Al-Rafidain Journal of Engineering Sciences

Journal homepage https://rjes.iq/index.php/rjes

ISSN 3005-3153 (Online)

Enhanced Classification of UML Class Diagram Components Using Deep Learning for IT and Engineering Solutions

Maha Khalil Ibrahim [1*], Nadia Mahmood Hussien [2]

[1]Mobile Communications and Computing Engineering Department, College of Engineering, University of Information Technology and Communications, Baghdad, Iraq

[2] Department of Computer Science, Collage of Science, Mustansiriyah University, Baghdad, Iraq

ARTICLE INFO

Article history:
Received 5 April 2026
Revised 5 April 2026,
Accepted 28 April 2026,
Available online 1 May 2026

Keywords:
Information Technology
UML Class Diagram
Deep Learning
CNN
Software Engineering Automation
Visual Classification
Reverse Engineering

ABSTRACT

UML class diagrams are key to object-oriented software development, capturing structural descriptions of software systems. But manual interpretation and analysis of UML diagrams is time-consuming, prone to errors and not scalable, especially during software maintenance and reverse engineering. In this paper, a deep learning model is presented to automate recognition of UML class diagram components by using a Convolutional Neural Network (CNN). The proposed approach does not use text-based features of UML class components and instead uses only visual features to classify UML class components into four classes: Entity, Service, Controller, and Utility. A dataset of real UML class diagrams was labeled for supervised learning. Our model outperformed baseline methods, with a classification accuracy of 92.3% on the test set and generalised well to different diagram types. This paper makes the following contributions: (1) a lightweight CNN-based model for UML component classification, (2) a benchmark labeled dataset, and (3) comparison of visual-only learning with traditional techniques in UML component classification. This research advances the field of Software Engineering, especially Computer-Aided Software Engineering (CASE), by facilitating smart automation in software modeling and documentation tools. The paper presents a feasible and effective roadmap to intelligent software modeling tools and automated documentation systems.

1. Introduction

Over the last few decades, there has been a rapid increase in the number of data-centric and networked software systems, scientific models and theories that has led to a demand for better modeling and documentation methods. Unified Modeling Language (UML) is a powerful and popular standard for the static and dynamic behavior of software systems.UML class diagrams document classes, attributes, methods and their associations, which are essential for designing an object-oriented system [1,2]. However, analysis of manually drawn UML diagrams is inefficient in software engineering. The understanding and classification of UML elements in software maintenance, reverse engineering and system migration, especially for legacy systems, are tedious, inconsistent, error-prone and lack scalability [3,4]. Automated tools typically use heuristics or Optical Charac-

Corresponding author E-mail address: maha.ibrahim@uoitc.edu.iq
https://doi.org/10.61268/g8frsb68

ter Recognition (OCR) to obtain textual and structural information. But, the diversity of diagram styles, low quality images, and misaligned text limit the reliability and accuracy of these techniques [5,6]. Deep learning techniques, such as Convolutional Neural Networks (CNNs), have greatly improved computer vision. CNNs have been successfully used for image recognition, including facial recognition, self-driving vehicles, and medical image analysis. They are particularly adept at learning spatial hierarchies and representations, which are ideal for UML diagrams [7,8].However, little work has been done on using CNNs for UML class diagram interpretation in software engineering. Existing solutions are mainly based on OCR or rule-based techniques which are very sensitive to layout and visual artifacts, and are not robust to different diagram styles [9]. Therefore, the main research question this study seeks to answer is: "How can UML class diagram components be automatically and correctly classified, without using OCR or rules, but just visual information?" The objectives of this research are: (1) to develop a CNN model for classifying UML class diagram components; (2) to develop a UML class component dataset for supervised learning; (3) to investigate the effectiveness of visual-only classification; and (4) to explore how the model performs with different diagram styles. The main problems are the visual similarities between UML components, lack of a labeled dataset and styles of diagrams. This study proposes a deep learning model which accepts the original images of UML class diagrams as input to solve these issues. The CNN classifies UML components (Entity, Service, Controller and Utility) purely from their visual appearance. This visual approach adds to the ability to generalize across different diagrams and eliminates the need for OCR or rules. The main contribution and novelty of this research is: (1) a dataset of UML class components; (2) a small but effective CNN model; and (3) a thorough evaluation of the model. This tool can be used for software design, reverse engineering and documentation. The rest of this paper is organized as follows: Section 2 discusses background research, Section 3 presents the proposed method, Section 4 presents the experimental results, Section 5 discusses the results and finally Section 6 concludes and suggests future work.

