Hand Sketched UML Diagram Recognition and Conversion

By:

Sikandar Ali Umrani

27666

Huzaifa Attiq

28546

Chand

27959

Supervised by:

Dr. Naveed Ikram

Faculty of Computing

Riphah International University, Islamabad

Spring 2024

A Dissertation Submitted To

Faculty of Computing,

Riphah International University, Islamabad

As a Partial Fulfillment of the Requirement for the Award of the Degree of

Bachelors of Science in Computer Science

Faculty of Computing

Riphah International University, Islamabad

Date: [date of final presentation]

Final Approval

This is to certify that we have read the report submitted by **Sikandar Ali Umrani (27666), Huzaifa Attiq (28546) and Chand (27959)** for the partial fulfillment of the requirements for the degree of the Bachelors of Science in Computer Science (BSCS). It is our judgment that this report is of sufficient standard to warrant its acceptance by Riphah International University, Islamabad for the degree of Bachelors of Science in Computer Science (BSCS).

Committee:

1 ______________________________

Dr. Naveed Ikram
(Supervisor)

2 ______________________________

Dr. Musharraf
(Head of Department)

Declaration

We hereby declare that this document **"Hand Sketched UML Diagram Recognition and Conversion"** neither as a whole nor as a part has been copied out from any source. It is further declared that we have done this project with the accompanied report entirely on the basis of our personal efforts, under the proficient guidance of our teachers, especially our supervisor **Dr. Naveed Ikram.** If any part of the system is proved to be copied out from any source or found to be reproduction of any project from anywhere else, we shall stand by the consequences.

Sikandar Ali Umrani
27666

Huzaifa Attiq
28546

Chand
27959

Dedication

This project is dedicated to the group family, friends, especially our supervisor “Dr. Naveed Ikram” who have been our mentor and guidance throughout this final year project, and whose support was invaluable in completing this final year project.

Acknowledgement

First of all, we are obliged to Allah Almighty the Merciful, the Beneficent and the source of all Knowledge, for granting us the courage and knowledge to complete this Project.

In this section, we would like to express our sincere gratitude to all the individuals who supported us throughout this project. Firstly, we are thankful to our supervisor "Dr. Naveed Ikram", who provided invaluable guidance and support throughout the entire process. In particular, we dedicate this project to him for his invaluable mentorship and Secondly, we extend our gratitude to all our teachers who have imparted knowledge and wisdom, enriching our understanding of the subject matter. Thirdly, we would like to acknowledge our fellow colleagues who have been a source of encouragement and assistance along the way.

Sikandar Ali Umrani

27666

Huzaifa Attiq

28546

Chand

27959

Table of Contents

List of Tables

List of Figures

Abstract

In many organizational processes, including software architectures and business workflows, diagrams are crucial conceptual models. Even though there are many tools available for diagram modeling, hand sketching is frequently the first step in the process of creating these models. On papers or whiteboards. But converting these manually drawn diagrams into digital formats for additional editing and integration with modeling software presents a major difficulty.

The goal of hand-drawn diagram recognition research is to automate this digitization process; however, there is a noticeable deficiency in the recognition of diagrams drawn on paper or whiteboard surfaces, particularly in the software modeling domain, where UML is the industry standard for software system representation. Studying UML models ideally from the documentation of industrial software projects is necessary to comprehend their usefulness in software development. Nevertheless, UML models are frequently created by hand and saved as pictures without the model data.

To address this challenge, our project, Hand Sketched UML Diagrams Recognition and Conversion, proposes a solution. Our tool extracts UML Model information from hand sketched diagrams, enabling their integration into modeling tools for further study. Additionally, our project recognizes both hand-sketched Class diagrams and Activity Diagrams, broadening the scope of UML diagram types and converts into a format that can be loaded to a modeling tool.

Chapter 1:

Introduction

Diagrams serve as essential conceptual models in various organizational processes, spanning from business workflows to software architectures. Despite the availability of numerous diagram modeling tools, the initial creation of these models often starts with hand-sketching on paper. However, transitioning these hand-drawn diagrams into digital formats for further editing and integration into modeling tools poses a significant challenge.

Research in hand-drawn diagram recognition aims to automate this digitization process, but there is a notable gap in recognizing diagrams sketched on paper especially in the field of software modeling, where UML serves as the standard for representing software systems. Understanding the effectiveness of UML models in software development requires studying them, ideally from documentation of industrial software projects. However, UML models are often sketched on paper and saved as images like PNG, lacking the model information available in CASE tool.

To address this challenge, our project aims to develop a software system capable of recognizing hand-sketched UML diagrams and converting them into editable digital representations. Specifically, we focus on class diagrams and activity diagrams, two of the most widely used UML diagram types. It's important to note that while we cover the most widely used elements of these diagrams, we are not encompassing all elements of these diagram. Additionally, our focus is on recognizing diagrams drawn on normal-sized paper, rather than those that are excessively large or small.

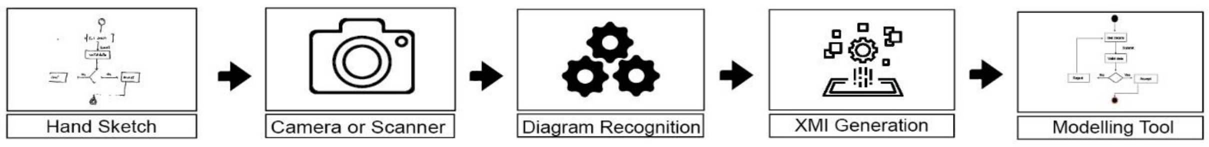


Figure 1.1: Exemplary use case for a diagram recognition system that is able to recognize hand-drawn diagrams in camera-captured images.

1.1 Goals and Objectives

- Develop a software system capable of recognizing hand-sketched UML diagrams (class and activity diagrams) accurately and efficiently.
- Automate the conversion process of hand-sketched UML diagrams into editable digital representations to save time and improve workflow efficiency.
- Enhance collaboration and communication in professional settings by providing an easy-to-use solution for converting hand-sketched UML diagrams into digital formats.

1.1.1 Objectives:

- Develop preprocessing techniques for standardization and noise reduction.
- Implement feature extraction methods to identify key UML Class Diagram symbols and relationships accurately.
- Integrate OCR technology for textual recognition to enhance the completeness of the digital representation.
- Design and implement a robust model architecture, possibly utilizing machine learning techniques, for accurate symbol recognition.
- Validate the solution to ensure effectiveness across various styles and complexities of hand-drawn UML Class diagrams.
- Present the output in a structured XML format, encapsulating both recognized symbols and textual annotations for seamless integration into existing UML tools.

1.2 Scope of the Project

The project scope includes the following key components:

- Creation of own custom dataset.
- Development of preprocessing techniques for standardization, noise reduction, and orientation correction of input images.
- Implementation of feature extraction methods for identifying key UML Class Diagram symbols and relationships.
- Integration of OCR technology for textual recognition to enhance the completeness of the digital representation.
- Design and implementation of a robust model architecture, utilizing machine learning techniques for accurate symbol recognition.
- Validation of the solution to ensure effectiveness across various styles and complexities of hand-drawn UML Class diagrams.
- Presentation of the output in a structured XML format, encapsulating both recognized symbols and textual annotations for seamless integration into existing UML tools.

Chapter 2:

Literature Review

2.1 Introduction

We will conduct a thorough analysis of relevant literature in this chapter, offering insights into the difficulties involved in transforming hand-drawn UML diagrams into digital formats. We will examine current approaches, their drawbacks, and the noted area of unmet research need. This chapter will also outline our project's history and problem statement, laying the groundwork for the creation of software that can recognize hand-sketched UML diagrams and transform them into editable digital representations. We hope that this review will provide some background and motivation for our work while emphasizing the need for an automated solution to improve workflow, efficiency, and collaboration in work environments.

2.2 Background and Problem Elaboration

During meetings or brainstorming sessions, people frequently turn to hand sketching their ideas and concepts in professional settings. One of the biggest challenges is converting these handdrawn sketches especially UML (Unified Modeling Language) diagrams—into digital formats. In fast-paced work environments, efficiency and collaboration are hampered by this labor- and time-intensive manual conversion process.

In the early stages of a project, UML models are usually hand-sketched; however, these sketches cannot be directly imported into UML tools that are currently in use. The inability of current modeling tools, like draw.io, to support image imports makes it challenging to automatically transform hand-drawn sketches into digital formats. The inability of UML CASE tools to extract UML models from images results in problems with interoperability between different tools and formats, even though they provide a variety of features for creating, editing, and exporting UML models, including exporting them as images.

Because of this, the process of manually converting hand-drawn UML diagrams into digital formats is still laborious and ineffective, which interferes with professional workflow and creates obstacles to collaboration. Our project intends to close this gap by creating a system that recognizes and converts activity diagrams and class diagrams automatically. This will make it simple for stakeholders to incorporate their hand-drawn sketches into current digital platforms for additional editing and sharing.

Professionals in a variety of industries who regularly generate and communicate ideas through hand-drawn sketches will greatly benefit from this advancement, as will software developers and engineers who must digitize hand-drawn UML diagrams for further development and documentation.

2.3 Detailed Literature Review

We offer a thorough analysis of the literature in this section that is pertinent to our project, which is to automatically extract UML Class and Activity diagrams from hand-drawn sketches. After providing definitions for important terms and concepts associated with our study, we critically examine previous studies conducted in the area.

