Multiclass Classification of Four Types of UML Diagrams from Images Using Deep Learning

Sergei Shcherban and Peng Liang*
School of Computer Science
Wuhan University, China
k1myriel@yandex.ru, liangp@whu.edu.cn

Zengyang Li
School of Computer Science
Central China Normal University, China
zengyangli@ccnu.edu.cn

Chen Yang
IBO Technology (Shenzhen) Co., Ltd.
Shenzhen, China
c.yang@ibotech.com.cn

***Abstract*—UML diagrams are a recognized standard modelling language for representing design of software systems. For academic research, large cases containing UML diagrams are needed. One of the challenges in collecting such datasets is automatically determining whether an image is a UML diagram or not and what type of UML diagram an image contains. In this study, we collected UML diagrams from open datasets and manually labeled them into four types of UML diagrams (i.e., class diagrams, activity diagrams, sequence diagrams, and use case diagrams) and non-UML images. We evaluated the performance of five popular neural network architectures using transfer learning on the dataset of 3231 images that contains 700 class diagrams, 454 activity diagrams, 651 use case diagrams, 706 sequence diagrams, and 720 non-UML images, respectively. We also proposed our neural network architecture for multiclass classification of UML diagrams. The experiment results show that our proposed neural network architecture achieved the best performance amongst the algorithms we evaluated with an accuracy of 98.65%, a precision of 96.76%, a recall of 96.48%, and an F1-score of 96.62%. Moreover, among the neural network architectures that we have evaluated, our proposed architecture has the least parameters (around 2.4 millions) and spends the least time per image (0.0135 seconds per image using GPU) for classifying UML diagrams.**

***Keywords*—UML Diagrams, Neural Network, Deep Learning, Multiclass Classification**

I. Introduction

To define and communicate the design and architecture of systems, software engineers are challenged by the increasing complexity of software systems, especially when groups of developers use different modelling notations in distributed development. Software developers, in particular architects and designers, are therefore using Unified Modeling Language (UML) [1] diagrams as a universal set of notations to promote the concept description and collaboration in the development of software systems. UML was stemmed as the union of three object-oriented design practices: the Booch method [2], the object modelling technique [3], and the Objectory method [4]. The UML standard was published and maintained by the Object Management Group (OMG). In addition, both in industry and academia, the use of UML diagrams as a standard for developing software [5]. Class diagrams, activity diagrams, sequence diagrams, and use case diagrams are the most common four types of UML diagrams used in industry [6]. UML class diagrams demonstrate the classes in a system, attributes and operations of each class, and the relationship between classes, and act as a focal role in defining software structure. Activity diagrams explain how activities are organized to deliver a service that can be at different levels of abstraction. Sequence diagrams are diagrams of interaction that detail how operations are done, and they capture the interaction between objects. Use case diagrams provide description of the interactions between users and the system.

In the research concerning UML modeling, researchers need repositories with a large number of samples of UML diagrams [7] [8] [9], such as the Lindholmen dataset [10]. These datasets can also be used to create helpful tools for developers, such as creating UML diagrams by using natural language specifications [7] or using UML diagrams to produce code [8]. Images are one of the most used ways of storing and sharing UML diagrams. Therefore, identifying whether an image belongs to UML diagrams or not is the issue of constructing such repositories. Non-UML images are often found, especially in large datasets, such as the Lindholmen dataset [10]. Sorting through thousands of pictures manually requires a significant amount of time and effort. It is therefore necessary to identify various UML diagrams from images automatically. Recently, many researches have been focused on classifying images by applying deep learning techniques (e.g., [11] [12] [13]), and practitioners and researchers are making these techniques accessible for use. Besides, in terms of precision of image classification, deep learning algorithms outperform classical machine learning algorithms [14]. Also, modern deep learning frameworks (such as Tensorflow or PyTorch) include the ability to use the GPU for neural network training and inference, which accelerates the use of deep learning-based image classification approaches.

In this paper, we presented an approach to identify various types of UML diagrams automatically from images using deep learning algorithms. Initially, we gathered a dataset of 3231 images (700 class diagrams, 454 activity diagrams, 651 use case diagrams, 706 sequence diagrams, and 720 non-UML images). An experiment was then performed with several popular neural network architectures [15] [16] that can be found in current deep learning frameworks [17] [18] and are widely used for applying neural networks (i.e., MobileNet [19], DenseNet [20], NasNet [21], ResNet [22], Inception [23]). We

* Corresponding author
DOI reference number: 10.18293/SEKE2021-185

evaluated these selected neural networks with semi-trainable transfer learning (the convolutional part of pre-trained neural networks was not trained) and fully-trainable transfer learning. Also, we proposed a new neural network architecture for multiclass classification UML diagrams.

