

UI Development and Functionality Testing Automation in Android Application

by

Shehjad Ali Taus

19101539

Riazul Hasan

19301168

Golam Rasul

19301126

Anila Tabassum

19101157

Khondoker Al Muttakin

22241176

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
School of Data and Sciences
Brac University
Summer 2023

© 2023. Brac University
All rights reserved.

Declaration

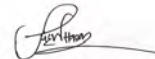
It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:



Shehjad Ali Taus
19101539



Riazul Hasan
19301168



Golam Rasul
19301126



Anila Tabassum
19101157



Khondoker Al Muttakin
22241176

Approval

The thesis/project titled “UI Development and Functionality Testing Automation in Android Application” submitted by

1. Shehjad Ali Taus (19101539)
2. Riazul Hasan (19301168)
3. Golam Rasul (19301126)
4. Anila Tabassum (19101157)
5. Khondoker Al Muttakin (22241176)

Of Summer, 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on Fall , 2023.

Examining Committee:

Supervisor:
(Member)



Md. Golam Rabiul Alam, PhD
Professor
Department of Computer Science and Engineering
Brac University

Co-Supervisor:
(Member)



Muhammad Iqbal Hossain, PhD
Associate Professor
Department of Computer Science and Engineering
Brac University

Thesis Coordinator:
(Member)



Md. Golam Rabiul Alam, PhD
Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

We officially declare that this thesis is supported by our study findings. This research work is free of plagiarism at all research levels. All other information sources have been acknowledged in the text. This thesis has not been submitted, in whole or in part, to any other university or institution for the purpose of degree-granting.

Abstract

Software Automation Process involves automating the software testing process entailing machine learning models and methodologies. This may entail procedures like test case prioritization, selection and test case generation. Machine learning can be used to rate problems and recommend fixes upon that, As well as to identify software faults. Additionally, Machine Learning (ML) can be used to analyze test coverage, improve test efficiency and optimize processes. Overall, The use of machine learning in software testing automation can help to improve the speed, accuracy and efficiency of the testing process, leading to higher-quality software and a quicker time to market. Finding and correcting software bugs requires a lot of work on the part of software engineers. Traditional testing requires human search and data analysis which is not time efficient. Errors are frequently ignored because people have a tendency to make false assumptions and arrive at prejudiced conclusions. Since machine learning enables systems to learn, adapt and use the learned knowledge in the future, software testers profit from more accurate understanding. Numerous sophisticated machine learning tasks including code completion, defect prediction, bug localization, clone recognition, code search and learning API sequences can be accomplished via deep learning. Over the years, Researchers have published a variety of methods for automatically switching between programmes. Ultimately, Machine learning for automated software testing is an intriguing field that has the possibility to entirely alter how software is tested. Before machine learning is widely used in software testing, There are still a few issues requiring to be solved. This paper represents, Sequence-to-sequence (Seq2Seq) modeling is a deep learning technique used in machine learning and natural language processing (NLP) for tasks involving sequences of data. It's particularly powerful for tasks where the length of input and output sequences can vary. Again, Seq2Seq models are widely used for translating text from one language to another. The encoder processes the source language, and the decoder generates the target language. Here, We also apply encoder-decoder techniques in machine learning. Encoder-decoder techniques are fundamental in machine learning, particularly in tasks involving sequence-to-sequence modeling, natural language processing (NLP), computer vision, and more. These techniques involve two key components: an encoder and a decoder. In NLP, for example, the encoder may be a recurrent neural network (RNN) or a transformer model like BERT. In computer vision, a convolutional neural network (CNN) can serve as the encoder. These models are designed to extract relevant features from the input data. The decoder receives the context vector from the encoder and initializes its internal state. It generates an output sequence step by step, often autoregressive. For each step, it produces an element of the output sequence and updates its internal state based on previous outputs.

Keywords: Deep Learning; Machine Learning; NLP; RNN; Seq2Seq; CNN

Dedication

Every difficult task requires personal effort and encouragement from our elders, especially those closest to our hearts. We dedicate our humble efforts to our loving parents, whose love, devotion, motivation, and nightly prayers have made us deserving of this achievement and honor, as well as all of the outstanding academics we met and learned from while pursuing our Bachelor's degrees, and especially our beloved supervisor, Dr. Md. Golam Rabiul Alam and Co-supervisor Dr. Muhammad Iqbal Hossain.

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis have been completed without any major interruption.

Secondly, to our Supervisor Dr. Md. Golam Rabiul Alam and Co-supervisor Dr. Muhammad Iqbal Hossain sir for his kind support and advice in our work. He helped us whenever we needed help.

And finally to our parents without their throughout support it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

Declaration	i
Approval	ii
Ethics Statement	iv
Abstract	v
Dedication	vi
Acknowledgment	vii
Table of Contents	viii
List of Figures	x
List of Tables	xi
Nomenclature	xi
1 Introduction	1
1.1 Motivation	1
1.2 Research Problem	2
1.3 Thesis Structure	3
2 Related Works	4
2.1 Literature Review	4
2.2 YOLOv5	11
2.3 YOLOv8	12
2.4 Pytorch Framework	12
2.5 Computer Vision	13
2.6 Labellmg	14
3 Model & Dataset	15
3.1 Dataset description	15
3.2 Data Preprocessing	16
3.2.1 Data Preprocessing	16
3.2.2 Feature selection	17
3.3 Model description	18
3.3.1 Single Stage Object Detector	20

3.3.2	Other important parts of an improved result	21
4	Implementation & Result Analysis	22
4.1	Implementation	22
4.1.1	Hardware Specification	22
4.1.2	Environment Setup	23
4.1.3	Package Installation	23
4.1.4	Custom Model Configuration	24
4.1.5	Automation System	25
4.2	Result Analysis	26
4.2.1	Comparative Analysis	26
4.2.2	Selected Model Result Analysis	35
4.2.3	System Analysis & Overview	36
5	Conclusion	40
5.1	5.1 Challenges	40
5.2	Future Prospect	40
5.3	Conclusion	41
	Bibliography	45

List of Figures

3.1	Figure: Summary of RICO dataset	15
3.2	Figure : Labeling images using Labellmg package	16
3.3	Figure: Data after Image Labeling	17
3.4	Figure:Visual Representation how labeling works	17
3.5	Figure: YOLOV5 architecture representation	19
3.6	Figure: YOLOV8 architecture	19
3.7	Figure: Figure: YOLOV8 model architecture representation	20
4.1	YOLOV8 model architecture representation	24
4.2	YOLOv5 Confusion Matrix	27
4.3	YOLOv8 Confusion Matrix	27
4.4	F1 Confidence Curve of YOLOv5	28
4.5	F1 Confidence Curve of YOLOv8	29
4.6	P Curve of YOLOv5	30
4.7	P Curve of YOLOv5	31
4.8	PR Curve of YOLOv5	32
4.9	PR Curve of YOLOv8	32
4.10	R Curve of YOLOv5	33
4.11	R Curve of YOLOv8	34
4.12	Overall Results of YOLOv8	35
4.13	System Overview	36
4.14	Application Screenshot and Design Screenshot Respectively	37
4.15	UI Elements Detected.	38
4.16	Application Testing Mechanism in terms of Design.	39

List of Tables

3.1	Figure: Features	18
-----	----------------------------	----

Chapter 1

Introduction

User Interface (UI) development is an important part of the software development life cycle and is crucial to user engagement and application success. A well-designed user interface not only draws users in, but it also promotes recurring usage and top-notch suggestions. However, An inadequately designed user interface might turn off consumers and reduce the app’s overall efficacy. The placement and seamless functioning of these components become critical aspects when it comes to Android applications, which employ a number of core and composite components. This complexity is increased by the requirement for functionality tests, sometimes known as ”Black-box” testing, to identify and correct non-functional components. As part of the current practice, Quality Assurance (QA) teams conduct a complete review, adding to a laborious process that can substantially hamper development deadlines. Our aim is to tackle this challenge by proposing a new approach to simplify and mechanize UI development and functionality testing. The primary goal is to give UI developers an efficient automatic suggestion system for core and composite component placement. By automating functionality testing, developers may ensure that project objectives are met more quickly and that they have the ability to independently assess the performance of UI components. Reducing the time and money that may be lost on manual testing is the aim of the research. By empowering UI developers to make knowledgeable choices about the location and functionality of components, this approach aims to increase overall UI development efficiency and facilitate the timely delivery of high-caliber Android applications.

1.1 Motivation

An organized Software Development Life Cycle (SDLC) is necessary for producing high-quality software projects, emphasizing methodical progress. In this case, the Software Testing Life Cycle (STLC) makes sure the software is error-free. However, during the UI development phase, a crucial stage in the SDLC, achieving optimal design, component placement, and functionality can be challenging. The conventional process involves UI developers and Quality Assurance (QA) teams exchanging iteratively, which dramatically extends development timelines and delays market release. We have to speed up the UI creation and testing process, which is why we are doing this research.

By releasing self-guided recommendations for UI developers and an automated func-

tional testing tool, our goal is to empower developers to put components together efficiently and test functionality with ease. This project aims to provide a system that recommends where core and composite components should be placed in Android applications in order to expedite functionality testing. Our ultimate objectives are to reduce schedule constraints, ensure the timely deployment of products onto the market, and optimize productivity in the UI development process. This research attempts to offer a solution that optimizes the UI development life cycle for the benefit of developers, QA teams, and the software development process overall.

1.2 Research Problem

Important UI development processes include the evaluation by QA testing teams and the ensuing black-box (functional) testing, which are labor-intensive processes meant to guarantee that the UI design meets requirements. Even when testing takes up a sizable (20–30% of development time, deadlines often pass before crucial application testing is completed. The extended time periods required to do rigorous testing and finish the application design are exacerbated by the continuous feedback loops that take place between QA testing teams and UI developers. User interface development lacks any procedures or tools that enable quick functional testing and feedback on component placements, in contrast to other software development phases that have access to automation technology. The primary question this study attempts to answer is how automation might assist UI developers in performing UI design and QA test jobs more quickly during the development process. The goal is to save time in both the SDLC and STLC stages to guarantee timely application deployment.

1.3 Research Objective : The overall goal of this research is to expedite and enhance the software development process, with a focus on user interface development. The goal is to give UI developers the tools they need to introduce automation technologies early in the development cycle and maximize their productivity. The main objective is to make it easy for UI developers to do auto-functional testing and to get recommendations for the optimal positions of Android composite and core components. Reducing the necessity for the manual testing, bug-repair, and retesting processes that are a part of the typical iterative software testing life cycle (STLC) is the aim of this study. The project's ultimate objective is to revolutionize user interface creation by reducing the requirement for post-development testing, which will save a substantial amount of time and provide a more seamless transfer from development to deployment.

Provide a framework that allows UI developers to work on the user interface while automatically suggesting locations for Android core and composite components. Give UI developers the tools they need to automatically test component implementations, saving them from having to perform manual testing iteratively. The Software Testing Life Cycle (STLC) can be shortened to speed up the fixing of issues and approval of applications for deployment. Automation approaches can help you expedite the UI development process and free up UI developers to independently check the quality and usability of the application without having to rely too much on QA testing teams.

1.3 Thesis Structure

Our study article looks at the significance of the Software Development Life Cycle (SDLC) and Software Testing Life Cycle (STLC) in software projects and how they impact successful deployment. We examine our study rationale in detail, outlining the driving forces for our investigation into different aspects of software engineering. We have outlined the study challenge as well as the objectives we hope to achieve in this field. In the background chapter, we conducted a comprehensive literature study and provided an overview of earlier studies that were relevant to your subject. This provides background information and highlights the knowledge gaps that our research aims to fill, thus establishing the context for our study. Additionally, we explained key concepts and illustrated their importance in our explanations of object identification, computer vision, the PyTorch framework, and YOLOv5 and YOLOv8. The collection and pre-processing of the dataset for training were covered in detail in the Model and Dataset chapter. The dataset's component features and quantities were provided in a thorough statistical analysis. This chapter also discussed the features of the YOLOv5 and YOLOv8 models and how they help to increase their accuracy. In the chapter on implementation and outcome analysis, we gave readers a comprehensive implementation guide so they could replicate our system step-by-step. After outlining the rationale for the selection of YOLOv5 and YOLOv8, the chapter compared several YOLOv5 and YOLOv8 iterations that were trained on the same dataset. The performance of the YOLOv5 and YOLOv8 models was shown, and the measurements and results were presented in detail, providing insights into their efficacy. Interestingly, we went beyond YOLOv5 in our analysis and added YOLOv8. This contribution enhances the comprehensiveness of our study by offering an analysis from which to contrast these two object detection models.

Chapter 2

Related Works

2.1 Literature Review

Starting with a research paper [6], it's an analysis regarding image rectification software test automation using robotic arms. Phone software features can be tested with the use of robotic arms. For example, - Image rectification or mobile devices. Basically, a precise test automation system for testing and validating the computer vision algorithms used for image rectification in a mobile phone. The robotic arm setup provides us with the flexibility to run our test cases using multiple speeds, rotations, and tilt angles. The objective behind this project is to design and develop the test automation for the image rectification feature and the use of a robotic arm for automating this use case. Traditional manual testing requires so much hard work yet is a lengthy process to accomplish , In this way , It will detect problems easily and make the work leniently.

In this research paper [17], Revisiting Test Impact Analysis in Continuous Testing from the Perspective of Code Dependencies,. As a continuous process, automated test cases are executed in a limited proportion to improve the quality of existing code. A continuous process ensures a better pattern and reduces hassles. If a code smell occurs, we have to be consistent through the development process. Continuous testing has a great impact on the quality of software. Term CI (Continuous Integration) is widely used in modern software development systems, including its practice of integrating developers code changes to a central code repository often. To ensure the quality of the integrated code , developers need to run sets of test cases for each code integration . Due to continuous change, there are possibilities to face a fall yet a A certain degree of dependency between test cases and source code files may remain. Within the following research paper [2] , the effectiveness of Metamorphic Testing to solve.

The Oracle problem has been discussed . Software testing systems have their own term, Oracle, which verifies the appropriate test case execution results. Problems occur when they do not exist or are out of budget. From that point of view , the metamorphic testing approach appears . It's a testing approach that solves the oracle problem in multiple techniques and ways . Even a small number of various metamorphic relations identify the fault and solve the oracle problem .In this current situation , software development systems are increasing with the massive demand of software usage among people . In such a case , effective testing alternatives must have been used. Metamorphic can be an alternative to prevent oracle

problems within a system. The research paper [19] introduces A new automation testing kit is announced and tested on nanosatellite flight software named CubeSAT Space Mission to ensure flight software quality. Before in the space field , several testing techniques were used to assess flight software . When the developers approached manual testing with classic testing strategies they found out 12 bugs were not covered in less than three days . The fuzz testing improved the whole quality of the software through automation. In this research paper [25], we find that they design and implement a tool for testing service level agreement. It is a technical document that contains the conditions that must be fulfilled during the provision and consumption of services. Using different combinatorial testing methods, they are designing and implementing a tool that is able to identify a set of constraints that avoid the obtaining of non feasible tests by analyzing the information contained in the SLA guarantee terms. By using this tool,they have gained the tests for critical E-Health scenarios and proposes in the FP7 European project.

