NEURAL-UML: Intelligent Recognition System of Structural Elements in UML Class Diagram

Aymeric Koenig, Benjamin Allaert, Emmanuel Renaux

NEURAL-UML: Intelligent Recognition System of Structural Elements in UML Class Diagram

Aymeric Koenig*, Benjamin Allaert* and Emmanuel Renaux*
*IMT Nord Europe, Institut Mines-Télécom, Univ. Lille, Centre for Digital Systems, F-59000 Lille, France
{aymeric.koenig, benjamin.allaert, emmanuel.renaux}@imt-nord-europe.fr

***Abstract*—Design models are essential for many tasks in software engineering, such as consistency checking, code generation and design-to-code tracing. Unfortunately, many UML class diagrams are stored as images, which limits their use and evolution. It is therefore important to identify the semantic elements of design models from images. Although a number of studies focus on the recognition of a UML class diagram, very few address semantic analysis, which is a relatively complex task. In this paper, we propose a framework for training a learning model to categorise and locate semantic elements in class diagram from an image. A large set of annotated design models is proposed and made available online. Qualitative and quantitative evaluations have been carried out on two subsets of data, giving accuracy scores of 92.59% and 94.11% respectively. Evaluations highlight the ability of the proposed learning model to generalise to a wide range of examples.**

***Index Terms*—Deep learning, Computer vision, object recognition, UML design, class diagram**

I. Introduction

The UML class diagram design process is a way of modeling and conceptualizing a complex idea. However, the conceptualization process is often carried out by several collaborators, using a variety of supports, e.g., discussion, drawing or post-it notes. Although this facilitates exchanges, a transcript step in UML modeling software is often necessary to preserve a standardized written track that can be interpreted by all. This is at the expense of a significant investment of time. In many cases, UML class models and their evolutions are rarely retrieved during the life of a project.

With the rise of neural networks, and their ability to generalize complex problems, it is conceivable to propose systems for interpreting and modeling UML class diagrams from a non-standardized source. Many recent studies focus on classifying UML diagrams according to their type (i.e., class, sequence, activity, etc), without focusing in depth on UML concepts [2]–[4]. However, to correctly interpret a UML class diagram, it is important to perform several tasks: 1) component detection, e.g., classes and type of relationships; 2) class content extraction, e.g., name, attributes and operations; 3) relationships between classes. To train a learning model to perform these tasks, annotated datasets are required. However, the great variability of perspectives of class diagram (i.e., conceptual, specification or implementation) and graphical format, depending on the tools used or the nature of the diagrams, e.g., manuscript, pdf or post-it, tends to increase the complexity of dataset creation and the annotation task.

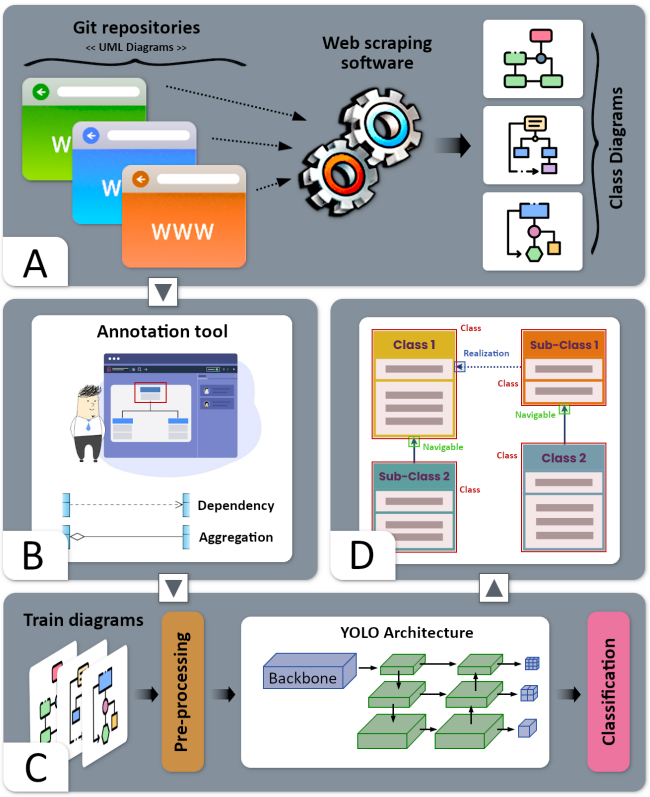

Fig. 1. Overview of the proposed framework. A) A web scraping step is used to retrieve UML class diagrams from a large number of Git repositories [1]. B) Each class diagram is annotated using an annotation tool developed for this study. C) A neural network is trained on the annotated data to identify the various components that make up a UML class diagram passed in parameter. D) All the classes and relationships are categorised and located on the diagram.

In this paper, we focus on the task of detecting and recognizing classes and type of relationships in a UML class diagram. Our proposed framework is illustrated in Fig. 1, and is based on two main contributions:

- A new annotated learning dataset based on a public data collection [1] are proposed to train a learning-based model to recognize classes and the six relationship types: Association, Inheritance, Aggregation, Composition, Dependency and Realization in a class diagram in image format. A scraping approach, combined with a pre-trained learning model proposed by Ho-Quang et al. [5], is used to extract only UML class diagrams from a large collection of diagrams.

- A new learning-based approach is proposed, for interpreting various UML diagrams, i.e., designed by several modeling tools, in image format. The system is evaluated according to several metrics as the intersection over union (IoU) and the score-F1 to verify its accuracy.

This paper is structured as follows: Section II presents a litterature on the various existing datasets for analyzing UML class diagrams, and on the learning-based approach used to interpret these diagrams. The proposed framework for designing the dataset, and to create the learning model is explained in Section III. Section IV contains the various evaluations carried out to validate the effectiveness of the proposed learning model. A discussion and a conclusion is given in Section V.

II. Related work

Interpreting a UML class diagram is a complex task that requires consideration of several elements, such as its graphical components, textual content and relational structure. In the literature, two points are generally addressed: 1) intelligent system design relying on feature extraction techniques to analyze the content of a UML diagram; 2) dataset design to train and evaluate these systems.