The contribution of this paper is threefold: (1) a dataset which contains UML class diagrams, activity diagrams, use case diagrams, sequence diagrams, and non-UML images, serving as a starting point for researchers to further investigate the UML diagram identification and classification problem, (2) an approach to automatically identify various types of UML diagrams by using deep learning techniques, and (3) a neural network architecture which can fast and effectively classify various types of UML diagrams from images.

The rest of the paper is organized as follows: Section II introduced related works. Section III describes the research questions, the data collection, the classification process, and the evaluation process, and the experimental setup. Section IV provides the experiment results. Section V presented the threats to validity. Section VI describe concludion of this work with further work directions.

II. Related Work

Recently, various techniques to extract features from diagrams have been introduced. Karasneh and Chaudron [24] introduced a process to extract UML class diagrams from images and transform them into XMI format. Fu and Kara [25] proposed a method for converting engineering diagrams into connected graphs. Ho-Quang et al. proposed an approach to classifying UML class diagrams from images automatically by using machine learning algorithms and different feature extraction techniques [9]. Despite its effectiveness, this method to image classification consumes 5.84 seconds per image, which can be problematic when using it with big datasets. Mohd Hafeez Osman et al. showed that reverse-engineered and forward-engineered UML class diagrams can be classified by using machine learning [26]. Ahmed and Huang also applied machine learning to classify role stereotypes of UML class diagrams in order to quickly get the knowledge about role stereotypes for developers [27]. They achieved an accuracy of 89.6% in the multiclass classification of role stereotypes of UML class diagrams using Random Forest with SMOTE oversampling. Rashid classified UML sequence diagrams by applying machine learning and computer vision algorithms to facilitate the creation of repositories containing UML diagrams [28]. His work achieved an accuracy of 90.8% using the methods from the OpenCV framework, such as Canny edge, probabilistic Hough lines transform, and FindCountors as feature extraction methods. Bian *et al.* proposed an approach to automatically grade students' UML class diagrams by using semantic, structural, and syntactic matches between the teacher's solutions and the students' solutions [29]. They received a variation of 14% between the teacher's grade and the grade received by using their tool by grade 20 students.

Because neural networks do not demand additional feature extraction algorithms for classification and preform feature extraction automatically using convolution layers [11], their classification speed is higher than that of approaches that used combination of feature extraction algorithms and classical machine learning algorithms. Moreover, in some computer vision tasks, such as object recognition, modern neural networks outperform humans in terms of accuracy [12].

In deep learning, the idea of reusing the acquired knowledge from one task to another is called transfer learning [13]. It has been proven as an effective way to implement neural networks without using a lot of resources on training and searching for a neural network's architecture. Transfer learning means applying the neural network's weights obtained within one task to complete or partially complete training on a new task.

To the best of our knowledge, there are no works that are directly aimed at the automatic multiclass classification of UML diagrams. Thus we open this line of research by proposing an approach to UML diagram classification and making available to the public the dataset used for the experiments.

III. Research Design

This research aims to study how we can automatically classify multiclass UML diagrams by using deep learning algorithms. In this study, we investigated four Research Questions (RQs):

RQ1: What is the best performance of semi-trainable transfer learning for multiclass classification of UML diagrams?

Rationale: This RQ aims to get the best classification algorithm (in terms of performance) when training algorithms to recognize UML diagrams. Various deep learning algorithms can produce different results, depending on the architecture, configurations, and datasets used. Our selection fell on five deep learning architectures, including MobileNet, DenseNet169, NasNetMobile, ResNet152V2, and InceptionV3, because they are commonly applied in image classification and can be founded in modern deep learning frameworks. We evaluate the performance of semi-trainable transfer learning of each algorithm using accuracy, precision, recall, and F1-score metrics.

RQ2: What is the best performance of fully-trainable transfer learning for multiclass classification of UML diagrams?

Rationale: The process of training all layers of neural networks takes more time and computation resources. The purpose of this RQ is to understand whether the cost of training all layers of neural networks will improve performance in the task of classifying UML diagrams. We evaluate the performance of fully-trainable transfer learning on the algorithms from **RQ1** with all trainable layers.

