DeepTestDroid: A Platform for Automated Application Testing Using Deep Learning

Nazia Tabassum Natasha

ID: 2019-1-60-202

Imran Fakir

ID: 2019-1-60-203

Md. Huzaifa

ID: 2019-1-60-079

Fabiha Bushra Fabin

ID: 2019-1-60-139

Sabiha Afsana Falguni

ID: 2019-1-60-261

A Capstone project report submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering

Department of Computer Science and Engineering East West University
Dhaka-1212, Bangladesh

9th August 2023

Declaration

We, **Nazia Tabassum Natasha**, **Imran Fakir**, **Md. Huzaifa**, **Fabiha Bushra Fabin** and **Sabiha Afsana Falguni** hereby, declare that the work presented in this capstone project report is the outcome of the investigation performed by us under the supervision of **Dr. Mohammad Rifat Ahmmad Rashid**, Designation, Department of Computer Science and engineering, East West University. We also declare that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma, except for publication.

Countersigned

. . . .

Dr. Mohammad Rifat Ahmmad Rashid
Supervisor

Signature

. . . .
Nazia Tabassum Natasha
2019-1-60-202

FabihaBushraFabin
. . . .
Fabiha Bushra Fabin
2019-1-60-139

[illegible]
. . . .
Md. Huzaifa
2019-1-60-079

[illegible]
. . . .
Imran Fakir
2019-1-60-203

[illegible]
. . . .
Sabiha Afsana Falguni
2019-1-60-261

Letter of Acceptance

The capstone project report entitled "DeepTestDroid: A Platform for Automated Application Testing Using Deep Learning" is submitted by Nazia Tabassum Natasha, Imran Fakir, Md. Huzaifa, Fabiha Bushra Fabin and Sabiha Afsana Falguni to the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh is accepted for the partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Engineering on (9/8/2023).

Board of Examiners

1. ____

Supervisor Name (Dr. Mohammad Rifat Ahmmad Rashid)

Assistant Professor
Department of Computer Science and Engineering
East West University

2. ____

Chairperson Name (Dr. Maheen Islam)

Chairperson and Associate Professor
Department of Computer Science and Engineering
East West University

Abstract

Black box and white box testing are used to measure an application. Testing can be complex and costly. Various testing techniques can measure multiple parts of an application. This work addresses the discussion of black box and white box testing types, utilization of testing and training data, data set categorization, implementation of deep learning for automated testing, construction of testing approaches, investigation into fundamental techniques and input information, processing of detailed test approach information, proposal for assessing automated testing effectiveness, utilization of quality attributes, metrics, datasets, and procedures for assessment, and employment of evaluation metrics to measure the deep learning approach performance. DeepTestDroid performs UI performance testing, which is black box testing. It can predict the density of an application's page from its wireframe picture. Higher density means a longer load time. UI performance testing can only be performed with human interaction. However, DeepTestDroid makes it automatic. To detect density, DeepTestDroid uses MobilenetV3Large. MobileNetV3Large was selected after an experimental analysis alongside other image processing models and almost 75% of test accuracy was achieved by MobileNetV3Large model, through this work. To reduce the usage of different tools for white box testing, XGBRegressor, RandomForestRegressor, and DecisionTreeRegressor models were trained and tested to generate the values of the application’s UI that the existing testing tool would predict and XGBRegressor was able to give the R2 (R-squared) score of 0.96. This project can reduce the cost, time, and step of UI performance testing as well as the white box testing. It will also make it easier to complete the testing phase of SDLC. This work introduces significant utilities that were not previously offered by existing literature.

Acknowledgments

As it is true for everyone, we have also arrived at this point of achieving a goal in our life through various interactions with and help from other people. However, written words are often elusive and harbor diverse interpretations even in one’s mother language. Therefore, We would not like to make efforts to find best words to express my thankfulness other than simply listing those people who have contributed to this thesis itself in an essential way. This work was carried out in the Department of Computer Science and Engineering at East West University, Bangladesh.

First of all, we would like to express my deepest gratitude to the almighty for His blessings on us. Next, our special thanks go to our supervisor, “Dr. Mohammad Rifat Ahmmad Rashid”, who gave us this opportunity, initiated us into the field of “DeepTestDroid: A Platform for Automated Application Testing Using Deep Learning”, and without whom this work would not have been possible. His encouragement, visionaries and thoughtful comments and suggestions, unforgettable support at every stage of our B.Sc. study was simply appreciating and essential. His ability to muddle us enough to finally answer our own question correctly is something valuable what We have learned, and We would try to emulate, if ever we get the opportunity.

We would like to thank our group members for their excellent collaboration during performance evaluation studies; overall support; and their helpful suggestions in solving tricky technical problems. Last but not least, we would like to thank our parents for their unending support, encouragement, and prayers.

There are numerous other people too who have shown me their constant support and friendship in various ways, directly or indirectly related to our academic life. We will remember them in my/our heart and hope to find a more appropriate place to acknowledge them in the future.

Nazia Tabassum Natasha
August 2023

Imran Fakir
August 2023

Md. Huzaifa
August 2023

Fabiha Bushra Fabin
August 2023

Sabiha Afsana Falguni
August 2023

Table of Contents

List of Figs

List of Tables

List of Algorithms

List of Acronyms

SDLC	Software Development Life Cycle
UX	User Experience
GUI	Graphical User Interface
RL	Reinforcement Learning
ML	Machine Learning
AI	Artificial Intelligence
R&D	Research and Development
TEMA	Testing and Monitoring for Android
BDD	Behavior-Driven Development
ARES	Automated Reinforcement learning-based Exploration of Android System
FATE	Fast Android Testing Execution
AUT	Application Under Test
PLOCs	Potential Lines of Codes
Mod. PLOCs	Modified Potential Line of Codes
TLOCs	Test Lines of Codes
Mod. TLOCs	Modified Test Lines of Codes
TLR	Test LOC’s Ratio
MTRL	Modified Test LOCs Ratio
MRTL	Modified Relative Test LOCs
TMR	Total Method Ratio
MMR	Modified Method Ratio
MCR	Modern Code Review
RFCR	Reached Full Code Coverage Requirement
FCR	Flow Coverage Requirement

Chapter 1

Introduction

1.1 Background

Black box testing is a powerful testing technique because it exercises a system end-to-end. Just like end-users don't care how a system is coded or architected, and expect to receive an appropriate response to their requests, a tester can simulate user activity and see if the system delivers on its promises. Along the way, a black box test evaluates all relevant subsystems, including UI/UX, web server or application server, database, dependencies, and integrated systems.

By leveraging deep learning, app testing tools can provide more efficiency for development teams in the creation and analysis phases of testing through both black-box and white-box testing [1]. Black box testing involves testing a system without knowing its internal workings. A tester provides input and observes the system-generated output under the test. This makes it possible to identify how the system responds to expected and unexpected user actions, its response time, usability issues, and reliability issues. White box testing is a technique that uses a program's internal or source code to design different test cases to check the program's quality; this technique, the internal structure, and the implementation of how an application works are known to the tester. Due to the rapid release cycle and limited human resources, creating test cases promptly and manually is challenging. As a result, numerous automated test input generators have been designed. Tools and software already enable us to automatically test apps, including Monkey, Dyno Droid, Espresso, etc. The success of an automated test relies on picking the appropriate interaction for a given UI, allowing access to new relevant UI states. However, it can be challenging for a machine to interpret the GUI and select the appropriate button to click or scroll. Because of this, most automated test generators use different types of GUI elements and instead use a random selection or brute force to choose which one to test. Human testers are better at this than automated test input generators because they can quickly determine which parts of the GUI need to be tested.

1.2 Problem Statement and Analysis

Android application's UI testing is a type of "black box" testing that focuses on ensuring the user interface of an application is functioning correctly. The purpose of UI testing is to ensure that the application's user interface is intuitive, easy to use, and that load performance is good and consistent with the design specifications. For "white box" testing, there are many tools that can predict some parameters' values of an application's UI. But there is no platform which automates and incorporates different tools while using deep learning methods for predicting those values. To perform UI testing, the following issues arise which makes the testing difficult to perform to ensure the quality of the interface of an Android application. In some cases, UI testing tools and models may not be able to adapt to changes in the graphical user interface of an application, leading to false positives or false negatives, which means a layout can be categorized as medium density but in the updated version of the app, it can be categorized as very-high density. As applications evolve and change over time, automated UI tests may need to be updated and maintained to ensure they are still relevant and accurate. Manual UI testing requires test data to be maintained and updated regularly, which can be time consuming (especially when testing large and complex applications) and error-prone.