According to the paper [11] , artificial intelligence (AI) has the potential to revolutionize the field of software testing by using smart algorithms to analyze complex data and improve the quality of software. In the future, AI is expected to play a key role in software testing, with testers focusing on training AI models and techniques to become smarter. AI testing algorithms will also be able to connect to new technologies like cloud computing, the Internet of Things, and big data to generate more accurate and effective test cases. Deep learning, natural language processing, and other techniques will also be important in software testing and may be supported by specialized tools and hardware.Overall,the use of AI in software testing is expected to improve the customer experience by providing defect-free applications and solutions. According to the research paper [15], we find the limitations of automation in testing and generating test cases . Also, it shows that the importance of the user environment is immense, and we should always pay attention to the pain-points of tool users . An user has no user manual, and environmental faults are difficult to debug, and this fault may lead to incorrect results or increase user complaints .

According to the research paper [11], artificial intelligence (AI) is a field of computer science and engineering focused on the creation of intelligent machines that can perform tasks without explicit human instruction. AI technologies and techniques, such as machine learning, deep learning, and natural language processing, have achieved significant progress in recent years and are being applied in a variety of fields, including software testing. In software testing, AI algorithms and techniques can be used to evaluate information, analyze data, and perform tasks in order to ensure the quality and functionality of software.

This particular review [13] is about automated online combat game testing using evolutionary deep reinforcement learning. The complicated method of automated game testing, which frequently includes manual testing, is covered in this paper. The writers emphasize how tricky it is to test video games since often flaws don't show up until you've successfully completed certain demanding tasks that call for intelligence. For automated game testing, they suggest a method known as Wuji, which combines evolutionary algorithms, deep reinforcement learning (DRL), and multi-objective optimisation. Wuji strikes a balance between playing the game to the best of his ability and looking for problems. The authors tested Wuji on two commercial games and a small game to see how good it was at finding bugs. Wuji made three significant discoveries in the commercial games that were previously un-

detected flaws and were validated by developers. This study indicates a possible path for enhancing automated game assessment with cutting-edge AI methods. Apparently, video game testing [23] is getting more difficult as a result of user expectations and the games' rising complexity. Traditional approaches, such as scripted automation and manual testing, can be expensive and ineffective in non-deterministic settings. SUPERNOVA (Selection of Tests and Universal Defect Prevention in External Repositories for Novel Objective Verification of Software Anomalies) was created as a solution to these problems. It functions as an automation hub as well as a system for test selection and fault avoidance. In order to help developers and quality assurance testers uncover flaws and reduce defects, SUPERNOVA incorporates data analysis, machine learning, and deep learning capabilities, eventually enhancing production stability and cost management. For a sports game that used these test selection optimisations, this strategy resulted in a considerable decrease (55% or more) in testing hours. Additionally, It can identify the risk that a change-list would introduce problems with 71% precision and 77% recall using a semi-supervised machine learning model, giving developers important information. These initiatives simplify processes and cut down on testing time for games that are still in production.

On the other hand , A deep learning software for automated kymograph analysis [12] stands for graphical representations of spatial position over time . In biology, kymographs are used to depict the movement of particles over time. Low signal-to-noise ratios make it difficult to analyze them numerically, and current techniques frequently call for personal intervention. A deep learning-based programme called "KymoButler" automatically monitors dynamic processes in kymographs. It is offered as a web-based "one-click" tool, performs on par with manual analysis, and promotes the use of machine learning in biological research by expediting data processing.

Besides , evaluation of reinforcement learning [18], we got to monitor methods of analyzing software quality . Software testing is essential for assuring software quality, but it takes a lot of time and money and frequently accounts for 50% of production costs. To lessen these responsibilities, automated methods are being sought. Automatic test data generation techniques have been developed over the past ten years to maximize fault detection with little data production. The problem is turned into a search task in this research, and meta-heuristic techniques are used to find a solution. The suggested method outperforms many previous evolutionary and meta-heuristic algorithms in terms of speed and coverage, with fewer evaluations, by combining a genetic algorithm with reinforcement learning as a local search strategy. Genetic algorithms, random search, particle swarm optimisation, the bee algorithm, ant colony optimisation, simulated annealing, hill climbing, and tabu search are examples of comparison algorithms.

Adaptive automation [4] which denotes software application suitability using automated software tools, has become a vital element for most of the organizations . Organizations need automated software testing, but current methods frequently can't adjust to unforeseen changes without manual intervention, which results in expensive maintenance costs. In order to manage unforeseen barriers and recover gracefully, this study suggests a solution employing machine learning techniques such as fuzzy matching and error recovery. The framework produces reports for

user inspection to decide if the recovery was satisfactory, and it modifies subsequent runs in accordance with user choices. With less frequent human participation, this method permits automated testing, lowering the possibility of schedule delays.

The paper [24] introduces Keeper, a cutting-edge testing tool made to handle problems brought on by the incorporation of cognitive machine learning (ML) into software applications. Robust testing approaches are required since cognitive ML solutions are becoming more common in the current software landscape. The major novelty of Keeper is the use of pseudo-inverse functions, which reverse the cognitive operations carried out by ML APIs and allow for thorough testing of the ML components of applications. The manual work necessary for creating and evaluating test input is reduced by this empirical approach. Testing is more effective thanks to Keeper’s incorporation of pseudo-inverse functions into a symbolic execution engine, which automates the production of pertinent inputs and the evaluation of output accuracy. Through its study of open-source applications, the research highlights Keeper’s applicability and effectiveness by highlighting notable gains in branch coverage and the discovery of previously undiscovered defects. This emphasizes its practical relevance and potential to raise the caliber of software. In conclusion, Keeper’s debut as a cutting-edge testing tool for software applications integrating cognitive ML represents a significant advancement in the industry. In an era where cognitive ML solutions play an increasingly important role, its automation of input creation, output evaluation, and empirical approach to difficult ML APIs are particularly noteworthy. These features present a viable path for enhanced testing and software quality.

This research [14] makes a substantial contribution to the field of deep learning (DL) by introducing Audee, a cutting-edge methodology created to evaluate and enhance the dependability and quality of the core DL frameworks. While most previous research has focused on assessing the performance of DL models, this work acknowledges the necessity of carefully examining the frameworks that support these models. By using a search-based methodology that includes three different mutation mechanisms, Audee stands out from the crowd. These techniques make it possible to create various test cases that span a range of topics, such as model structures, parameters, weights, and inputs. System crashes, Not-a-Number (NaN) issues, and logical inconsistencies are three crucial types of bugs that Audee is particularly adept at spotting. Its use of cross-referencing approaches to identify behavioral differences between various DL frameworks sheds light on this in particular. Audee also makes use of causal-testing methods to pinpoint the particular layers and variables that lead to errors or problems in DL frameworks. A thorough assessment of Audee using four well-known DL frameworks shows how effective it is at spotting problems. As a result, many previously unidentified defects were found, seven of which have already been confirmed and fixed by developers.

In this paper [30], it clearly states the criteria for software testing tool evaluation . Given the wide range of available test tools, both commercially and as open source options, conducting a comprehensive evaluation to determine their suitability for a specific organizational or project context appears to be a daunting task. To address this challenge, we propose a systematic approach for establishing evaluation criteria that are easily verifiable for test tools. These criteria encompass both quality and functional aspects. By employing the TIRE methodology, we pinpoint activities that have the potential for automation or at least benefit from support provided by

a test tool. Using these identified activities as a starting point, we derive evaluation criteria for test tools. Our emphasis is on criteria that can be assessed based solely on the instructions provided by vendors, without the need for extensive laboratory testing. The outcome of our efforts is a practical set of criteria that enables an efficient classification and initial screening of test tools. In an initial phase, we applied our criteria to assess three capture & replay tools. In doing so, our approach demonstrated its value. We were able to discern fundamental distinctions between the tools and establish additional criteria specific to this particular test automation technique.

From this paper[10], we find The rapid advancement of artificial intelligence technology and data-driven machine learning techniques has made the development of high-quality AI-based software a prominent research area in both academic and industry circles. Nowadays, numerous machine learning models and artificial technologies have emerged for creating intelligent application systems based on multimedia inputs, enabling functionalities like recommendation systems, object detection, classification, prediction, natural language processing, and translation. This surge in AI-driven applications underscores the pressing need for quality assurance and validation of AI software systems. However, existing research tends to overlook crucial aspects of AI software testing, including associated questions, challenges, and validation methodologies with well-defined quality requirements and criteria. This paper addresses the validation of AI software quality, covering key areas of focus, features, processes, and potential testing approaches. Additionally, it introduces a testing process and a classification-based test model tailored for testing AI classification functions. In conclusion, the paper delves into the existing challenges, issues, and requirements in the realm of AI software testing.

In this paper [31], we find that , despite the presence of highly skilled quality assurance teams and advanced tools, software testing has consistently proven to be a time-intensive endeavor. This is because software development is a complex process that involves many different components and interactions. As a result, it is difficult to test all possible combinations of inputs and outputs manually. Test automation can help to address this challenge by automating the testing process. This can save time and resources and can also help to improve the quality of software by identifying and fixing bugs early in the development process. Test automation can have a significant impact on the total cost, quality, and timeline of software production. In this paper, we sought to identify key factors pertaining to test automation and its cost-effectiveness. We analyzed the effects of test automation on the cost, timeline, and quality of three distinct software projects. The results of our experiments unequivocally demonstrate the favorable impact of test automation on the cost, quality, and speed of bringing the software to market. Specifically, we found that test automation can help reduce the cost of software development by up to 50%. It can also help to improve the quality of software by up to 30%. Additionally, test automation can help to speed up the development process by up to 20%. Overall, our results suggest that test automation is a valuable tool that can help to improve the quality, cost, and speed of software development.

In this article [7], we find that the primary objective of this paper was to examine the functioning of artificial intelligence in the realm of software test automation. Within the domain of software engineering, artificial intelligence (AI) has exerted a substantial impact, and software testing is no exception. With the integration

of artificial intelligence (AI), achieving the goal of automating software testing appears more attainable than ever before. Over the past two decades, there has been a shift in paradigm. The entire testing process has seen a positive evolution, moving from manual testing to automated testing, with Selenium being recognized as one of the premier automation tools. Consequently, in today's fast-paced IT environment, software testing must adopt novel approaches founded on sound research. The introduction of AI-based testing has proven highly advantageous for this purpose [1]. AI algorithms and machine learning (ML) enable a computer to learn autonomously without human intervention. While AI and ML necessitate the creation of distinct and tailored algorithms to process and learn from data, identifying patterns to draw conclusions, these predictive capabilities are intended to be fully leveraged in software testing.

According to this research [21], Consumer Electronic Manufacturing (CEM) companies often struggle to maintain quality standards during frequent product launches, risking product recalls. To address this challenge, a universal automated testing system is proposed. This system features a universal hardware interface for connecting commercial off-the-shelf (COTS) test equipment to the unit under test (UUT). Additionally, a machine learning-based software application is developed in LabVIEW. This application automates the selection of COTS test equipment drivers, interfaces with the UUT, collects real-time test measurement data, performs analysis, generates reports and key performance indicators (KPIs), and provides recommendations. According to this paper [22], software testing is a critical activity in the software development process, with two main approaches: manual testing and automation testing. Automation testing involves writing programming scripts to automate the testing process. The choice between manual and automated testing depends on various factors, including project requirements, the project team's expertise, the technology used, and the target audience. In this research paper, a machine learning model is developed to predict the adoption of automated testing for software development projects. The study uses the chi-square test to identify correlations between different factors and employs the PART classifier to build the prediction model. The proposed model demonstrates a high accuracy rate of 93.1624%, offering valuable insights for making informed decisions regarding the suitability of automated testing in software development projects. Within the following research paper, this paper introduces an innovative automated layer defect detection system designed specifically for construction 3D printing. The system's development involves a structured approach, beginning with the creation of a deep convolutional neural network (CNN) capable of receiving image inputs and performing semantic pixel-wise segmentation to distinguish concrete layers from their surrounding environment. To enhance CNN's effectiveness, data augmentation techniques are employed, resulting in the generation of 1 million labeled images used for both training and testing purposes. Subsequently, the paper presents a defect detection module that leverages the CNN model to identify deformations within the printed concrete layers, as extracted from the images. The system's performance is rigorously evaluated using metrics such as accuracy, F1 score, and miss rate, demonstrating its commendable capabilities in effectively identifying layer defects. This research presents a valuable contribution to quality control in construction printing, offering a promising solution for detecting and addressing structural issues early in the building process.

In this research paper [16], automated software testing plays a crucial role in Agile methodologies like Scrum, where the definition of done” includes completing tests. However, As software projects progress, the number of tests can become overwhelming, slowing down deployment. This paper introduces a novel approach that leverages machine learning to prioritize automated testing, focusing on tests with a higher likelihood of failure to provide early feedback to developers. The technique involves collecting various metrics about the software under test, including cyclic values, Halstead-based values, and Chidamber-Kemerer values. Additionally, historical commit messages from the source code control system are analyzed to identify past defects in source code classes. Using this data, a file is generated containing metrics and past defect information, which is then processed using Weka to create a decision tree model. This model aids in predicting potential defects in source code files and guides the prioritization of testing efforts, ultimately improving software quality and efficiency in Agile development environments.

According to this research paper [8], machine learning plays a vital role in analyzing and deriving insights from large biomedical datasets, contributing significantly to biomedical research and healthcare improvement. However, before training a machine learning model, users typically face the complex task of selecting an appropriate algorithm and configuring hyperparameters. This process demands expertise and manual iterations, making it challenging for individuals with limited computing knowledge. To democratize machine learning for non-expert users, computer science researchers have introduced automatic methods for algorithm and hyperparameter selection in supervised machine learning problems. This paper provides a comprehensive review of these methods, highlighting their significance in the context of big biomedical data. It also acknowledges certain limitations associated with these techniques in this specific environment and offers initial insights into potential solutions.