2.3.1 Definitions

Before delving into the literature review, it's important to define key terms used in our project:

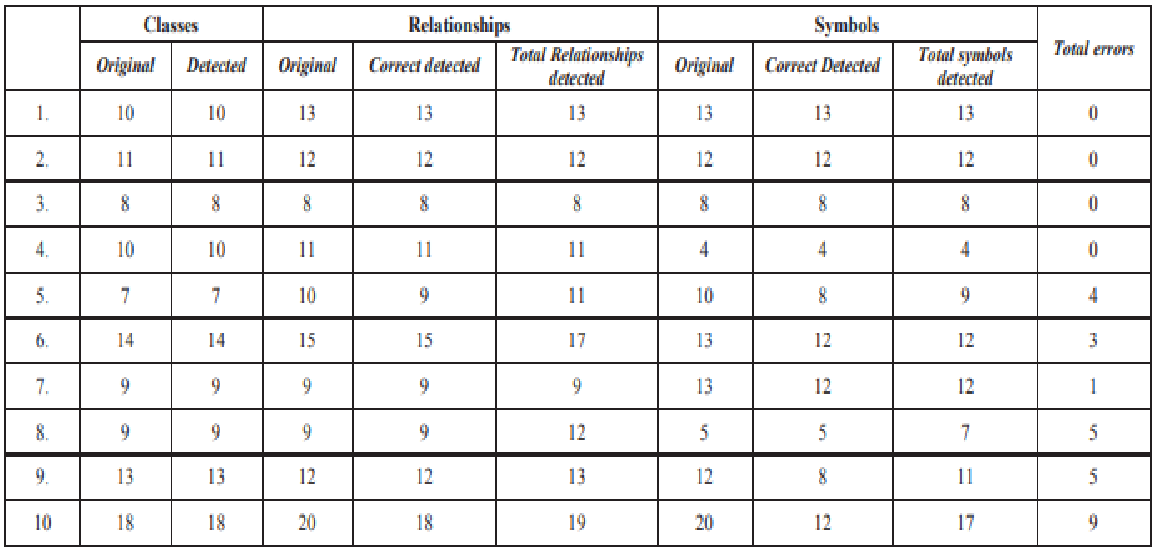
- UML (Unified Modeling Language): A standardized modeling language used in software engineering for visualizing, specifying, constructing, and documenting the artifacts of a software system.
- UML Class Diagram: A type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.
- UML Activity Diagram: A type of behavior diagram that depicts the dynamic aspects of a system by modeling the flow of control from activity to activity.

2.3.2 Related Research Work 1

Several research works have explored the automatic extraction of UML diagrams from hand-drawn sketches. One notable study is "Extracting UML models from images" by Rodrigues et al. (2013). This research presents the Img2UML tool, which uses image processing techniques to recognize classes, relationships, and symbols from hand-drawn UML diagrams. The tool aims to address the challenge of updating UML models that are stored in image formats, making them difficult to modify. Rodrigues et al. demonstrate the functionality of Img2UM and validate its accuracy in detecting classes, relationships, and symbols.

2.3.2.1 Strengths:

Img2UML tackles a prevalent issue in software engineering by offering a workable solution for extracting UML Class models from images.
As can be seen in the table, the tool's validation results show how well it can identify classes, relationships, and symbols, suggesting its potential utility.



	Classes		Relationships			Symbols			Total errors
	Original	Detected	Original	Correct detected	Total Relationships detected	Original	Correct Detected	Total symbols detected	
1.	10	10	13	13	13	13	13	13	0
2.	11	11	12	12	12	12	12	12	0
3.	8	8	8	8	8	8	8	8	0
4.	10	10	11	11	11	4	4	4	0
5.	7	7	10	9	11	10	8	9	4
6.	14	14	15	15	17	13	12	12	3
7.	9	9	9	9	9	13	12	12	1
8.	9	9	9	9	12	5	5	7	5
9.	13	13	12	12	13	12	8	11	5
10	18	18	20	18	19	20	12	17	9

Table 2.1: Validations results of IMg2UML tool

2.3.2.2 Limitations:

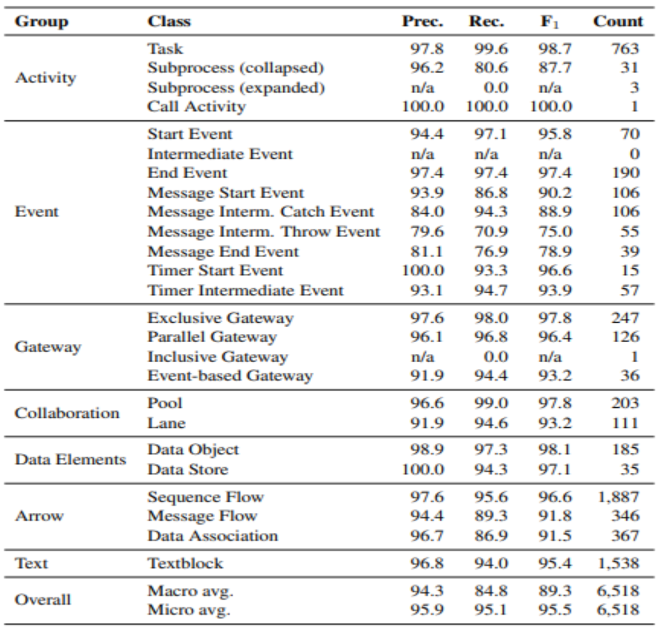
The tool's performance can vary based on the complexity and quality of the input images, especially in recognizing symbols and text. Moreover, it's essential to note that the tool doesn't recognize hand-drawn sketches rather, it focuses on the recognition and conversion digital images. Additionally, it's important to highlight that the tool does not recognize all the elements of a class diagram; instead, it covers only a subset of the elements typically found in class diagrams. Furthermore, while the tool addresses the recognition and transformation of UML Class Diagrams, it does not explore the extraction of UML Activity diagrams, which are also vital in software design.

2.3.3 Related Research Work 2

Sketch2Process is a tool that automates the recognition of hand-drawn diagrams, with a specific focus on business process diagrams created on paper. It introduces the hdBPMN dataset, comprising 702 business process diagrams collected from 107 participants, along with BPMN annotations, made publicly available to facilitate further research. Addressing the challenges inherent in recognizing diagrams drawn on paper, Sketch2Process uses neural network-based techniques to detects shapes, arrows, and textblocks, while also recognizing edges and labels, ultimately generating BPMN XML files suitable for process modeling tools. Overall, Sketch2Process contributes valuable insights and methodologies to automated diagram recognition, offering practical solutions to longstanding challenges.

2.3.3.1 Strengths:

Sketch2Process shows its strong strengths in detection and recognition of various BPMN elements as shown in table 2.2. It provides a strong solution for recognition of BPMN models and also provides an environment where these models can be edited in a real time after recognition and extraction from images and generates a BPMN XML files which can be imported to modeling tools for further editing and utilization.



Group	Class	Prec.	Rec.	F_1	Count
Activity	Task	97.8	99.6	98.7	763
	Subprocess (collapsed)	96.2	80.6	87.7	31
	Subprocess (expanded)	n/a	0.0	n/a	3
	Call Activity	100.0	100.0	100.0	1
Event	Start Event	94.4	97.1	95.8	70
	Intermediate Event	n/a	n/a	n/a	0
	End Event	97.4	97.4	97.4	190
	Message Start Event	93.9	86.8	90.2	106
	Message Interm. Catch Event	84.0	94.3	88.9	106
	Message Interm. Throw Event	79.6	70.9	75.0	55
	Message End Event	81.1	76.9	78.9	39
	Timer Start Event	100.0	93.3	96.6	15
	Timer Intermediate Event	93.1	94.7	93.9	57
Gateway	Exclusive Gateway	97.6	98.0	97.8	247
	Parallel Gateway	96.1	96.8	96.4	126
	Inclusive Gateway	n/a	0.0	n/a	1
	Event-based Gateway	91.9	94.4	93.2	36
Collaboration	Pool	96.6	99.0	97.8	203
	Lane	91.9	94.6	93.2	111
Data Elements	Data Object	98.9	97.3	98.1	185
	Data Store	100.0	94.3	97.1	35
Arrow	Sequence Flow	97.6	95.6	96.6	1,887
	Message Flow	94.4	89.3	91.8	346
	Data Association	96.7	86.9	91.5	367
Text	Textblock	96.8	94.0	95.4	1,538
Overall	Macro avg.	94.3	84.8	89.3	6,518
	Micro avg.	95.9	95.1	95.5	6,518

Table 2.2: Results of Sketch2Process Object detection results per class.

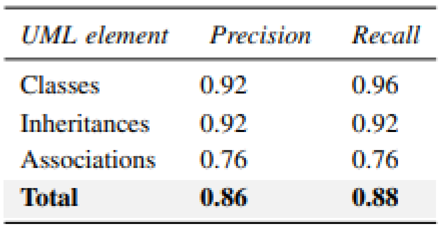
2.3.3.2 Limitations:

Sketch2Process provides a strong solution for automating the process of BPMN models but with a partial capability to recognize activity diagrams as most of the elements of activity diagrams are similar to flow charts. However, it lacks the ability to recognize class diagrams.

2.3.4 Related Research Work 3

SketchToPlantUML is a pioneering tool designed to bridge the gap between informal hand-drawn UML Class Diagrams and formal PlantUML models, a crucial step in software engineering. By harnessing the power of the OpenCV library, this tool automates the transformation process, significantly reducing the time and effort required for manual conversion. It preprocesses images, segments UML elements, identifies geometric features, and classifies relationships, culminating in the generation of equivalent PlantUML models.

This tool addresses the inherent challenge of manually transforming transient sketches into reusable and shareable formal models. Its effectiveness is underscored by rigorous testing against a dataset of 70 sketched UML Class Diagrams, where it achieved impressive Precision and Recall values of 88% and 86% respectively. Notably, it excels in accurately recognizing classes, with scores of 0.92 for Precision and 0.96 for Recall, while also demonstrating competence in handling association relationships.



UML element	*Precision*	*Recall*
Classes	0.92	0.96
Inheritances	0.92	0.92
Associations	0.76	0.76
Total	**0.86**	**0.88**

Tale 2.3: Precision and Recall Evaluation Metrics for UML Element Classification.

2.3.4.1 Strengths:

Sketch2Plant UML shows its high accuracy in recognition of UML elements such as classes, inheritance, associations and conversion to formal PlantUML models as shown in figure 2.3.