RQ3: Is transfer learning essential for multiclass classification of UML diagrams?

Rationale: Transfer learning can speed up the process of training algorithms and can improve the accuracy of classification. However, images from the most popular dataset for transfer learning (ImageNet [30]) are not like UML diagrams from our dataset. This RQ aims to understand whether transfer

learning is better for classifying UML diagrams than training neural networks from scratch. We measure the performance of MobileNet and our proposed neural network without using transfer learning, and compare them with the best results from RQ1 and RQ2.

RQ4: What is the best performance on time per image of neural networks for multiclass classification of UML diagrams?

Rationale: The aim of this RQ is to investigate the performance of classification on time per image based on various neural networks. Since image classification can be used to collect datasets, refine search results, etc., the use of a classification algorithm should not be too time-consuming. The performance time is measured by using the GPU.

A. Data Collection

For the experiments, we created a dataset based on several existing datasets [10] [31] [9] [26], in which the Lindholmen dataset [10] is the largest one. We scrapped more than 10000 images from these datasets, and we then manually removed the duplicates and labeled four types of UML diagrams: class diagrams, activity diagrams, use case diagrams, sequence diagrams. Non-UML images were collected from [31] and manually filtered to remove UML diagrams. Overall we collected 3231 images (inlcluding 700 class diagrams, 454 activity diagrams, 651 use case diagrams, 706 sequence diagrams, and 720 non-UML images). Our dataset has been provided online for replication and reproduction purposes [32].

B. Image Classification Process

The process of classifying UML diagrams is composed of four phases:

Phase 1: Input Data. The input is the dataset which contains 700 class diagrams, 454 activity diagrams, 651 use case diagrams, 706 sequence diagrams, and 720 non-UML images, and we further split the dataset into a training set, a testing set, and a validation set.

Phase 2: Preprocess Images. The images were converted to a JPG format and to a size of 224x224 or 299x299.

Phase 3: Train Classification Algorithms. We trained different pre-trained neural networks and some neural networks without pre-trained weights.

Phase 4: Evaluate Trained Classification Algorithms. We evaluated the performance of each algorithm from **Phase 3** using multiple performance measures.

In **Phase 1** (i.e., input data), we split the dataset into *validation*, *testing* and *training* sets: 10% of images (323) as the validation set, 20% of images (646) as the testing set, and 70% of images (2262) as the training set. Deep learning frameworks often do not work with all image formats, so we converted all images to a JPG format. Also, for pre-trained models, we needed to bring all the images to the particular size, in this case, we brought the images to the size of 299x299x3 for InceptionV3 and 224x224x3 for the rest of the neural networks that we used (recommended image sizes for transfer learning). All the images were normalized by changing the range of pixel intensity values between 0 to 1. The training dataset was augmented with a horizontal flip, slight shifts in the horizontal and vertical axis (up to 20%).

To raise the accuracy and decrease the training time in **Phase 3**, we trained neural networks with and without using transfer learning, which is a method to reuse the information received during training on one task to new tasks. We used models pre-trained on the ImageNet task [30], including MobileNet, DenseNet169, NasNetMobile, ResNet152V2, and InceptionV3. Convolutional layers were not trained during the semi-trainable transfer learning process and all layers were trained during the fully-trainable transfer learning process. The output from pre-trained models was fed to a fully-connected layer with 1024 neurons and the relu activation function, then through dropout layer to the next fully-connected layer with 512 neurons and the relu activation function, and after the last dropout layer to a fully-connected layer with five neurons and a softmax activation function.

Our proposed neural network was inspired by MobileNetV3 [33] and ResNet [22]. Figure 1 provides the details about our proposed neural network architecture. The input layer is a convolutional layer with kernel size 3x3, 32 filters and ReLU as an activation non-linear function. Next are the architecture blocks, which are repeated throughout the architecture. The input of each logical block is a convolutional layer with twice as many filters as the previous layer and ReLU as an activation function. The next layer is the batch normalization layer. The output of the batch normalization layer is then split into two single outputs, one of which is called a shortcut which goes directly to the end of the logic block. Shortcut was made by using a maximum pooling layer. The second output goes through a separable convolution layer with half as many filters as the input layer and a 1x1 kernel size with ReLU activation function. It then goes into a convolution layer identical to the input layer of the logic block with stride 2 and the HardSwish activation function. Then goes to a similar layer but with LeakyReLU activation function. And after the batch normalization layer is concatenated with the shortcut output to the next logical block. Each logic block of our architecture contains convolution layers with different kernel size and stride, separable convolution layer [34]. We also used ReLU, LeakyReLU, and HardSwish [35] as activation functions. Following the sequential use of several logical blocks, the Global average pooling layer is used. Instead of using classical fully-connected layers at the end of the neural network, we used two convolutional layers with 1x1 kernel and the numbers of filters are 1024 and 5 (i.e., number of classes in the dataset). This allows us to reduce the number of parameters and speed up the neural network.

