also can observe that Inception V3 achieved better results in both experiments.

Results obtained are quite encouraging, especially those obtained using the Inception V3 architecture. Regarding **RQ1**, the experimental results showed that Inception V3 using the augmented dataset and the Strategy 2 for transfer learning achieved the best results in terms of accuracy.

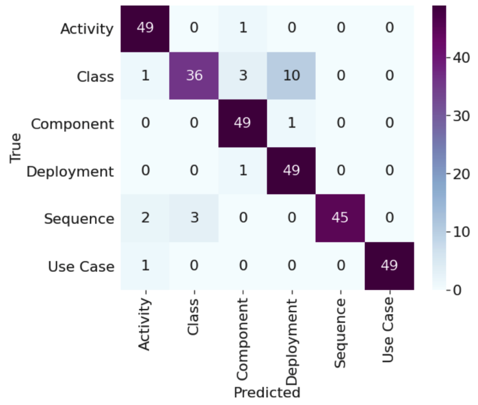

Fig. 3. Confusion Matrix of the Experiment 3.

Taking into account the superior results obtained by Inception V3, we performed *Experiment 3* using the model trained in *Experiment 2* using Inception V3. Figure 3 show the confusion matrix of the *Experiment 3*. In a nutshell, the confusion matrix discriminates how the samples of each class were classified towards all the classes of the problem.

As observed in Figure 3, InceptionV3 has achieved at least 98% of accuracy for activity, component, deployment and use case diagrams and 90% for sequence diagrams. On the other hand, it had the worst performance for class diagrams (72%). The biggest number of confusion occurrences happened when class diagrams were classified as deployment diagrams. This kind of misclassification can be explained, in part, because the wrongly classified images contain elements that can resemble those present in the deployment diagrams. Furthermore, the reading of some associations of class diagrams may have been impacted by any of the data augmentation strategies adopted. In fact, the class diagram is more complex to classify (i) due to the diversity of association types, and (ii) because the visual representation of some associations of class diagrams are changed according to the position of the classes.

Furthermore, the images of our datasets have different abstraction levels. The biggest range in the level of abstraction occurs on class diagram, contributing to the low accuracy of the classifier for this type of diagram. Approximately 20% of the images correspond to complex class diagrams, which contain attributes and their types, methods and their parameters and return types, classes inside packages, stereotyped classes, a high diversity of associations, and so on. On the other hand, most images (≈80%) represent simple class diagrams, composed only of classes, their attributes and methods (without details), and the most common associations. This lack of uniformity tends to drop the hit rates, but the classifier developed in this way is supposed to perform better in more realistic scenarios. Hence, these results showed that a more in-depth study needs to be done to improve the accuracy, especially for class diagram.

Regarding **RQ2**, the accuracy of the generated classifier using the test dataset varied according to the type of UML diagram. InceptionV3 achieved quite encouraging results for most UML diagrams, however it had low accuracy in class diagram.

Hence, the experimental results provided initial evidence that a single classifier can efficiently predict different types of UML diagrams, but there is still a room for improvement regarding the classification of class diagrams in particular.

VI. Lessons Learned

From the experimental results we learned some lessons about the UML diagram classification problem that can be useful to guide further works. These lessons are:

- *Data augmentation is beneficial for the problem*. However, the horizontal and vertical flips applied to obtain the image variations might damage the prediction of some UML diagrams. The transformations to be applied need to be carefully planned and applied according to specific characteristics of each type of diagram.

- *Transfer learning is a good practice for the problem*, as results show that transfer learning contributed to achieve a better accuracy.

- *The UML diagram classification is not a so simple problem*. We conjecture that this fact can happen due to: (i) complex and different diagrams, even within the same category, and (ii) low number of images. There is a need to deepen the concepts of backpropagation and automated extraction of features in the context of UML diagrams in order to seek for improvements on the classifier's accuracy.

- *A huge dataset of UML diagram images is need to perform further studies*. Related work [9]–[12] used datasets that contain around 1000 images of class diagrams. In our study, we used only 200 images per type of diagram due to the lack of images available for some types of diagrams. Then, we applied data augmentation techniques to obtain 1000 images per category.

VII. Concluding Remarks

In this study, we explored the application of CNN using transfer learning and data augmentation techniques to predict the type of UML diagram aiming to provide information that enables the creation of didactic material and book for visual impaired people. The results suggest that the UML classification is a complex problem that needs classifiers carefully created considering specific types of UML diagrams.