2.3.4.2 Limitations:

SketchToPlantUML excels in accurately recognizing hand-sketched UML Class Diagrams, focusing primarily on identifying classes, inheritance, and association relationships. However, its scope is limited to class diagrams and also shows its limitations in recognizing text from sketches. Additionally, while the tool simplifies the transformation process, users need prior knowledge of PlantUML coding conventions to fully utilize its capabilities.

2.4 Literature Review Summary Table

The columns in the table depend upon your problem and should be specific to your project.

Table 2.4: History of Hand-Drawn Diagram Recognition and Conversion Methods

The summary of various methods and tools developed for recognizing and converting hand-drawn diagrams into digital formats

No.	Name, reference	Inventor	Year	Input	Output	Description
1.	Sketch 2Process	B. Schäfer et al.	2023	Hand- drawn Process Models	Digital BPMN Models	Automates the conversion of hand-drawn process models into digital BPMN models using neural network-based techniques.
2.	Img2UML	Michel Chaudron, B. Karasneh	2013	Digital Images of UML diagrams	Extracted UML Models From Images	Aims to extract UML models from images, assisting in the digitization of UML diagrams for further editing and use
3.	Sketch 2PlantUML	Monique Axt	2023	Hand Sketch of Class Diagram	Formal PlantUML Model.	Automates the transformation of hand-drawn UML Class Diagrams into formal PlantUML models using the OpenCV library.

2.5 Research Gap

In the field of UML diagram recognition and conversion, several tools have emerged, each addressing specific needs but also presenting significant limitations. Img2UML, for instance, extracts UML Class models from image files and converts them into XMI files compatible with CASE tools like StarUML. However, it fails to recognize hand-drawn sketches, which limits its applicability in scenarios where such sketches are common.

On the other hand, Sketch2PlantUML focuses on transforming hand-drawn UML Class Diagrams into formal PlantUML models using OpenCV for image processing. Despite its automation capabilities, its output can only be viewed with the PlantUML tool, requiring users to have knowledge of PlantUML coding for any modifications.

SketchUML offers a natural, paper-like interface for students to create UML class diagrams directly within a digital environment, enhancing the learning experience. However, it does not support converting hand-drawn sketches into an editable digital format.

Lastly, Sketch2Process automates the recognition of hand-drawn business process diagrams, generating BPMN XML files for use in process modeling tools. While effective for business process models, it only partially recognizes activity diagrams and does not support class diagrams.

The primary research gap lies in the absence of a comprehensive system capable of recognizing and converting various types of diagrams, such as activity and class diagrams, into formats that can be seamlessly exported to popular modeling tools. Existing tools either focus on specific diagram types or lack integration with multiple modeling tools. For example, Img2UML offers compatibility with tools like StarUML but does not support hand-drawn diagrams. Similarly, Sketch2Process excels at business process models but falls short on class diagram recognition.

Therefore, there is a significant need for a versatile and efficient system that can handle multiple diagram types and provide direct integration with popular modeling environments.

2.6 Problem Statement

People commonly use hand sketching in business contexts to communicate ideas and concepts, especially during brainstorming or meeting sessions. However, there is a big obstacle in translating these hand-drawn sketches especially UML (Unified Modeling Language) diagrams into digital formats. The labor-and time-intensive nature of the manual conversion process hinders productivity and teamwork in hectic work settings. Inefficiencies like time consumption, loss of detail, obstacles to collaboration, and disruption of workflow result from this lack of an automated solution.

One of the most commonly used types of UML diagrams in professional settings are class diagrams and activity diagrams. Despite their importance, there is a noticeable gap in automated tools capable of recognizing hand-drawn UML diagrams and converting them into editable digital representations. Existing solutions either focus on specific diagram types, lack compatibility with popular modeling tools, or are limited in functionality.

A flexible and effective system that can manage various diagram types and offer direct integration with well-known modeling environments is therefore desperately needed. We are proactively addressing this gap by creating a solution that recognizes and converts class diagrams and activity diagrams automatically. Although our system might not be able to address every issue that currently exists, it is a significant step in the right direction toward a comprehensive tool that improves accessibility and usability for users in a variety of contexts.

2.7 Conclusion:

This chapter outlined the project's background and problem statement, as well as offered a thorough analysis of pertinent literature. The literature review addressed the difficulties in transferring hand-drawn UML diagrams into digital formats as well as the shortcomings of current solutions. This review revealed that, although certain types of diagrams have tools available, or tools with limited functionality, there is a glaring gap in the creation of a comprehensive system that can identify and translate various UML diagram types into editable digital representations.

Our project aims to address this gap by developing a software system that can recognize class diagrams and activity diagrams, and convert them into editable digital formats. By automating this process, our system will enable professionals to seamlessly integrate their hand-drawn sketches into existing digital platforms, improving collaboration, saving time, and enhancing workflow efficiency.

We have defined the background and problem statement, carried out an extensive review of relevant literature, and laid the groundwork for our project in this chapter. Continuing from the insights gathered from the literature review, Chapter 3 will concentrate on detailing the methodology and approach we will use to develop our solution.

Chapter 3:

Requirements and Design

The design and methodology of our project, which recognizes and converts hand-drawn UML diagrams into digital formats automatically, are covered in Chapter 3. It covers prerequisites for software and hardware, necessary functional and non-functional requirements, and the particular libraries and frameworks that are used. The process of creating and annotating a custom dataset, developing preprocessing, feature extraction, and OCR integration algorithms, and training and assessing these algorithms are all covered in the methodology section. The system architecture is shown, including the user interface and recognition engine details, as well as the parts that go into identifying UML elements and creating XMI files. A use case discussion on creating XMI files from hand-drawn UML diagrams closes the chapter.

Our project focuses on automating the recognition and conversion of UML diagrams, specifically targeting the most widely used notations for class diagrams and activity diagrams as specified by the OMG.

3.1 Requirements

3.1.1 Functional Requirements

ID	Requirements
FR - 1.1	Users should be able to upload hand-drawn UML diagrams in Jpg and PNG format
FR - 1.2	The system should accurately identify and recognize UML elements such as classes, attributes, methods, relationships, and activities from the uploaded images.
FR – 1.3	Upon recognition of UML elements, the system should generate a Star UML-compatible XMI file representing the digital version of the hand-drawn UML diagram.

3.1.2 Non-Functional Requirements

These are following non-functional requirements:

3.1.2.1 Performance:

The system should process and recognize UML diagrams.

3.1.2.2 Scalability:

The system should be scalable, allowing for numerous simultaneous uploads by users without sacrificing performance.

3.1.2.3 Security:

UML diagrams submitted should be securely saved and processed. To safeguard users' privacy, their data should be erased from the server once processing is completed.

3.1.2.4 Reliability:

The system should operate reliably.

3.1.3 Software Requirements:

3.1.3.1 Roboflow:

The roboflow platform will be used for custom dataset labelling and annotating.

3.1.3.2 Pycharm:

This platform will be used for writing python code.

3.1.3.3 Programming Languages:

The primary language for developing the software system is likely Python, given its popularity in machine learning and image processing domains.

3.1.4 Libraries and Frameworks:

3.1.4.1 OpenCV:

Computer vision library will be used for image preprocessing, feature extraction to standardize, reduce noise, and correct the orientation of hand-drawn UML diagrams.

3.1.4.2 Model:

Deep learning object detection model will be trained on the custom dataset for detecting UML elements in image.

3.1.4.3 TensorFlow or PyTorch:

Deep Learning frameworks will be used for constructing and training complex neural network models for recognizing UML symbols and relationships in images.

3.1.4.4 EasyOCR:

OCR will be used for recognizing textual annotations within the diagrams due to its flexibility and ease of integration with python.

3.1.4.5 XML Processing:

XML Processing Libraries like xml.etree.ElementTree in Python for parsing and generating XML files, ensuring compatibility with existing UML tools.

3.2 Proposed Methodology

3.2.1 Dataset:

One of the most significant challenges we face is the absence of available datasets. Currently, there are no datasets specifically tailored to hand-sketched activity diagrams and class diagrams. This absence of relevant datasets has hindered progress in developing recognition systems for these types of diagrams. To address this gap, we are taking steps to create a new dataset explicitly designed for recognizing hand-sketched activity and class diagrams.

3.2.2 Dataset Labelling and Annotation:

The next big challenge to tackle is the annotation of the dataset. As, the dataset creation of was custom so we had to annotate the images because the deep learning-based models required the dataset into a labelled format which contains images folder, and a label's folder, so the we had to manually annotate the entire dataset which consisted of drawing bounding boxes and assigning a label to each images based on the behavior of each diagram element.

3.2.3 Development of Algorithms

3.2.3.1 Preprocessing Methods:

Develop techniques for noise reduction, orientation correction, and image standardization to enhance the quality and consistency of input images.

3.2.3.2 Feature Extraction:

Design algorithms to extract important UML relationships and symbols from preprocessed images, distinguishing between various UML elements.

3.2.3.3 OCR Integration:

Integrate OCR technology to recognize textual annotations within the diagrams, ensuring accurate extraction of textual information.

3.2.3.4 Model Architecture:

Design and implement a robust model architecture, potentially utilizing deep learning methods, for precise symbol identification. Experiment with different architectures and configurations to optimize performance.

3.2.4 Training and Evaluation

3.2.4.1 Training Process:

Train the developed algorithms and model architectures using the annotated dataset, ensuring diverse representations of different symbol types.

3.2.4.2 Validation:

Split the dataset into training, validation, and test sets to evaluate the performance of the trained models. Employ cross-validation techniques to ensure robustness.

3.2.4.3 Evaluation Metrics:

Define evaluation metrics such as accuracy, precision, recall, F1-score, and mean Intersection over Union (mIoU) for comprehensive assessment of model performance.

3.2.5 Deployment and Integration

3.2.5.1 XMI Format Compatibility:

Develop the software system to generate XMI files that adhere to the standard format accepted by existing UML tools.