C. Performance Evaluation

We measured the performance of each algorithm by using accuracy, precision, recall, and F1-score, which are usually used in performance evaluation of image classification. The images from one class that have been correctly classified by a classifier are considered as True Positive (TP), while the

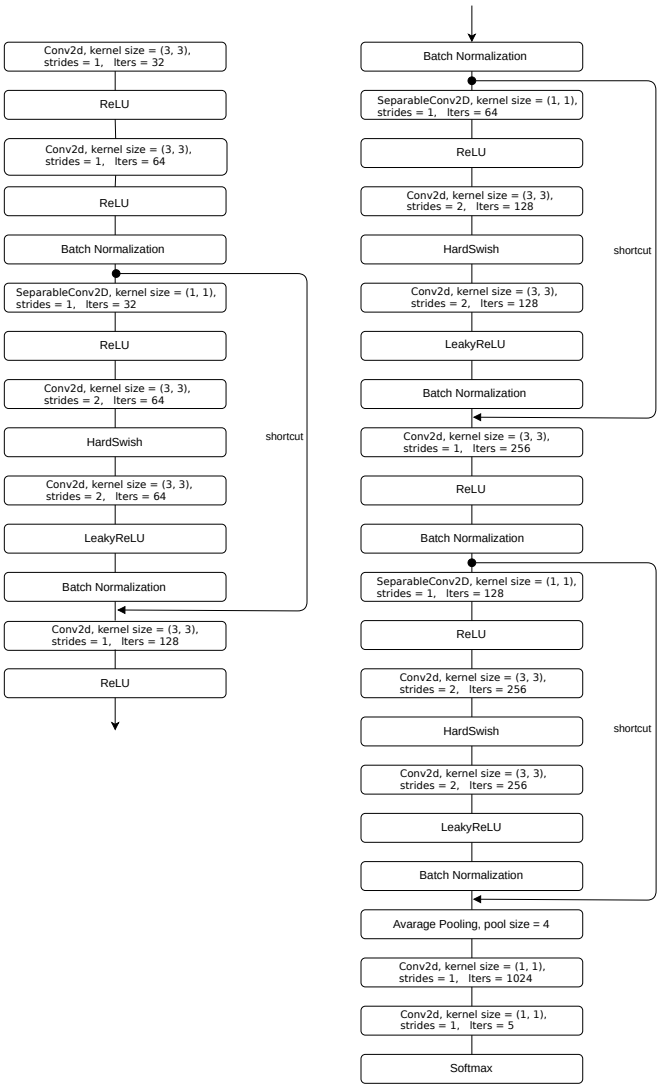

Fig. 1: Specification for our neural network architecture

images from another class that have been incorrectly classified are considered as False Positive (FP). The images from one class incorrectly identified as images from other classes are considered as False Negative (FN). The images from another class that have been correctly identified by a classifier are considered as True Negative (TN).

Accuracy (ACC) is the proportion of accurately predicted set from the whole observations. Precision is the ratio of images correctly identified as one class to all images identified as this class. Recall is the fraction of all images of one class correctly demarcated. The harmonic mean of precision and recall is called F1-score or F1 Measure. We calculated those metrics using the following equations:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$F1 - score = \frac{2TP}{2TP + FP + FN} \tag{2}$$

$$precision = \frac{TP}{TP + FP} \tag{3}$$

$$recall = \frac{TP}{TP + FN} \tag{4}$$

To measure the time spent, we used the free version of Google Colaboratory. This will make the results reproducible and verifiable. The free version of Google Colaboratory uses Nvidia Tesla T4 as the GPU device.