UI testing tools may have limitations in their ability to interact with certain parts of an application or handle certain types of user inputs, which can impact the accuracy and completeness of the testing. For example, a UI testing tool may not be able to interact with elements that are dynamically generated or hidden, such as pop-ups or overlays that appear only under certain conditions. Test data that are manually classified for the model will have a big impact on the prediction of the model, as it might affect the prediction of the model if the classified data are not sorted carefully.

Here are our few research questions:

- Which type of black box and white box testing is addressed in our paper?
    - What type of data will we use for testing and training the model?
    - How will we categorize the data set?
    - How will we automate the testing using a deep learning approach?
- How are the testing approaches built?
    - How will we investigate the fundamental techniques leveraged as well as the amount of input information that is required to perform testing?
    - How will we process detailed information on the design and implementation of test approaches?
- How can we measure the effectiveness of our proposed automated testing approach?
    - Which quality attributes, metrics, datasets, and procedures for the m literature can be used for measuring the effectiveness of our approach?
    - Which evaluation metrics can be used to measure the performance of the deep learning approach?

In order to automate UI testing, first we have to create five categories according to the number of components present in the UI layout. Then according to these categories, data needs to be sorted. With the sorted data a model needs to be trained. Finally, a platform will be created to perform UI testing for android applications which will determine which class the app’s layouts belong to. Then a model will be created which will determine the values for white box testing to automate and eliminate the need of other existing testing tools.

1.3 Project Objectives

To build a system that predicts test outcomes using deep learning. This paper integrates black-and-white-box testing with deep understanding, a novel approach. The goal is to automate black-box and white-box tests and generate test data for a deep learning algorithm. The objective is to achieve the most accurate prediction using the deep learning algorithm, ensuring Android applications' durability, vulnerability, and performance. Automated UI testing requires the creation of five categories based on the number of components in the UI layout. The data needs to be sorted according to these categories. Using the sorted data, a model can be trained. Finally, a platform will be developed for UI testing of Android applications, classifying the layouts into their respective categories.

Fig 1.1 shows the comparison graph between black box and white box testing in various papers, 52 papers worked on black box testing versus 14 papers worked on white box testing. No single paper incorporated black box and white box testing in their model.

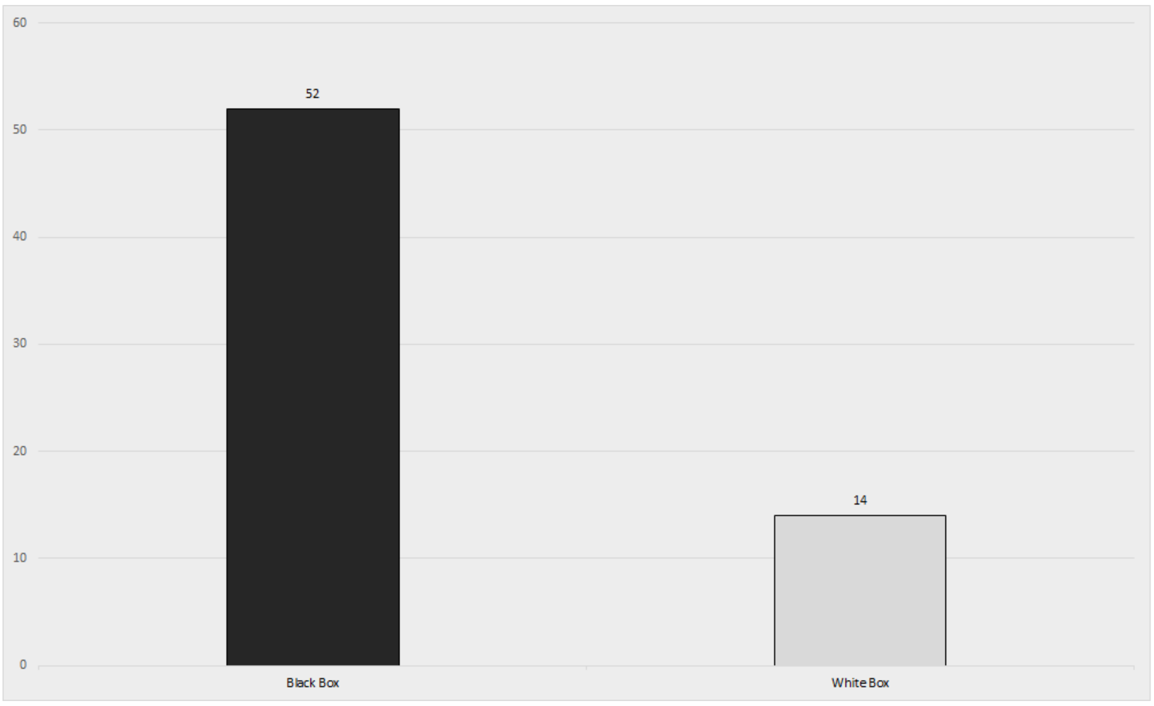


Fig 1.1: Comparison Between Black Box and White Box Testing in Various Papers

1.4 Project Contributions

Testing an Android application's user interface (UI) is a sort of "black box" testing that focuses on making sure the user interface is operating properly. The goal of UI testing is to confirm that the user interface of the program is clear and simple to use, and that load performance is satisfactory and consistent with the design standards. The following challenges make it challenging to conduct UI testing to guarantee the quality of an Android application's user interface. When UI testing methods and models are unable to adapt to changes in an application's graphical user interface, false positives or false negatives might result, classifying a layout as medium density yet in the updated version. Even the first initiative for white box testing has been taken.

The contributions of this paper are:

1. We must first build five categories based on the number of components in the UI layout in order to automate UI testing. The data must then be sorted in accordance with these categories.
2. It is necessary to train a model using the sorted data. Last but not least, a platform will be developed to do UI testing for Android applications, identifying the class to which the app's layouts belong. Transfer learning, such as EfficientNetB3, ResNet50, and MobileNetV3, are used so the model continuously improves.
3. For white box testing, the model created for this project will generate the values of the existing testing tools which will reduce the need for using different testing tools. XGBRegressor, LinearRegression, ElasticNet, RandomForestRegressor, DecisionTreeRegressor models have been tested in order to determine the best working model.

1.5 Project Outlines

The remainder of the essay is structured as follows. Section 2 covers the literature reviews, Section 3 describes the suggested model and its components for testing the UI of the apps, Section 4 shows the results and discussion, and Section 5 concludes the study. Fig 1.2 shows the sequence of this book.

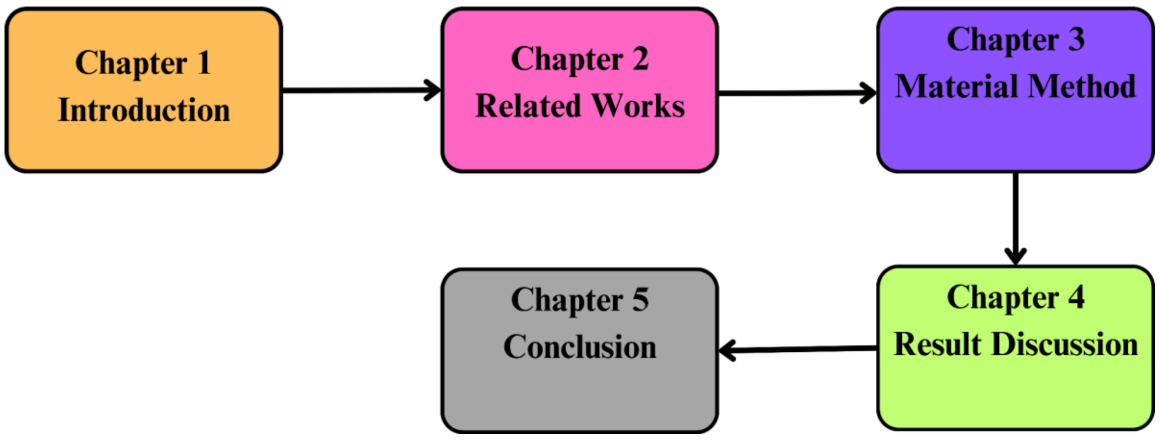


Fig 1.2: Flowchart of this Book

Chapter 2

Related Works

2.1 Survey of the State-of-the-art

Deep Xplore, introduced in a research publication, revolutionizes the systematic testing of real-world deep learning systems [2]. This system efficiently discovers improper corner case behavior in state-of-the-art deep learning models with thousands of neurons trained on popular datasets. Deep Xplore generates the wrong test inputs in one second on a laptop. Unlike differential testing approaches, deep Xplore detects errors even when profound neural network outputs are consistent. Most deep neural networks are designed and trained independently, making it unlikely that all of them will commit the same error in practice.