In future work, there is a need for improving the transformations applied in the data augmentation, and also for understanding how the CNN reacts to each of the different diagrams, in order to improve the classification model. The goal is to create a specific CNN model that takes into account the specific characteristics of each type of diagram and that improves the results achieved with transfer learning. For doing so, it is important to work on object recognition within the diagrams, to make possible to extract the object's information and transform it into accessible information.

Acknowledgment

This work is supported by the Brazilian National Council for Scientific and Technological Development (CNPq).

References

[1] B. Beck-Winchatz and M. A. Riccobono, "Advancing participation of blind students in science, technology, engineering, and math," *Advances in Space Research*, vol. 42, no. 11, pp. 1855–1858, 2008.

[2] OMG, "UML: Infrastructure specification," 2017, https://www.omg.org/spec/UML/2.5.1/PDF.

[3] G. Engels, J. H. Hausmann, M. Lohmann, and S. Sauer, "Teaching uml is teaching software engineering is teaching abstraction," in *Satellite Events at the MoDELS 2005 Conference*, 2006, pp. 306–319.

[4] A. M. Fernández-Sáez, M. Genero, D. Caivano, and M. R. V. Chaudron, "On the use of uml documentation in software maintenance: Results from a survey in industry," in *18th MoDELS '15*, 2015, p. 292–301.

[5] G. Scanniello, C. Gravino, and G. Tortora, "Investigating the role of uml in the software modeling and maintenance - a preliminary industrial survey." in *ICEIS 2010*, vol. 3, 01 2010, pp. 141–148.

[6] J. Choma Neto, L. H. T. C. Bento, E. OliveiraJr, and S. R. S. Souza, "Are we teaching UML according to what IT companies need? A survey on the São Carlos - SP region," in *EduComp*, 2021, pp. 34–43.

[7] L. M. S. Bine, Y. M. G. Costa, and L. B. R. Aylon, "Automata classification with convolutional neural networks for use in assistive technologies for the visually impaired," in *11th Pervasive Technologies Related to Assistive Environments Conference*, 2018, p. 157–164.

[8] M. J. R. Torres and R. Barwaldt, "Approaches for diagrams accessibility for blind people: a systematic review," in *2019 IEEE Frontiers in Education Conference (FIE)*, 2019, pp. 1–7.

[9] G. Bethany, S. Chowdhuri, J. Singh, M. Gupta, and A. Mishra, "Automatic classification of uml class diagrams using deep learning technique: Convolutional neural network," *Applied Sciences*, 05 2021.

[10] T. Ho-Quang, M. Chaudron, I. Samuelsson, J. Hjaltason, B. Karasneh, and M. H. Osman, "Automatic classification of UML class diagrams from images," 12 2014. [Online]. Available: 10.1109/APSEC.2014.65

[11] V. Moreno, G. Génova, M. Alejandres, and A. Fraga, "Automatic classification of web images as uml diagrams," in *Proceedings of the 4th Spanish Conference on Information Retrieval*, ser. CERI '16, 2016.

[12] N. Best, J. Ott, and E. Linstead, "Exploring the efficacy of transfer learning in mining image-based software artifacts," *Journal Of Big Data*, vol. 7, 08 2020.

[13] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, 2018, pp. 117–122.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[18] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, pp. 1–98, 06 2017.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, "Imagenet: a large-scale hierarchical image database," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 06 2009.

[20] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *CoRR*, vol. abs/1605.07678.

[21] P. Marcelino, M. de Lurdes Antunes, E. Fortunato, and M. C. Gomes, "Machine learning approach for pavement performance prediction," *Intl. Journal of Pavement Engineering*, vol. 22, no. 3, pp. 341–354, 2021.

[22] B. Karasneh and M. R. Chaudron, "Img2uml: A system for extracting uml models from images," in *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, 2013, pp. 134–137.

[23] B. Karasneh and M. Chaudron, "Online Img2UML repository: An online repository for uml models," in *EESSMOD@MoDELS*, 2013.

[24] ——, "Extracting UML models from images," in *5th Intl. Conf. on Computer Science and Information Technology*, 03 2013, pp. 169–178.

[25] G. Robles, T. Ho-Quang, R. Hebig, M. R. V. Chaudron, and M. A. Fernandez, "An extensive dataset of UML models in github," in *14th Intl. Conference on Mining Software Repositories*, 2017, p. 519–522.