IV. Results and Analysis

To answer the RQs, we used deep learning algorithms. Overall, we experimented with 12 (5 classification algorithms as a semi-trainable transfer learning + 5 classification algorithms as a fully-trainable transfer learning + 2 classification algorithms without pre-trained weights) experiment configurations. These configurations were applied on a training set of 2262 images, a testing set of 646 images, and a validation set of 323 images. Since the neural networks have some bias in performance results, each experiment configuration was evaluated 5 times and the results bellow are an average of the 5 times when the experiment configuration is repeated.

RQ1: What is the best performance of semi-trainable transfer learning for classification of UML diagrams?

To answer RQ1, we applied several pre-trained neural networks such as MobileNet, DenseNet169, NasNetMobile, ResNet152V2, and InceptionV3 without training convolution layers. Table I shows the precision, recall, and F1-score for each algorithm. The best performed algorithm was MobileNet (accuracy = 96.79%, precision = 92.77%, recall = 91.06%, and F1-score = 91.91%). In the five classification algorithms, MobileNet outperforms other algorithms.

TABLE I: Performance in precision, recall, and F1-score by using semi-trainable transfer learning

	Recall	Precision	Accuracy	F1-score
ResNet152V2	88.28%	91.87%	96.09%	90.04%
InceptionV3	89.52%	91.88	96.32%	90.69
MobileNet	91.06%	92.77%	96.79%	91.91%
DenseNet169	89.41%	92.26%	96.38%	90.81%
NasNetMobile	85.41%	89.26%	95.03%	87.29%

RQ2: What is the best performance of fully-trainable transfer learning for classification UML diagrams?

To answer RQ2, we applied all pre-trained algorithms from **RQ2** with training all layers. Table II shows the precision, recall, and F1-score for each algorithm. The best performed algorithm was DenseNet169 (accuracy = 97.76%, precision = 94.44%, recall = 94.35%, and F1-score = 94.40%). In the five classification algorithms, DenseNet169 outperforms other algorithms. Moreover, DenseNet169 with all trainable layers outperforms all semi-trainable algorithms from **RQ1**. However, the training process for all layers takes more time than the training process of semi-trainable transfer learning.

RQ3: Is transfer learning essential for multiclass classification of UML diagrams?

Since the images from the ImageNet dataset [30] used for transfer learning are not similar to the images from our

TABLE II: Performance in precision, recall, and F1-score by using fully-trainable transfer learning

	Recall	Precision	Accuracy	F1-score
ResNet152V2	93.11%	93.50%	97.33%	93.31%
InceptionV3	94.14%	94.24	97.68%	94.19
MobileNet	93.83%	94.22%	97.62%	94.03%
DenseNet169	94.35%	94.44%	97.76%	94.40%
NasNetMobile	87.15%	90.21%	95.54%	88.65%

dataset, the question arises about the need for transfer learning. **To answer RQ3**, we compared the best results from RQ1 (MobileNet (STTL)) and RQ2 (DenseNet169 (FTTL)) with the results of MobileNet and our proposed neural network architecture, which were trained without using transfer learning (WPW). The comparison shows that the pre-trained out-of-the-shelf neural networks are better than the out-of-the-shelf neural networks that are trained without using transfer learning, but worse than our proposed neural network at classifying UML diagrams. Table III shows the comparison between the pre-trained and not pre-trained neural networks. Moreover, it is possible to pick up a custom neural network architecture that will cope almost as well as off-the-shelf neural networks, but the process of picking architecture parameters takes time.

TABLE III: Comparison between pre-trained and not pre-trained neural networks (STTL = semi-trainable transfer learning, FTTL = fully-trainable transfer learning, WPW = training without pre-trained weights)

	Recall	Precision	Accuracy	F1-score
MobileNet (STTL)	91.06%	92.77%	96.79%	91.91%
DenseNet169 (FTTL)	94.35%	94.44%	97.76%	94.40%
MobileNet (WPW)	88.59%	89.23%	95.58%	88.90%
Our Solution (WPW)	96.48%	96.76%	98.65%	96.62%

RQ4: What is the best performance on time per image of neural networks for multiclass classification of UML diagrams?

We measured time per image by using Google Colaboratory **to answer RQ4**. Table IV shows the measured time for our approach and comparison with the solution from [9]. For the transfer learning approach, MobileNet is the most effective algorithm in term of performance time (0.014 second per image). For all the approaches, our proposed neural network showed the best performance time (0.0135 second per image).