2. Related Works

The use of automated classification and generation of UML diagrams has been of great interest over the last few years. Several approaches that use deep learning, natural language processing (NLP), and rule-based systems to simplify the UML diagram processing have been suggested by researchers. Nevertheless, the majority of the methods have a limited scope or generalization capacity.

Silva and Carvalho [10] sought to categorize 6 types of UML diagrams with the help of Convolutional Neural Networks (CNNs) and transfer learning. Their research showed the usefulness of transfer learning and data augmentation, which yielded reasonable results in terms of classification with VGG16, ResNet50, and InceptionV3 on small datasets.

The study of Azmi and Mehmood [11] was done on a large scale where 4,706 images in 11 categories (10 types of UML diagram and non-UML images) were used. They compared seven state-of-the-art CNN models and demonstrated that a lightweight architecture is as good as exception in performance, but at a lower cost of computation.

Javed and Malik [12] made particular attention on UML class diagram classification. Their CNN-based tool was able to support multiple types of image formats (JPEG, PNG, GIF, TIFF) and to automatically localize UML class diagrams without any labeled training examples (which increases automation).

Hussain and Nawaz [13] took the task of defining the textual software requirements as UML class diagrams. They combined a rule-based mechanism and a Naive Bayes classifier to

detect named entities, which narrows the ambiguity and a recall of 89.44.

Reyes and Olivas [14] came up with an educational support system in which the neural network based on features differentiated UML diagrams. Their approach was effective than the traditional image-based classifier and allowed instructors to assess student submitted diagrams more effectively.

To learn the structural connections among the UML diagrams, Zhang and Chen [15] suggested Graph Convolutional Networks (GCNs). Their model demonstrated how GCNs can offer structural information CNNs do not offer through the use of topological and spatial variables.

These papers show how diverse UML diagram automation can be, ranging all the way to vision-based CNNs to structural GCNs up to text-based rule-based systems. Table 1 is a comparison between these past efforts and the one proposed in this study.

Table 1: Comparative Analysis between Prior Work and the Proposed System

Study / Reference	**Objective**	**Methodology**	**Classification Target**	**# of Classes**	**Accuracy / F1**	**Generalization Scope**
Silva & Carvalho (2021)	Classify UML diagram types	CNN + Transfer Learning	6 UML diagram types	6	Moderate (not specified)	Limited dataset
Azmi & Mehmood (2021)	Multiclass UML diagram classification	Transfer Learning on CNNs (e.g., Xception)	10 UML types + non-UML	11	F1 = 92.73% (Xception)	Strong across types
Javed & Malik (2022)	Identify UML class diagrams vs. non-UML	CNN	Binary (class vs. non-class)	2	Not stated	Good image variety
Hussain & Nawaz (2023)	Generate class diagrams from text	Rule-based + Naïve Bayes	UML class elements (from text)	4 relationships	Precision = 83.72%, Recall = 89.44%	Text-based scope
Reyes & Olivas (2024)	Educational support tool for UML diagram classification	Feature-based ML + CNN	Image-based UML types	Not stated	Better than baseline	Educational domain
Zhang & Chen (2024)	Structure-aware diagram analysis	Graph Convolutional Network	UML diagram structure	Structural-level	Not specified	High for structural parsing
Proposed System **(Our Work)**	Fine-grained classification of class components	CNN trained on visual UML data	UML class roles: Entity, Service, Controller, Utility	4	**Accuracy = 92.3%**	High generalization across styles

Despite significant progress on the analysis of UML diagrams using CNNs, OCR-based methods and graph-based methods, there are several drawbacks in the existing literature. CNNs are primarily limited to coarse-grained classification tasks and may not capture fine-grained details of components. The use of OCR can suffer from quality, font and layout variations, making them less practical. On the other hand, graph-based techniques such as GCNs would require explicit graph structures, which are not easily accessible from UML diagrams. Moreover, existing models are usually based on large datasets and complex models, which might be impractical for light and efficient systems. Therefore, there is a need for an efficient and lightweight visual-only approach to directly process UML diagrams with high accuracy and generalization.

3. Methodology

In this study, a high-resolution, image-based CNN-based classification network of UML class diagram components is provided. The phases of methodological pipeline are dataset development and annotation, picture preprocessing, architectural selection and training approach. In automated software modeling environments, scalability, generalizability, and practicality were taken into account at every step.

3.1 Dataset Construction and Annotation

Learning under supervision needs good data. The training data were generated based on real-life UML class diagrams (academic and open-source). Annotations of the XML were made in each diagram with the coordinates of the bounding boxes of the class components.