A. Categorise and localize semantic elements

Among semantic analysis methods focusing on detecting the elements that compose a UML diagram, there are few handcrafted vision-based techniques. Mennesson et al. [6] use Hough transform method with an SVM classifier to encode the geometry of elements such as classes and relations. Other approaches focus on textual content analysis. Perianez-Pascual et al. [7] outlines different strategies that can be applied to improve the recognition quality of OCR engines for DSL expressions. In their study, the authors agree that OCR can be coupled with visual semantic analysis to facilitate the interpretation of UML diagrams. Torres et al. [8] offer a uniform approach that is independent of the particularities of template notation, based on OCR techniques. For this purpose, they select graphical models from various domains that typically combine textual and graphical elements. Based on image descriptors and OCR methods, Chen et al. [4] propose a system for detecting the various elements within a UML class diagram, while interpreting the content of classes. Finally, several works focus on the hierarchy of structural elements within the class diagram. Babur et al. [9] propose representing metamodels in a vector space model, and applying hierarchical clustering techniques to compare and visualize them as a tree structure. Strüber el al. [10] transpose class diagrams into graphical form to group them into clusters of interest, thus facilitating their interpretation. Although these methods have proved effective on UML class diagrams of low complexity, where the structure follows a certain graphical standard and relationships do not cross, they do not generalize to more complex data.

Recently, a few works have used deep learning approaches to analyze class diagrams thanks to their ability to generalize more easily. Bnouni et al. [3] combine two types of CNN architectures: Mobilenet and VGG16 and demonstrates that the synergy between them improves the performance of the learning model. Wang et al. [2] propose a method based on a convolutional neural network (CNN), combined with a graph in the attention module to aggregate features, so that the network can capture more important features. The authors justify the poor performance of their system by the lack of training data. To overcome the lack of training data, Gosala et al. [11] uses transfer learning and artificial data augmentation techniques. Although these recent techniques have been shown to be effective for class diagram classification, it remains to be proven for more complex tasks such as semantic analysis of a diagram. To our knowledge, there is currently no neural network-based method for efficiently performing semantic analysis on a class diagram. This is partly due to the lack of training data and the complexity of the task.

B. Datasets

Several datasets are available for training and evaluating systems to interpret UML diagrams. As observed in Table I, the majority of existing datasets are designed to train learning models to recognize diagram type without focusing on diagram content. Hebig et al. [1] currently offer the largest database with a total of 93,607 UML diagrams. Although the dataset is public, no annotation is provided on the diagrams. Moreover, the dataset corresponds to a compilation of git repositories, where it is necessary to apply scraping and filtering operations in order to retrieve the desired data. According to the various users of this dataset, around 32,000 diagrams correspond to class diagrams, although the exact number is not known because this information is not included in the metadata. It should be noted that since the dataset was published, many repositories are no longer maintained, which means that access to all diagrams is not guaranteed. Ho-Quang et al. [5] and Shcherban et al. [12] both propose different datasets containing UML diagrams, where the aim is to be able to identify the type of diagram from several categories. In addition to providing no further annotation, only a small subset of the data corresponds to UML class diagrams, which is difficult to exploit for training neural networks for semantic analysis. Karasneh et al. [13] use a dataset containing rich annotations on the elements, content and structure of UML class diagrams. However, the dataset is very small, and the data is not made available. Although there are several datasets available for

TABLE I
List of datasets proposed in the literature for training and evaluating systems to interpret UML diagrams.

	[1]	[12]	[5]	[13]	Proposed
Number of diagrams	93,607	1,300	3,231	N/A	14,500
Class diagram	$\sim$ 32,000	700	650	10	14,500
Type	✗	✓	✓	✗	✓
Structure	✗	✗	✗	✓	✗
Elements	✗	✗	✗	✓	✓
Content	✗	✗	✗	✓	✗
Availability	✓	✓	✓	✗	✓

training systems to interpret the content of a UML class diagram, only a small amount of data is actually usable at present for semantic analysis task.

III. Framework

The NEURAL-UML (iNtelligent rEcognition of strUctuRAl eLements in UML class diagram) framework proposes a strategy for training and evaluating systems for interpreting UML diagrams using deep neural networks. One of the major contributions of this paper is the implementation of a strategy for intelligently collecting and clustering relevant UML diagrams from the various repositories provided in the Lindholmen dataset [1]. Then, the collected data are annotated using a specially developed annotation tool to design a dataset that can be easily exploited by the scientific community. The code implementation for reproducing the dataset and performing a semantic analysis of a class diagram is available online. [1] The metadata from sets A and B in Table II, used to train and evaluate the learning model, are grouped in two respective folders. For set A, the metadata contains the url links for each image, and the coordinates and labels of the various elements contained in the diagrams. For Set B, the metadata is in the same format, with the exception of the url links. In this folder, simply download and move the dataset given by Shcherban et al. [12] into the folder. For both sets, code is provided to load the metadata and display annotations on the images. All the instructions and commands required to run the code and reproduce the experiments are included in the repository.

A. Collecting UML diagrams

To design a comprehensive dataset adapted for semantic analysis, a restructuring of the Lindholmen dataset [1] was proposed. Firstly, a scraping method was applied to the 93,607 web repositories in order to harvest only those files corresponding to UML diagrams. Several links no longer existing, implied a significant reduction of the initial data number. In addition, most of the UML diagrams are not class diagrams. In order to retain only the class diagrams, the pre-trained learning model proposed by Ho-Quang et al. [5] was applied. The result is a subset of 13,800 unannotated class diagrams (Set A). Some elements, such as classes or associations, are represented more than others in class diagrams. Annotating all class diagrams leads to a significant imbalance in annotations. This class imbalance can lead to problems of over-fitting. If the model encounters the same class too often, it will favor that class when learning to maximize its overall performance. However, other poorly represented classes will be neglected and will not be learned or detected by the model. For this reason, only a subset was annotated to overcome this problem. To assess the ability of the proposed approach to generalize well to class diagrams, a second dataset was studied. The second dataset corresponds to the subset of 700 class diagrams proposed by Ho-Quang et al. [5], on which a manual annotation has been performed (set B). This choice is related to the fact that this dataset is easily accessible and the quality of provided data is good.