One research study outlines constructing and testing a mobile app-based artificial intelligence prototype that mimics user activities [3]. Deep learning predicts screen taps and actions. Convolutional neural networks encode visuals and screen elements, while long-short-term memory remembers to touch positions on successive screens. Deconvolutional neural networks indicate tap location and linear neural networks' action type. The research shows how deep learning can automate mobile app testing through model construction and training. More tests are planned to improve outcomes. They want to experiment with longer training times, larger datasets, and transfer learning to retrain network components before training others. Another option is to share learning with a more prominent public dataset before fine-tuning their own. After improving the findings, link the network's output to testing software and test its multi-screen navigation. The final goal is to train the prototype and utilize it as an automatic input generator for testing.

Another work proposes new testing methods for system faults. The proposed method uses a web scraper, ML, and Selenium [4]. Machine learning and Selenium are used to improve web-based error detection and discovery. The researchers plan to develop a web testing tool that delivers accurate URL test results. They want to provide specialized testing methods for distinct web elements and expand test cases to support diverse testing tactics.

Another study automates and scales Android app security and robustness testing. The paper introduces an Android-specific software analysis technique that creates several fuzz test scenarios [5]. They also offer a cloud-based test infrastructure that runs these test cases on several simulated Android devices. This method underlines the need for smartphone-specific guided fuzzing strategies rather than brute-force fuzzing. The white-box strategy tries to identify vulnerabilities efficiently, decreasing the time and computational resources needed for real-world fuzz testing.

In another study, the authors propose a new Android app testing automation method [6]. Machine learning and frequent test situations improve testing. Empirical analysis shows that their improved testing tool outperforms conventional methods in natural settings. Activity classifications and preset features limit the researchers' approach. They are working with the Test Project R&D team to add activity categories and reduce the testing algorithm's reliance on hard-coded test cases.

Another research study introduces Crash Scope, an automated Android developer tool that creates crash reports and test scripts [7]. It gives developers thorough error reports and detects a similar number of crashes as other advanced tools. The crash report contains screenshots, steps to reproduce the crash, stack traces of exceptions, and a script that can consistently recreate the issue. The researchers hope to use model-based GUI testing and static analysis to streamline bug reports and improve their systematic exploration method.

Humanoid is a new method for automated black-box testing of humanoid Android apps. Deep learning generates GUI test inputs from human interactions [8]. A deep neural network model accurately predicts application user interactions. This learned model is used to create an Android input generator. The input generator is tested on open-source and commercial programs to verify real-world dependability and applicability. Humanoid has limits. System broadcasts and sensor events are ignored. It also does not predict text input. The researchers propose to enhance the interaction model or use alternative text input generation methods to address these constraints.

In a research study, the researchers proposed automating Android app testing to find GUI problems [9]. The researchers found existing and undetected system weaknesses, providing vital insights to avert future failures. However, limits must be acknowledged. Defects beyond designated activity, event, or type categories cannot be detected. Researchers propose a dualistic approach to avoid these restrictions. First, they suggest state machines for I/O and concurrency primitive invocations. Developers can compare application logs to this model to better spot anomalies. Second, they want Java static analysis tools to use model-based verification. This compile-time technique can identify pattern and state-machine violations during development, minimizing dependency on automated testing and improving system dependability.

A grey literature review on using artificial intelligence (AI) to improve test automation is presented in another study [10]. AI is being used to create and enhance test code, according to 136 papers found in 1,200 sources. AI-enabled problems and solutions are categorized in the report. Code and automated test generation are examples of common issues and solutions. The report also suggests adding literature sources and using developer surveys and interviews to corroborate the findings on AI in test automation. Traditional and AI-based methods can be compared to assess AI's benefits in testing.

Software automation and machine learning (ML) frameworks were extensively explored in the paper [11]. Testing tools were rated on test performance, accuracy, scope, time, expertise, and manual labor. To improve software quality and ML frameworks for automation software. Future studies will examine and organize software testing and machine learning research to help researchers and engineers develop guidelines for using ML approaches to software testing.

This case study presents a novel automated testing framework that blends user interaction characteristics, historical bug data, and an interest point detector and descriptor to find user interaction concerns effectively [12]. This methodology detects user interaction problems empirically. The researchers want to create a customized exploration environment to influence target application event sequences during testing. Each application's scope will be expanded to assess its usefulness and resilience further.

The researchers created a mechanism to generate tests for Android resource leak problems automatically [13]. They propose a resource-efficient, neutral GUI event flow. Their resource leak testing method surpasses a non-automated approach from earlier research. The study shows that static control flow analysis can automatically develop comprehensive Android GUI tests for repeating behaviors. Static or dynamic analysis-based test execution, monitoring, and prioritizing methodologies need improvement.

AM-TaaS, a cloud-based automated mobile app testing system, is described in the paper. The framework gives users on-demand testing services [14]. Results reveal that framework-designed automated test cases tested all emulated devices. Future research will optimize and extend the cloud infrastructure, automate more test cases based on specified criteria, build new evaluation methods for test cases, and expand the framework to accommodate Windows Phone and iOS.

In the paper, the researchers address the performance degradation of mobile devices due to the simultaneous use of multiple mobile applications [15]. They have developed a performance testing tool (PTT) with a user-friendly interface and accurate performance testing results. The plan for PTT includes enhancing performance indicators such as electricity and GPU.

The research paper examines model-based testing of Android user interfaces, notably the BBC News widget case study [16]. Case study models and tools are open-source. The researchers emphasize that although the technology (TEMA) employed still needs to provide automatic tools for reverse engineering models, no technical limitations hinder this capacity. The device could include a scripting language with keywords or a Robot Framework library.

The research paper presents an innovative method for automating the functional testing of mobile software [17]. This method combines machine learning techniques with standard test scenarios for effective and practical testing. They demonstrate that their system can perform a significant portion of the tasks typically done by human testers, potentially reducing manual testing efforts.

This paper examines a sophisticated Android remote UI testing infrastructure. Developers and researchers can test scripts on several Android versions to find bugs across platforms quickly [18]. The platform cannot imitate real-world use; therefore, it cannot test applications primarily relying on speech, gesture, or movement inputs. Multitouch events and device orientation testing are planned to improve the platform.

This paper discusses automated software interface testing, particularly behavior-driven testing, a key component of BDD. Behavior-based testing is used for Android app user interface automation [19]. Behavior-driven testing involves constant reworking to adapt to environmental changes; therefore, upgrading the instrument to enable Cucumber may not be viable for existing products with instrumented tests.

A privacy framework in the paper limits undesired permissions and prevents programs from collecting user data [20]. The proposed technique uses data mining to identify superfluous permissions and modify permission information for privacy and application operation. The framework may not work for sealed, protected applications in sectors like finance, which require additional protection. The researchers want to strengthen the framework for such applications' security and protection procedures.

In this paper, mobile applications face fragility in GUI testing, as desktop applications are less rigorous [21]. This research investigates the dissemination of test classes using common GUI Automation Frameworks for Android applications, the extent of changes needed to maintain relevance, code turnover, and the fundamental adjustments in the AUT that prompted these modifications. The study uses 12 metrics and a classification system to analyze test class progression and code turnover. Results show no significant adoption of GUI automation frameworks within open-source Android projects hosted on GitHub. However, test suites require frequent modifications, with 8% of developers' modified lines of code involving test code and 50% involving changes in the graphical user interface. This may hinder developers' adoption of automated testing. Assessing maintenance requirements and categorizing factors that require modifications can serve as a standard for software developers and provide a foundation for creating practical recommendations and designing automated utilities to address this issue.

Finally, researchers in the paper used Deep Reinforcement Learning (RL) to automate Android app exploration [22]. ARES and FATE, their model-based Android testing tools, highlight their approach's efficacy. Coverage and problem discovery are better with configuration structures. The process may degrade, and system-level events that require rooted devices cannot be performed. Android app fault categories and iOS adaptations are planned. The papers propose novel methods and frameworks for automating Android application development, addressing constraints, and suggesting further study.

2.2 Summary

Table 2.1 summarizes all the papers that have been discussed so far. This table highlights each work's methodology, drawbacks and the solutions that have been addressed in our work.