TABLE IV: Performance in time per image (seconds) and required parameters (millions)

	ResNet 152V2	InceptionV3	MobileNet	DenseNet169	NasNetMobile	Our Solution
GPU	0.0206	0.0183	0.0140	0.0151	0.0142	0.0135
Num of params	60.9	24.4	4.8	14.8	5.8	2.4

Summing up the results of the RQs, we can say that the DenseNet169 with all trainable layers is a good and out-of-the-shelf choice to classify UML diagrams from images. We have also found that it is possible to find an architecture that performs better than the pre-trained out-of-the-shelf neural networks, but that architecture selection takes time. Our proposed neural network architecture showed the best results (accuracy = 98.65%, precision = 96.76%, recall = 96.48%, and F1-score = 96.62%) and it also has the least parameters (2.4millions) and spends the least time per image (0.0135 seconds per image using GPU) for classifying UML diagrams.

V. Threats to Validity

Internal validity concentrates on whether the results can be drawn from the data, and one of the threats to the internal validity of this study are the overfitting of neural networks and the manual labelling of the dataset used in the experiment. Since pre-trained models are trained on a complex multi-class task of 1000 classes, they can be retrained when we move on to five class classification. To avoid overfitting, we used a Dropout layer after each fully connected layer. This allows us to ignore some of the information coming from fully connected layers and increases the stability of models. The images used in the experiment were manually labelled, which may introduce selection bias. It is possible that images were incorrectly labelled, leading to incorrect classification results. To mitigate this threat, the labeled data was manually checked by the first author, and for any unsure labels the first author discussed with the second author to get a consensus.

External validity reflects the extent to which the study results and findings can be generalized in other cases with similar characteristics. The potential threat to the external validity is about the diversity of the dataset used in the experiment, which was created based on several existing datasets [10] [31] [9] [26], in which the Lindholmen dataset [10] is the largest UML models repository from OSS projects and the other three datasets were retrieved by Google Images search. We believe that the experiment results can be applicable to the classification of UML digrams in practice to a large extent.

Construct validity in this work focuses on whether the evaluation metrics are suitable and measured correctly. A set of metrics (i.e., accuracy, recall, precision, and F1-score) were used to measure the performance of the classification algorithms, which have been widely used in assessing the quality of algorithms in image classification.

Reliability refers to whether the experiment yields the same results when it is replicated. In this work, this validity is mainly related to the dataset and the execution of the experiment. We defined the protocol for the classification process and evaluation metrics, which were confirmed and followed by all the researchers. We also made our experiment dataset available for replication purposes [32].

VI. Conclusions and Future Work

In this work, we automatically identify four most popular types of UML diagrams (class diagrams, use case diagrams, activity diagrams, and sequence diagrams) and non-UML

diagrams from images by using five popular neural network architectures (including MobileNet, DenseNet, NasNet, ResNet, and Inception) using transfer learning. We scrapped over 10000 images from open datasets and manually labelled it into four types of UML diagrams. In total, we have collected a dataset that contains 3231 images (700 class diagrams, 454 activity diagrams, 651 use case diagrams, 706 sequence diagrams, and 720 non-UML images). Then, we used transfer learning to classify UML diagrams by applying the neural network architectures. We also proposed our neural network architecture for multiclass classification of UML diagrams. The experiment results show that our proposed neural network architecture outperformed the existing neural network architectures with an accuracy of 98.65%, a precision of 96.76%, a recall of 96.48%, and and F1-score of 96.62%. To measure the time spent for the classification, we used a free version of Google Colaboratory. We found that the most efficient algorithm by using GPU is our proposed neural network architecture with 0.0135 seconds per image for classifying UML diagrams and our proposed neural network architecture also has the least parameters (around 2.4 millions).

In the next step, we plan to (1) construct cost-effective neural network architectures for classifying major types of UML diagrams from images; and (2) automatically recover the relationships between various types of UML diagrams from images within a project in order to establish the tractability between UML models.

REFERENCES

[1] OMG, "About the unified modeling language specification version 2.5," 2015. [Online]. Available: https://www.omg.org/spec/UML/2.5/About-UML

[2] J. Aranda, S. Easterbrook, and G. Wilson, "Requirements in the wild: How small companies do it," in *Proc. of the 15th IEEE International Requirements Engineering Conference (RE)*. IEEE, 2007, pp. 39–48.

[3] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[4] I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Pearson Education, 1993.