The detection of instances by cutting down each box of the classes into an image patch according to these annotations enabled the detection of components on an instance level. According to the software architecture tasks, each of the cropped pictures was classified manually in one of four semantic categories:

- Entity: Model of domain models or basic data structures.
- Service: Service is defined as business logic units or processing units.
- Control: Manages the interaction of models and views, input/output flow.
- Utility: Module wide stateless or helper functions.

The hierarchical system to keep the categories in subfolders under /classes_split/ enabled learning of label automatically depending on the folder names.

This structured and annotated data has become a reference point in categorizing UML components on a fine-grained scale and a competent foundation on CNN models that understand semantic diagrams.

3.2 Image Preprocessing

All images in the cropped class component were preprocessed through a standardized pipeline so that they had uniformity and computational efficiency. The pictures were scaled to 128 x 128 pixels, brought to the [0, 1] scale, and loaded with the help of the ImageFolder utility of PyTorch. This process balanced the trade-off between the retention of detail and the memory and processing efficiency, fostering quicker learning and diminishing the data processing overhead cost. This is an optimized dataset utilized in learning on a graphics card, which boosted the models training efficiency.

3.3 CNN Architecture Design

In order to reduce overfitting, and to take advantage of the resources at hand, a lightweight custom CNN architecture was devised to be deployable on resource constrained devices given the structured complexity of UML class diagrams and the limited number of class categories.

- Convolutional Layer 1: This layer captures the basic low-level spatial features, such as edges and textures.
- ReLU Activation: Adds non-linearity to the model, which allows it to learn elaborate feature hierarchies.
- MaxPooling Layer: Spatial dimensions are reduced, and it is translation-invariant.
- Convolutional Layer 2: The high-level semantic features are extracted in connection with UML classification.
- Activation Relu + Maxpooling: Further abstracts and eliminates the feature maps.
- Fully Connected Layer: It flattens the features and uses a dense transformation to project the extracted features into scores on the classes.

- Softmax Output Layer: This is done through converting the scores into a

probability distribution over the four class labels.

The architecture was selected as it has few parameters, easy to interpret, and being suitable in learning domain-specific UML components, in terms of computational efficiency and classification accuracy.

3.4 Training Protocol and Optimization

The model was trained using the following set up:

i. Optimizer: Adam optimizer was chosen as it has adaptive learning rate and converges at faster rates than the conventional Stochastic Gradient Descent (SGD).
ii. Learning Rate: A starting learning rate of 0.001 was taken, which was successful in the empirical studies of related image classification problems.
iii. Batch Size: It was selected that mini-batch size should be 32 to balance between the training speed and stable gradient updates.
iv. Epochs: The epochs of the model were 5 that was enough to converge without overfitting.

PyTorch was used to train with the help of hardware with a built-in GPU. The accuracy and loss of the training and validation sets were monitored after each epoch, which allowed monitoring and early breaking of the process to prevent overfitting.

More importantly, the method is based purely on image-level features, and it does not require Optical Character Recognition (OCR) or rule-based extraction methods. This is an image-only classification strategy that has a number of major benefits:

- Domain independence (independence of certain structure or formatting of text)
- Layout resilience (tolerant to drawing styles)
- Increased tolerance to old or noisy UML diagrams in which text parsing is not reliable.

The model can be used to learn at the pixel-level data and therefore, it illustrates that purely visual UML understanding is viable, especially in reverse engineering. Table 2 summarizes the system's functional and non-functional requirements. In this study, the hyperparameters were not randomly selected, but determined empirically based on the validation results. The learning rate was set to 0.001 as it resulted in smooth convergence without oscillations in initial experiments. The batch size of 32 was chosen as a compromise between efficiency and stability of the gradient, especially on GPUs. The training epochs were set to 5 epochs as continuing training did not result in improved validation accuracy and signs of overfitting appeared. Based on the above considerations, the hyperparameters were determined for optimal performance and generalization. To assess the generalizability of the proposed model, we initially divided the dataset into training (70%), validation (15%) and testing (15%) sets with stratification to maintain class distribution. While k-fold cross-validation is often used to improve statistical performance, it was not used here because the dataset was small and the computational cost high. But the inclusion of a validation set was beneficial in tracking model performance and early detection of overfitting. In future work, k-fold cross-validation will be used to enhance the model's performance.

Table 1: Functional and Non-Functional Requirements of proposed system.