[1]Code repository: https://gitlab.univ-lille.fr/emmanuel.renaux/neural-uml

TABLE II
Number of manual annotations performed on the two class diagram data subsets.

	Class	Asso.	Inher.	Aggre.	Compos.	Depend.	Realiz.
Set A	1,776	1,777	1,230	1,608	1,433	1,629	1,172
Set B	5,968	1,551	2,839	552	523	353	211
Total	7,744	3,328	4,069	2,160	1,956	1,982	1,383

The dataset used in the proposed framework accounts for a total of 14,500 class diagrams. Of these, seven elements are annotated: classes, association, inheritance, aggregation, composition, dependency and realization. The number of annotations obtained for sets A and B is given in Table II. Set A contains 13,800 class diagrams out of the 32,000 existing diagrams in the Lindholmen dataset [1], after filtering out corrupted url links and unusable diagrams. Only 1,318 were annotated to obtain the annotations presented in Table 2. Indeed, as the annotation task is relatively long (between 10 to 60 seconds depending on the complexity of the diagram), and the number of annotated data sufficiently exhaustive and balanced, annotating all the diagrams doesn't seem essential. However, further enrichment of the dataset with other interesting annotations, such as notes or interfaces, is not excluded in the future. More complex annotations, such as structural relationships between classes, or textual content, can also enrich existing metadata. Set B contains 700 class diagrams, corresponding to all the data contained in the dataset provided by Shcherban et al. [12]. All annotations were made by hand, by three annotators. Despite having only one annotator per diagram to guarantee a large number of annotations, a visual check was carried out by at least one other annotator.

B. Annotation tool

To annotate the data, a tool was designed specifically for this purpose. The tool enables all class diagrams to be browsed and visualized in image format, as illustrated in Fig. 2. The tool consists of two parts: a viewer and a manager. In order to guarantee the accuracy of annotations and adapt to the highly variable size of diagrams, the viewer allows users to scroll through the image using directional arrows. For each image, the annotator can annotate the elements contained in the diagram, from among the various possible relationships. Right-click to assign a relationship to the cursor position in the image. Left-click to delimit the class bounding box. Once an image has been annotated, it is possible to move on to other images, or go back to check or modify annotations. The tool saves each modification in real time, enabling the annotation process to be resumed at any time. As for annotations, every coordinate is saved, along with the associated label in csv format for easy data import and export. The tool is multi-platform: Windows, Unix and Mac.

C. Learning data processing

Several strategies are used to make it easier to learn the model, and thus improving the accuracy of classification and localization of elements within a class diagram.

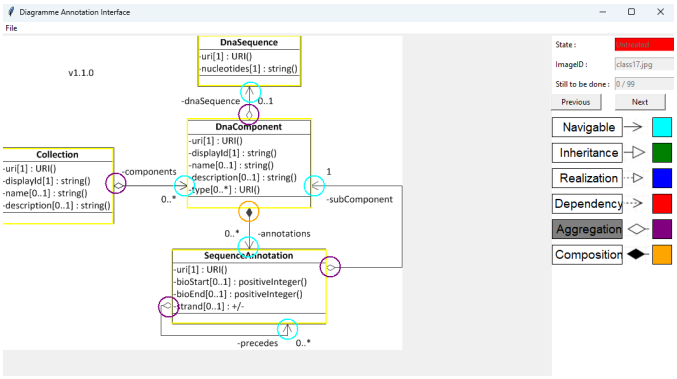

Fig. 2. Annotation tool for class diagrams in two parts. Left, the viewer for browsing the image and viewing annotations. Right, the manager for selecting the type of relationship to assign to the mouse click, and for swapping images.

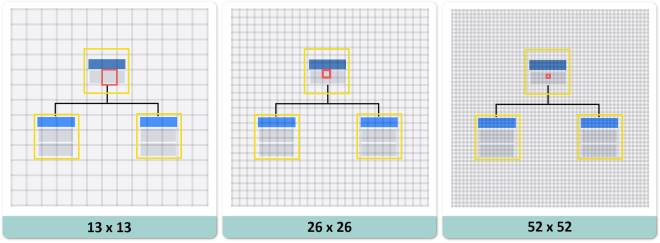

Fig. 3. How YOLO works to analyse images on several levels of scale, and thus identify objects of varying size in the images.

a) Dealing with the variable dimension of the diagram: Learning-based approaches, such as neural networks, generally require the images used as model inputs to be of the same dimension, to guarantee consistency in the dimension of the feature vectors. In this case, a normalization step is applied to resize the images. In some works, as proposed by Gosala et al. [11], the image sizing parameter plays a significant role in model performance. This step is empirical and computationally time-consuming. In addition, image resizing has consequences for data quality, where a compression effect tends to degrade important information such as text, or small details. In the proposed framework, YOLOv8 [14] is used for its ability to adapt to image size, but also for its accuracy and speed of execution. YOLOv8 learning architecture overcomes the normalization step by including multi-scale detection in the convolution layers. The architecture is well suited to the analysis of UML class diagrams, as the dimensions of classes and relationships vary greatly from one modeling tool to another. To perform this detection, YOLO uses three different scales to adapt to different object sizes, using 32, 16 and 8 divisors. Thus, for a 416 x 416 image, YOLO performs a first detection on a 13 x 13 image, a second detection on a 26 x 26 image and a third detection on a 52 x 52 image, as illustrated in Fig. 3.