This paper [5] gives a description of how software testing entails investigating the behavior of software systems to detect flaws. Due to the complexity and costliness of testing, a practical approach has been to automate it. There is a growing interest in utilizing machine learning (ML) to automate various aspects of software engineering, including testing. This study reviews the current state of applying ML to streamline software testing and conducts a systematic mapping analysis at the intersection of these two fields. 48 primary studies were chosen and categorized by study type, testing activity, and ML algorithm employed. The findings highlight prevalent ML algorithms and suggest potential avenues for future research. ML has primarily been utilized for generating, refining, and evaluating test cases. It has also been employed in evaluating test oracle construction and predicting testing-related costs. The results of this study provide insight into the frequently used ML algorithms for automating software testing, aiding researchers in comprehending the current research landscape. Furthermore, there is a need for more comprehensive empirical studies on the utilization of ML algorithms in automating software testing.

This article [9] states that recent progress in evolutionary test generation has improved testing for object-oriented (OO) software. This paper introduces a method that combines evolutionary testing with reinforcement learning to generate test cases for OO software, especially those with Inherited Class Hierarchies (ICH) and Non-public Methods (NPM). The approach, called EvoQ, outperforms state-of-the-art methods in branch coverage within the same time frame. This paper describes the

study that assesses room speech intelligibility using parameters like T30, Ts, EDT, D50, C50, U50, and STI. It analyzes factors including background noise, ceiling material absorption, confinement, and occupancy. Findings challenge previous research, suggesting D50 may not be a reliable metric for speech intelligibility assessment. This paper [3] is about using neural networks for the software testing process. The use of artificial neural networks as automated oracles in software testing is a topic that is introduced in this study. With test cases applied to the original software version, the neural network is trained using the backpropagation method, utilizing a "black-box" strategy where only inputs and outputs are taken into account. This well-trained network functions as an oracle to assess the accuracy of fresh, possibly flawed software versions' output. A two-layer neural network has shown potential in experiments to identify various inserted fault types in credit approval applications. This paper [27] is about an innovative tool for automated testing of GUI-driven software. The article underlines the necessity for diverse testing methodologies and explores the significance of GUI testing in contemporary software applications. A new tool named GUITAR is introduced, which stands out owing to its design, which employs plug-ins and provides flexibility and extensibility. This architecture enables programmers and quality control specialists to design unique toolchains, processes, and measurement tools for GUI testing. Through a number of case examples, the paper illustrates GUITAR's capabilities.

This paper [1] is about automated testing of classes. The reliance between an object's state and its behavior is discussed as a barrier in testing object-oriented programmes. For this situation, conventional testing techniques are insufficient. The paper provides a way to generate sequences of method invocations for testing using data flow analysis, symbolic execution, and automated deduction. Despite issues with symbolic execution and automated deduction, these methods can nonetheless produce important testing data.

2.2 YOLOv5

You Only Look Once Model 5, or YOLOv5 [28], is an actual-time object identification approach that is part of the You Only Look Once (YOLO) series. Its precise performance is derived from detecting objects in a single forward run through the neural network, which isn't the same as greater conventional -step strategies together with place proposal networks and categorization. YOLO splits the entered image into grid cells and offers each grid cell the job of estimating bounding packing containers and class probabilities for objects that can be inside its borders.

The use of anchor packing containers within the education section of YOLOv5 is a noteworthy feature that enhances the accuracy of bounding box predictions. Pre-hooked up at some stage in education, those anchor bins assist the set of rules to learn to count on bounding container coordinates more as it should be. YOLOv5 additionally does fantastically properly in class prediction, giving elegance possibilities for each bounding field. This shows that severe bounding bins, every related to an exclusive class possibility, can also perceive a single object. In addition, YOLOv5 regularly consists of characteristic pyramid networks (FPN) or associated strategies to accumulate records at numerous scales, permitting the version to perceive gadgets of different sizes. Typically, the network structure consists of a deep neural

community with a backbone network for efficient feature extraction. Deep neural networks are frequently constructed on convolutional neural networks (CNNs). The aim of the training method is to minimize the distinction between the predicted and actual ground truth bounding packing containers by optimizing the community's parameters. Operating as an open-supply mission, YOLOv5 promotes cooperation by permitting teachers and builders to use, alter, and upload to the coding, which is commonly available on GitHub. It is crucial to notice that after my closing expertise update in January 2022, YOLOv5 can also have skilled upgrades or enhancements. For the most current data, customers are urged to reference the most up-to-date guides or manuals.

2.3 YOLOv8

YOLOv8 version [29] does not exist promptly as it's still under development. Apparently, YOLOv4 changed into the maximum current generation of the YOLO series. You Only Look Once, or YOLO, is widely recognized for its capacity to recognise gadgets in real time. Notable upgrades introduced by YOLOv4 consist of the Mish activation function, PANet for function aggregation, and CSPDarknet53 as the spine structure. These enhancements have been intended to improve the model's standard efficacy and precision in object detection in pics. Since deep learning and laptop vision are fields with speedy development, it is essential to keep up with the modern day discoveries. Users are encouraged to seek advice from the official repositories, documentation, or modern-day guides for the most updated records if there were any observe-up releases, which include YOLOv8 or extra iterations in the YOLO collection. By making use of open-source collaboration, the YOLO challenge regularly allows academics and builders to each make a contribution to and take advantage of the most recent trends in object identification algorithms. It is recommended to consult reliable sources for up to date statistics about YOLOv8 or any later variations to be able to realize the functions, architecture, and upgrades added inside the maximum cutting-edge releases.

2.4 Pytorch Framework

The AI Research Lab at Facebook (FAIR) created the open-source gadget learning framework PyTorch [20]. Because it gives a dynamic and adaptable computational graph, it is thoroughly-liked by lecturers and builders. It is drastically utilized for deep learning programs. PyTorch is prominent through its outline-via-run method, which is a dynamic computational graph. This implies that, unlike static graph frameworks, the graph is built dynamically as the code is being completed, making debugging easier and expressing complex designs more certainly viable.

PyTorch is based on the idea of tensors, which are multidimensional arrays with GPU acceleration features that resemble NumPy arrays. Tensor operations routinely monitor the computational graph, permitting computerized differentiation. Tensors are the primary construction factors of PyTorch. Because the graph can be modified dynamically depending on the entered facts, this dynamic computation

graph may be very useful for building state-of-the-art and dynamic neural community topologies.

A significant array of libraries and gear, inclusive of torch. Nn for neural community construction, torch. Optim for optimisation techniques and torchvision for computer vision programs are also covered inside the framework to assist with distinct areas of deep learning. The clean integration of PyTorch with famous libraries like NumPy allows switching among these equipment. Moreover, PyTorch has a vibrant and increasing network that helps its increase by means of supplying fashions, tutorials, and different resources which will increase its attraction for deep gaining knowledge of beginners and professionals alike. Overall, PyTorch is a robust and nicely-liked framework for gadget getting to know and deep getting to know packages because of its adaptability, dynamic computational graph and strong network aid.

2.5 Computer Vision

Through the multidisciplinary nature of computer vision, machines can realize and interpret visual statistics from their environment in a way similar to that of the human visible system. Fundamentally, computer vision ambitions are to permit robots to come across, understand, and make decisions primarily based on visible input, thereby permitting them to derive significant insights from pictures and videos. From basic photo processing strategies to difficult responsibilities like object identification, photo segmentation, and facial popularity, the region covers a huge spectrum of activities.

Image processing, which entails using algorithms to alter and observe an image's pixels to be able to enhance its great or extract specific capabilities, is a basic aspect of laptop imagination and prescience. Techniques that include aspect detection, image filtering, and color change can be used for this. The location has developed to include increasingly complicated techniques as generations have advanced, along with deep studying and device mastering. With the use of those strategies, computer imaginative and prescient systems can examine and discover styles and characteristics in the visible center, paving the way for increasingly more state-of-the-art uses, which include object popularity, scene interpretation, and image categorization. One vital use of laptop imagination and prescience is item popularity, which involves recognising and categorizing matters in a picture or video. Machine learning algorithms, together with convolutional neural networks (CNNs), are often used for this process. CNNs have proven to have amazing effects on object reputation and class across a lot of visual datasets. In addition, laptop imaginative and prescient encompasses three-dimensional imaginative and prescient, which lets machines look at and recognize the 3-dimensional geometry of conditions and gadgets. This capability is critical for applications including augmented reality, robots, and independent automobiles, in which selection-making depends on having a unique hold close to the encompassing surroundings.

The area of laptop imaginative and prescient has witnessed wonderful development nowadays due to the accessibility of great datasets, strong processing abilities, and improvements in deep learning architectures. Specifically, convolutional neural networks have emerged as a key factor in cutting-edge PC imaginative and prescient systems, permitting computerized getting to know of hierarchical representations

of visible traits. Real-world programs were made possible by those trends in some regions, consisting of healthcare, retail, safety, and entertainment. The field of imaginative and prescient laptops has the capability to transform how machines view and engage with the visual world because it develops, leading to advances that will have an international impact on everyday life and many sectors.

2.6 Labelling

A graphical picture annotation device referred to as LabelImg [26] is available without spending a dime and is supposed to make the procedure of manufacturing labeled datasets for laptop imaginative and prescient model education easier. Tzutalin's LabelImg tool offers an easy-to-use interface for annotation of item bounding packing containers in snapshots. This tool is very helpful in the subject of gadget studying, in which segmentation, type, and object identity models require labeled datasets for schooling.

By permitting customers to create bounding containers around things of hobby interior and picture, LabelImg aims to streamline the annotating technique. The tool is compatible with principal deep learning frameworks like TensorFlow and PyTorch since it supports many annotation formats, along with YOLO and Pascal VOC. In order to create annotations that act as floor truth statistics for training gadget mastering fashions, users can establish instructions and give labels to gadgets.

LabelImg may be utilized by beginners and seasoned researchers alike due to its consumer-friendly layout and cross-platform interoperability. Among its features are the capability to label things with bounding containers, navigate amongst photos, and store the annotations in an organized manner. LabelImg, an essential part of the device mastering system, simplifies the process of creating annotated datasets, saving developers and researchers a lot of time while producing training data for laptop imaginative and prescient applications.

Chapter 3

Model & Dataset

3.1 Dataset description

3.1 Dataset description A research team produced the dataset that we utilised. We used a dataset created by the RICO data-driven design group at the University of Illinois for our research. The RICO dataset contains 66,261 unique screenshots of the user interfaces of 9,772 Android apps.

There were several categories that were omitted, such as media players and photo editors. These 9,772 Android applications have an average rating of 4.1 stars from the Google Play Store.

In order to compile this dataset, RICO downloaded 9,722 applications from the Google Play Store. Thirteen UpWork employees were employed to take screenshots. These workers were instructed to capture screenshots of every screen that each programme offered and to limit their time on any given programme to no more than 10 minutes.

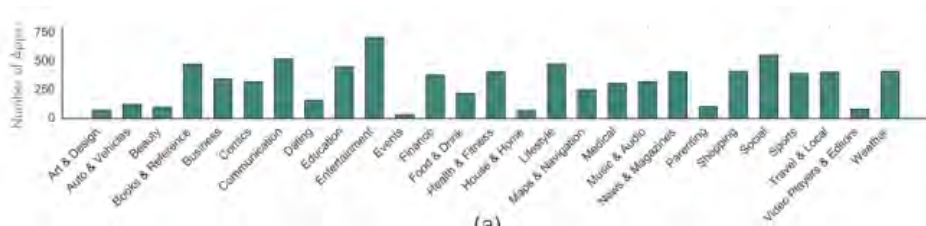


Figure 3.1: Figure: Summary of RICO dataset

We found that when we utilized a dataset greater than 2000 images, we were obtaining the error "data-loader exit unexpectedly" during training. Our system could not train this massive dataset given by RICO, so we had to reduce the quantity of data utilized for both model verification and training. We were limited to using 2000 screenshots from the RICO dataset, which contained some composite and core Android components, due to this hardware constraint. Furthermore, we found that, on the hardware we had available, a dataset of 1,700–1,800 pictures yielded consistent and optimal results for training and testing the model. As a result, our thesis study employed 2000 images.

3.2 Data Preprocessing

3.2.1 Data Preprocessing

We had to first label each core and composite component using the Labellmg programme available on GitHub before we could utilize our dataset for training. Tzutalin created Identifying Image, a well-known photo annotation programme, with further help from other volunteers. The Labellmg package is no longer being maintained, but this flexible image labeling tool has joined the LabelStudio community. We created a box with Labellmg for each part we need for our system's inquiry. The system stored a text file with the serial numbers of each component once the parts were labelled. Including the component name that we gave and the four coordinate values x (max, min) and y (maxx, min) that indicated the box that was placed on the component so that our model could train. An example is given below,

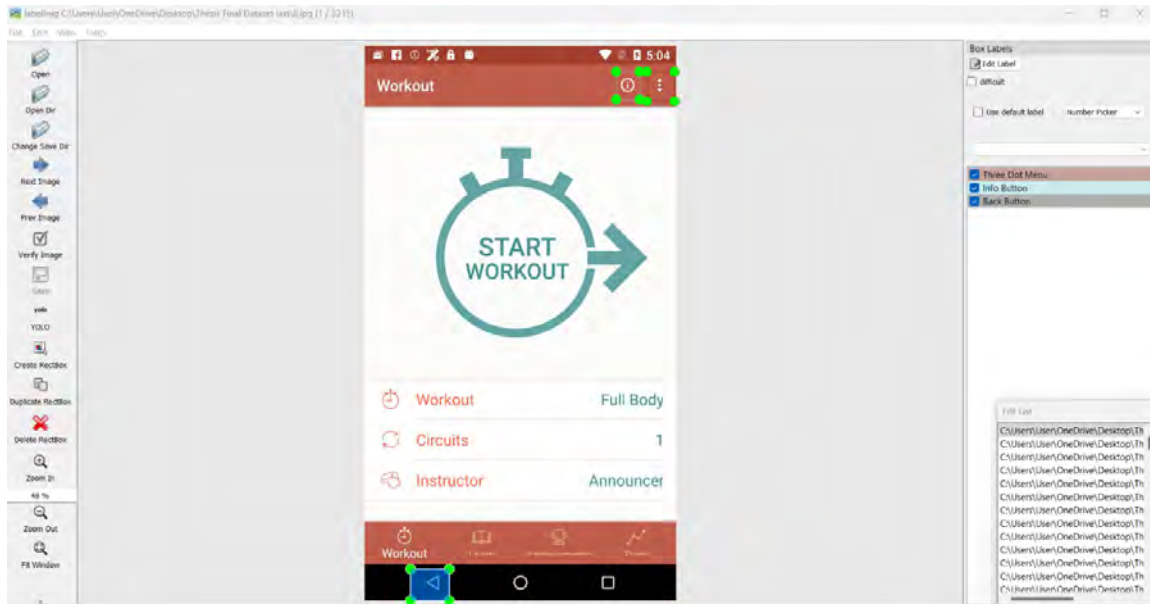


Figure 3.2: Figure : Labeling images using Labellmg package

We obtain a text file once the labels are added to the image above. Below is an

example of the text file. In order to train and validate our model, we have labeled all core and composite components observed on the screenshots after scanning through all 1100 photos. Below is the data's visual representation following image labeling.