3.2.5.2 Seamless Incorporation:

Enable users to import the generated XMI files directly into their UML tool.

3.3 System Architecture

3.3.1 User Interface (UI) Layer:

The UI Layer allows users to interact with the system by uploading hand-drawn UML diagrams (such as class and activity diagrams) and downloading generated XMI files. It is the principal interface via which users interact with the recognition system.

3.3.2 Recognition Engine:

This component processes hand-drawn UML diagrams to identify and interpret UML elements and their relationships.

3.3.3 Preprocessing Module:

Standardizes image dimensions, reduces noise, and corrects alignment for accurate UML component detection and recognition.

3.3.4 Object Detection Module:

Employs YOLO-based models to detect and localize UML symbols (e.g., classes, associations, etc.) within uploaded images.

3.3.5 Feature Extraction Module:

Extracts distinct characteristics of detected elements, such as class attributes, methods, and relationships, refining the recognition process to ensure accurate identification.

3.3.5 OCR Integration (Easy OCR):

Integrates OCR functionality to extract and interpret textual data from UML diagrams, including labels, notes, and other relevant text content.

3.3.6 Output Generation Module:

Converts recognized symbols and accompanying text to structured digital output, specifically XMI files that are compatible with popular UML modeling tools. This allows for simple integration and exploitation inside existing software development processes.

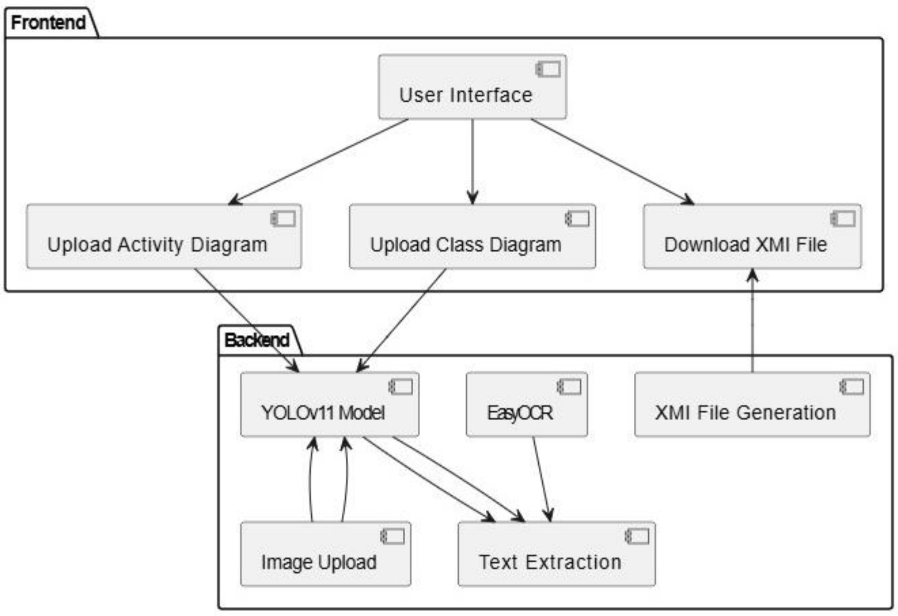


Fig: 3.1 Architecture Diagram

3.4 Use Cases

3.4.1 Generate XMI File from Hand-Drawn UML Diagram

Name	Generate XMI File from Hand-Drawn UML Diagram		
Actors	User		
Summary	The user uploads a hand-drawn UML diagram, and the software system processes it to generate an XMI file and Json file containing recognized UML symbols and textual annotations.		
Pre-Conditions	The user has access to the software system. Hand-drawn UML diagrams are available in digital format.		
Post-Conditions	The user obtains an XMI file representing the recognized UML symbols and textual annotations from the hand-drawn UML diagram.		
Special Requirements	None		
Basic Flow			
Actor Action		**System Response**	
1	The user opens the software interface.	2	The software interface is displayed, prompting the user to upload an image.
3	The user uploads a hand-drawn UML diagram.	4	The software processes the uploaded diagram using recognition algorithms.
		5	The software generates an XMI file containing recognized symbols and annotations.
6	The user retrieves the generated XMI file.	7	The XMI file is provided to the user for download.
Alternative Flow			
3	The user uploads an invalid or unreadable image.	4-A	The system responds with an error message: Invalid or unreadable image uploaded.

3.5 Use Case diagram

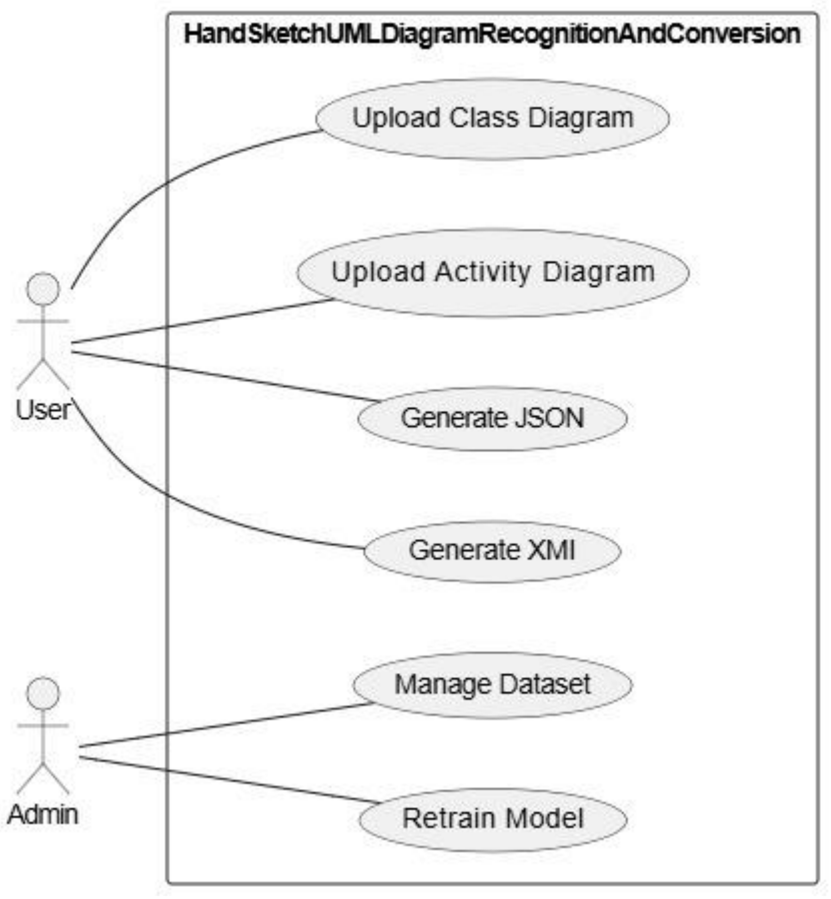


Fig: 3.2 Use-case diagram

3.6 Methodology Diagram

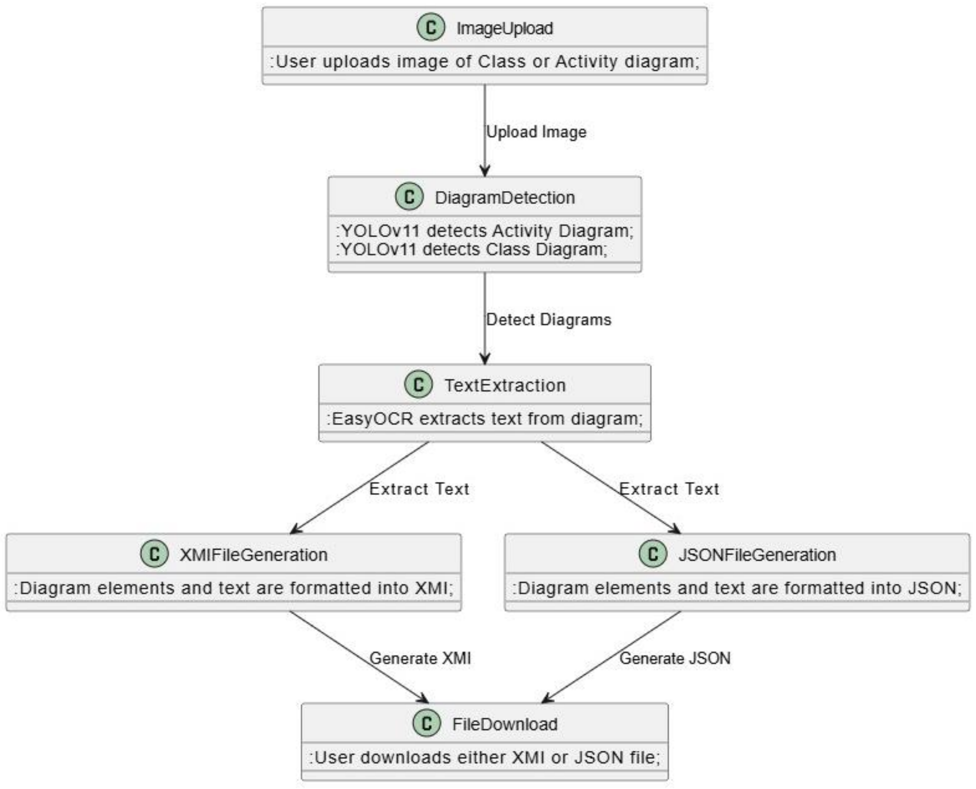

C ImageUpload
:User uploads image of Class or Activity diagram;
Upload Image
C DiagramDetection
:YOLOv11 detects Activity Diagram;
:YOLOv11 detects Class Diagram;
Detect Diagrams
C TextExtraction
:EasyOCR extracts text from diagram;
Extract Text
Extract Text
C XMIFileGeneration
:Diagram elements and text are formatted into XMI;
C JSONFileGeneration
:Diagram elements and text are formatted into JSON;
Generate XMI
Generate JSON
C FileDownload
:User downloads either XMI or JSON file;