b) Dealing with unbalanced data: As mentioned in Section III-A, the elements contained in the UML diagrams of set A used to train the model, are not all annotated in order to avoid a major imbalance in the categories. It is therefore not possible to send complete images to YOLO, where some elements are not annotated. This would prevent the learning model from converging. Indeed, in the case where some diagrams are partially annotated while others are fully annotated, an element is considered to belong to class A on one diagram and to class B on another. In this context, the model is unable to interpret the exact class of this element if it is classified differently from one image to another. This is equivalent to an element being incorrectly annotated. To overcome this problem, each class diagram is split into a subset of patches located on each annotated element. For example, if the diagram contains two classes and a navigable relationship between these classes, then three patches are created: two patches corresponding to the classes and one patch corresponding to the navigable relationship. To create these patches, an enclosing window is created around the annotated element.

c) Avoid handling non-essential data: Information contained in diagrams, such as text, manual annotations or elements that are not considered to be learnable, can be considered as a reject class for the learning model. In this study, metadata do not contain elements such as interfaces or notes. Consequently, when the model is trained, these elements are considered as belonging to the ”background” class, which contains all the elements to be excluded during the inference stage. The same applies to text outside classes. On the other hand, text within a class can have an impact on the recognition rate of a class. This is because, during learning, the model focuses on the overall structure of a class, and often on the structuring elements (i.e., border, corner, number of vertical and horizontal strokes). If the name of a class always has a particular feature, e.g., being bold or capitalized, the model will interpret this information as important in its decision-making. On the other hand, if the structure of the classes is similar from one diagram to another, but the textual content always varies, then the model will consider that this information is not redundant, and therefore not essential. This is why, if you want to deal with information other than element structure, it’s essential to take this need into account when designing the apprentice model. In the current study, the YOLO architecture is not suited to interpreting textual content, and tends to ignore it in the learning phase.

D. Implementation details

The architecture of object detectors is divided into three parts: the backbone, the neck and the head, as illustrated in Fig. 4. The backbone is crucial for extracting valuable features from input images, typically using a convolutional neural network (CNN) trained on large-scale image classification tasks. In this study, the architecture is pre-trained on COCO [15]. The backbone captures hierarchical features at different scales. Lower-level features (such as edges and textures) are extracted in previous layers, and higher-level features (such as object parts and semantic information) are removed in deeper layers. The neck is an intermediate component that connects the spine to the head. The YOLO model works through these different steps:

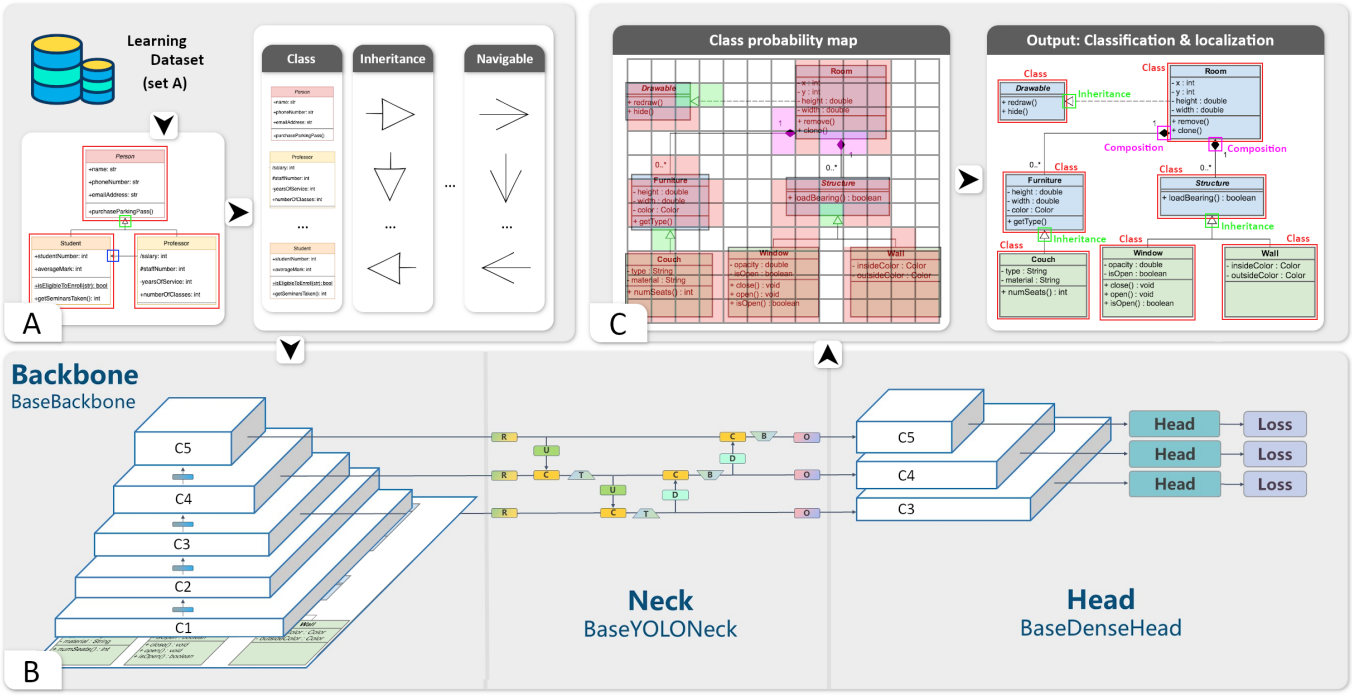

Fig. 4. Overview of the learning method. A) The class diagrams of set A are divided into small patches to obtain clusters of images for each type of structural element. B) The data is injected into a YOLOv8 architecture in order to train the model to recognise these elements in a class diagram. C) Once trained, the learning model takes as input a class diagram of any dimension and predicts both the category of the elements and their localization.

Residual blocks: The first step starts by dividing the original image of the class diagram (A) into NxN grid cells of equal shape, where N in our case is 4 shown on the image on the right. Each cell in the grid is responsible for localizing and predicting the class of the object that it covers, along with the probability/confidence value.