File	Edit	View
6	0.950000	0.073438 0.088889 0.053125
5	0.844907	0.071875 0.075000 0.050000
1	0.209259	0.971615 0.124074 0.056771

Figure 3.3: Figure: Data after Image Labeling



Figure 3.4: Figure: Visual Representation how labeling works

3.2.2 Feature selection

As for the core and some composite components that we have taken into consideration were number picker, back button, settings button, add button, share button, info button, three dot menu button, switch button, cross button, three bar menu button, reload button, search button and check button, Login Button, Signup button

and Notification button. After labeling each of the images for our training and validating our model, we get the following numbers of core and composite components shown in the table below,

Table 3.1: Figure: Features

Name of the Component	Numbers of component detected
Number Picker	32
Back Button	1641
Settings Button	112
Add Button	137
Share Button	103
Info Button	142
Three Dot Menu	386
Switch Button	76
Cross Button	204
Three Bar Menu	413
Reload Button	51
Search Button	233
Check Box	516
Login Button	76
Signup Button	93
Notification Button	103

3.3 Model description

Yolov5 is a real-time, single-stage object identification model that uses a single convolutional neural network (CNN) architecture to categorize cases and identify objects in a single forward pass network. It entails taking a picture as input and creating an output case that gives the bounding boxes of each component as well as the class probabilities of each component that can be identified independently in the image. This raises the efficiency of the model. The employment of cross-scale feature pyramids, strong backbone networks, and anchor boxes contributes to its accuracy, which is also rather excellent. There are three primary components to the YOLO network. We refer to these three major components as the head, neck, and backbone. CNN helps maintain the backbone's functionality. A range of pixelated pictures are crammed into the backbone of the system in order to train it. The photos are combined and mixed up by the neck. Subsequently, the combined photos are transmitted forward, enabling the detection of the target. While the head functions as a head and makes predictions in the same

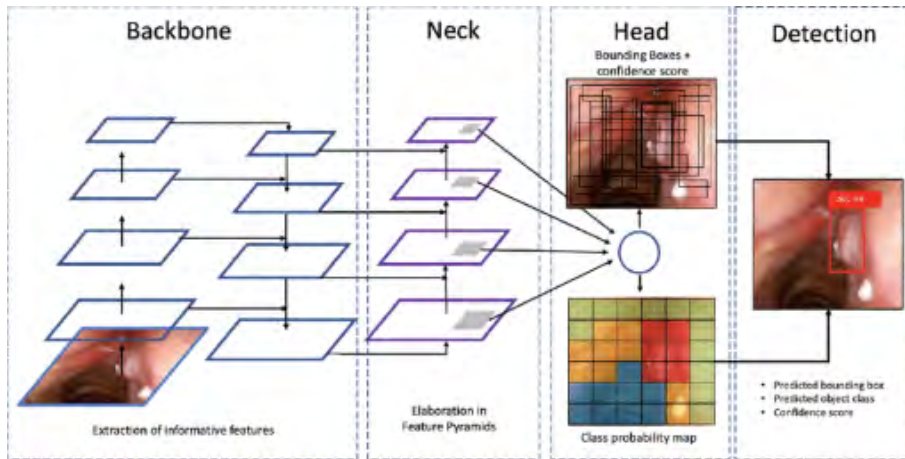
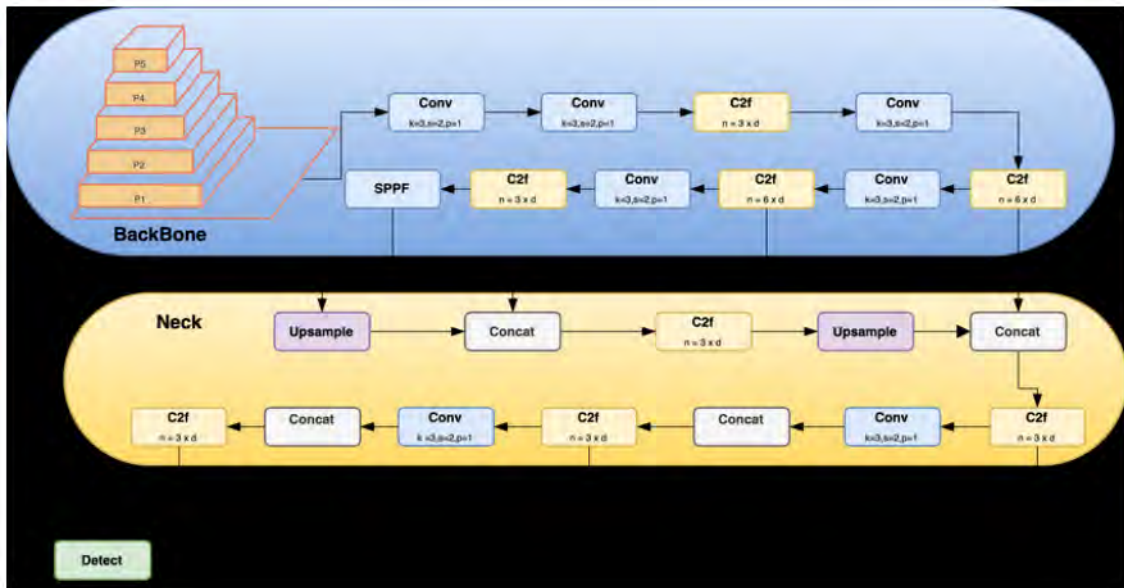


Figure 3.5: Figure: YOLOV5 architecture representation

The most recent and advanced YOLO model, YOLOv8, is applicable to tasks like instance segmentation, object detection, and image classification. The company Ultralytics, which also developed the well-known and industry-defining YOLOv5 model, is the creator of YOLOv8. Many architectural and developer enhancements and modifications over YOLOv5 are included in YOLOv8. YOLOv8 has a high rate of accuracy, as measured by Microsoft COCO and Roboflow 100. YOLOv8 comes with a lot of developer-convenience features, from an easy-to-use CLI to a well-structured Python package. There is a large community around YOLO and a growing community around the YOLOv8 model, meaning there are many people in computer vision circles who may be able to assist you when you need guidance.



YOLOv8 architecture.

Figure 3.6: Figure: YOLOV8 architecture

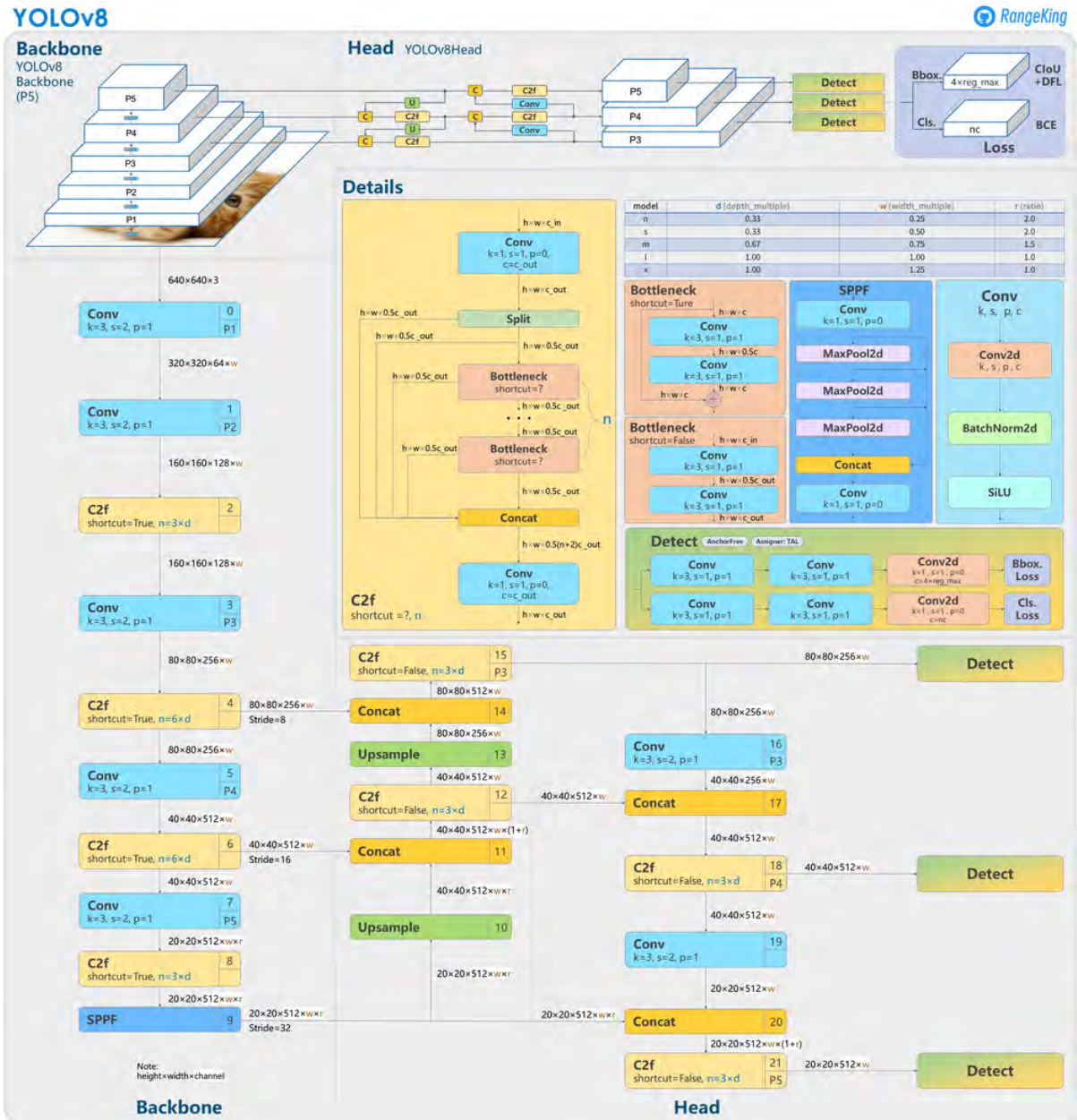


Figure 3.7: Figure: Figure: YOLOV8 model architecture representation

3.3.1 Single Stage Object Detector

Yolov5 employs a single CNN Network—which consists of the Backbone, Neck, and Head—during a single forward pass for the purposes of both object detection . Backbone suggests a pretrained network that extracts the salient features from an image and decreases the spatial resolution while raising the feature resolution. Model necks with pyramids are used to accurately generalize to varying component sizes and scales and obtain the desired feature. Lastly, the model head is the most crucial component that improves accuracy by applying bounding boxes with certainty, indexing the component, and carrying out the last stage action. Yolov5’s head and neck use PANet (Path Aggregation Network) and SPP (Spatial Pyramid Pooling), while the Darknet53 Convolutional network, also known as CSPDarknet53, provides

the backbone utilizing the CSP (Cross Stage Partial) technique.

3.3.2 Other important parts of an improved result

YOLOv5 employs the sigmoid function as the activation function and the sigmoid linear unit (Silu). The output layer uses the sigmoid function, whereas the hidden layer uses Silu for convolution operations. For the loss functions for classes, objects, and locations, respectively, BCE (Binary Cross Entropy) and CIoU (Complete Intersection Over Union) are utilized. Additionally, a focus layer is employed to increase speed by swapping out the first three layers and lowering settings. This makes certain modifications to mAP (mean average precision). Because the grid sensitivity has been removed, the model can now detect components even when they are at the edge, something that the previous Yolo version found difficult to do.

Chapter 4

Implementation & Result Analysis

4.1 Implementation

The most integral part of the thesis was to make a model that is well equipped to detect functional items from the UI of Android applications. In order to make such a model, there is a discussion of the required hardware specification, environment setup, required packages, custom model configuration, and more, which collectively helped us achieve success with the model and finally give results that are free of any errors. In order to achieve accuracy across different user interfaces, our implementation process is the one that took us a long way without any errors or bugs.

4.1.1 Hardware Specification

In terms of developing the system, the hardware specification played an important part in training the model on the labeled dataset. It made sure that the system has everything it needs to handle the computation, along with other functionalities, and ensure deliverability. The central processing unit plays a vital role in efficiently handling general computational duties and coordinating various operations throughout the system. A powerful processor is critical for taking on numerous processing-intensive tasks simultaneously and keeping the workflow moving smoothly. So, the machine is powered by an AMD Ryzen 5 5500U processor that has 6 cores underneath. A dedicated GPU also ensures speeding up the training process on numerous epochs and other complex scenarios that require speeding up the processes. Therefore, the machine that we are using has an Nvidia Geforce 1050TI with 4GB of ram. The operating system has Windows 11 installed and uses a 512GB M.2 SSD. Sufficient RAM is crucial for processing substantial datasets and meeting the considerable memory needs of sophisticated deep learning models. While the datasets and models were sizable, having enough RAM enabled efficient handling of large amounts of data and complex neural networks. This is important for training models on vast troves of information and implementing intricate architecture capable of discerning intricate patterns. Thus, the RAM of the overall system was 32GB of DDR4 at 2400 MHz. But, alternatively, the model can also do well enough in Google Colab, running on a Python Google Compute Engine with a RAM of 12.67 GB and a disk space of 78.19GB.

4.1.2 Environment Setup

In order to configure the model and get the desired outputs, we need to set up a proper environment with all the prerequisites to ensure deliverability across the lifetime of the model being active. In order to do so, we used a Python environment of 3.9 in a basic Python environment. The terminal was used often to install the required packages. Some alternative approaches that also worked were Google Colab, which has a separate runtime with Python version 3.10.12. Python played a pivotal role in the technical environment as it served as the foundational language for data analysis and machine learning applications. Given its vast library of modules for tasks like data wrangling, visualization, and predictive modeling, in addition to its large international community of users, Python was the natural selection as the main technology stack. Its wide range of libraries and functionality allowed for extensive experimentation and custom solutions to be developed. Further, with Python's popularity among professionals in adjacent fields like statistics and research, it ensured that solutions and results would be understood by relevant stakeholders.