ID	Requirement Type	Requirement Description
FR-01	Functional	The system shall allow the user to upload UML class diagram images.
FR-02		The system shall parse the corresponding XML annotation files for bounding box data.
FR-03		The system shall crop and extract individual class boxes from UML diagrams.
FR-04		The system shall classify each class box into Entity, Service, Controller, or Utility.
FR-05		The system shall display the classification results to the user.
FR-06		The system shall allow exporting classification results in formats like CSV or JSON.
FR-07		The system shall integrate with modeling tools or IDEs for reverse engineering.
NFR-01	Non-Functional	The system shall achieve at least 92% classification accuracy.
NFR-02		The system shall return classification results within 5 seconds per image.
NFR-03		The system shall support image formats such as JPEG and PNG.
NFR-04		The system shall maintain uptime of 99% or higher.
NFR-05		The system shall be scalable for larger UML datasets.
NFR-06		The system shall ensure data privacy and secure access control.
NFR-07		The system shall provide a user-friendly and intuitive graphical interface.

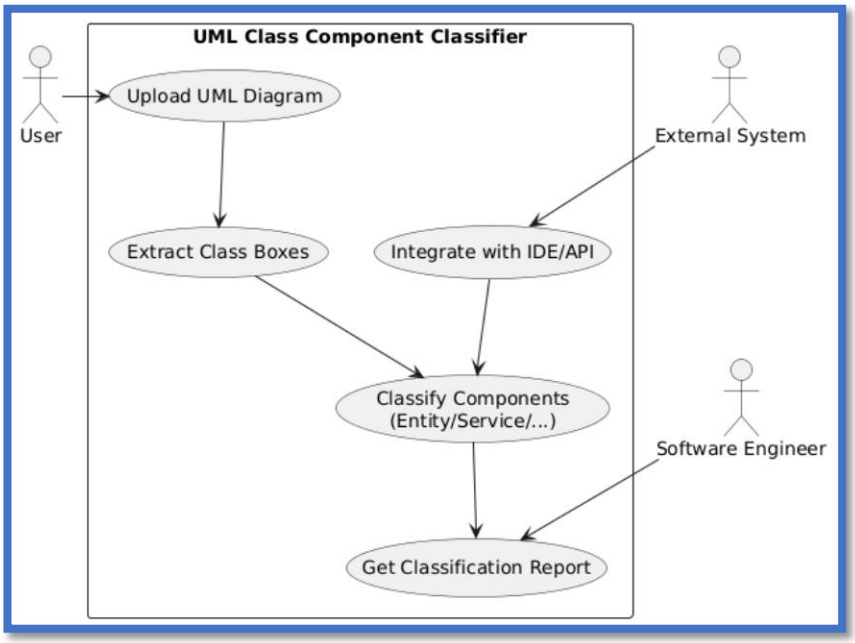

Figure 1: Use Case Diagram for CNN-Based UML Component Classification System

The use case diagram [16] shows the basic interactions of the end users (e.g., a researcher or a developer) to the proposed system. Users start with uploading UML class diagram images and the XML annotation files of the images. These

inputs are then broken down by the system, and the applicable elements of classes are harvest-

ed, and they are sorted into semantic groups: Entity, Service, Controller, or Utility.

The diagram also indicates the system integration with external tools, including software engineering IDEs, which can consume and use the results of the classification to develop, analyze, or reverse engineer workflow further [17]. Figure 1 reflects this interaction graphically and the flow of collaboration between users and the system is evident.

The following is the class diagram [18] of the proposed system:

- UMLImageProcessor: This module is in charge of input parsing, class boxes This architecture is also modular and reusable and is in line with the principles of the object-oriented design.