Bounding box regression: The next step is to determine the bounding boxes which correspond to rectangles highlighting all the objects in the image. We can have as many bounding boxes as there are objects within a given image. YOLO determines the attributes of these bounding boxes using a single regression module in the following format, where Y is the final vector representation for each bounding box. Y = [pc, bx, by, bh, bw, c1, c2, ..., c7] where pc corresponds to the probability score of the grid containing an object. bx, by are the x and y coordinates of the center of the bounding box with respect to the enveloping grid cell. bh, bw correspond to the height and the width of the bounding box with respect to the enveloping grid cell. $c1, c2, ..., c7$ correspond to the seven element classes selected in this study.

Metrics: Most of the time, a single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant. The goal of the IOU (a value between 0 and 1) is to discard such grid boxes to only keep those that are relevant. YOLO computes the IOU of each grid cell which is the Intersection area divided by the Union Area. Finally, it only takes into account predictions whose IOU is greater than a fixed threshold.

In this article, the latest version of YOLO is used. As a state-of-the-art model, YOLOv8 builds on the success of previous versions, introducing new features and enhancements to increase performance, flexibility and efficiency. YOLOv8 supports a full range of visionary AI tasks, including detection, segmentation, pose estimation, tracking and classification. YOLO is based on the idea of segmenting an image into smaller images. The image is split into a square grid of dimensions $S \times S$. The cell in which the center of an object resides, is the cell responsible for detecting that object. Each cell will predict B bounding boxes and a confidence score for each box. The default for this architecture is for the model to predict two bounding boxes. The classification score will be from 0.0 to 1.0, with 0.0 being the lowest confidence level and 1.0 being the highest; if no object exists in that cell, the confidence scores should be 0.0, and if the model is completely certain of its prediction, the score should be 1.0. These confidence levels capture the model's certainty that there exists an object in that cell and that the bounding box is accurate. Each of these bounding boxes is made up of 5 numbers: the x position, the y position, the width, the height, and the confidence. The coordinates (x, y) represent the location of the center of the predicted bounding box, and the width and height are fractions relative to the entire image size. The confidence represents the IOU between the predicted bounding box and the actual bounding box, referred to as the ground truth box. The IOU stands for Intersection Over Union and is the area of the intersection of the predicted and ground truth boxes divided by the area of the union of the same predicted and ground truth boxes. In addition to outputting bounding boxes and confidence scores, each cell predicts the class of the object. This class prediction is represented by a

one-hot vector length C, the number of classes in the dataset. Thus, each prediction from a grid cell will be of shape C + B x 5, where C is the number of classes and B is the number of predicted bounding boxes. B is multiplied by 5 because it includes (x, y, w, h, confidence) for each box. Because there are $S \times S$ grid cells in each image, the overall prediction of the model is a tensor of shape $S \times S \times (C + B \text{ x } 5)$.

Compared with older versions of YOLO, YOLOv8 is an anchor-free model. This means it directly predicts the centre of an object instead of the offset from a known anchor box. Anchor boxes are a predefined set of boxes with specific heights and widths, used to detect classes of objects with the desired scale and aspect ratio. They are chosen according to the size of the objects in the training dataset and are arranged in a mosaic over the image during detection. The network generates probabilities and attributes such as background, aperture angle and offsets for each tiled box, which are used to adjust the anchor boxes. Multiple anchor boxes can be defined for different object sizes, serving as fixed starting points for estimating bounding boxes. The anchor-free analysis of YOLOv8 tends to directly predict the centre of an object instead of the offset from a known anchor box. The advantage of anchor-free detection is that it is more flexible and efficient, as it does not require the manual specification of anchor boxes, which can be difficult to choose and can lead to suboptimal results in previous YOLO models such as v1 and v2.

Another interesting feature for our case study has been introduced since version 7 of the model. YOLOv7 also introduces a new multi-scale training strategy, which involves training the model on images at several scales and then combining the predictions. In addition, YOLOv7 incorporates a new technique called 'Focal Loss', designed to address the problem of class imbalance that often arises in object detection tasks. Focal Loss gives greater weight to data that is more complex to encode for the model, and reduces the influence of data that is less complex to interpret or more redundant. This involves influencing the learning of the model by forcing it to focus on certain details at the expense of others. This technique helps the model to converge when there is a strong imbalance of classes, or when learning is conditioned by expert knowledge.

IV. Experimentation

In this section, we present an evaluation for the semantic analysis of a class diagram using the proposed NEURAL-UML framework. Using the set A and B as input to a neural network, we perform an evaluation of the model predict element types for seven cases, including the classes and the six relationship types: Association, Inheritance, Aggregation, Composition, Dependency and Realization. That is, for a given class diagram, the aim is to classify and locate all the structuring elements contained in the image.

A. Evaluation protocol

In order to evaluate the decision making model, a k-fold cross validation is used, with k = 5. This protocol aims to check the good performance of the model when faced with unknown data. The performance of the model is measured by the mean accuracy over all k-folds, as well as the standard deviation. As shown in Fig. 5, in this scenario the method consist to divide the dataset into five folds. The model uses the first fold in the first iteration to test the model (test). It uses the other datasets to train the model. The second fold helps to test the dataset (validation) and the others support the training process (train). The same process is repeated until the test set uses each of the five folds. In this study, an extension of classical k-fold cross-validation is applied. Instead of the divisions being completely random, we apply stratified cross-validation, which consists of ensuring that the ratio between the target classes is the same in each fold as in the entire data set.

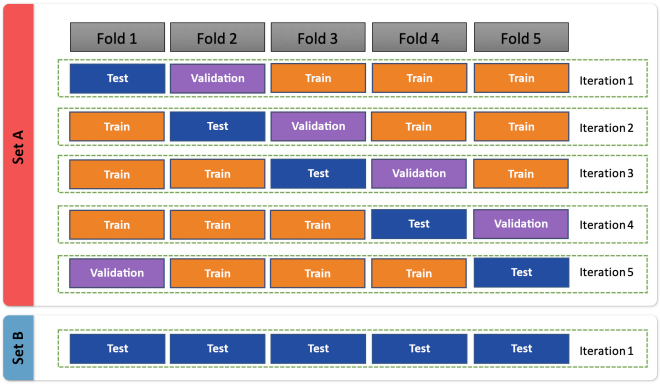

Fig. 5. K-fold Cross-validation, with k = 5. Set A is divided into three parts: Train, Validation and Test at each iteration. Only set A is used to train the model. Set B is used only to test the model inference, in a single iteration.