Fig: 3.3 Methodology Diagram

3.7 Activity Diagram

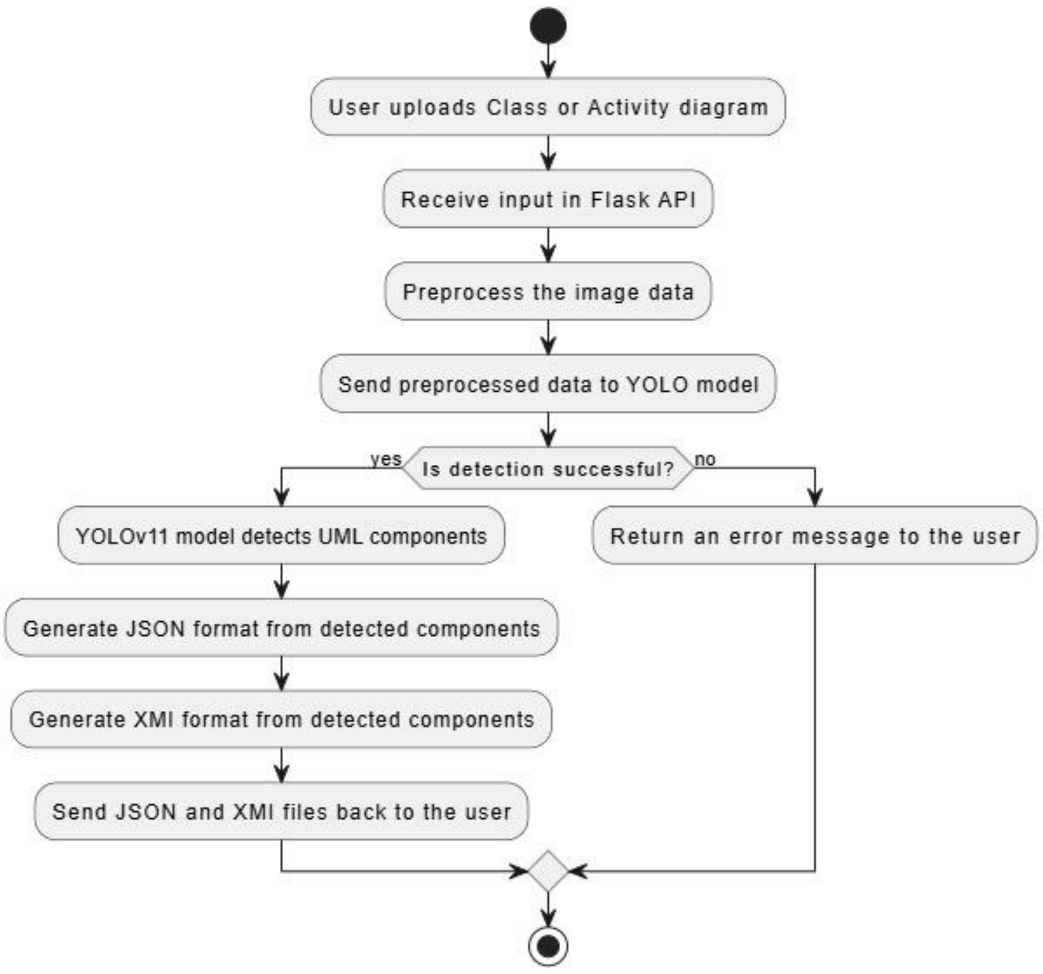
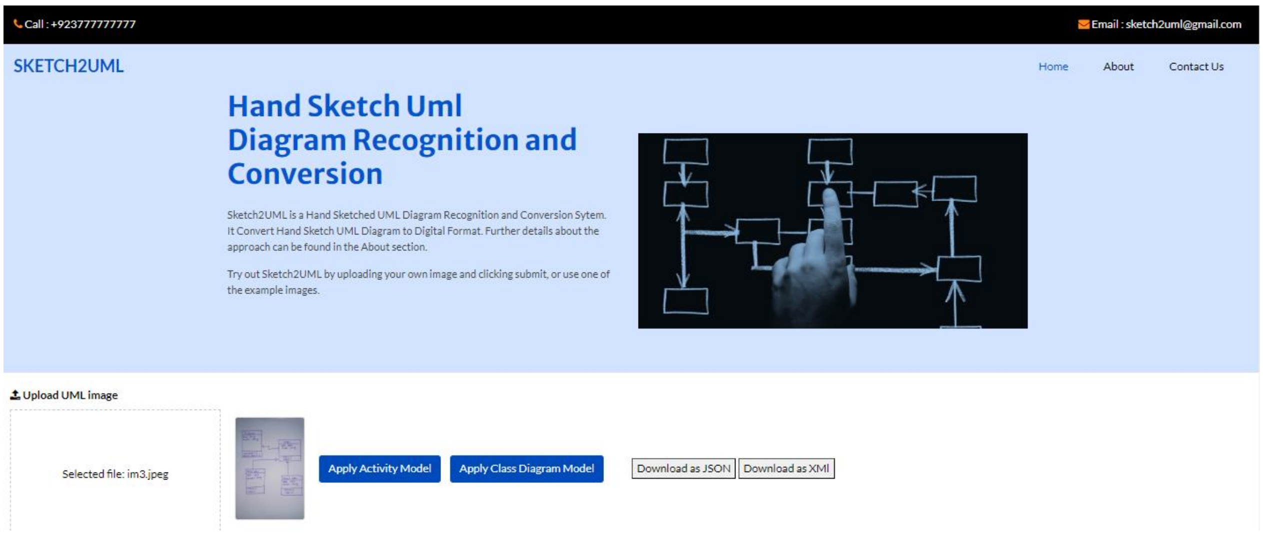

User uploads Class or Activity diagram
Receive input in Flask API
Preprocess the image data
Send preprocessed data to YOLO model
yes
Is detection successful?
no
YOLOv11 model detects UML components
Return an error message to the user
Generate JSON format from detected components
Generate XMI format from detected components
Send JSON and XMI files back to the user

Fig: 3.4 Activity Diagram

3.8 GUI


Call : +923777777777

Email : sketch2uml@gmail.com

SKETCH2UML

Hand Sketch Uml Diagram Recognition and Conversion

Sketch2UML is a Hand Sketched UML Diagram Recognition and Conversion Sytem. It Convert Hand Sketch UML Diagram to Digital Format. Further details about the approach can be found in the About section.

Try out Sketch2UML by uploading your own image and clicking submit, or use one of the example images.

Upload UML image

Selected file: im3.jpeg

Fig 3.5: Graphical User Interfaces (GUI)

3.9 Conclusion

This chapter provides a comprehensive overview of the design and implementation plan for our project. It establishes the functional and non-functional requirements necessary for the system to operate efficiently and effectively. The chapter also highlights the importance of creating a custom dataset and the steps taken to annotate it accurately. The development and training of the recognition algorithms, along with the system architecture, are explained in detail to ensure clarity in understanding the project's approach. By automating the recognition and conversion of hand-drawn UML diagrams into digital formats, our system aims to streamline the software development process, making it easier for developers to update and maintain UML models.

Chapter 4:

Implementation and Test Case

In this chapter, we present the detailed implementation of the developed system for detecting and extracting data from hand-sketched UML diagrams, along with the description of test cases. This system leverages advanced computer vision and machine learning algorithms, including YOLO object detection models and Optical Character Recognition (OCR), to analyze and process diagrams. The implementation focuses on both frontend functionalities, which allow users to upload images and download extracted data, and backend processes, including model training and data extraction. The chapter also outlines test cases designed to validate the system's functionality, accuracy, and robustness.

4.1 Implementation

The implementation of this project is centered around building an efficient system capable of recognizing hand-sketched UML diagrams and extracting relevant data. Below, we detail the major components of the system, the algorithms used, the technologies implemented, and the overall architecture of the system.

4.1.1 Implementation Details:

4.1.1.1 Platform:

Roboflow

4.1.1.2 Functionality:

For dataset labelling annotation and dataset management.

4.1.2 Implementation of YOLOv11 Model for Diagram Detection:

Objective: The goal is to detect hand-sketched UML diagrams, specifically Class Diagrams and Activity Diagrams.

4.1.3 Model Training

YOLOv11 is trained on custom dataset to detect Class Diagrams and Activity, focusing on identifying classes, attributes, and relationships between them.

4.1.4 Text Extraction Using EasyOCR

Objective: Extract text labels and annotations from the detected diagram elements.

4.1.4.1 Implementation Details:

4.1.4.2 Platform: Python

Functionality: After detecting diagram elements with YOLO model, EasyOCR is used to extract text, which may include class names, activity labels, and other annotations present within the hand-sketched diagrams.

4.1.5 Conversion to XMI File

Objective: Convert detected diagram elements and extracted text into an XMI file that can be imported into UML modeling tools.

4.1.5.1 Implementation Details:

4.1.5.2 Platform: Python

4.1.5.3 Library: xml.etree.ElementTree

Process: The system converts detected elements and extracted text into an XMI file format by creating an XML structure representing the diagram's components. The XMI file can be downloaded and imported for further editing and refinement.

4.1.5 Frontend Functionalities

Objective: Provide an intuitive interface for users to interact with the system.

4.1.5.1 Functionalities:

1. Upload Activity Diagram: Users can upload an image containing an Activity Diagram for processing.

2. Upload Class Diagram: Users can upload an image containing a Class Diagram for processing.

3. Download XMI/JSON File: Once the image is processed, users can download the data in an XMI file format.

4.2 Test Case Design and Description

The test cases for this project are designed to validate the functionality and accuracy of the system, ensuring reliable diagram detection, text extraction, and conversion to XMI format.