4.1.3 Package Installation

A critical aspect of setting up the environment for the project involved the installation of various Python packages. These packages provided the necessary tools and functionalities required for the development and execution of our machine learning model. Below is a detailed overview of the key packages installed and their roles in the project:

YOLO Dependencies

The YOLOv5 and YOLOv8 dependencies are essential for utilizing the YOLOv5 framework, which is central to our object detection tasks. Installation: These dependencies were installed as part of the YOLOv5 and YOLOv8 repository setups.

Roboflow

Roboflow is a tool for managing and processing machine learning datasets. It was used for handling the dataset involved in training our model, providing functionalities like annotation, normalization, and augmentation.

PyTorch

PyTorch is a fundamental library for building and training deep learning models. In our project, PyTorch is used to construct and train the custom model for UI item detection.

Requests and Python-utils

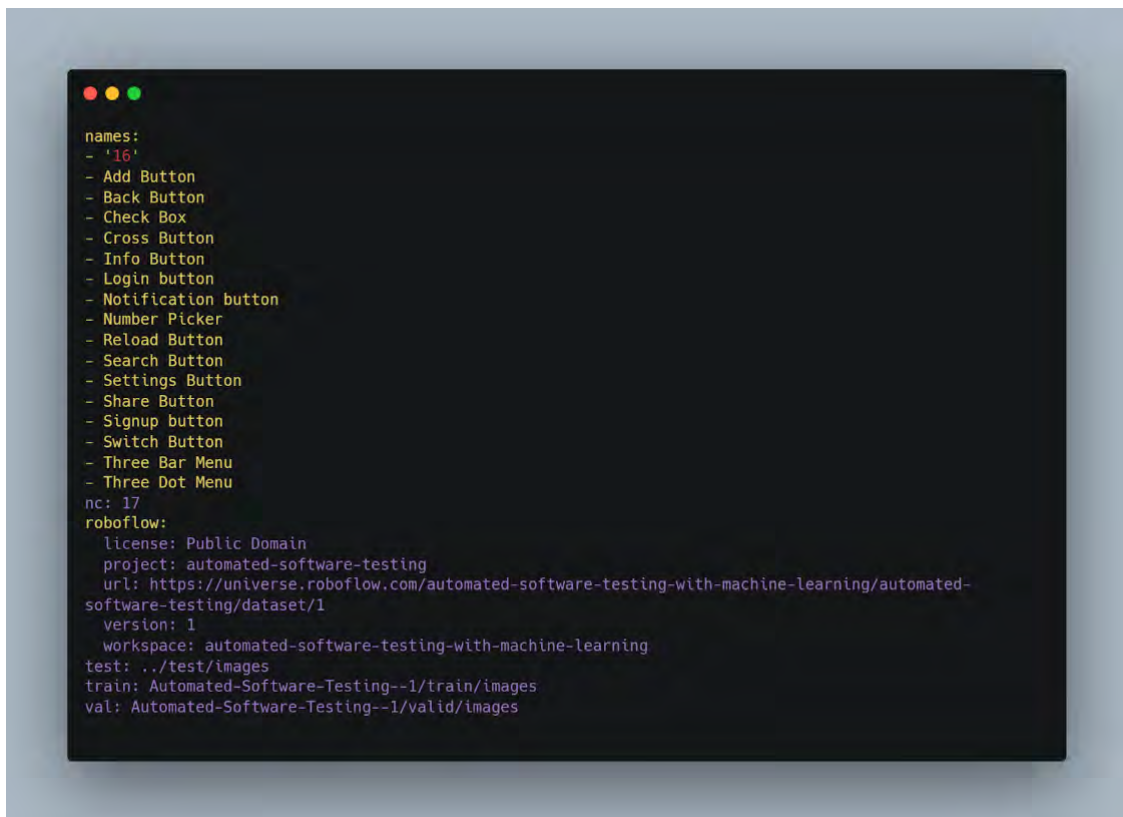
The requests library is used for making HTTP requests, which is crucial for fetching data or interacting with web services. The python-utils package provides a range of utility functions, enhancing the versatility of our Python scripts.

4.1.4 Custom Model Configuration

As we opt to use YOLO models, specifically the YOLOV5 and YOLOV8 models, we are one step ahead of the traditional approach of making something from scratch. But, yet, these models were fine tuned to our selective dataset of functional UI screenshots of Android applications. As mentioned earlier, we have opted to select 16 distinctive features for the detection of functional ui elements in applications. In order to get the desired outputs, we opted to make different modifications to the model and tailor it to our specific needs.

YAML File Configuration

The custom YAML file serves as the central blueprint by defining various parameters and metadata that are important for training the model.



```
names:
- '16'
- Add Button
- Back Button
- Check Box
- Cross Button
- Info Button
- Login button
- Notification button
- Number Picker
- Reload Button
- Search Button
- Settings Button
- Share Button
- Signup button
- Switch Button
- Three Bar Menu
- Three Dot Menu
nc: 17
roboflow:
  license: Public Domain
  project: automated-software-testing
  url: https://universe.roboflow.com/automated-software-testing-with-machine-learning/automated-software-testing/dataset/1
  version: 1
  workspace: automated-software-testing-with-machine-learning
test: ../test/images
train: Automated-Software-Testing--1/train/images
val: Automated-Software-Testing--1/valid/images
```

Figure 4.1: YOLOV8 model architecture representation

The names section defines the class names and their corresponding identifiers. Each class represents a specific type of UI item that our model aims to detect. The list

under names specifies the labels for UI elements, like "Add Button," "Back Button," etc. The nc (number of classes) is set to 17, indicating that our model is trained to recognize 17 distinct types of UI items. The license indicates the licensing of the dataset (Public Domain in this case). The YAML file also specifies the project details within Roboflow. Finally, the directory contains training images, validation images, and test images, which are used to train, validate, and evaluate the model.

Training Custom Model

In order to make our desired model, we opted for fine tuning two versions of a preexisting model named YOLO. We are using YoloV5 and YoloV8 as our backbones for the model. But while fine tuning we configured the required configuration files and our custom dataset of Android application screenshots. Our preprocessed data, labeled with the help of LabelImg, has played a pivotal role in integrating the dataset with YOLO models. YOLOv5 is comparatively easier to setup and use. It's also evident that YOLOv5 is easier to deploy. And moving on, YOLOv8 is faster and more accurate than YOLOv5. The claims are based on proper research by the organizations responsible for building YOLO models. But the simplified comparative analysis is yet to be encountered as we move forward with the model and its outputs.

Training on Custom Dataset Configuration

The custom model has been developed with 500 epochs and a batch size 16. The batch size 16 is responsible for keeping a proper balance between the memory usage of the GPU and the overall training speed. Moreover, to avoid any sort of overfitting across the lifespan of training, early stopping has been implemented. The image size of the input images are set to 640 pixels, which is well accepted as a decent image size across all YOLO models. The cache flag is also used to speedify the process of training epochs by caching the dataset and ensuring efficiency.

4.1.5 Automation System

In order to have a proper automated flow for such use cases, it's mandatory to preload the custom model that we have trained on our dataset before taking a screenshot. A proper image cropping mechanism system has to be implemented that gets rid of all the unnecessary parts of the screenshots. Our proposed automation system can be divided into three main components - detection mechanism that we have developed using YOLOv5/YOLOv8. After that, the system moves on to the component that's responsible for having a proper real time connection between the Android application and the detection system. And then, based on the detection, we move on to the last part of the architecture, which is the UI elements testing system. Here, the screenshot of the application is tested to match the screenshot of the design file for that specific screen. And based on that, the end user gets the status of misplaced UI elements or successful placement of UI elements. The automation system is thus completed to its core functionalities without any hassle

or difficulties.

4.2 Result Analysis

The obtained results from the trained models are to be analyzed over certain metrics, and thus we can decide on the better performing version to take an effective approach.

4.2.1 Comparative Analysis

Moving on, the YOLOv5 model was chosen simply because it's one of the simplest ones of all the YOLO models and it's easy to set up and train comparatively. On the contrary, the YOLOv8 model was chosen as an option because of its speed and accuracy over real time visuals. Though app screenshots are not real time moving visuals, we have still found these existing reasons enough to include them in our research and thus find the one that fits perfectly with the purpose of the use case.

Confusion Matrix

As the backbone of the analysis, we are deciding on the confusion matrix for now as an integral metric. Often, for classification models, a confusion matrix is the one that shows the quantity of correct as well as incorrect predictions made by that model.

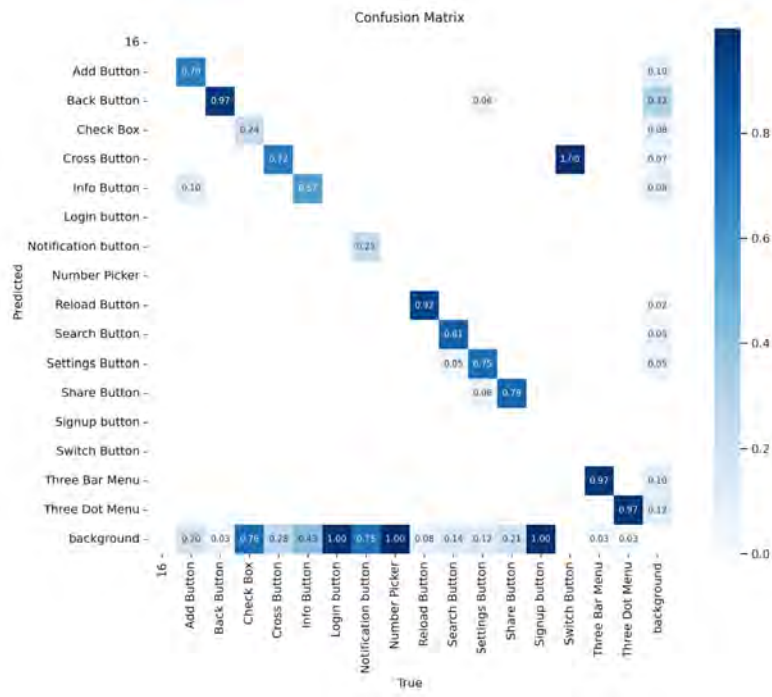


Figure 4.2: YOLOv5 Confusion Matrix

The confusion matrix for YOLOv5 displays strong diagonal values for classes like “Back Button”, “Number Picker,” and “Search Button,” which indicates that the model is classifying or detecting these classes well enough. However, it’s easy to see that there are high caliber confusions between classes like “Add Button” and “Info Button”. Because a significant number of “Add Buttons” instances were misclassified as “Info Button,”.

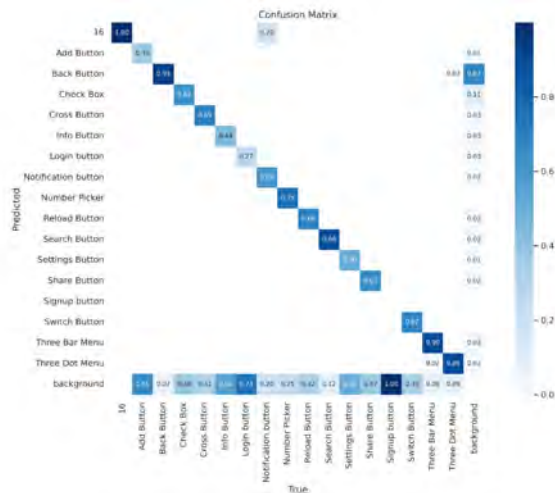


Figure 4.3: YOLOv8 Confusion Matrix

In contrast, the YOLOv8 confusion matrix reveals an improvement in distinguishing between the “Add Button” and “Info Button” classes. Moreover, this model also shows a near-perfect classification for the “Back Button” but exhibits some confu-

sion between "Switch Button" and "Three Bar Menu," which was not as prevalent in the YOLOv5 matrix.

Both models are able to demonstrate high accuracy for certain classes but YOLOv8 shows an overall improvement in detecting accurately across most UI elements. YOLOv5 struggles to differentiate similar types of elements, but YOLOv8 is more effective in understanding the nuanced differences between those classes. The instances of misclassification rates are more evident in terms of YOLOv5 but YOLOv8 wins this comparison metric as well. And collectively, both models face difficulties with classes that have lesser training examples which is obvious.

F1 Curve

The F1 curve is one of the most driving factors towards judging the performance of a model. It's a harmonic mean of the existing model by taking in account its precision and recall. The term Precision is a measure of quality and recall is a measure of quantity. So, it's evident and easy to understand that both of these measuring metrics really give a meaningful collective metric to judge a model well enough. Now, we have obtained F1 curves for YOLOv5 and YOLOv8 models for the specific dataset and selective features or classes.

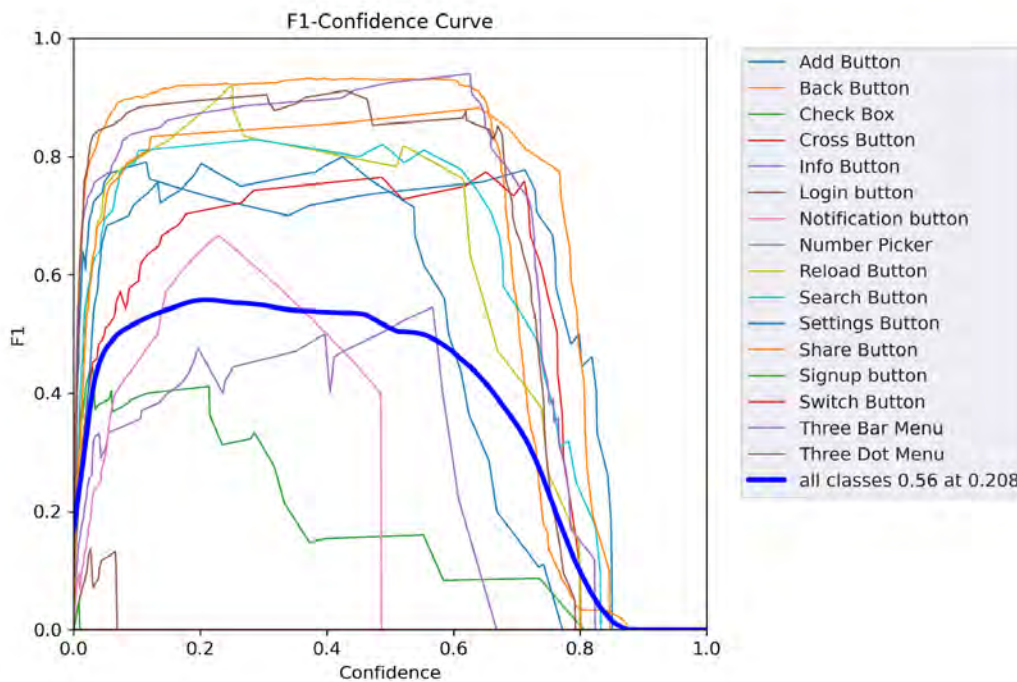


Figure 4.4: F1 Confidence Curve of YOLOv5

The F1-Confidence curve for YOLOv5 displays a high F1 score for certain classes such as the "Back Button" and "Search Button" across a range of confidence thresholds, showcasing the model's reliability in detecting these elements. However, the curve for "Info Button" and "Number Picker" indicates a drop in F1 score as confidence increases, suggesting a trade-off between confidence and accuracy for these classes. The overall F1 score across all classes peaks at a specific threshold, beyond

which the score declines, indicating a sweet spot for the model's confidence level.