Table 2.1: Summary of Related Works

Paper Title	Methodology	Drawbacks	Solution
DeepXplore: Automated Whitebox Testing of Deep Learning Systems [2]	It effectively identifies numerous rare edge case issues in advanced deep learning models containing thousands of neurons trained on widely-used datasets.	Here, researchers only performed white box testing. They did not test it from the interface, which is black box testing.	A deep learning approach was utilized in black-box testing to determine the density of Android applications' UI.
Visual GUI testing in practice: challenges, problems and limitations [3]	This study identified four high-level solutions and metrics on cost and return on investment, indicating that VGT is valuable, flexible, and cost-effective for industrial practitioners.	Study limitations is VGT transitions were performed on safety-critical systems with legacy code and similar characteristics.	The UI testing was done by the deep learning approach.
A Whitebox Approach for Automated Security Testing of Android Applications on the Cloud [4]	In this paper, researchers provide an overview of a multi-faceted project targeted at automatically testing the security and robustness of Android apps in a scalable manner.	They did not implement deep learning to automate testing. Also, they did not perform any black box testing for security testing.	The methodology proposes black-box testing with the implementation of deep learning for automating Android application testing.
Humanoid: A Deep Learning-based Approach to Automated Black-box Android App Testing [5]	A deep neural network model was designed and implemented to analyze user interactions with an app. An Android app input generator was also developed based on this model and evaluated on open-source and market apps.	This paper has limitations regarding including system broadcasts and sensor events as inputs. Furthermore, it does not cover white box testing or the ability to predict text when sending text input actions.	In DeepTestDroid, black-box (UI testing) was performed with a deep-learning approach.

A Deep Learning Based Approach to Automated App Testing [6]	Researchers designed and implemented an artificial intelligence prototype that mimics real user behavior using deep learning and neural networks.	The prototype was not trained using reinforcement learning to identify bugs and glitches.	Here the automation was done through GUI testing of apps screenshots.
User interface test automation for an Android application [7]	This paper evaluates the applicability of behavior-based testing for automated user interface testing of an android application, focusing on Agile software development and its application in Agile software development.	Drawbacks in the use of behavior driven testing as it requires refactoring to keep up-to-date with changes in the environment.	The interface testing is done properly by the model trained with deep learning approach.
Automating GUI testing for Android applications [8]	Researchers propose an automated Android application testing approach for graphical user interface bugs, identifying existing and new issues, and preventing future issues.	There were some limitations such as some bugs did not fall into the activity/event/type categories.	Here android apps UI was taken for testing purpose
Automation of Android Applications Testing Using Machine Learning Activities Classification [9]	An approach for automating Android application testing using machine learning and reusing test scenarios, outperforming standard methods in realistic settings, is presented.	Limited to the activity types and the functionalities which have been pre-defined.	Deep learning was taken to perform black box and ML approach was taken to perform white box testing.
An Approach of Automated Testing on Web Based Platform Using Machine Learning and Selenium [10]	Their approach trained web pages from human perception, intelligently performing sanity and smoke tests on elements, and classifying outputs as error pages or not intended.	Limitation testing all web elements for a given URL or testing specific web elements for specific types.	Both white box and black box testing was performed.

AI-based Test Automation: A Grey Literature Analysis [11]	This research reviews the application of artificial intelligence in test automation procedures, analyzing over 1,200 sources and identifies six common tools and their AI-enabled solutions, with manual code development and automated test generation being the most frequently reported problems.	They did not survey and interviews with developers will confirm findings, and comparison tests between traditional and AI-based methods to evaluate AI's advantages.	This work provides solutions that are also based on area of AI.
Evaluation of an Automated Testing Framework: A Case Study [12]	They suggested automated testing framework using user-interaction characteristics, historical bug data, and interest points detector for defect detection.	Researchers did not create a personalized exploration environment for better control over event-sequences and application scope.	Both black box and white box testing was automated in the platform.
CrashScope: A Practical Tool for Automated Testing of Android Applications [13]	CrashScope is an automated tool for Android developers, generating crash reports and test scripts, providing detailed information, including screenshots, reproduction steps, exception stack trace, and a reliable script.	Researchers did not explore bug report reduction using model-based GUI testing and static analysis.	GUI testing was performed by a model trained by deep learning approach.
Automated Mobile Testing as a Service (AM-TaaS) [14]	This introduces AM-TaaS framework, providing automated mobile application testing on-demand using cloud infrastructure, enabling 100% device testing using the framework's test cases.	Researchers did not optimize cloud infrastructure, automate test cases, develop new evaluation techniques, and develop frameworks for Windows Phone and iOS platforms.	DeepTestDroid platform can automatically test wireframes including android, iOS, and windows apps.
Privacy Protection Framework for Android [15]	This paper proposes a privacy-preserving secure framework to restrict unnecessary permissions, ensuring proper functioning and user data protection.	Their approach may not work for sealed protected finance/payments applications due to additional security, preventing installation.	This platform tests the app's screenshots to find the density of it.

Design and Development of Android Performance Testing Tool [16]	Mobile performance declines due to simultaneous application usage; less efficient testing tools exist; PTT offers a strong, attractive interface for accurate results.	Increasing some performance indicators, such as electricity, GPU, etc. by researchers.	This work has only one indicator that can increasing the performance of DeepTestDroid which is by increasing categories.
Experiences of system-level model-based GUI testing of an android application [17]	This paper discusses model-based user interface testing of Android applications, focusing on the BBC News widget, using open-source models and tools.	TEMA lacks automatic reverse-engineering tools, but technical limitations can prevent it.	Our approach was for both black box and white box testing.
Automation of Android Applications Functional Testing Using Machine Learning Activities Classification [18]	Using ML techniques, mobile software functional testing can be automated by reusing generic test scenarios.	The limitation of their work is that they implement only black-box testing using machine learning.	Deep knowledge was incorporated into black-box testing (UI testing) for automation.
Automated Testing with Machine Learning Frameworks: A Critical Analysis [19]	This paper analyzes machine learning frameworks in software automation, focusing on test performance, accuracy, scope, time, and knowledge requirements while measuring manual labor effort to ensure excellent outcomes and software quality.	The study aims to review and organize previous work on software testing and machine learning, enabling future researchers and engineers to create rules for ML approaches.	This work incorporated both deep learning and ML methods.
Testdroid: automated remote UI testing on Android [20]	The platform provides test results that help developers detect potential crashes or failures in different media.	The testing limitations arise when evaluating applications that rely on voice, gesture, or movement input, making it challenging to simulate the actual use context.	In the paperwork, data on human interaction will be collected and analyzed to generate test inputs, aiming to avoid any simulation problems.

Scripted GUI testing of Android open-source apps: evolution of test code and fragility causes [21]	“Grounded Theory” methodology was used for manual analysis of differential files for test classes, aiming to create a comprehensive classification system for this paper.	The type of investigated open-source app was not considered as a factor.	Open-source apps wireframes were tested in the paper.
Automated Test Generation for Detection of Leaks in Android Applications [22]	Researchers have developed an automated method for finding resource leak bugs in Android applications using a neutral flow of GUI events. The experimental analysis supports the use of these methodologies for efficient, broad, and automated tests.	Automatically generate tests for recurring behaviors in Android GUIs using static control-flow analysis. Improve execution and monitoring, develop test prioritization strategies, and automate major leak detection.	This platform automates testing which is trained by deep learning and ML approach.

Chapter 3

Materials and Method

3.1 Materials

In order to train the model for the black-box testing datasets, TensorFlow, Python, Kaggle, and Rico, mobile app datasets for building data-driven design applications, were used.

- **TensorFlow** is an open-source framework developed by Google primarily for deep learning applications. It also supports traditional machine learning. We used tensor flow for the pre-training model.
- **Python** is commonly used for developing websites and software, task automation, data analysis, and data visualization. We used Python because it is familiar to us and easy to use.
- **Jupyter Notebook** is for sharing computational documents and creating original web applications. We used it to handle the code easily so that it could show the output quickly.
- **Kaggle Notebook** is a cloud-computed notebook used for codes that are computed on their cloud servers. We used it because it is faster than Google Collab and easy to train the model.
- **Django** used for making a web application using our model. It takes image and view types from frontend and processes them for black box and takes input data for the white box.
- **Tailwind** and **DaisyUI** are helping us to make beautiful frontends.
- **Ajax** sits between the frontend and the backend. Maintaining the UX is better. It is sending images and other data to the backend. Then, when the backend sends the processed result, It implements the result on the frontend without a refresh or extra request.