To learn the model, cross-validation is applied only to the set A. Set B is used only to assess the capacity of the model to generalise to unknown data, with different formats to the data used for training. In this scenario, all the data is grouped together in a unique test fold. Only the inference of the model is evaluated, and no training is carried out on these data.

B. Performances metrics

To determine and compare the predictive performance of the proposed model, three most common evaluation metrics are used: Intersection over Union (IoU), F1 score, and k-fold cross-validation accuracy metrics.

a) Intersection over Union (IoU): The union intersection is a popular metric for measuring the localization accuracy and calculating the localization errors in object detection models. To calculate the intersection between the predicted bounding boxes and the ground truth bounding boxes, the area of intersection between the two corresponding bounding boxes for the same object must be calculated. Next, the total area covered by the two bounding boxes, also known as the "Union", and the area of overlap between them, known as the "Intersection", is calculated. The intersection divided by the union gives the ratio of the overlap to the total area, which provides a good estimate of how close the prediction bounding box is to the original bounding box. The IoU threshold α decide whether the prediction is True Positive (TP), False Positive (FP), or False Negative (FN). The Fig. 6 illustrate predictions with the

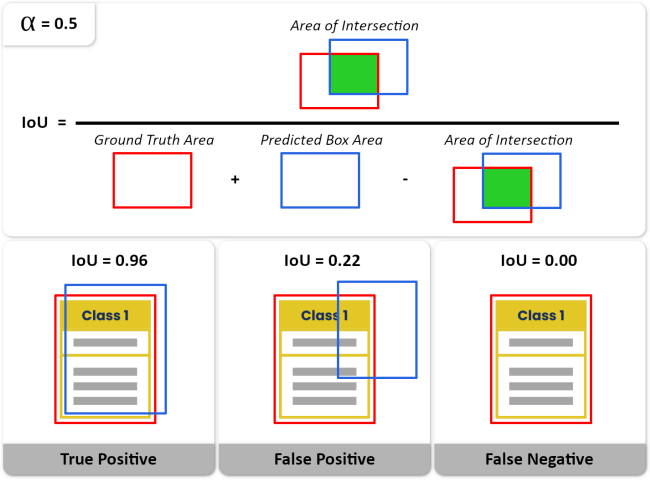

Fig. 6. An example of computing Intersection over Unions for various bounding boxes on a UML class diagram, where α threshold is set at 0.5.

IoU threshold α set at 0.5. The decision of making a detection as True Positive or False Positive completely depends on the requirement. The first prediction is True Positive as the IoU α threshold is 0.5. If the threshold is set at 0.97, it becomes a False Positive. Similarly, the second prediction shown above is False Positive due to the threshold but can be True Positive if the threshold set at 0.20. Theoretically, the third prediction can also be a True Positive, if the threshold is lowered to 0.

b) F1 score: F1 score is an evaluation metric that assesses the predictive skill of a model by elaborating on its class-wise performance rather than an overall performance as done by accuracy. F1 score combines two competing metrics-precision and recall scores of a model. F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. It is calculated from a confusion matrix. A confusion matrix represents the predictive performance of a model on a data set.

Based on the True Positive (TP), False Positive (FP), and False Negative (FN), for each labeled class, two parameters: precision and recall are calculated.

Recall gives the percentage of positives correctly predicted by the model. In other words, it is the number of well-predicted positives (True Positives) divided by the total number of positives (True Positives + False Negatives). Its mathematical form is as follows:

$$Recall = \frac{TP}{TP + FN}. \tag{1}$$

The higher it is, the more the model maximises the number of True Positives. However, this does not mean that the model does not make mistakes. When the recall is high, it means that it will not miss any positives. However, this does not give any information about its predictive quality on negatives.

Precision is a measure of the number of positive predictions made. It is the number of correctly predicted positives (True Positives) divided by the total number of predicted positives (True Positives + False Positives). In mathematical form, this gives :

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

The higher it is, the more the model minimises the number of False Positives. When the accuracy is high, this means that the majority of the positive predictions made by the model are well-predicted positives.

Although they are useful, neither precision nor recall can fully evaluate the model. Taken separately, these two metrics are useless. If the model predicts 'positive' all the time, recall will be high. On the other hand, if the model never predicts "positive", precision will be high. The F1 Score provides a good assessment of the performance of the model. It is calculated as follows:

$$\begin{aligned} F1 &= \frac{2 * Precision * Recall}{Precision + Recall} \\ &= \frac{2 * TP}{2 * TP + FP + FN} \end{aligned} \tag{3}$$

c) K-fold cross validation accuracy: The most common way to measure the predictions made by the model match the observed data is by using the mean squared error (MSE), which is calculated as:

$$MSE = (1/n) * \sum (y_i - f(x_i))^2 \tag{4}$$

where n is the total number of observations, y_i correspond to the response value of the i^{th} observation and f(x_i) is the predicted response value of the i^{th} observation. The closer the model predictions are to the observations, the smaller the MSE will be. Typically, the test is performed on a subset of the dataset. It gives an idea of a model's performance on data it has never seen before. However, the disadvantage of using only one test set is that the MSE can vary considerably depending on the observations used in the training and test sets. One way to avoid this problem is to fit a model several times using a different training and testing set each time, then calculating the test MSE to be the average of all of the test MSE's. This general method is known as cross-validation and a specific form of it is known as k-fold cross-validation, as illustrated in Fig. 5. The dataset is randomly divided into k groups, or "folds", of roughly equal size. One of the groups is used as a test set and the others as a training set. The MSE is then calculated from these two subsets of data. The process is repeated k times, each time using a different set as the test set. The overall test accuracy being the average of the k test MSEs and is calculated as follows:

$$MSE = (1/k) * \sum MSE_i \tag{5}$$

where k in the number of folds and MSE_i is the test MSE on the i^{th} iteration. The standard deviation calculated from the k-fold accuracies obtained ensures that the model generalizes well when trained on different folds.