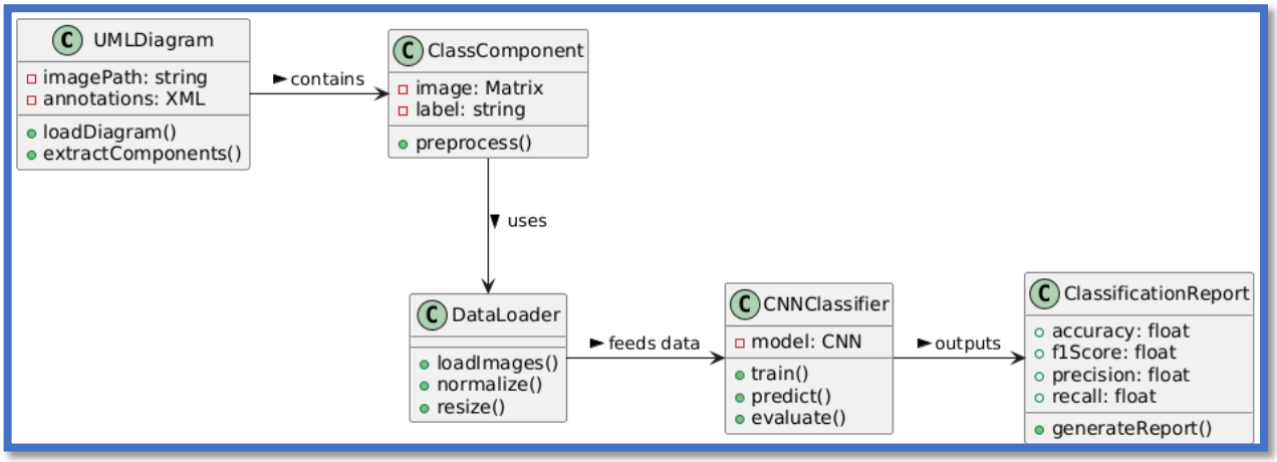

Figure 2: Class Diagram Representing the Internal Structure of the UML Classifier System

this model in prediction of the semantic label of each cropped image.

- ComponentLabel: This is used to define and map the semantic roles of UML elements (e.g., Entity, Service, Controller, Utility).cropping of the UML image, and the start of the classification pipeline.
- CNNClassifier: This is the CNN model that has been trained and encapsulates
- ClassificationResult: This stores and processes the classification results to be consumed later either by downstream processing or visualization.

The activity diagram [19], shown in Figure 3, outlines the step-by-step flow of the UML component classification system This diagram captures the dynamic behaviour of the system in a process-oriented view, offering insight into the sequential logic and data transformations.

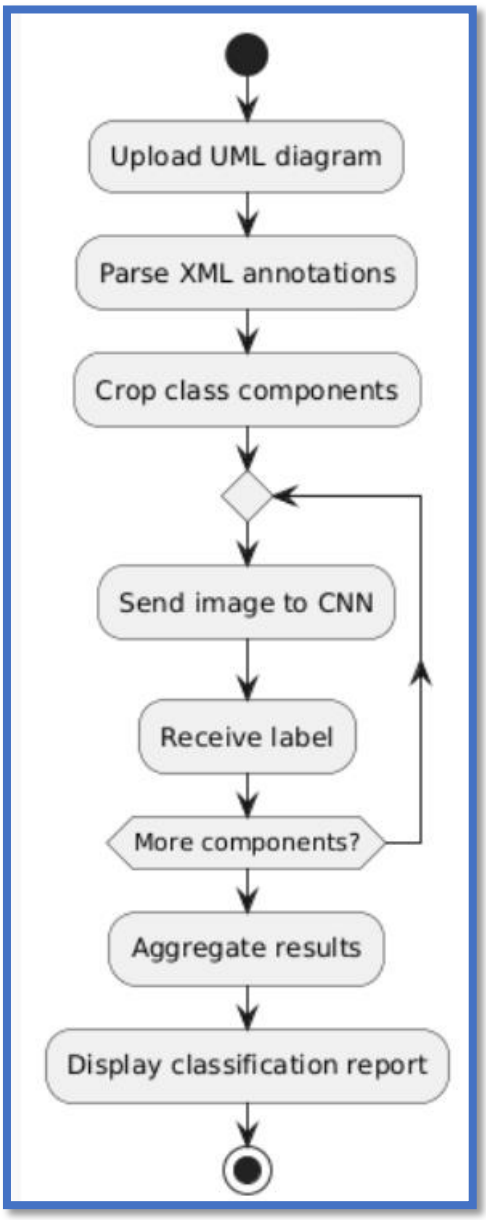

Figure 3: Activity Diagram Illustrating the Component Classification Workflow

As illustrated in Fig. 4, the sequence diagram [20] shows the timing relationship between system components and the user:

• The user provides the application with a UML class diagram along with an XML file via the user interface.

• The UMLImageProcessor reads the XML, crops the image, and feeds each cropping to the CNNClassifier.

• The CNNClassifier performs some operations on the inputs, and then sends the output labels back.

• The outputs will be sent to the ResultViewer and displayed to the user.

Such a sequence diagram indicates how data is processed in real time within components.