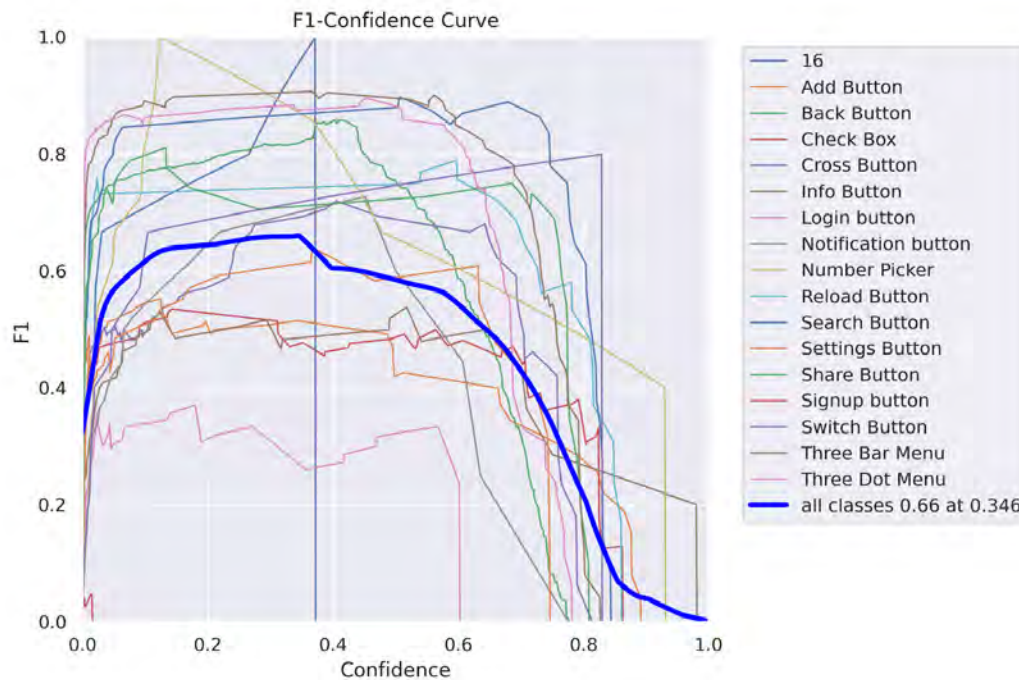


Figure 4.5: F1 Confidence Curve of YOLOv8

In contrast, YOLOv8 shows an improved F1-confidence profile with a generally higher curve across most classes. This indicates a better balance between precision and recall, even at higher confidence levels. Notably, classes like "Switch Button" and "Three Dot Menu" maintain a high F1 score over a wider range of confidence thresholds, showing robustness in detection. The aggregate F1 score across all classes for YOLOv8 is higher than that of YOLOv5, and the optimal confidence threshold is also at a higher value, suggesting increased overall confidence in detections.

Now, it's easy to comprehend that YOLOv8 surpasses YOLOv5 in the overall F1 score, indicating superior aggregate performance in detecting UI elements. Moreover, YOLOv8 demonstrates more consistent F1 scores across different UI elements, especially at higher confidence thresholds. The analysis also shows that YOLOv8 maintains a higher F1 score at elevated confidence levels, suggesting that it can operate with higher certainty without compromising accuracy. While both models perform well on common UI elements, YOLOv8 shows less performance degradation on less common or more challenging elements.

P Curve

Now, precision is a critical metric in object detection which can easily measure the accuracy of the predictions where the model is confident enough. The Precision-Confidence curve is an essential diagnostic tool as it illustrates the trade-off between precision and the confidence threshold for a given model.

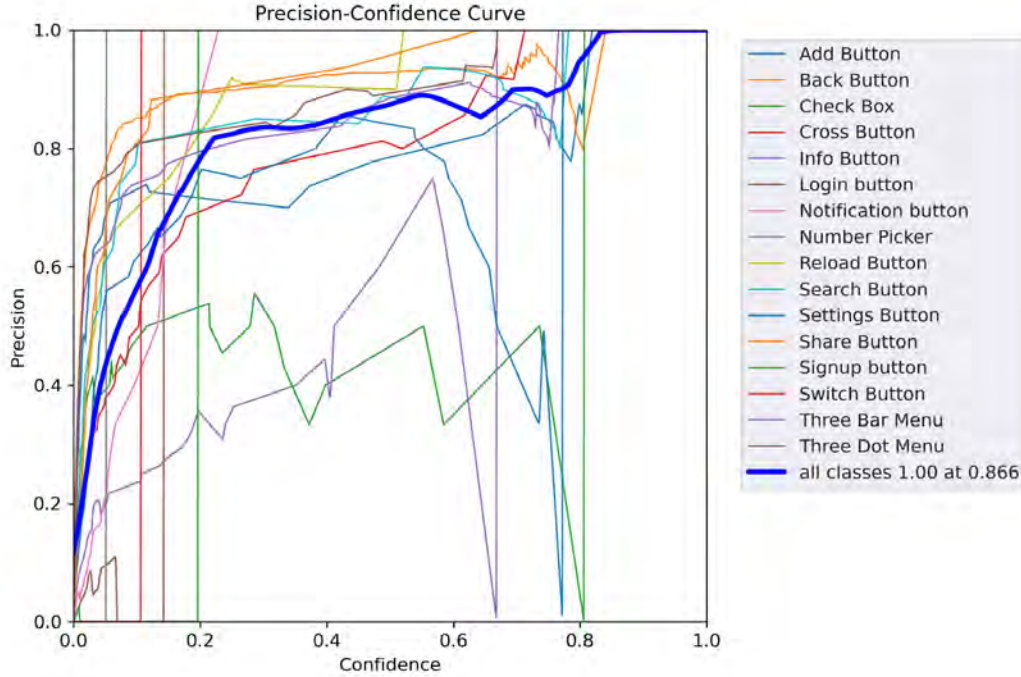


Figure 4.6: P Curve of YOLOv5

The YOLOv5 model demonstrates high precision at lower confidence thresholds for some of the classes, which indicates a strong ability to correctly identify UI elements with less certainty. However, as soon as the confidence threshold increases, we see a decline in precision for several classes, such as "CheckBox" and "Info Button," suggesting that the model becomes less accurate when it is more certain about its predictions. Moreover, the "Add Button" and "Back Button" maintain a relatively high precision across a wide range of confidence levels, indicating consistent performance for these classes.

On the contrary, the P curve maintains a higher precision across nearly all confidence levels for most classes. This clearly suggests that YOLOv8 is more accurate in its predictions and more reliable when it asserts high confidence. It's important to note that the classes "Search Button" and "Settings Button" show superior precision at high confidence levels compared to YOLOv5. The overall precision for all classes combined is also higher for YOLOv8, indicating a better performance across the board.

It's a clear win for YOLOv8 because it is seen that both models start with high precision at low confidence thresholds, but YOLOv8 maintains higher precision as the confidence increases. Each class has a unique precision-confidence profile and for sure YOLOv8 outperforms YOLOv5, especially in classes where there are possibilities of different variations. YOLOv8 exhibits a higher overall precision than YOLOv5, especially at the mid-to-high confidence ranges, which are crucial for practical applications.

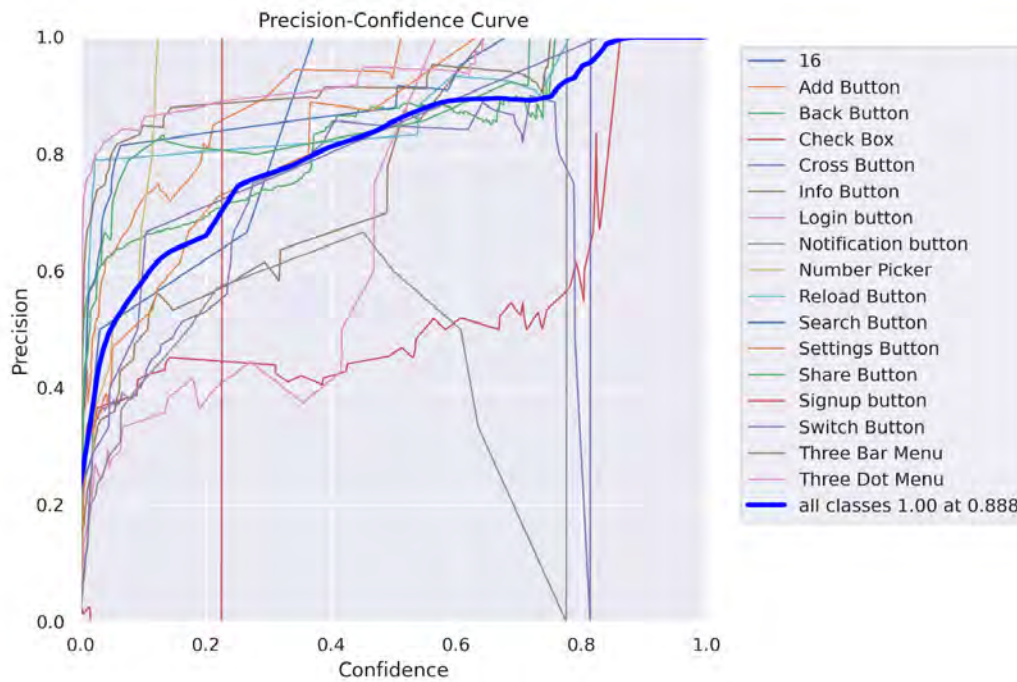


Figure 4.7: P Curve of YOLOv5

PR Curve

We use the Precision-Recall curve to gauge how well a model works. It's especially important when your data isn't balanced. It shows the balance between precision (how accurate your model is) and recall (if your model can detect everything it needs to). Let's look at how the YOLOv5 and YOLOv8 models perform when tasked with spotting UI elements in Android apps.

YOLOv5's curve isn't always consistent. Take the "Add Button" category for ex-

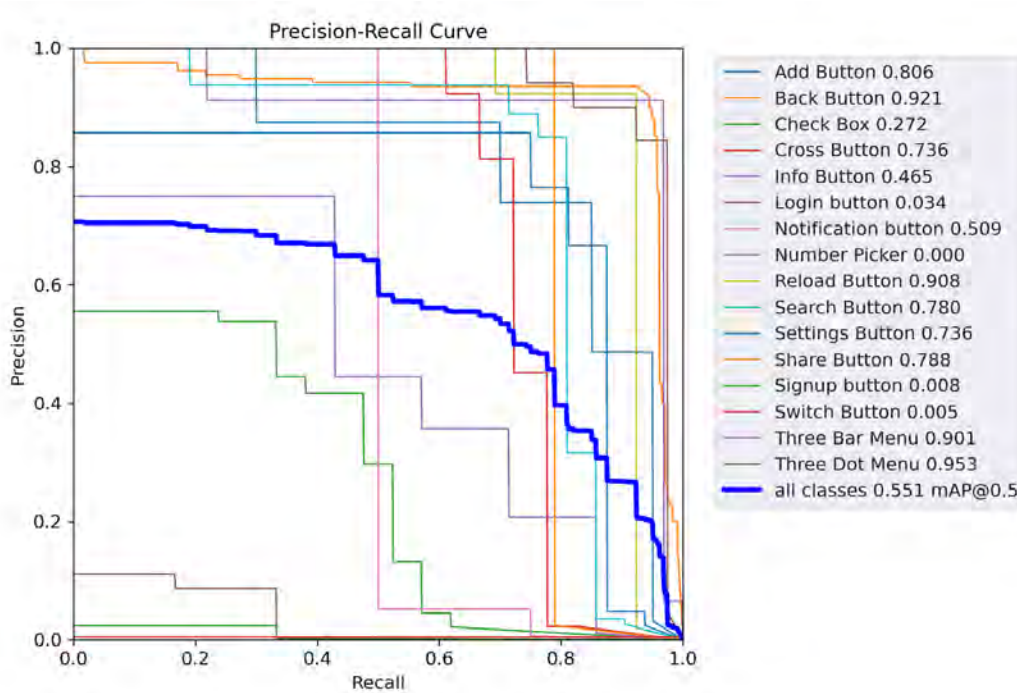


Figure 4.8: PR Curve of YOLOv5

ample. It's precise but not so good on the recall side, meaning it misses some things. Other categories, like "Reload Button" and "Three Bar Menu," score high on both, telling us YOLOv5 does well with those. Overall, its mean Average Precision (mAP) at a 0.5 IoU threshold is 0.551, giving a general idea of how it performs. YOLOv8,

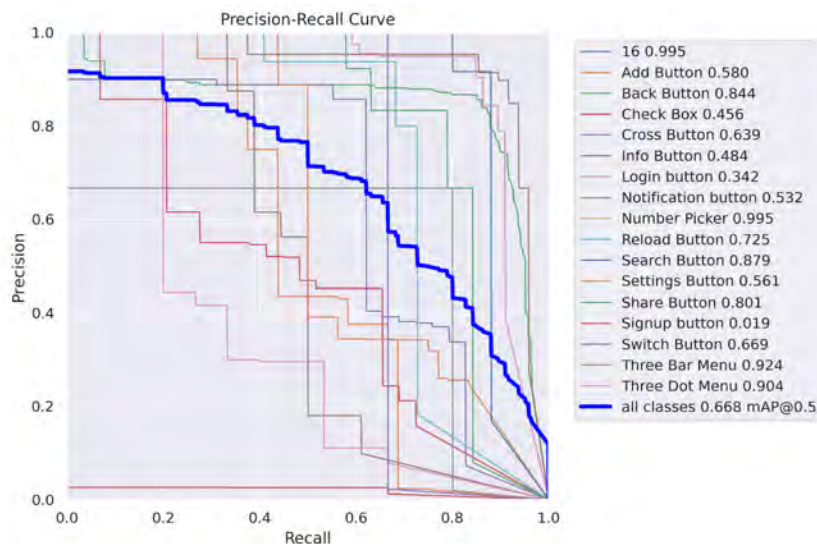


Figure 4.9: PR Curve of YOLOv8

on the other hand, tends to do better. It has high levels of precision and recall in most categories. "Number Picker" and "Three Dot Menu" perform exceptionally well. With YOLOv8, precision and recall levels are usually better balanced, leading to superior performance. Plus, the mAP for YOLOv8 is 0.668, which shows it outperforms YOLOv5.