4.2.1 Test Case 1

< Unsupported Image Format Error>			
<Reference>			
Test Case ID:	TC-001	**Test Date:**	*01-Oct-2024*
Test case Version:	v1.0	**Use Case Reference(s):**	*Format Validation*
Revision History:	*None*		
Objective	*Validate only supported formats (JPEG, PNG)*		
Product/Ver/Module:	*File Upload Module*		
Environment:	File Validation System		
Assumptions:	Unsupported formats trigger error message		
Pre-Requisite:	Ensure system rejects unsupported files		

Step No.	**Execution description**	**Procedure result**
1	*User uploads unsupported image format (e.g., BMP, TIFF)*	*System rejects upload and displays an error*
Comments:		

Passed/Failed	*passed*

4.2.2 Test Case 2

< Correct Diagram Detection >			
<Reference>			
Test Case ID:	TC-002	**Test Date:**	*10-OCT-2024*
Test case Version:	V2.0	**Use Case Reference(s):**	*Class Diagram Detection*
Revision History:	*None*		
Objective	*Ensure correct Class diagram detection functionality*		
Product/Ver/Module:	*UML Class Diagram Detection Module*		
Environment:	*YOLOv11 Model, Python, TensorFlow/YOLO Environment*		
Assumptions:	*User correctly selects diagram options*		
Pre-Requisite:	*YOLOv11 model trained on class diagrams*		

Step No.	**Execution description**	**Procedure result**
1	*User selects Class diagram for detection*	*System detects and verifies the uploaded image*
2	*User uploads a class diagram for detection*	*System should recognize the diagram and ensure correct detection*
Comments:		

Passed/Failed	*Passed*

4.2.3 Test Case 3

< Correct Diagram Detection >			
<Reference>			
Test Case ID:	TC-003	**Test Date:**	*15-Oct-2024*
Test case Version:	V3.0	**Use Case Reference(s):**	*Activity Diagram Detection*
Revision History:	*None*		
Objective	*Ensure correct Activity diagram detection functionality*		
Product/Ver/Module:	*UML Activity Diagram Detection Module*		
Environment:	*YOLOv11 Model, Python, TensorFlow/YOLO Environment*		
Assumptions:	*User correctly selects diagram options*		
Pre-Requisite:	*YOLOv11 model trained on Activity diagrams*		

Step No.	**Execution description**	**Procedure result**
1	*User selects Activity diagram for detection*	*System detects and verifies the uploaded image*
2	*User uploads a Activity diagram for detection*	*System should recognize the diagram and ensure correct detection*
Comments:		

Passed/Failed	*Passed*

4.2.4 Test Case 4

< Blank Image Upload >			
<Reference>			
Test Case ID:	TC-004	**Test Date:**	*20-Oct-2024*
Test case Version:	V4.0	**Use Case Reference(s):**	*Image Content Validation*
Revision History:	*None*		
Objective	*Ensure blank images are not processed*		
Product/Ver/Module:	*YOLOv11*		
Environment:	*YOLOv11 Model, Python, TensorFlow/YOLO Environment*		
Assumptions:	*Blank images return unchanged*		
Pre-Requisite:	*Blank image file available*		

Step No.	**Execution description**	**Procedure result**
1	*User uploads blank image (e.g., white page*	*System detects no content and returns image unchanged*
2	*User uploads a class diagram or activity diagram and requests for detection.*	*System should recognize and detect successfully*
Comments:		

Passed/Failed	*Passed*

4.2.5 Test Case 5

< Incorrect Detection for Unsupported Diagram Types >			
<Reference>			
Test Case ID:	TC-005	**Test Date:**	*20-Oct-2024*
Test case Version:	V5.0	**Use Case Reference(s):**	*Unsupported Diagram Detection*
Revision History:	*None*		
Objective	*Ensure unsupported diagrams are not mistakenly detected as class or activity diagrams*		
Product/Ver/Module:	*UML Diagram Detection Module*		
Environment:	*YOLOv11 Model, Python, TensorFlow/YOLO Environment*		
Assumptions:	*Only activity and class diagrams should be detected correctly*		
Pre-Requisite:	*System is not trained to handle unsupported diagrams*		

Step No.	**Execution description**	**Procedure result**
1	*User uploads a non-UML or unsupported diagram (e.g., use case, sequence)*	*The System shows wrong detection.*
2	*User uploads supported diagrams (Class, activity)*	*System should recognize and detect correctly.*
Comments:		

Passed/Failed	*Passed*

4.3 Test Metrics

The Coverage and effectiveness of our system is shown below.

4.3.1 Sample Test case Matric

Metric:	**Calculation**	**Result**
Number of Test Cases:	Total test cases developed: 5	5
Number of Test Cases Passed:	Number passed: TBD after execution	5
Number of Test Cases Failed:	Number failed: TBD after execution	0
Test Case Defect Density:	(No of test cases failed * 100) No of test cases executed	(0 * 100) / 5 = 0%
Test Case Effectiveness:	No of defects detected using test cases *100 Total number of defects detected	0% (no defects detected)

4.4 Conclusion:

This chapter elaborated on the implementation details of the system components, including YOLO-based detection, OCR-based text extraction, and XMI file conversion. We also outlined test cases to validate system functionality, ensuring robust detection and accurate data extraction for hand-sketched UML diagrams. The test cases and metrics provide a structured approach to evaluating system performance and traceability, ensuring each feature aligns with specified requirements.

Chapter 5:

Experimental Results and Analysis

This chapter presents the results and detailed analysis of the implemented methods for detecting and extracting information from hand-sketched UML diagrams. The experimental evaluation covers both the machine learning-based approach (using YOLO models and OCR for text extraction) and a traditional image processing approach (using contour detection and Canny edge detection). We discuss the results obtained from each method, compare their performance, and provide insights into their strengths and limitations.

This chapter further explores the transformation of detected diagram elements into XMI files, evaluating their compatibility within existing UML modeling tools. The subsequent sections outline the challenges encountered during detection, the superiority of the YOLO-based method for complex diagrams, and opportunities for further optimization. By comparing modern deep learning techniques with conventional image processing, this chapter provides a comprehensive view of their effectiveness in real-world scenarios and paves the way for future advancements.

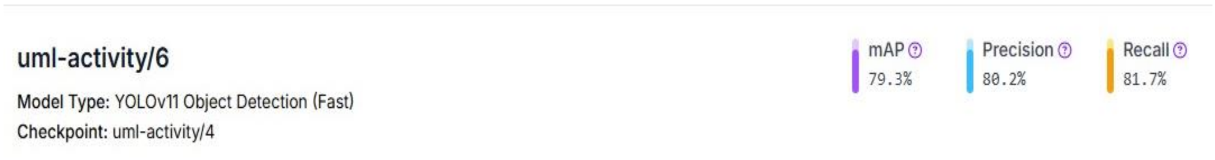
5.1 Experimental Setup

For the trials, a set of manually drawn UML diagrams (Class Diagrams and Activity Diagrams) were utilized. The YOLO V11 object detection model was trained on a custom dataset, and the dataset was annotated using RoboFlow and includes diagrams supplied by different users to ensure diversity.

5.2 YOLO-Based Detection and OCR Results

5.2.1 Detection Accuracy

The YOLOv11 was trained on annotated dataset of Class Diagram and Activity Diagram to detect Activity Diagrams and Class Diagrams, respectively.

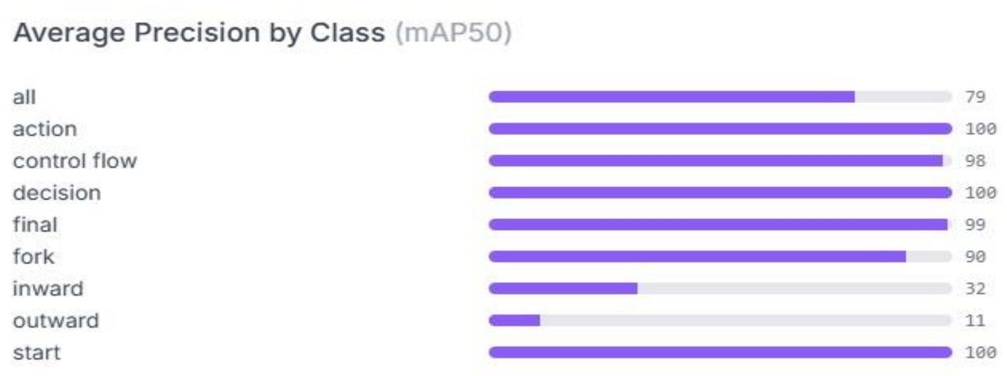


uml-activity/6

Model Type: YOLOv11 Object Detection (Fast)
Checkpoint: uml-activity/4

mAP	Precision	Recall
79.3%	80.2%	81.7%

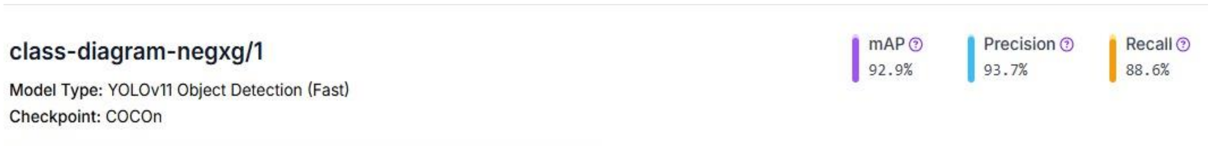
Figure 5.1 illustrates the performance metrics of the YOLOv11 object detection model on Activity Diagram Datset, highlighting its mAP, precision, and recall rates.



Average Precision by Class (mAP50)

all	79
action	100
control flow	98
decision	100
final	99
fork	90
inward	32
outward	11
start	100

Figure 5.2: Shows the average precision by class for various elements detected in the test set.



class-diagram-negxg/1

Model Type: YOLOv11 Object Detection (Fast)
Checkpoint: COCOn

mAP	Precision	Recall
92.9%	93.7%	88.6%

Figure 5.3: illustrates the performance metrics of the YOLOv11 object detection model on Class Diagram Datset, highlighting its mAP, precision, and recall rates.