3.1.1 Dataset Collection

The dataset used in this study is called Rico. Mining Android applications created Rico while it was running, using a combination of programmed and manual investigation [21]. The infrastructure used for app mining in Rico does not require access to or modification of an app's source code. Crowd workers accessed the applications through a web interface after downloading them from the Google Play Store. The dataset includes photographs and hierarchies with semantic annotations for Android applications' UI. For Whitebox testing, we considered 22 classes with 17713 data numeric data. And we collected the data from Scripted GUI testing of Android open-source apps: evolution of test code and fragility causes. The data is divided into six files. Because it was created using six testing tools. These six testing tools were performed testing on the open-source application and then recorded data to see the change ratio between app versions.

3.1.2 Dataset Exploration

Rico dataset consists of over 66,000 UI screens and hierarchies, with semantic annotations that describe the meaning and usage of elements on the screen [23]. These annotations provide insights into what different UI components, buttons, and icons signify. Semantic screenshots in Rico utilize a distinctive color scheme, where each class is assigned a unique color. The mapping between semantic concepts and their corresponding colors is documented in three files known as "component_legend.json," which correspond to specific component categories. For this study, approximately 60,600 datasets were available. The datasets were sorted according to the categorized UI screenshots of the apps. The sorting technique employed is known as “crowdsourcing.”

Below Fig 3.1 are user interface displays from the dataset and their respective semantic annotations.

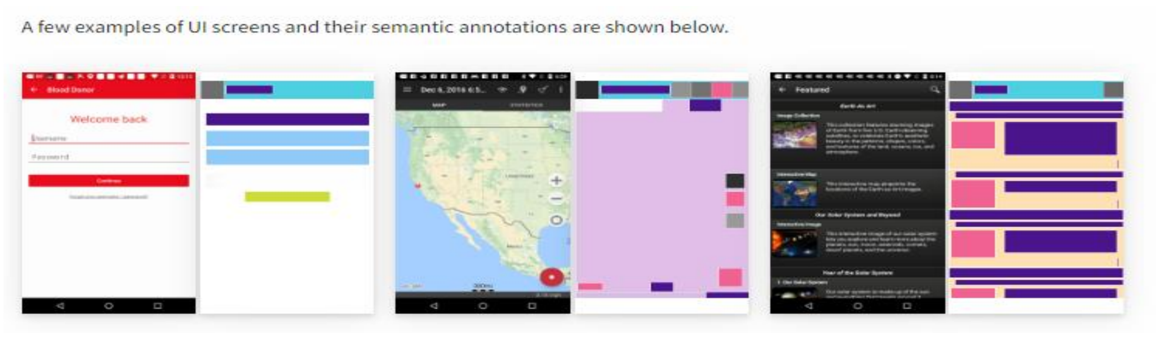

A few examples of UI screens and their semantic annotations are shown below.

Fig 3.1: Screenshot of RICO Dataset

The class we considered for white box testing in the project is shown in the Table 3.1. In the left column, these are the short form of classes for input and output and in the right column is the explanation for understanding the classes.

Table 3.1: List of Classes

Class	Explanation
Plocs	Number of lines of code in the Project
Mod. Plocs	Modified line of code in the project
Tlocs	Total line of code of selected class file
Mod. Tlocs	Modified line of code on Tlocs
Classes	Number of classes in the project
Added Classes	Number of added classes from the previous version
Deleted Classes	Number of deleted classes from the previous version
Mod. Classes	Number of Modified Class
Methods	Total Methods of the Classes
Added Methods	New added method from the previous version
Deleted Methods	Deleted method from the previous version
Mod. Methods	The modified method from the previous version
Classes with Mod. Methods	Number of classes with the modified method
TLR	Test LOCs Ratio
MTRL	Modified Test LOCs Ratio
MRTL	Modified Relative Test LOCs
TMR	Total Method Ratio
MMR	Modified Method Ratio
Tool	Tool to perform the testing

3.1.3 Dataset Sampling

A UI was designed to facilitate sorting datasets based on categories. This UI feature is for managing and organizing datasets. Sorting datasets based on categories can greatly improve the user experience and make it easier for users to find and analyze specific data. It provides a clear structure to the dataset. It reduced the likelihood of errors that may occur during manual data sorting or when using complex commands. The automated process ensures accurate data organization, minimizing potential mistakes that could arise from manual intervention. Semantic annotations provide meaningful labels and metadata to categories, enabling users to comprehend the content and context of each data group effortlessly. This, in turn, expedites data retrieval, making it easier for users to find relevant information quickly. By capturing the inherent meaning and semantics of categories, users can perform advanced queries and filtering operations with greater precision. This capability enables users to uncover deeper insights and identify patterns that might be challenging to detect using conventional sorting methods.

In Fig 3.2, these are the sample wireframes for black box testing. These are different wireframes of different applications from component range low to high sequentially.

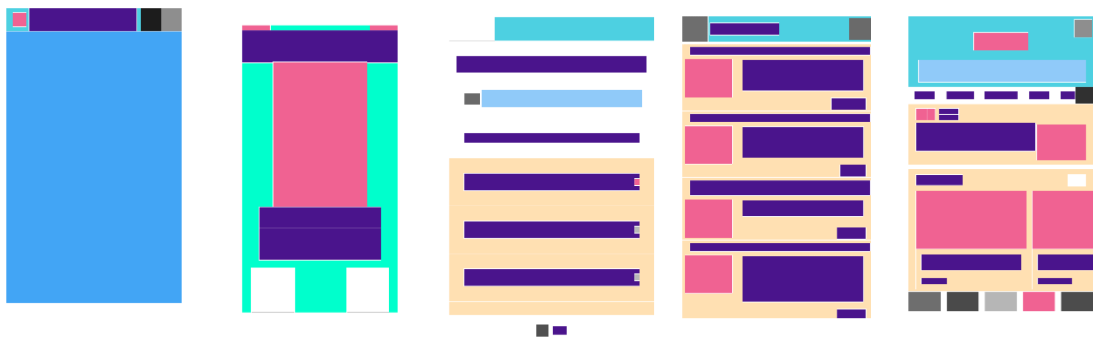


Fig 3.2: Data Sample for Black Box Testing

We merge all data from six files and then assign each row with its corresponding testing tools. Then we remove blank data. And gave all non-numeric data a value. So, we can create the model from it more easily. After more preprocessing, we applied machine learning data. Below Table 3.2 represents a sample dataset for white box testing, where the first 14 data are the input data, and last 8 data, which are colored, are the output data.

Table 3.2: Dataset Sample for White Box Testing

Plocs	1899	1899	20159	67267
Mod. Plocs	58	58	5021	7121
Tlocs	471	471	1746	1265
Mod. Tlocs	6	6	100	77
Classes	1	1	11	11
Added Classes	0	0	0	5
Deleted Classes	0	0	0	0
Mod. Classes	1	1	3	4
Methods	17	17	98	62
Added Methods	0	0	1	27
Deleted Methods	0	0	0	0
Mod. Methods	3	3	12	2
Classes with Mod. Methods	1	1	2	2
Tool	uiautomator	espresso	robolectric	robotium
TLR	0.248025	0.248025	0.086611	0.018806
MTRL	0.012739	0.012739	0.05787	0.103914
MRTL	0.103448	0.103448	0.019916	0.010813
TMR	0.416209	0.416209	0.220071	0.959082
MCR	1	1	0.272727	0.666667

MMR	0.176471	0.176471	0.123711	0.057143
RFCR	1	1	0.666667	0.5
FCR	1	1	0.181818	0.333333

3.1.4 Dataset Processing

JSON files of the layouts were utilized to generate data, which was then classified into five density categories based on the number of components present in the UI.

Fig 3.3 shows the flowchart of dataset preprocessing for black box testing where a png file has the corresponding json file and then from the json file, the number of components gets extracted and classify the image according to it.