YOLOv8 has a good balance between precision and recall. This means it has fewer false positives and negatives. Again, YOLOv8 shows consistent performance across different classes. This suggests it can handle a variety of UI elements. With a higher mAP score, YOLOv8 proves to be a reliable model for spotting UI elements in Android apps.

R Curve

Basically, recall is how well a model can spot the right instances of a class. We can see how good this is by looking at the Recall-Confidence curve. Right now, we're comparing YOLOv5 and YOLOv8. We're looking at how it picks up on UI elements.

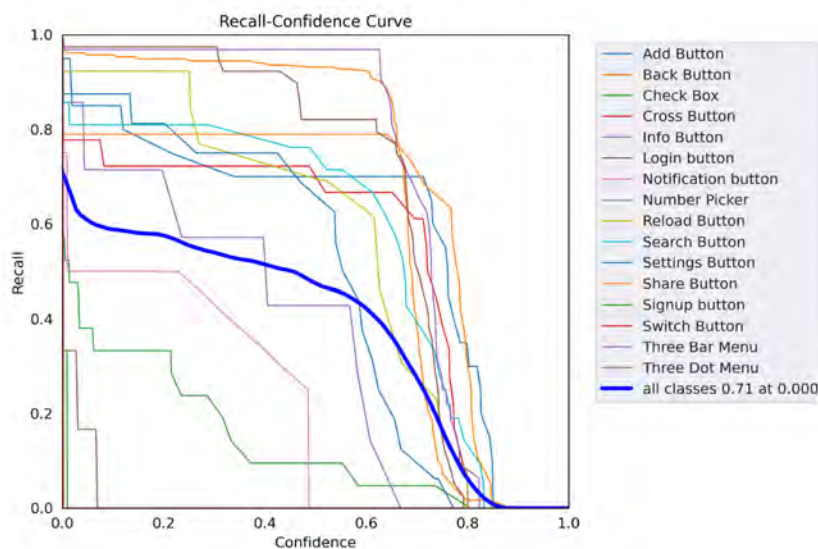


Figure 4.10: R Curve of YOLOv5

With YOLOv5, the "Back Button" and "Reload Button" classes do well, even when there's not a lot of confidence. However, "Login button" and "Signup button" don't do as well when the confidence goes up. The total recall starts to go down when the confidence gets really high.

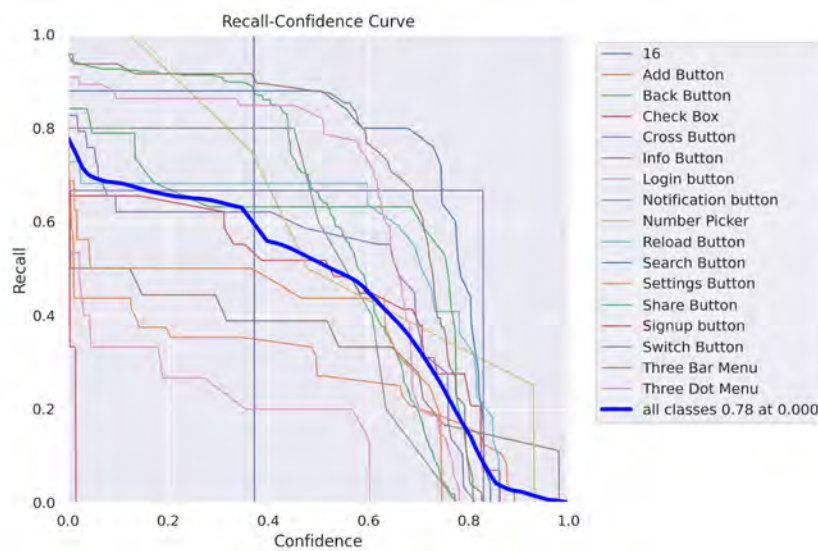


Figure 4.11: R Curve of YOLOv8

YOLOv8 does things a bit differently. It does a good job with most of the classes even when the confidence scores get higher. "Three Dot Menu" and "Number Picker" classes stay strong at any confidence level. Overall, YOLOv8 is better at spotting positive instances.

Both models have high recall at the beginning, showing they can detect well. As the bar is raised, YOLOv8's recall decreases slowly. YOLOv5, however, falls more quickly.

YOLOv8 keeps a high reputation for all classes. This hints that it's less likely to overlook positive items. YOLOv8 stands out when recalling most classes, providing a dependable model for spotting a variety of UI elements.

4.2.2 Selected Model Result Analysis

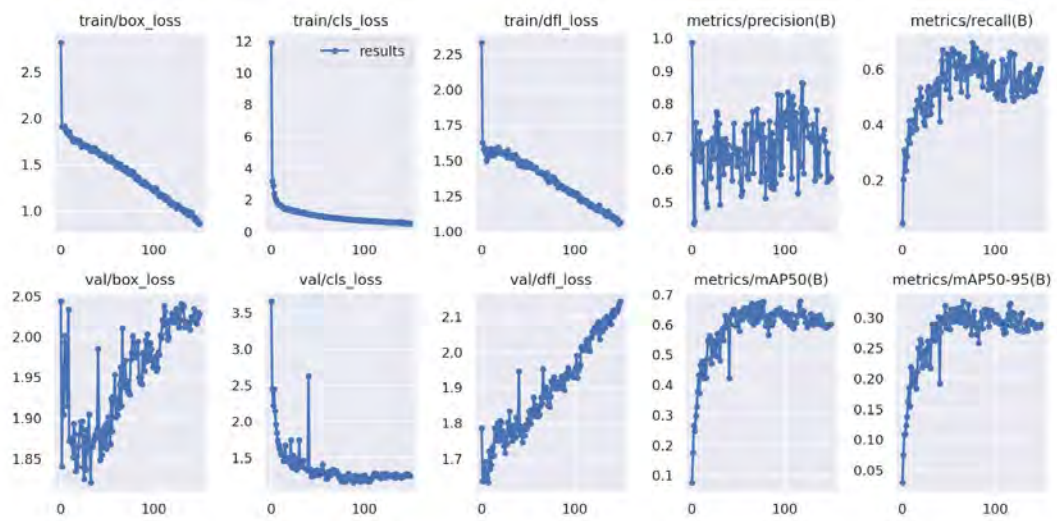


Figure 4.12: Overall Results of YOLOv8

Therefore, based on all of the important metrics and observations, we can surely come to the conclusion that YOLOv8 is the clear winner in terms of performance, efficiency, speed, and accuracy with respect to time, data size, and any other possible variable for the given use case.

4.2.3 System Analysis & Overview

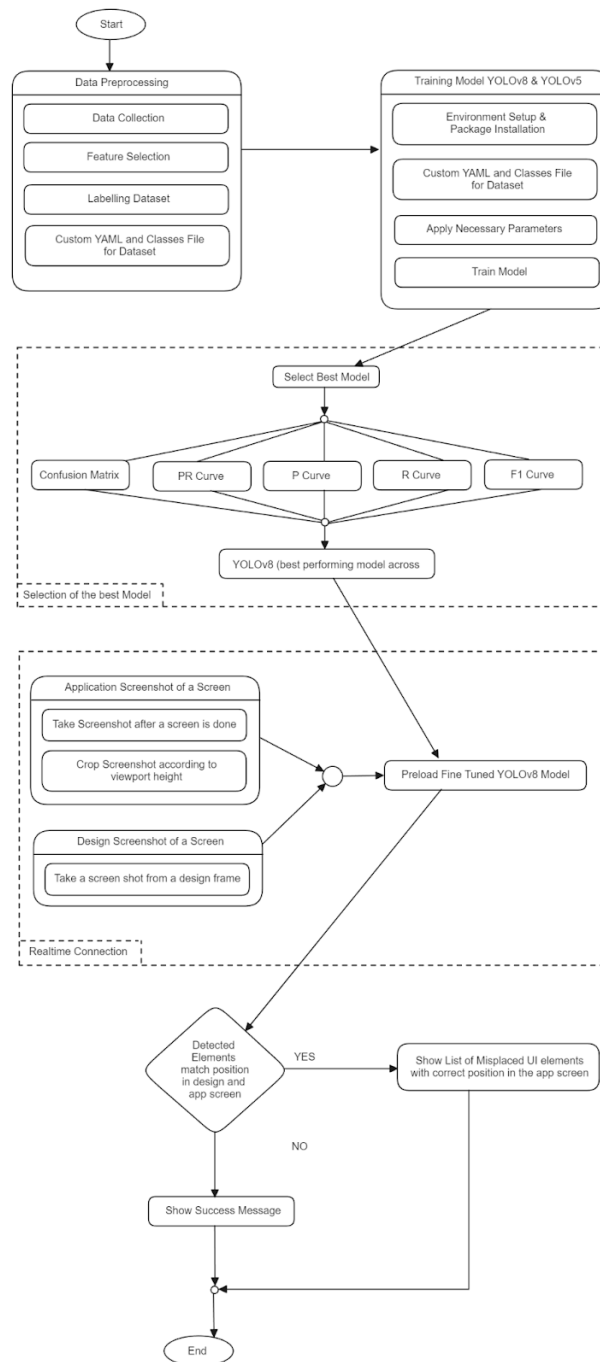


Figure 4.13: System Overview

The automated tool is advanced yet easy to use. It finds and examines UI elements. Different parts work together in this tool. Think of it like a combo of machine learning and software testing needs. This helps validate the design and use of user interfaces in Android apps. We took a lot of care to make the architecture of this tool. It runs smoothly, gets results right, and provides fast feedback.

We're bonding YOLOv5 and YOLOv8 to smooth out UI tests in Android apps. First, we prep data—gather it, label it, and set it up for the models to learn from. Then, we train. Models get fine-tuned, and the one doing best based on precision and recall measurements is chosen. There's a live link between the app and system for instant spotting and contrasting of UI bits with design files. It wraps up with an automated UI testing part that gives immediate feedback. It flags errors and okays correct placements in UI elements. All of these parts work together to make a strong system that ups the speed and correctness of checking UI consistency. This is a valuable tool for software testers and those developing the visual front-end. First of all the necessary UI elements are detected from two images - application screenshot and design screenshot with the help of the fine tuned YOLOv8 model.

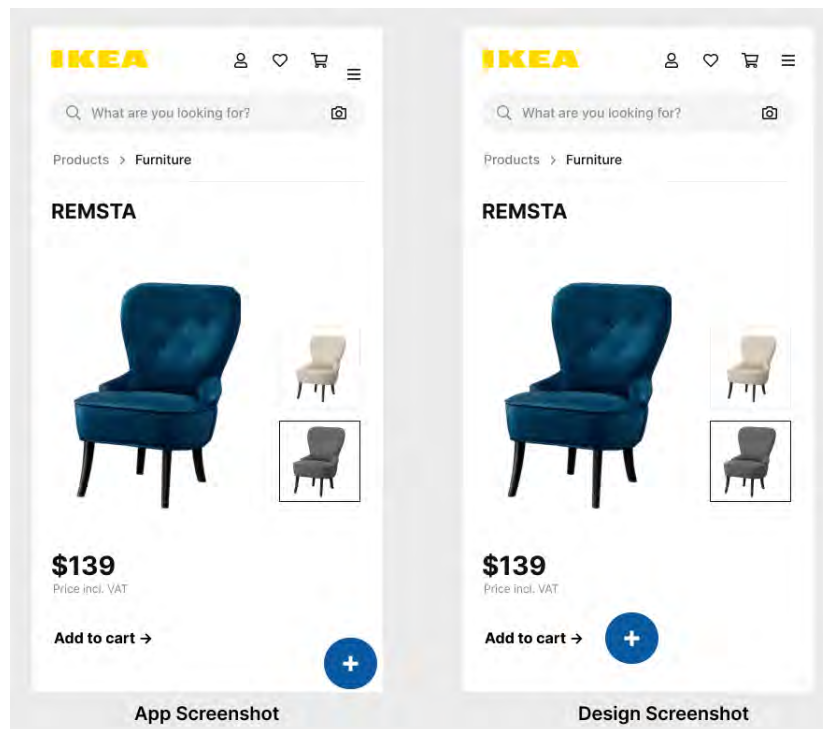


Figure 4.14: Application Screenshot and Design Screenshot Respectively

The end user inputs two screenshots, and now the ai model tries its best to detect UI elements. The classes or features detected from both are “Three Bar Menu” and “Add Button”

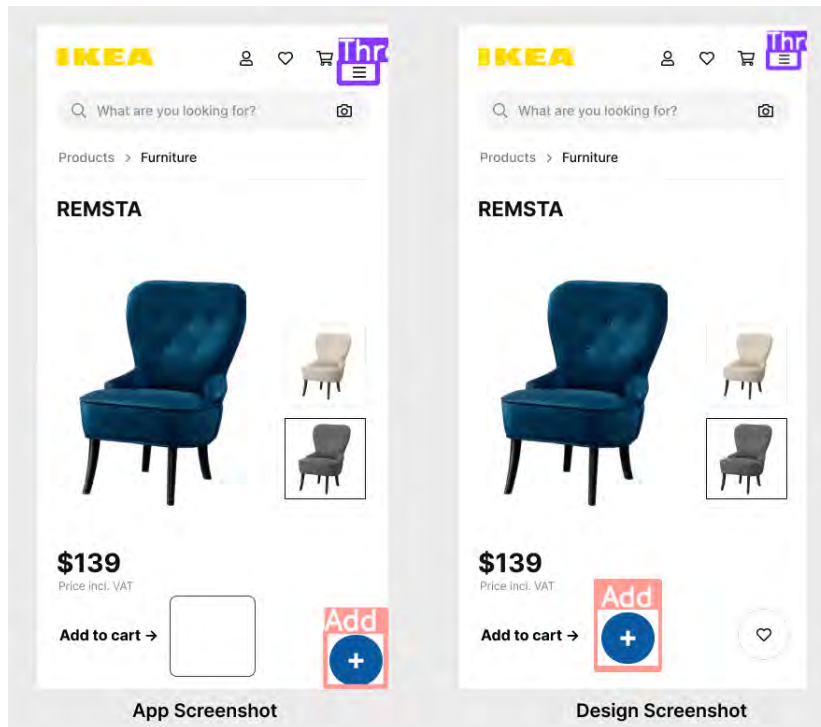


Figure 4.15: UI Elements Detected.