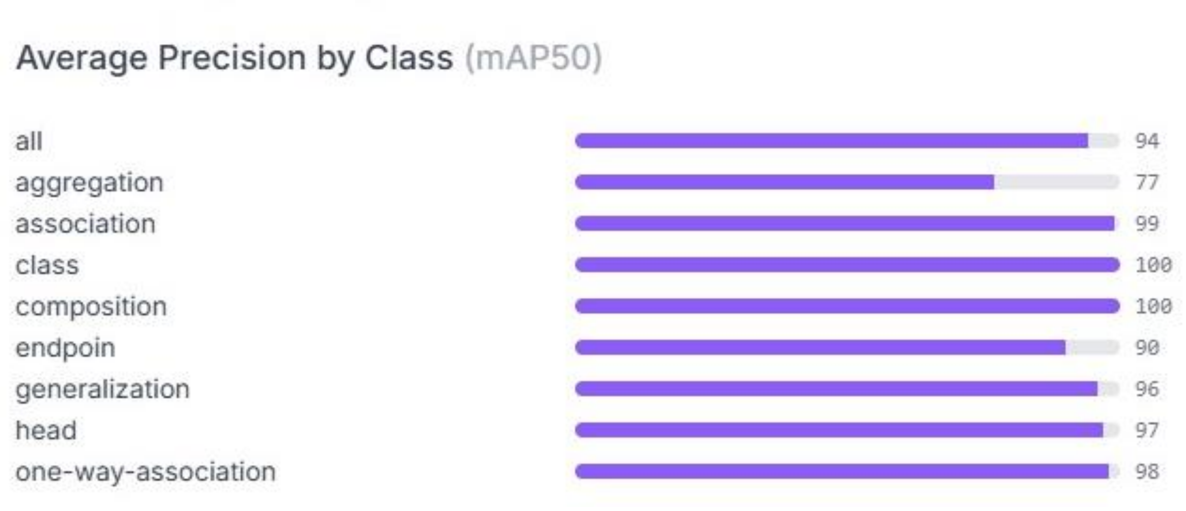


Figure 5.4: Shows the average precision by class for various elements detected in the test set.

Analysis: The YOLO object detection model demonstrated good precision and recall values, indicating reliable performance for detecting diagram elements.

5.2.2 Text Extraction Using EasyOCR

We evaluated several OCR solutions, including Tesseract and other frameworks, to extract text from detected diagram elements. Among these, EasyOCR delivered superior performance due to its accuracy and ability to handle the unique characteristics of hand-sketched UML diagrams. In Addition we have integrated **`SpellChecker`** library to correct any spelling errors in the extracted text, enhancing accuracy and reliability, particularly for hand-sketched input. The corrected text is then stored and used for further processing or output.

5.2.3 Yolo Based Detection of Elements and Text in Activity Diagram

We detected this hand sketched Activity Diagram using our YOLO object detection model and detected and extracted the text using EasyOCR and after detection we created xmi file of this Diagram and opened it in modeling tool.

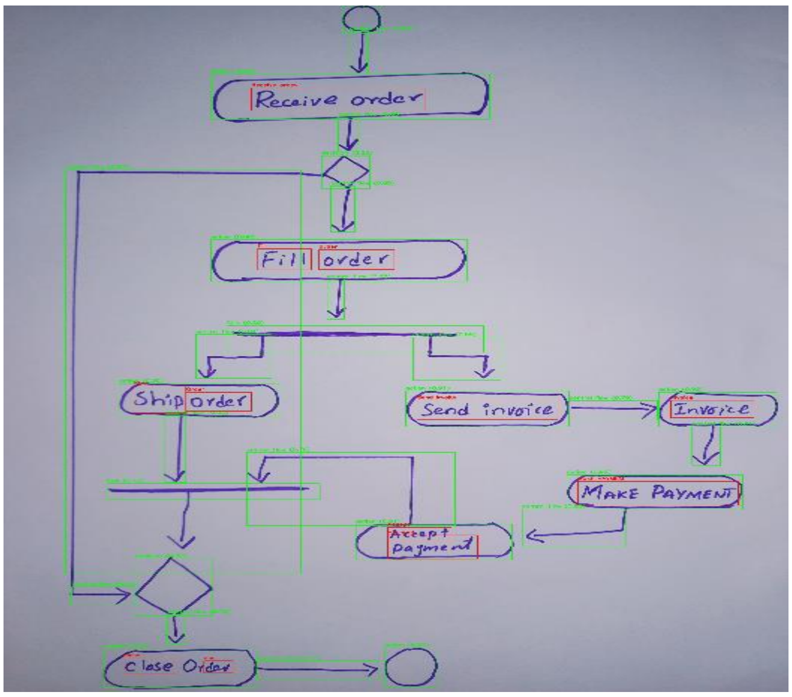


Fig: 5.5 Yolo Based Detection of Elements and Text in Activity Diagram

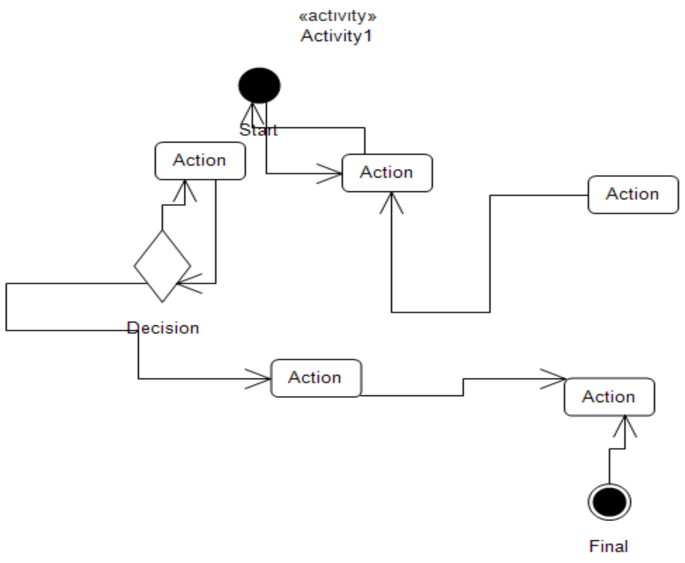


Fig: 5.6 Result After Opening the Detected Diagram in Modeling Tools

5.2.4 Yolo Based Detection of Elements and Text in Class Diagram

We detected this hand sketched class Diagram using our YOLO object detection model and after detection we created xmi file of this Diagram and opened it in modeling tool.

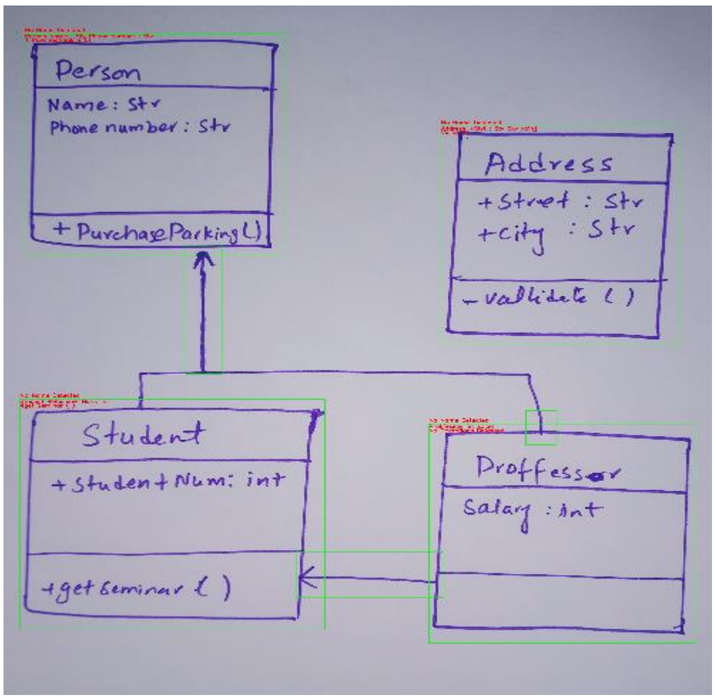


Fig: 5.7 Yolo Based Detection of Elements and Text in Class Diagram

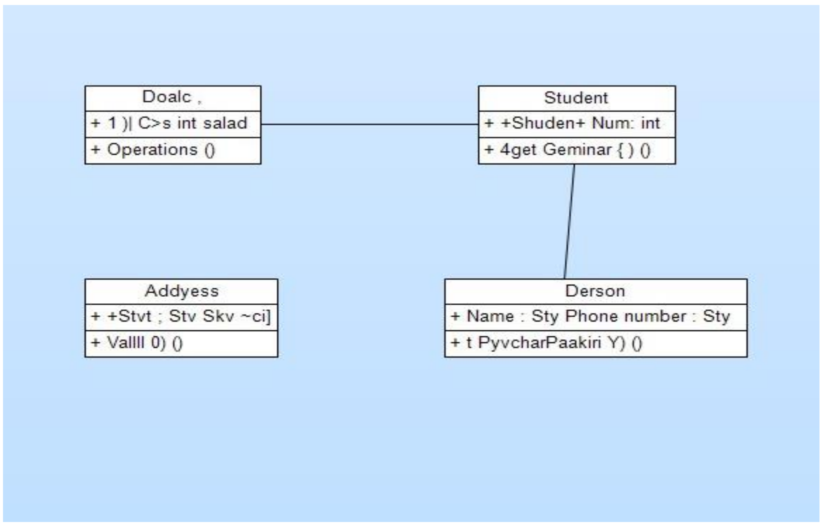


Fig: 5.8 Result After Opening the Detected Diagram in Modeling Tools

5.2.3 XMI Conversion

The detected elements and extracted text were converted to XMI format using Python's `xml.etree.ElementTree`. The generated XMI files were compatible with standard UML modeling tools, successfully capturing the key structure and elements of the diagrams

5.3 Traditional Image Processing-Based Detection Results

In addition to the YOLO-based approach, a traditional image processing method was employed, utilizing Canny edge detection, Hough Line Transform, and contour analysis to detect lines and shapes (e.g., rectangles, diamonds, and arrows).We used the same images figure 5.5 and 5.7 and after performing detection using traditional methods and creating xmi we opened it tool and the detection results were really poor as shown in figure 5.9 and 5.10.