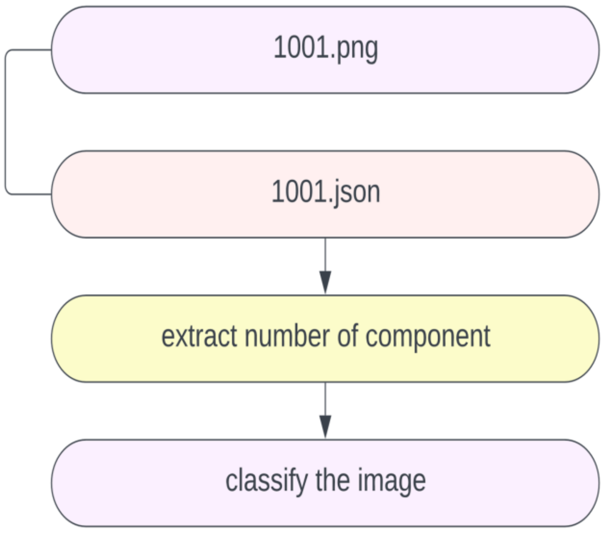


Fig 3.3: Dataset Preprocess of Black Box Testing

Table 3.3: Density Class Example

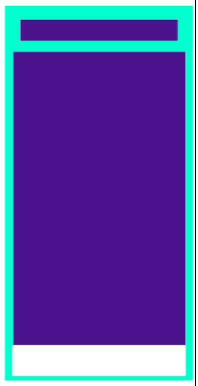
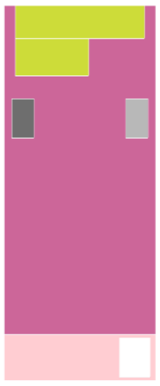
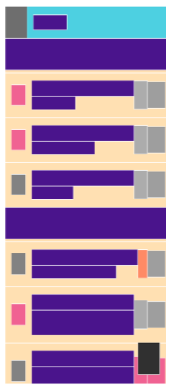
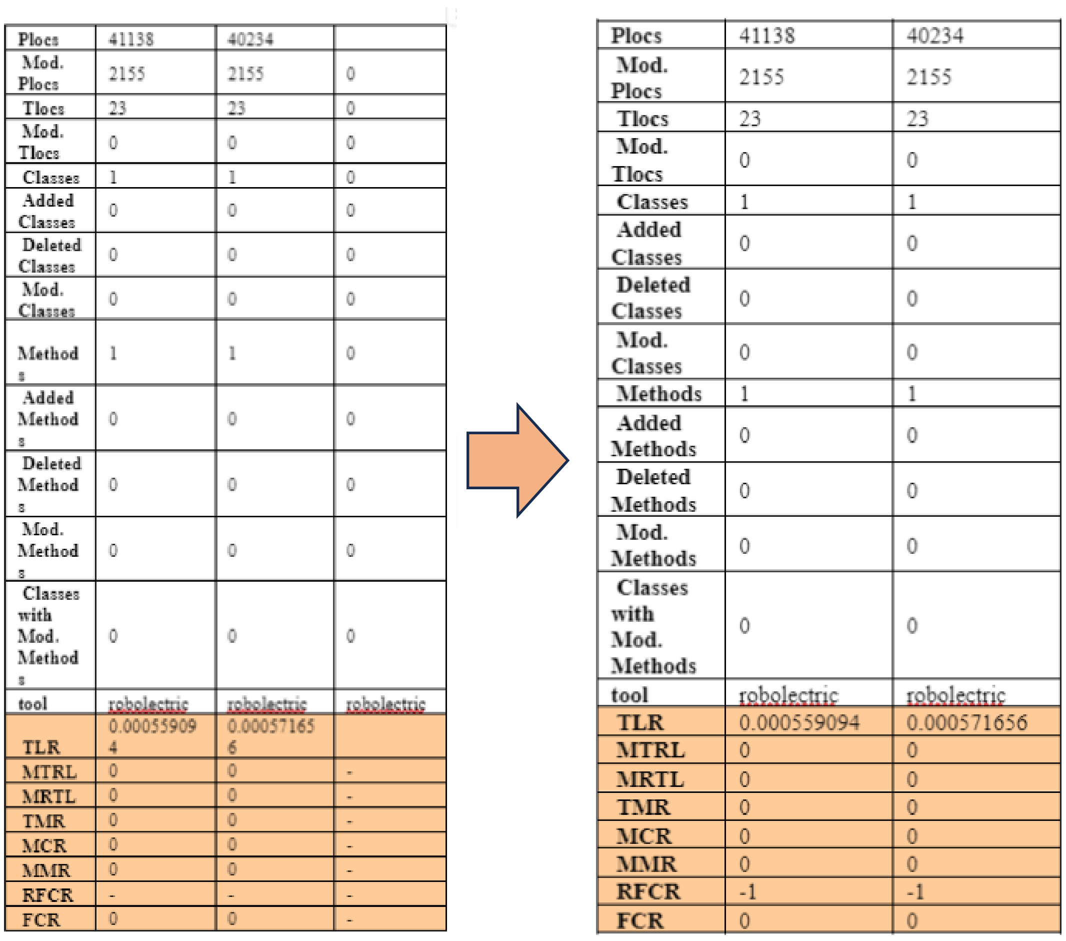
Component Range	1-6	7-11	12-18	19-30	31-60
Density	Very low	Low	Medium	High	Very high
Screenshots					

Table 3.3 shows the component range and their corresponding density type and screenshots. The component range was created based on the number of components presented in a wireframe. Then each component was named by the density type and with the screenshots examples. Multiple methodologies were employed to sort the data, but the final sorting method was chosen based on its reliability compared to other methods.

For Whitebox, all the data is processed by MS Excel, pandas, and sklearn. There are '-' on data. It represented no change had been on the version or value cannot be calculated. So, we changed those using -1. The number of blank rows was meager. So, we just deleted the blank row. Fig 3.4 shows the altered dataset, and each where Plocs, Mod. Plocs, Tlocs, Mod.Tlocs, Classes, Added Classes, Deleted Classes, Mod. Classes, Methods, Added Methods, Deleted Methods, Mod. Methods, Classes with Mod. Methods, tool are the 14-input metrics and TLR, MTRL, MRTL, TMR, MMR are the 8-output metrics.



Plocs	41138	40234	
Mod. Plocs	2155	2155	0
Tlocs	23	23	0
Mod. Tlocs	0	0	0
Classes	1	1	0
Added Classes	0	0	0
Deleted Classes	0	0	0
Mod. Classes	0	0	0
Methods	1	1	0
Added Methods	0	0	0
Deleted Methods	0	0	0
Mod. Methods	0	0	0
Classes with Mod. Methods	0	0	0
tool	robolectric	robolectric	robolectric
TLR	0.000559094	0.000571656	
MTRL	0	0	-
MRTL	0	0	-
TMR	0	0	-
MCR	0	0	-
MMR	0	0	-
RFCR	-	-	-
FCR	0	0	-

Plocs	41138	40234
Mod. Plocs	2155	2155
Tlocs	23	23
Mod. Tlocs	0	0
Classes	1	1
Added Classes	0	0
Deleted Classes	0	0
Mod. Classes	0	0
Methods	1	1
Added Methods	0	0
Deleted Methods	0	0
Mod. Methods	0	0
Classes with Mod. Methods	0	0
tool	robolectric	robolectric
TLR	0.000559094	0.000571656
MTRL	0	0
MRTL	0	0
TMR	0	0
MCR	0	0
MMR	0	0
RFCR	-1	-1
FCR	0	0

Fig 3.4: Before and After of the Dataset Alteration

Fig 3.5 for white box testing, first, it starts with two sets of data: data from the test inputs and data from the test results. Then combining these two sets of data into a single set of data that matches input data with results. After merging, changing every '-' in the dataset with a '-1'. This step is meant to handle values that are lost or can't be found. After that, getting rid of any data points that are blank or empty. This cleaning step makes sure that the file only has information that is useful. Lastly changing all the remaining data in the dataset to a consistent type, especially to floating-point numbers. This step makes sure that the data is in the same format, so it can be analyzed or processed further.

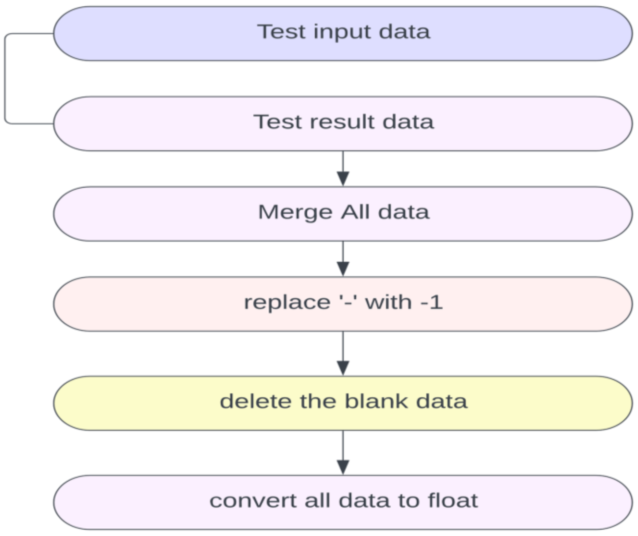


Fig 3.5: Dataset Preprocess of White Box Testing

3.1.5 Research Environment and Devices

A UI was designed to facilitate sorting datasets based on categories. Here is a screenshot of the UI.