Now that we have detected the elements, we can focus on testing the frontend of the Android application with the design. And then, look for any potential misplacements of any of the detected objects in terms of design screenshot.

We can clearly see and come to the conclusion that this system is following a considerable good approach towards bringing UI development and UI functional item detection a huge boost. First, the system checks if the detected elements of the design screenshot are absent in the application screenshot. If even one of those elements is missing in the application, then an error is given to the end user about that specific class name. Moving on, the system checks the tensor position, and if the position of the detected UI elements matches, then the end user gets a success message that it's on the right track. But if it doesn't match, then error messages are shown with proper mentions of the specific UI elements and suggested position from each.

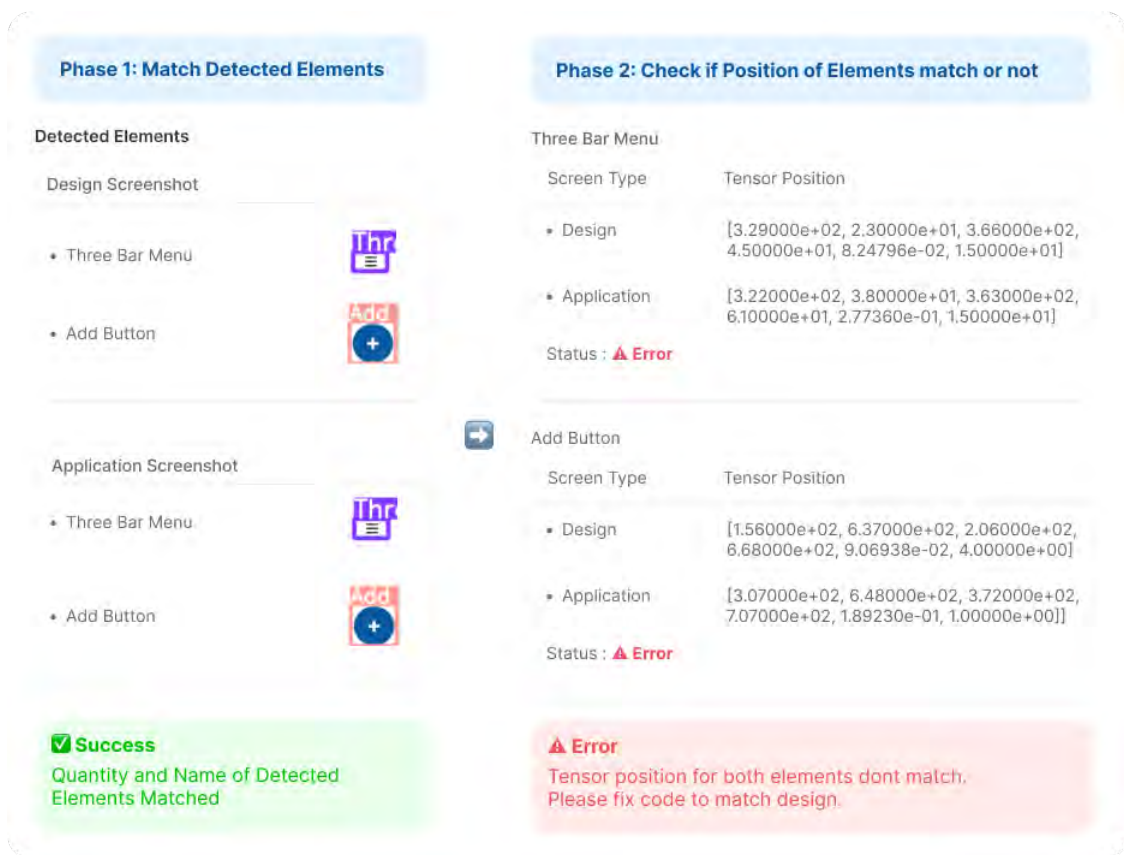


Figure 4.16: Application Testing Mechanism in terms of Design.

Chapter 5

Conclusion

5.1 5.1 Challenges

During the course of our investigation, we met a series of serious problems that greatly affected the conclusion of our study. Firstly we did not selected the dataset randomly . We had to categorize the screenshots which has ideal measurement of each and every buttons so that we can have good mAP of each and every class . Then we focused on specific screenshots which had more buttons . One noteworthy problem resulted from the hardware limits inherent in processing Android screenshots for object recognition utilizing the YOLOv5 and YOLOv8 models. These limits created constraints on the efficiency and speed of our image processing activities, warranting careful study and optimization of computing resources. Another difficulty surfaced in the process of categorizing smaller border boxes inside the Android screenshots collection. The intricacy of precisely identifying these tiny components required painstaking attention to detail, since faults in labeling might possibly impair the accuracy of the object identification models. Additionally, the cleansing of datasets offered its own set of issues. Ensuring the quality and consistency of the Android screenshot dataset needed a rigorous data cleaning procedure to minimize noise and extraneous information, boosting the reliability of later analysis. Tuning parameters for the YOLOv5 and YOLOv8 models added another degree of complexity. Fine-tuning these parameters to enhance the performance of the object identification models needed sophisticated knowledge of model behavior and responsiveness, necessitating repeated modifications and assessments. In navigating through these challenges related to hardware limitations, labeling precision, dataset cleaning, and parameter tuning, our research not only highlights the intricacies involved in utilizing object detection models for Android screenshots but also emphasizes the importance of addressing such challenges for the advancement of accurate and efficient image processing in the realm of UI development and testing.

5.2 Future Prospect

Looking toward the future, the combination of ensemble training with YOLOv5 and YOLOv8 offers good promise for improving object identification algorithms in UI development and testing. As technological landscapes improve, the continued refinement and optimization of ensemble training methodologies are projected to contribute considerably to the efficiency and accuracy of UI-related image pro-

cessing tasks. Future potential includes the exploration of updated ensemble procedures that leverage the powers of YOLOv5 and YOLOv8 even more effectively. This includes looking into sophisticated ensemble designs, adaptive learning rate procedures, and innovative model coordination approaches. As improvements in hardware capabilities continue, the opportunity for overcoming hardware limitations—previously a challenge—opens prospects to more optimization and speedier ensemble training methods. Additionally, the integration of ensemble training with YOLOv5 and YOLOv8 could grow beyond ordinary UI development. The transferability of this technology to other sectors, such as computer vision applications and autonomous systems, presents fascinating possibilities for wider technological applications. As the research community continues to examine ensemble training with expanding object detection models, future efforts may focus on creating standardized approaches and frameworks for seamless integration. Collaboration and information sharing between research groups will be key to establishing the future landscape of ensemble training methodologies and ensuring their wide adoption and use in different real-world scenarios.

5.3 Conclusion

Finally, This research journey has explored the sensitive areas of UI design and functionality testing in Android applications. Understanding the critical role of the Software Development Life Cycle (SDLC) and Software Testing Life Cycle (STLC), We conducted a study to address challenges and expedite industry advancements. The intricacy of creating user interfaces was brought to light, especially when considering Android apps and their abundance of core and composite components. We identified the critical need to expedite UI design and automate functionality testing, highlighting challenges including labor-intensive testing methodologies, extended development timelines, and a dearth of quick functional testing protocols. By providing an automated recommendation system for component placement and functionality testing, our proposed solution aims to transform user interface development by reducing development time and increasing overall productivity. The research proceeded by means of an extensive examination of extant literature and a profound exploration of fundamental concepts, such as object identification, computer vision, and the YOLOv5 and YOLOv8 models. Handling Android screenshot datasets presented a number of difficulties, from hardware limitations to the laborious task of locating smaller border boxes and dataset cleaning. The tale was further expanded by parameter tinkering and ensemble training complexity with YOLOv5 and YOLOv8, bringing to light the complexities of utilizing cutting-edge technology in UI design. I think there are a tonne of great opportunities for UI development in the future. In addition to addressing current issues, the proposed paradigm, when paired with automation techniques, offers a foundation for ongoing innovation. Our work advances UI development methodologies by enabling UI developers to do auto-functional testing and get recommendations for optimal component placements. In summary, This study sheds light on the challenges and opportunities facing UI development, emphasizing the need for automation and efficient testing techniques. Our findings contribute to the ongoing conversation about UI development as we navigate a rapidly evolving technology landscape and offer a path forward for further research and development in this important area. This project aims to be a

catalyst for innovative advances in the dynamic field of UI creation in Android apps by combining theoretical research with real-world problems.

Bibliography

- [1] M. Vanmali, M. Last, and A. Kandel, “Using a neural network in the software testing process,” *Journal Name*, vol. 1, Jan. 2002.
- [2] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, “How effectively does metamorphic testing alleviate the oracle problem?” *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 4–22, 2013.
- [3] W. He, R. Zhao, and Q. Zhu, “Integrating evolutionary testing with reinforcement learning for automated test generation of object-oriented software,” *Journal Name*, vol. 1, Jan. 2015.
- [4] R. Mathur, S. Miles, and M. Du, “Adaptive automation: Leveraging machine learning to support uninterrupted automated testing of software applications,” Aug. 2015.
- [5] G. Luo, “A review of automatic selection methods for machine learning algorithms and hyper-parameter values,” *Netw Model Anal Health Inform Bioinforma*, vol. 5, no. 18, 2016.
- [6] D. Banerjee, K. Yu, and G. Aggarwal, “Image rectification software test automation using a robotic arm,” *IEEE Access*, vol. 6, pp. 34 075–34 085, 2018.
- [7] D. S. Battina, “Artificial intelligence in software test automation: A systematic literature review,” *International Journal of Emerging Technologies and Innovative Research*, vol. 6, no. 12, pp. 1329–1332, Dec. 2019.
- [8] L. Butgereit, “Using machine learning to prioritize automated testing in an agile environment,” in *2019 Conference on Information Communications Technology and Society (ICTAS)*, Durban, South Africa, 2019, pp. 1–6. DOI: 10.1109/ICTAS.2019.8703639.
- [9] V. H. S. Durelli *et al.*, “Machine learning applied to software testing: A systematic mapping study,” *IEEE Transactions on Reliability*, vol. 68, no. 3, pp. 1189–1212, Sep. 2019. DOI: 10.1109/TR.2019.2892517.
- [10] J. Gao, C. Tao, D. Jie, and S. Lu, “Invited paper: What is ai software testing? and why,” in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, San Francisco, CA, USA, 2019, pp. 27–2709. DOI: 10.1109/SOSE.2019.00015.
- [11] H. Hourani, A. Hammad, and M. Lafi, “The impact of artificial intelligence on software testing,” in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, IEEE, 2019, pp. 565–570.
- [12] M. A. Jakobs, A. Dimitracopoulos, and K. Franze, “Kym butler: A deep learning software for automated kymograph analysis,” *eLife*, vol. 8, e42288, 2019.

- [13] Y. Zheng *et al.*, “Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA, 2019, pp. 772–784. DOI: 10.1109/ASE.2019.00077.
- [14] “Audee: Automated testing for deep learning frameworks,” in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE ’20)*, IEEE/ACM, 2020, pp. 486–498. DOI: 10.1145/3324884.3416571.
- [15] D. Banerjee and K. Yu, “3d face authentication software test automation,” *IEEE Access*, vol. 8, pp. 46 546–46 558, 2020.
- [16] O. Davtalab, A. Kazemian, X. Yuan, and B. Khoshnevis, “Automated inspection in robotic additive manufacturing using deep learning for layer deformation detection,” Oct. 2020.
- [17] Z. Peng, T.-H. Chen, and J. Yang, “Revisiting test impact analysis in continuous testing from the perspective of code dependencies,” *IEEE Transactions on Software Engineering*, 2020.
- [18] M. Esnaashari and A. Dania, “Automation of software test data generation using genetic algorithms and reinforcement learning,” *Expert Systems with Applications*, 2021. DOI: 10.1016/j.eswa.2021.115446.
- [19] T. Gutierrez, A. Bergel, C. E. Gonzalez, C. J. Rojas, and M. A. Diaz, “Systematic fuzz testing techniques on a nanosatellite flight software for agile mission development,” *IEEE Access*, vol. 9, pp. 114 008–114 021, 2021.
- [20] J. Hitchcock, M. Hundertmark, D. Foreman-Mackey, *et al.*, “Pytorchdia: A flexible, gpu-accelerated numerical approach to difference image analysis,” *Monthly Notices of the Royal Astronomical Society*, vol. 504, pp. 3561–3579, 2021. DOI: 10.1093/mnras/stab1114.
- [21] A. Siddiqui, M. Y. I. Zia, and P. Otero, “A universal machine-learning-based automated testing system for consumer electronic products,” *Journal Name*, vol. Volume, no. Issue, 2021.
- [22] M. N. Noor, T. A. Khan, F. Haneef, and M. I. Ramay, “Machine learning model to predict automated testing adoption,” *International Journal of Software Innovation (IJSI)*, vol. 10, no. 1, 2022.
- [23] A. Senchenko, N. Patterson, H. Samuel, and D. Ispir, “Supernova: Automating test selection and defect prevention in aaa video games using risk based testing and machine learning,” in *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*, Valencia, Spain, 2022, pp. 345–354. DOI: 10.1109/ICST53961.2022.00043.
- [24] C. Wan, S. Liu, S. Xie, *et al.*, “Automated testing of software that uses machine learning apis,” Jul. 2022.
- [25] Author(s), *Manual and automation testing challenges*, Unpublished, 2023.
- [26] H. Bandyopadhyay. “A friendly guide to labeling [+open datasets, models, alternative tools],” V7. (Apr. 2023), [Online]. Available: <https://www.v7labs.com/blog/labelimg-guide>.

- [27] E. O. D. Nascimento and P. H. T. Zannin, “Comparison of the sensitivity of room acoustic parameters on speech intelligibility using artificial neural networks and multiple linear regression,” Aug. 2023, SSRN Working Paper.
- [28] A. Sharma. “Training the yolov5 object detector on a custom dataset - pyimagesearch,” PyImageSearch. (Jun. 2023), [Online]. Available: <https://pyimagesearch.com/2022/06/20/training-the-yolov5-object-detector-on-a-custom-dataset/>.
- [29] P. Skalski. “Train yolov8 on a custom dataset,” Roboflow Blog. (Oct. 2023), [Online]. Available: <https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/>.
- [30] T. Illes, A. Herrmann, B. Paech, and J. Rückert, “Criteria for software testing tool evaluation – a task oriented view,” Institute for Computer Science, University of Heidelberg, Im Neuenheimer Feld 326, 69120 Heidelberg, Germany, Tech. Rep., Year.
- [31] D. Kumar and K. K. Mishra, “The impacts of test automation on software’s cost, quality, and time to market,” Year.