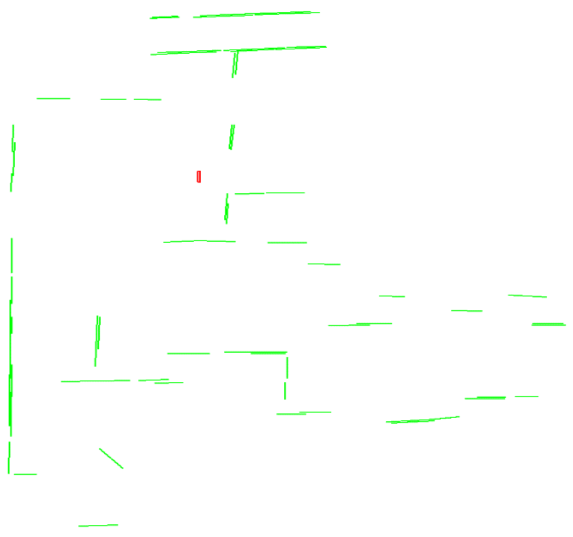


Figure5.9: Result of Activity Diagram after opening the detected image in modeling tool.

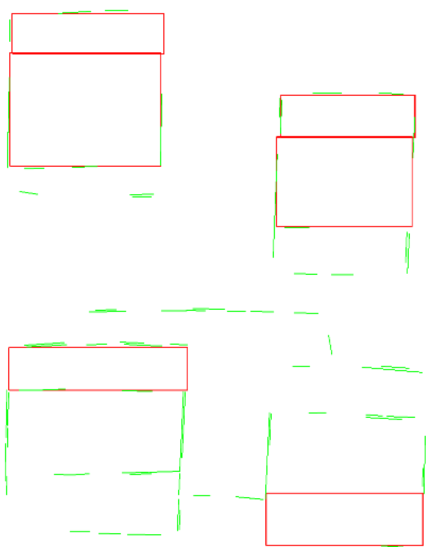
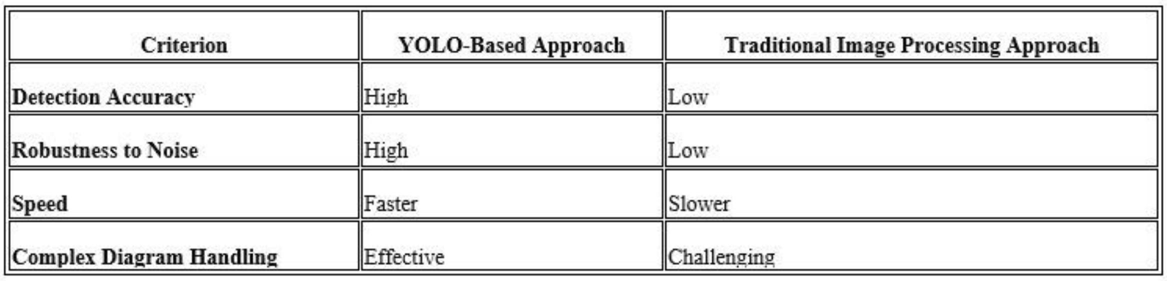


Figure5.10: Result of Class Diagram after opening the detected image in modeling tool.

5.4 Analysis

The YOLO-based detection method outperformed the traditional approach, offering superior accuracy and robustness in handling complex shapes and variations in hand-sketches. In contrast, the traditional method struggled with noise and was less effective overall.



Criterion	YOLO-Based Approach	Traditional Image Processing Approach
Detection Accuracy	High	Low
Robustness to Noise	High	Low
Speed	Faster	Slower
Complex Diagram Handling	Effective	Challenging

Table 5.1: Comparison of YOLO-Based and Traditional Method

5.4.1 Strengths & Limitations

5.4.1.1 YOLO-Based Method:

Offers high detection accuracy, faster processing, and better capability for handling complex diagrams. However, it may require significant computational resources for training.

5.4.1.2 Traditional Methods:

Simple to implement and doesnot performs well with hand sketched diagrams and struggles with noise, complex inputs and was less effective overall.

5.5 Experimental Results Summary

Overall, the YOLO-based approach combined with EasyOCR for text extraction demonstrated superior performance in detecting and processing hand-sketched UML diagrams. The traditional image processing method was less robust for complex and noisy diagrams. Future work could explore further optimization of the YOLO models, fine-tuning the OCR engine, and integrating hybrid approaches to improve accuracy and speed.

5.6 Conclusion

This chapter analyzed and compared the results of the YOLO-based detection and traditional image processing methods. While the YOLO-based approach outperformed in terms of accuracy and speed, the traditional method provided insights into simpler and resource-constrained applications. The results form a foundation for further enhancement and development of robust solutions for analyzing hand-sketched diagrams in real-world scenarios.

Chapter 6:

Conclusion and Future Directions

6.1 Conclusion and Summary of Work

This research investigated the creation of a system that uses YOLO-based object detection techniques to identify and transform hand-drawn UML diagrams. Using sophisticated machine learning models and EasyOCR text recognition, the solution concentrated on identifying and extracting elements from activity and class diagrams. After identifying UML symbols in the input designs, they were transformed into standardized XMI files that could be integrated with UML tools. The method fared better than OpenCV's typical image processing techniques, showing improved detection accuracy, noise resistance, and effective handling of intricate diagrams.

One of the main conclusions is that YOLO models perform better at identifying hand-drawn UML elements. This demonstrates deep learning's proficiency in pattern recognition challenges. Traditional approaches were useful for simpler situations, but they had trouble understanding the intricate structures, misalignments, and irregularities that come with hand-drawn designs. Therefore, by creating a system that can identify and convert hand-sketched UML diagrams with a respectable level of accuracy, the project's scope was mostly met, while there is still opportunity for improvement.it was difficult to maintain complex diagrams' precise shapes and proportions, and further work is needed to improve visual accuracy. To handle a variety of handwriting styles and lower errors, text recognition which is essential for deciphering labels and annotations within diagrams can also be significantly enhanced. The accuracy, dependability, and flexibility of the system for a range of user inputs will be further improved by future research concentrating on these areas.

6.2 Challenges Faced and Scope Coverage

Although the project's main objectives were achieved, a number of difficulties surfaced, especially with regard to dataset development, annotation, and labeling because there were no easily available specialized datasets for hand-sketched UML diagrams. To produce a varied dataset that could capture variations in sketching methods, problems with diagram alignment, and occlusions, a significant amount of manual labor was required. Despite its effectiveness, the YOLO model needed a well-annotated dataset in order to continue exhibiting consistent detection performance. Variability in handwriting styles also presented challenges for text extraction, affecting the precise identification of annotations and labels.
Finding and maintaining the integrity of text and diagram structures when they are imported into modeling tools is another crucial area that has to be improved. It was difficult to write and maintain the produced XMI files' compliance with different modeling programs, which hindered smooth integration. To better manage real-time processing and guarantee precise word detection across intricate designs, the system's overall responsiveness might also use some improvements. Future research addressing these constraints would greatly increase the accuracy, resilience, and adaptability of the system for wider use scenarios.

6.3 Future Recommendations

6.3.1 Dataset Diversity and Expansion:

Expanding the training dataset to include more varied samples and new UML shapes is essential to improving the detection's resilience. The system can more effectively handle the variability of the real world by integrating a variety of diagrams and elements

6.3.2 Improving Detection Accuracy:

It is crucial to improve the system's detection accuracy for both text components and diagram forms. Fine-tuning procedures, enhanced model architecture, and more accurate object detection algorithms can all help achieve this.

6.3.3 Enhancing Text Recognition Precision:

Increasing text extraction accuracy should be the main goal of future development especially for a variety of handwriting styles.

6.3.4 Compatibility with More Modeling Tools

The goal should be to increase compatibility with different modeling tools. This would increase the system's usability, guarantee smooth integration with widely used platforms, and provide more flexible outputs.

6.3.5 Maintaining Diagram Shapes and Text Integrity

Higher fidelity outputs will result from efforts to better maintain the accuracy of the text and diagram shape during the transformation process. For system dependability and user confidence, it is essential that diagrams be digitized while maintaining their original structure and text intelligibility.

Future revisions of this project will improve its efficacy and provide a more reliable, flexible solution for hand-sketched UML diagram detection and transformation by concentrating on these areas.

References

[1] B. Karasneh, and M. Chaudron, “Extracting UML Models from Images,” in Proc. International Conference on Computer Science and Information Technology (CSIT), pp. 1- 15, 2013. DOI: 10.1109/CSIT.2013.6588776.

[2] B. Schafer, “Recognizing Hand-drawn Diagrams in Images,” Ph.D. dissertation, Univ. of Mannheim, Mannheim, 2023.

[3] M. Axt, “Transformation of sketchy UML Class Diagrams into formal PlantUML models, Final Project, Main field of study: Computer Engineering BA ©, Credits: 15, Semester/year: Spring 2023, Supervisor: R. Jolak, Examiner: F. Dobslaw, Course code/registration number: DT133G, Programme: Software Engineering.

[4] V. Deufemia and M. Risi, "Multi-domain recognition of hand-drawn diagrams using hierarchical parsing," Dipartimento di Informatica, Università di Salerno, pp. 1-10, Aug. 2020.

[5] L. Qiu, "SketchUML: The design of a sketch-based tool for UML class diagrams," Department of Computer Science, State University of New York at Oswego, Oswego, New York, USA.

[6] T. A. Hammond and R. Davis, "Tahuti: A sketch recognition system for UML class diagrams - extended abstract," in Proc. Conference on Sketch Recognition Systems, July 2002.