C. Quantitative results

Applying the proposed learning model to the set A, using stratified k-fold cross-validation (k=5), we obtained an average accuracy of 92.59% over seven categories. Considering the performance given in the confusion matrix Fig. 7, the model performs well on all categories. Among the errors observed, the categories corresponding to the {Inheritance-Realization} relationships are the least well classified. The model sometimes tends to confuse classes with relationships. This can be explained by the fact that the annotations often overlap, given their close proximity in the class diagram.

As for the evaluation on the set B, by only performing inference with the model trained exclusively on set A, the results observed are very positive, with an average accuracy of 94.11%. The confusion matrix in Fig. 8 shows that the model generalises perfectly from one set to another. It is important to note that there is less confusion between classes and relationships. The majority of errors observed concern the {Inheritance-Realization} and {Navigable-Dependency} relationships. This can be explained by their close similarity in their graphical representation in the class diagram.

The results observed on both datasets A and B are very positive and highlight the ability of the proposed model to correctly classify the seven structural elements included in a class diagram. In addition, the two annotated datasets respond perfectly to the need to develop new models to facilitate the emergence of automatic classification and interpretation of class diagrams.

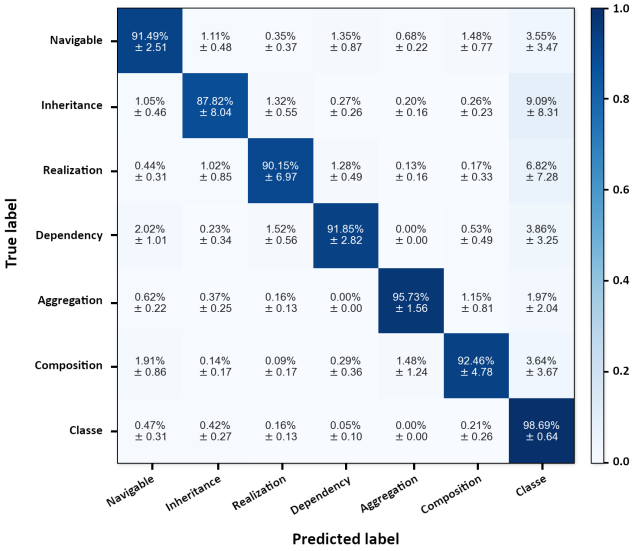

Fig. 7. Confusion matrix on the set A consisting of 10,625 annotations divided into 7 categories. The results correspond to the average precision obtained in each fold, as well as the standard deviation for each category.

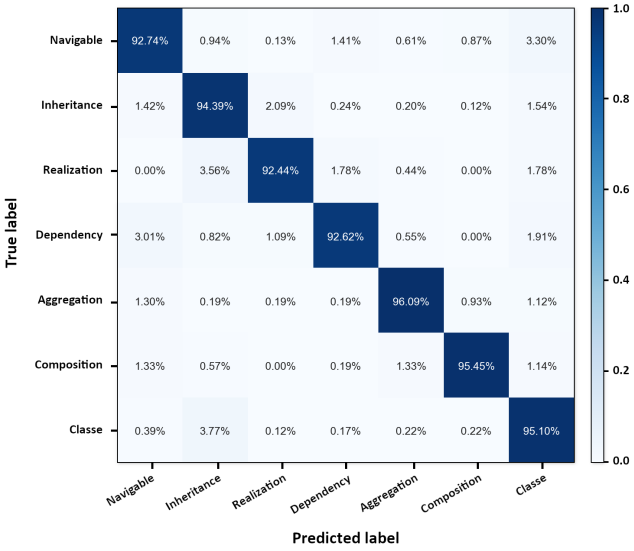

Fig. 8. Confusion matrix on the set B consisting of 11,997 annotations divided into 7 categories. The results correspond to the precision obtained on all the data for each category.

D. Qualitative results

In addition to the quantitative study, a qualitative study was carried out to examine the capacity of the model to localise predictions. Two metrics were used: the IoU and the F1 score. For set A, the calculated IoU is 85%. The F1 score is calculated according to the distance between the centroid of the region corresponding to the ground truth and the centroid of the predicted region. The distance between centroids corresponds to the tolerance threshold that defines whether a prediction is considered a true positive or a false positive. This distance is estimated in pixels. As shown in Fig. 9, the model quickly achieves good accuracy with a relatively small deviation of just 2 pixels from the ground truth. The Aggregation and Composition relationships are easier for the model to locate. This is due to their easily identifiable symbol in the class diagram. On the other hand, the Navigable relationship remains the most complex to localize. The model tends to localize this relationship to within 8 pixels of its exact position. Compared with the other relationships, this

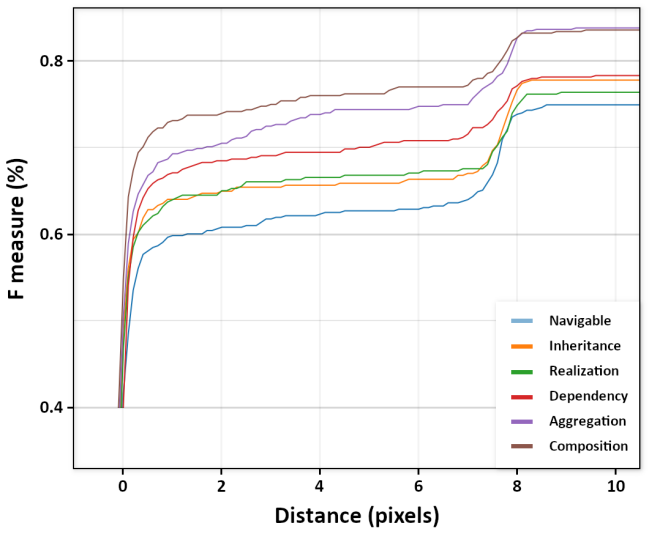

Fig. 9. Localization accuracy of structural elements in set A as a function of the distance tolerance between predicted and real Localization.