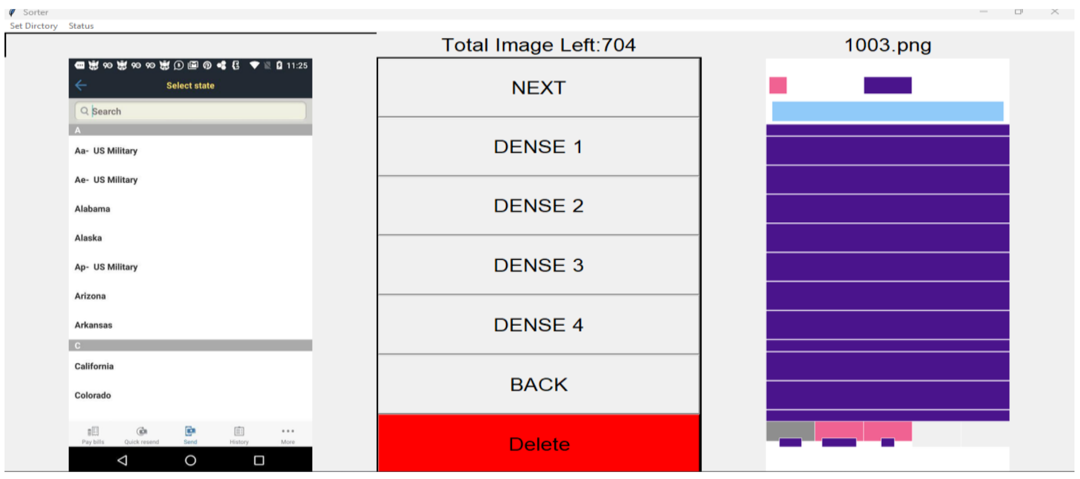


Fig 3.6: Tools for Data Sorting

In Fig 3.6, the user has to set the directory path of the directory that the semantics annotations are in. Then along with the wireframe, corresponding screenshot of the wireframe will also be loaded and then the user can either select the density class just by looking at the congestions in the wireframe or go to the next png file or delete the current png file. After selecting the density four folders will be created which will be named dense 1, dense 2, dense 3, dense 4 and then finally the density class that the user clicked only the wireframe will be stored in the corresponding folder to it of the selected density class.

3.2 Method

In this project, for black box testing, six models have been trained and analyzed to see which model performs better to determine the density of applications' screenshots. For white box testing, a model has been created which can automatically predict and generate the values of the tools such as Robolectric.

3.2.1 Proposed Model

The proposed "DeepTestDroid" model was designed to address the mentioned problems and perform UI testing using a deep learning approach. The implementation of the model utilized MobileNetV3, ResNet50, EfficientNet_b0, EfficientNet_b1, and EfficientNetV2_b0 neural networks for black box testing. Python was used for the performance, and specific libraries were employed to handle the app layouts based on the created categories. And for white box testing XGBRegressor, LinearRegression, ElasticNet, RandomForestRegressor, DecisionTreeRegressor model has been trained and tested to see their performance.

MobileNetV3 is a convolutional neural network architecture widely adopted in various mobile applications, including TensorFlow [24]. It offered the capability to run on embedded systems and was implemented in small and large versions. We used MobileNetV3small and MobileNetV3large.

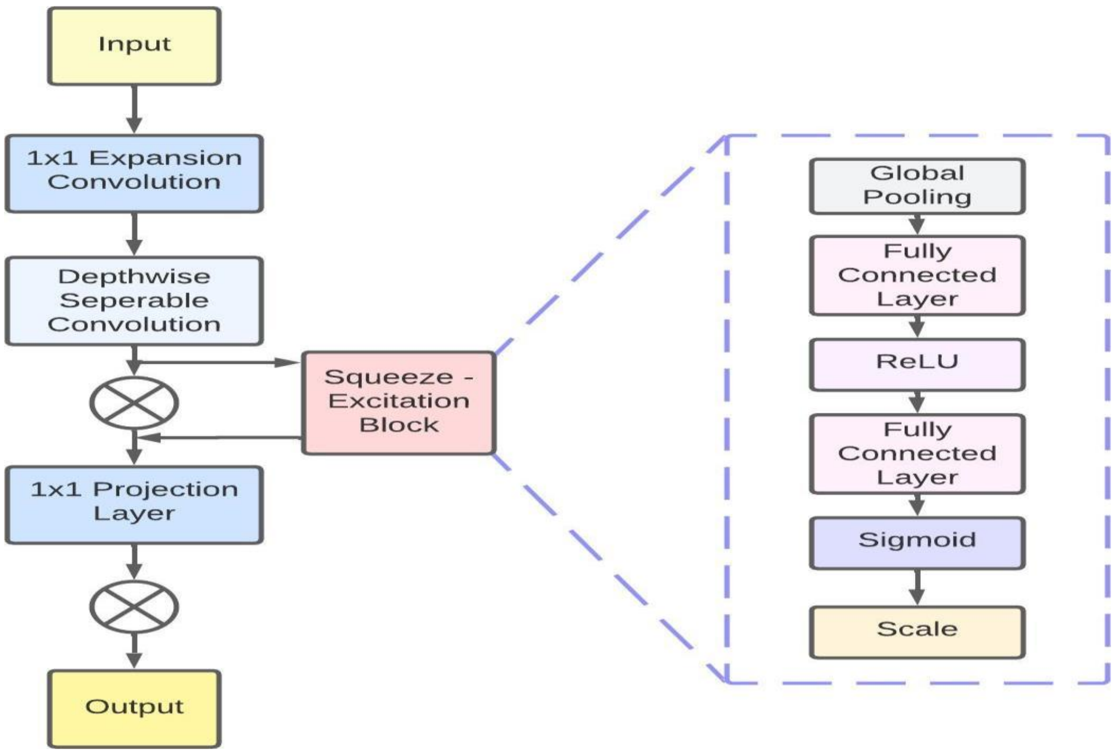


Fig 3.7: MobileNet Architecture

In the above Fig 3.7 the mobile net architecture is shown, where the input of the model was checked by expansion convolutional, and the depth wise projection layer will be detected. The 1×1 expansion block is the core building block of the model. Depthwise separable convolution block significantly reduces computation and parameters while maintaining performance. The SE module block interacts with depthwise separable convolution and projection layer block. The projection layer further transforms the feature maps, projects them into a new space, and generates the output.

ResNet50 is a state-of-the-art convolutional neural network model for image classification. It was trained on ImageNet, a large-scale classification dataset [25].

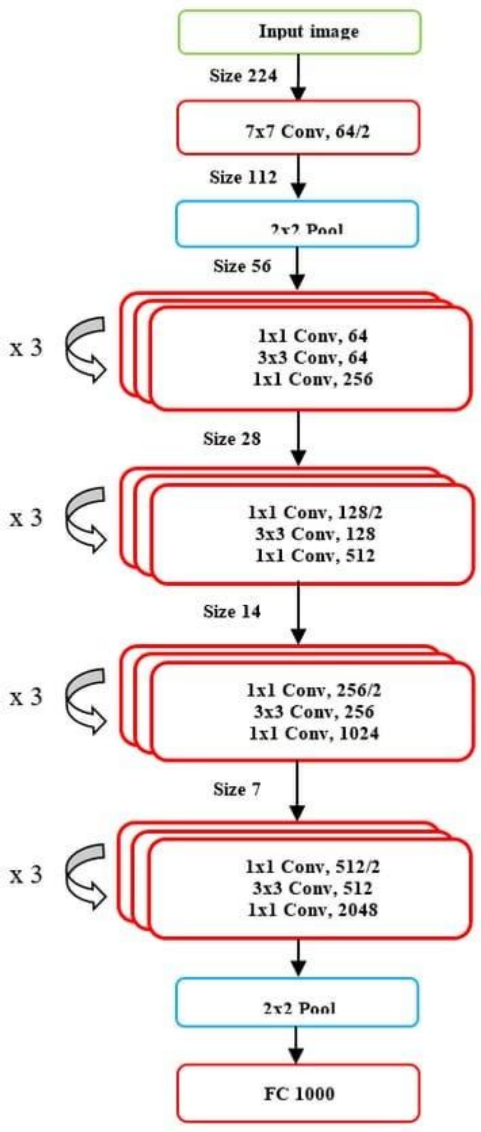


Fig 3.8: ResNet Architecture

In the above Fig 3.8 the ResNet architecture is shown whereas after giving the input there will be checking by zero padding and step by step work after that the output will be average pooling.