[5] M. R. Chaudron, W. Heijstek, and A. Nugroho, "How effective is uml modeling?" *Software and Systems Modeling*, vol. 11, no. 4, 2012.

[6] W. Lynch, "A comprehensive guide to 14 types of UML diagram," 2019. [Online]. Available: https://medium.com/@warren2lynch/a-comprehensive-guide-to-14-types-of-uml-diagram-affcc688377e

[7] P. More and R. Phalnikar, "Generating uml diagrams from natural language specifications," *International Journal of Applied Information Systems*, vol. 1, no. 8, pp. 19–23, 2012.

[8] M. Usman and A. Nadeem, "Automatic generation of java code from UML diagrams using ujector," *International Journal of Software Engineering and Its Applications*, vol. 3, no. 2, pp. 21–37, 2009.

[9] T. Ho-Quang, M. R. Chaudron, I. Samúelsson, J. Hjaltason, B. Karasneh, and H. Osman, "Automatic classification of UML class diagrams from images," in *Proc. of the 21st Asia-Pacific Software Engineering Conference (APSEC)*, vol. 1. IEEE, 2014, pp. 399–406.

[10] R. Hebig, T. H. Quang, M. R. Chaudron, G. Robles, and M. A. Fernandez, "The quest for open source projects that use UML: mining github," in *Proc. of the 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)*. IEEE, 2016, pp. 173–183.

[11] Y. LeCun and Y. Bengio, *Convolutional Networks for Images, Speech, and Time Series*. MIT Press, 1998, vol. 3361, pp. 255–258.

[12] R. Geirhos, D. H. Janssen, H. H. Schütt, J. Rauber, M. Bethge, and F. A. Wichmann, "Comparing deep neural networks against humans: object recognition when the signal gets weaker," *arXiv abs/1706.06969*, 2017.

[13] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Proc. of the 28th International Conference on Artificial Neural Networks (ICANN)*. Springer, 2018, pp. 270–279.

[14] S. Loussaief and A. Abdelkrim, "Deep learning vs. bag of features in machine learning for image classification," in *Proc. of the International Conference on Advanced Systems and Electric Technologies (IC_ASET)*. IEEE, 2018, pp. 6–10.

[15] J. Jordan, "Common architectures in convolutional neural networks," 2018. [Online]. Available: https://www.jeremyjordan.me/convnet-architectures/

[16] M. Hollemans, "New mobile neural network architectures," 2020. [Online]. Available: https://machinethink.net/blog/mobile-architectures/

[17] T. Contributors, "Torchvision.models," 2019. [Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html

[18] TensorFlow, "Module: tf.keras.applications," 2019. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/applications

[19] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv abs/1704.04861*, 2017.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[21] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018, pp. 8697–8710.

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

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

[24] B. Karasneh and M. R. Chaudron, "Img2UML: A system for extracting UML models from images," in *Proc. of the 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2013, pp. 134–137.

[25] L. Fu and L. B. Kara, "From engineering diagrams to engineering models: Visual recognition and applications," *Computer-Aided Design*, vol. 43, no. 3, pp. 278–292, 2011.

[26] M. H. Osman, T. Ho-Quang, and M. Chaudron, "An automated approach for classifying reverse-engineered and forward-engineered UML class diagrams," in *Proc. of the 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018.

[27] J. Ahmed and M. Huang, "Classification of role stereotypes for classes in UML class diagrams using machine learning," Master's thesis, University of Gothenburg, 2020.

[28] S. Rashid, "Automatic classification of UML sequence diagrams from images," Bachelor of Science Thesis, University of Gothenburg.

[29] W. Bian, O. Alam, and J. Kienzle, "Automated grading of class diagrams," in *Proc. of the 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2019, pp. 700–709.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2009, pp. 248–255.

[31] SunEdition, "Graphs dataset," 2010. [Online]. Available: https://www.kaggle.com/sunedition/graphs-dataset

[32] S. Shcherban, P. Liang, Z. Li, and C. Yang, "Dataset of the paper: Multiclass classification of four types of UML diagrams from images using deep learning," March 2021. [Online]. Available: http://doi.org/10.5281/zenodo.4595956

[33] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proc. of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2019, pp. 1314–1324.

[34] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 1251–1258.

[35] R. Avenash and P. Viswanath, "Semantic segmentation of satellite images using a modified cnn with hard-swish activation function." in *Proc. of the International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*, 2019.