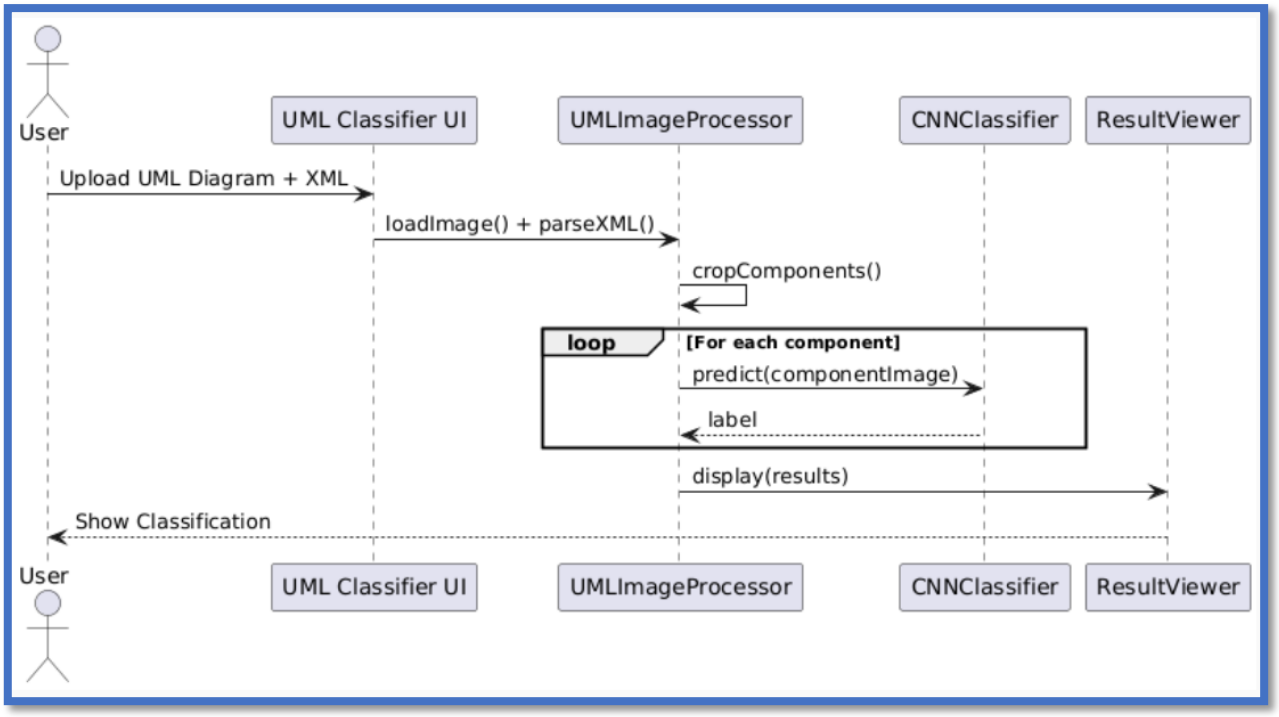

Figure 4: Sequence Diagram Depicting the Interaction Between System Components

4.Experiments and Results

The CNN model designed to classify the components of UML class diagrams was evaluated using a carefully prepared dataset. The experiment considered the experimental setup, training process, classification outcomes, and behavior of the model with conventional evaluation measures.

4.1 Experiment Setup

The experiments were performed using PyTorch in combination with a CUDA-enabled GPU. Three datasets were employed, which comprised 70% for training, 15% for validation, and 15% for testing. The model was trained for 5 epochs, and the early stopping technique was used to avoid overfitting.

Performance evaluation measures include the following:

• Accuracy: It refers to the proportion of accurate classification instances.

• Precision: It is the ratio of true positives to total positives predicted for every class.

• Recall: It is the ratio of true positives to total instances belonging to each class.

• F1-Score: It is a balance of precision and recall.

4.2 Training Performance

Table 3 summarizes training and validation performance across all epochs.

Table 2: the training and validation results at each epoch

Epoch	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
1	74.5%	76.2%	0.63	0.59
2	84.2%	85.7%	0.42	0.38
3	89.6%	90.1%	0.28	0.25
4	92.8%	91.6%	0.19	0.21
5	**94.3%**	**92.3%**	**0.13**	**0.18**

Fast convergence was observed in the model, along with improvement throughout each epoch. There is close proximity between validation accuracy and training accuracy, reflecting generalization and no sign of overfitting.
This can be seen in Figure 5 below.

The least F1-score, Controller with an F1-score of 0.89, is still regarded as highly reliable for classification purposes. The marginal decline in the recall score for the Controller, and precision for the Service class, could be attributed to the similarity in visual appearance in the diagrams' patterns. This is evidenced by the confusion matrix in Table 5, where semantic roles that have similar visual representations may cause some confusion for the classifier.