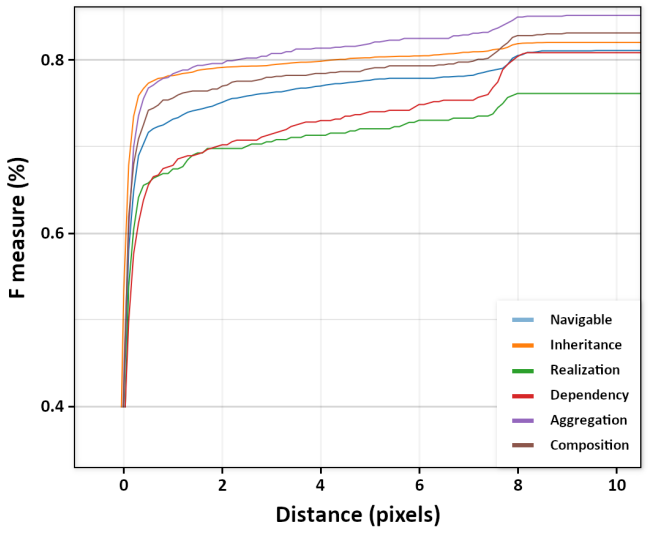

Fig. 10. Localization accuracy of structural elements in set B as a function of the distance tolerance between predicted and real Localization.

can be explained by the highly variable graphical style of the Navigable relationship in the various class diagrams analysed.

The same qualitative study was carried out on set B. The IoU calculated was 87.95%. Fig. 10 shows the F1 score according to the distance between the predicted centroids and the ground truth centroids. Although the model is more accurate, the results follow the same trend as in the previous study. The performance for the navigable relationship is better on this dataset. This may be due to the fact that the graphical style of the class diagrams is more standardised, which reduces the appearance of complex formats.

V. Discussion and Conclusion

UML notation is used throughout the software development process as a common representation. Thanks to this lingua franca, it is possible to hold a design meeting, build a shared model of the problem and propose a design plan. However, as the UML tools are perceived as heavy to use and the UML language difficult to learn, designers prefer to write the UML manually using a pencil and a sheet of paper or a whiteboard and in an informal manner. In practice, UML diagrams are often simply photographed and stored as images, which limits their use and evolution.

In this article, we presented an experiment using the advances in the field of deep learning to recognise UML class diagrams. We propose a framework for training a learning model to categorise and locate semantic elements in class diagram from an image. The performance (Set A- IoU: 85%,ACC: 92.59% ; Set B- IoU: 87.95%, ACC: 94.11%) obtained by the proposed model highlights the capacity of the model to generalise to a large dataset.

One of the main contributions of this paper is the implementation of a sourcing strategy to gather and group relevant UML diagrams from the different repositories available on the web. The other contribution is the annotation tool that allows to annotate the collected data. This tool has been specially developed for creating a dataset and can be easily exploited by the scientific community. Finally, the aim of the paper is to allow reproduction of the data set and to carry out a semantic analysis of a class diagram.

In view of the performance achieved by the proposed approach, we can expect UML practice to evolve towards a more flexible software design method. There is still work to be done to improve the accuracy of recognising diagrams from modelling tools and then to tackle freehand diagrams. The next step is to use this semantic extraction to think about usage in software design. In particular, precision can be improved by cross-referencing with the theme of the project, its lexical field and the overall context of the IT project (Jira, Kanban board...). And if we go even further, we can imagine a recommendation engine that helps the designer to draw up a model of a software using augmented reality, without being a UML expert and using a simple whiteboard.

References

[1] 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 *Proceedings of the ACM/IEEE 19th international conference on model driven engineering languages and systems*, 2016, pp. 173–183.

[2] F. Wang, "Uml diagram classification model based on convolution neural network," *Optik*, p. 170463, 2022.

[3] N. Bnouni Rhim, S. Cheballah, and M. Ben Mabrouk, "Cross synergetic mobilenet-vgg16 for uml multiclass diagrams classification," in *International Conference on Innovations in Bio-Inspired Computing and Applications*. Springer, 2022, pp. 24–30.

[4] F. Chen, L. Zhang, X. Lian, and N. Niu, "Automatically recognizing the semantic elements from uml class diagram images," *Journal of Systems and Software*, vol. 193, p. 111431, 2022.

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

[6] T. De-Wyse, E. Renaux, and J. Mennesson, "Using sketch recognition for capturing developer's mental models," in *The 3rd Workshop on Human Factors in Modeling*, 2018.

[7] J. Perianez-Pascual, R. Rodriguez-Echeverria, L. Burgueño, and J. Cabot, "Towards the optical character recognition of dsls," in *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, 2020, pp. 126–132.

[8] W. Torres, M. G. van den Brand, and A. Serebrenik, "Xamã: Optical character recognition for multi-domain model management," *Innovations in Systems and Software Engineering*, pp. 1–25, 2022.

[9] Ö. Babur, L. Cleophas, and M. van den Brand, "Hierarchical clustering of metamodels for comparative analysis and visualization," in *European conference on modelling foundations and applications*. Springer, 2016, pp. 3–18.

[10] D. Strüber, M. Selter, and G. Taentzer, "Tool support for clustering large meta-models," in *Proceedings of the workshop on scalability in model driven engineering*, 2013, pp. 1–4.

[11] B. Gosala, S. R. Chowdhuri, J. Singh, M. Gupta, and A. Mishra, "Automatic classification of uml class diagrams using deep learning technique: convolutional neural network," *Applied Sciences*, vol. 11, no. 9, p. 4267, 2021.

[12] S. Shcherban, P. Liang, Z. Li, and C. Yang, "Multiclass classification of four types of uml diagrams from images using deep learning," *mayo de*, 2021.

[13] B. Karasneh and M. R. Chaudron, "Extracting uml models from images," in *2013 5th International Conference on Computer Science and Information Technology*. IEEE, 2013, pp. 169–178.

[14] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[15] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312