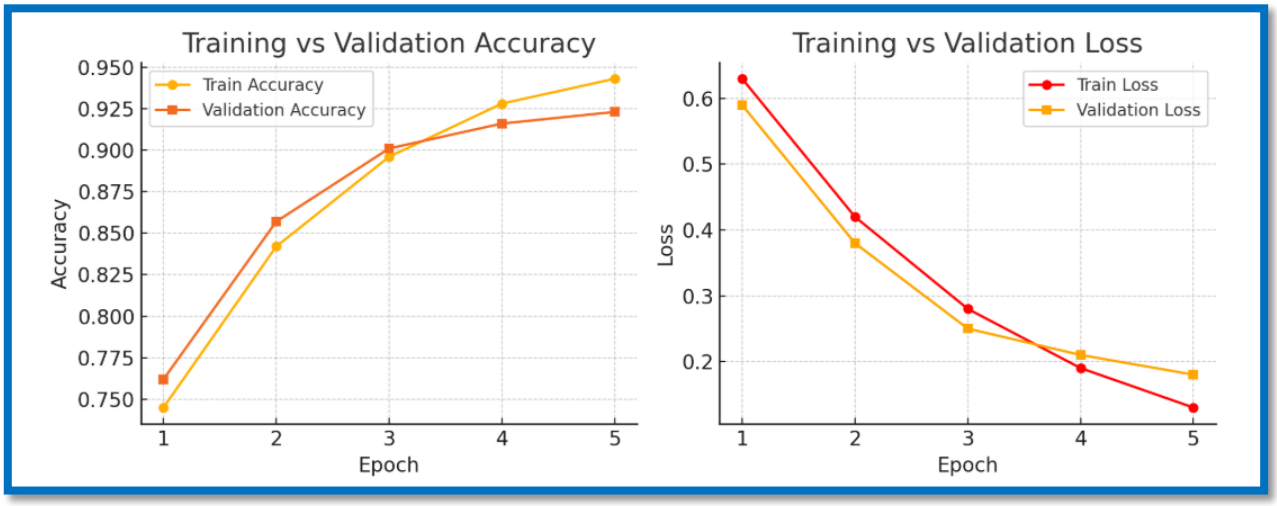

Figure 5: Accuracy and Loss over 5 Epochs

Table 3:Confusion Matrix

	Pred: Entity	**Pred: Service**	**Pred: Controller**	**Pred: Utility**
True: Entity	**91**	**2**	**5**	**2**
True: Service	**1**	**94**	**3**	**2**
True: Controller	**3**	**2**	**88**	**7**
True: Utility	**2**	**1**	**2**	**95**

4.3 Final Classification Results

On the test set, the model achieved a final accuracy of 92.3%, confirming its capability to distinguish between the four UML component types. Detailed performance per class is shown in Table 4.

Table 4: Evaluate the results of classes type

Class Type	**Precision**	**Recall**	**F1-Score**
Entity	0.94	0.91	0.93
Service	0.91	0.94	0.92
Controller	0.90	0.88	0.89
Utility	0.93	0.95	0.94

The classifier has excellent discriminative capacity with very low errors, which occur mainly because of visually similar roles such as Controller and Entity owing to their visual similarity. The significant outcomes from these experiments were:

a)The role Utility scored the maximum F1-score of 0.94, possibly because of commonality in the visual aspects.

b)The errors made were relatively small and mainly between adjacent roles semantically.

c)This experiment validated the feasibility of learning with only visuals and the superiority of skipping OCR and rules.

5. Discussion

Looking at the test outcomes, the new method using CNNs to sort UML class diagram parts by meaning works well. It reached 92.3% correct answers during testing. Strong numbers in precision, recall, and F1-score show it performs reliably across different types of diagrams. What makes this approach stand out is learning directly from visuals alone. Unlike systems needing text extraction or fixed grammar rules, this one skips those steps entirely. Because of that, it handles varied layouts without depending on written labels or structure hints. Performance stays consistent even when design styles change

i. Independence from Layout and Style: The model is strong enough to work with different UML software and style guidelines, which is a problem for OCR-based models.

ii. Fewer Engineering Steps: Not doing text analysis makes the pipeline simpler and stronger.

iii. Works with Scanned Images: The framework works with scanned UML class diagrams where the text can't be changed. Confusion among Entity and Controller elements suggests that semantic similarity may lead to visual ambiguity. Future work could improve the method through:

• Hybrid Models: Using visual features and text for improved disambiguation.

• Powerful Architectures: Vision transformers and attention-based CNNs could yield better performance.

6. Conclusion and Future Work

The CNN-based approach achieved an accuracy of 92.3% in identifying the elements of a UML class diagram (Entity, Service, Controller, Utility) based on visual attributes without OCR and rule-based classification. The results show that deep learning can be applied to automatic UML diagram recognition and can be used to model and reverse engineer software. However, the study is limited to the dataset size is small and the styles of diagrams are similar. Also, the model used is a simple CNN and k-fold cross-validation was not possible due to time and computational constraints. Furthermore, model's efficiency (Speed of inference, model size) was not taken into account. The work will be continued to address the limitations by expanding the dataset, applying cross-validation techniques and using more robust models, such as Transformer and attention-based CNN models to improve the performance of the model.

References

1. Ciccozzi, F., Malavolta, I., & Selic, B. (2019). Execution of UML models: a systematic review of research and practice. Software & Systems Modeling, 18(3), 2313-2360.
2. Rumpe, B. (2016). Modeling with UML (Vol. 98). Cham: Springer.
3. Ozkaya, M. (2019). Are the UML modelling tools powerful enough for practitioners? A literature review. IEt software, 13(5), 338-354.
4. Ciccozzi, F., Malavolta, I., & Selic, B. (2019). Execution of UML models: a systematic review of research and practice. Software & Systems Modeling, 18(3), 2313-2360.
5. Avyodri, R., Lukas, S., & Tjahyadi, H. (2022, September). Optical character recognition (ocr) for text recognition and its post-processing method: A literature review. In 2022 1st International Conference on Technology Innovation and Its Applications (ICTIIA) (pp. 1-6). IEEE.
6. Mohaideen Abdul Kadhar, K., & Anand, G. (2024). Optical character recognition. In Industrial Vision Systems with Raspberry Pi: Build and Design Vision Products Using Python and OpenCV (pp. 215-254). Berkeley, CA: Apress.
7. Li, P., Zhou, B., Wang, C., Hu, G., Yan, Y., Guo, R., & Xia, H. (2024). CNN-based pavement defects detection using grey and depth images. Automation in Construction, 158, 105192.
8. Nazir, Md Imran, Afsana Akter, Md Anwar Hussen Wadud, and Md Ashraf Uddin. "Utilizing customized CNN for brain tumor prediction with explainable AI." Heliyon 10, no. 20 (2024).
9. Chen, Y., & Wang, S. (2024, July). Research on software classification based on LSTM and CNN. In Third International Symposium on Computer Applications and Information Systems (ISCAIS 2024) (Vol. 13210, pp. 213-219). SPIE.
10. Silva, A. D., & Carvalho, T. C. M. B. (2021). UML diagram classification using CNN and transfer learning. *Applied Sciences, 11*(9), 4267. https://www.mdpi.com/2076-3417/11/9/4267
11. Azmi, A. M., & Mehmood, R. (2021). Deep learning approaches for identifying UML diagrams. *International Journal of Software*

Engineering and Knowledge Engineering, 31(10), 1387–1406. https://doi.org/10.1142/S0218194021400179

12. Javed, H., & Malik, M. H. (2022). Automated extraction of UML class elements using CNN. In *2022 International Conference on Artificial Intelligence and Smart Systems (ICAIS)* (pp. 56–61). IEEE. https://ieeexplore.ieee.org/abstract/document/9680314
13. Hussain, M. F., & Nawaz, S. (2023). Natural language processing techniques for UML class generation. *Journal of Computer and Biomedical Informatics, 7*(1). https://www.jcbi.org/index.php/Main/article/view/546
14. Reyes, J., & Olivas, J. A. (2024). Improving educational support with efficient UML diagram classification. In *Artificial Intelligence in Education* (Vol. 14763, pp. 79–94). Springer. https://link.springer.com/chapter/10.1007/978-3-031-75605-4_6
15. Zhang, Y., & Chen, B. (2024). Structural analysis of UML diagrams using graph convolutional networks. In *Advances in Software Engineering and Data Mining* (Vol. 14271, pp. 235–251). Springer. https://link.springer.com/chapter/10.1007/978-3-031-63031-6_16
16. Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language User Guide* (2nd ed.). Addison-Wesley.
17. *YM Mohialden, HA Abdulbaqi, NM Shati ,Developing collaboration tool for virtual team using UML models,Indonesian Journal of Electrical Engineering and Computer Science 22 (1), 38*–44
18. Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd ed.). Prentice Hall.
19. Xu, Z., Sun, F., & Zhang, W. (2024, June). Research on Activity Diagram Testing method based on UML Testing Profile. In 2024 6th International Conference on Electronic Engineering and Informatics (EEI) (pp. 434-439). IEEE.
20. Ferrari, A., Abualhaijal, S., & Arora, C. (2024, June). Model generation with LLMs: From requirements to UML sequence diagrams. In 2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW) (pp. 291-300). IEEE.