EfficientNet_b0 is another convolutional neural network architecture trained on a vast dataset from the ImageNet database [26]. It can classify images into one thousand object categories, including various objects and animals.

EfficientNet_b1 is a scalable convolutional neural network architecture that uniformly scales dimensions using a compound coefficient [27].

EfficientNetV2_b0 is a convolutional neural network architecture that performs highly on image classification tasks while maintaining parameters and computational cost efficiency [28].

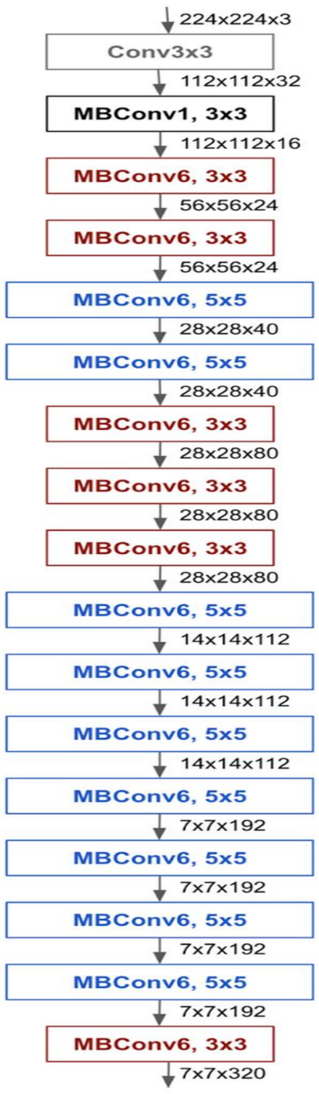


Fig 3.9: EfficientNet Architecture

In Fig 3.9, EfficientNet employs a structural component known as the "Efficient Block," which integrates depthwise separable convolutions with inverted residual connections, akin to the architectural design of MobileNetV2. These blocks are designed to maximize the balance between the size of the model and its performance. The concept of depth scaling pertains to the manipulation of the network's layer count, whereas width scaling involves modifying the number of channels within each layer. Additionally, resolution scaling entails the alteration of the input image dimensions.

For the white box, we can use the ML model. We tested out five models, XGBRegressor, LinearRegression, ElasticNet, RandomForestRegressor, DecisionTreeRegressor and among them XGBRegressor, DecisionTreeRegressor and RandomForestRegressor gave the best result. We used these models by sklearn python learn.

XGBRegressor is a highly effective machine learning model for regression tasks [29]. It is a member of the XGBoost (Extreme Gradient Boosting) family of ensemble learning algorithms, which are renowned for their high predictive accuracy and efficient performance. XGBRegressor is designed to manage regression problems in which it is necessary to predict continuous numerical values.

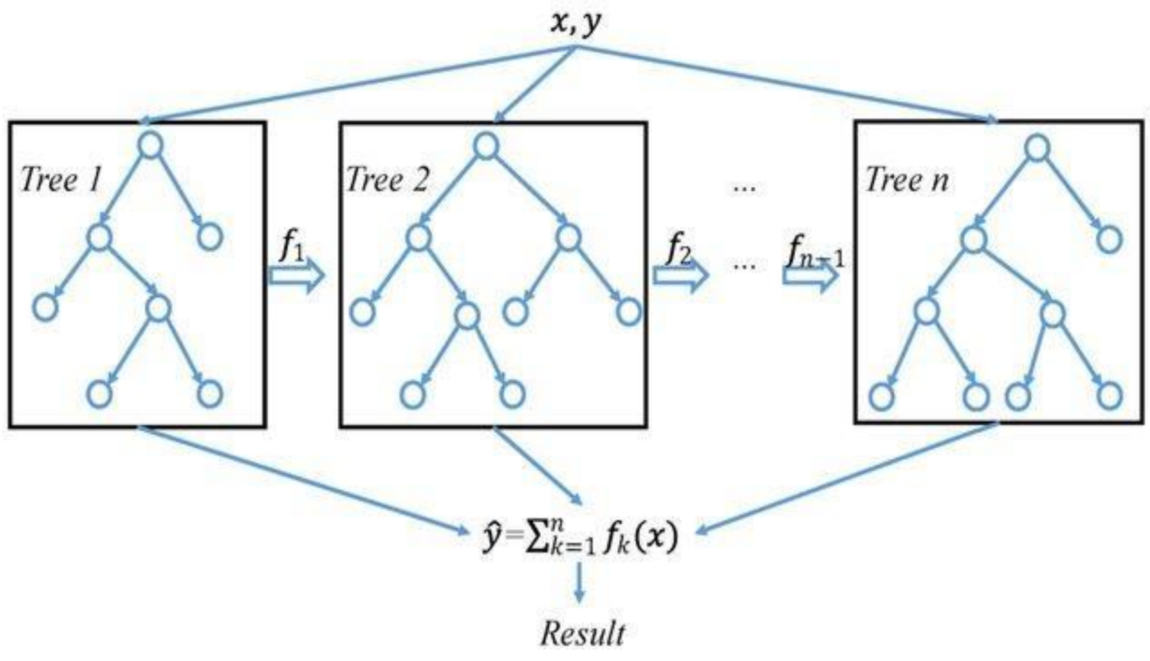


Fig 3.10: XGBoost Architecture

Fig 3.10 shows the architecture of XGBoost, using decision trees as the base learners. These decision trees are known as CART, which stands for Classification and Regression Trees.

In the fields of ML and data analysis **LinearRegression** is a standard and useful statistical method [30]. To predict continuous numerical values from input features, this supervised learning approach is employed. The model presupposes that the input variables have a linear connection with the outcome variable. Finding the line that minimizes the error between predicted and target values is the purpose of Linear Regression.

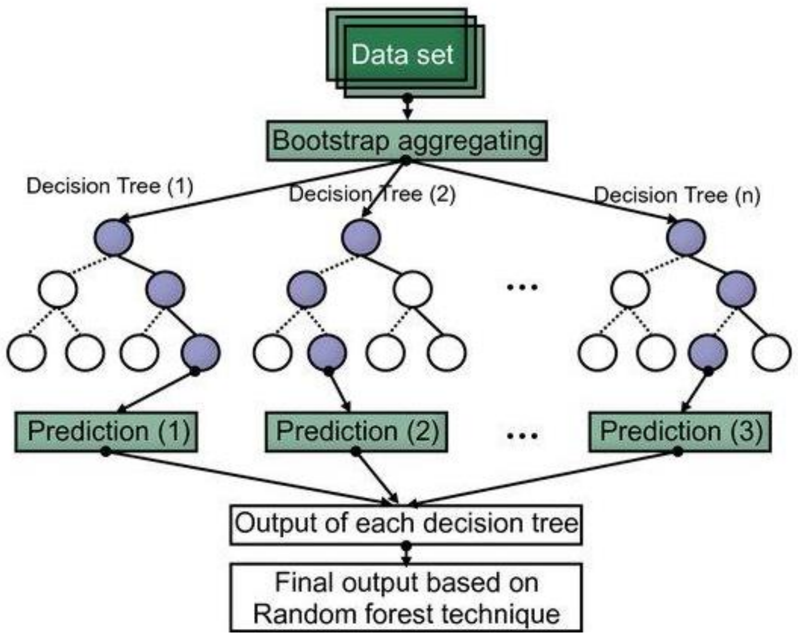


Fig 3.11: LinearRegression Architecture

Fig 3.11 shows the architecture of LinearRegression architecture. Decision Trees and Linear Regression are widely used in predictive modeling to determine patterns and correlations between input data and target variables. Linear regression uses linear equations, while decision trees have hierarchical structures with core nodes indicating decisions and leaf nodes reflecting predictions. These strategies are based on criterion-based data splitting and feature values.

Ridge regularization approaches are frequently employed in machine learning and statistics for applications such as regression and feature selection [31]. By balancing feature selection and regularization, the **ElasticNet** model addresses some of the shortcomings of Lasso and Ridge regression.

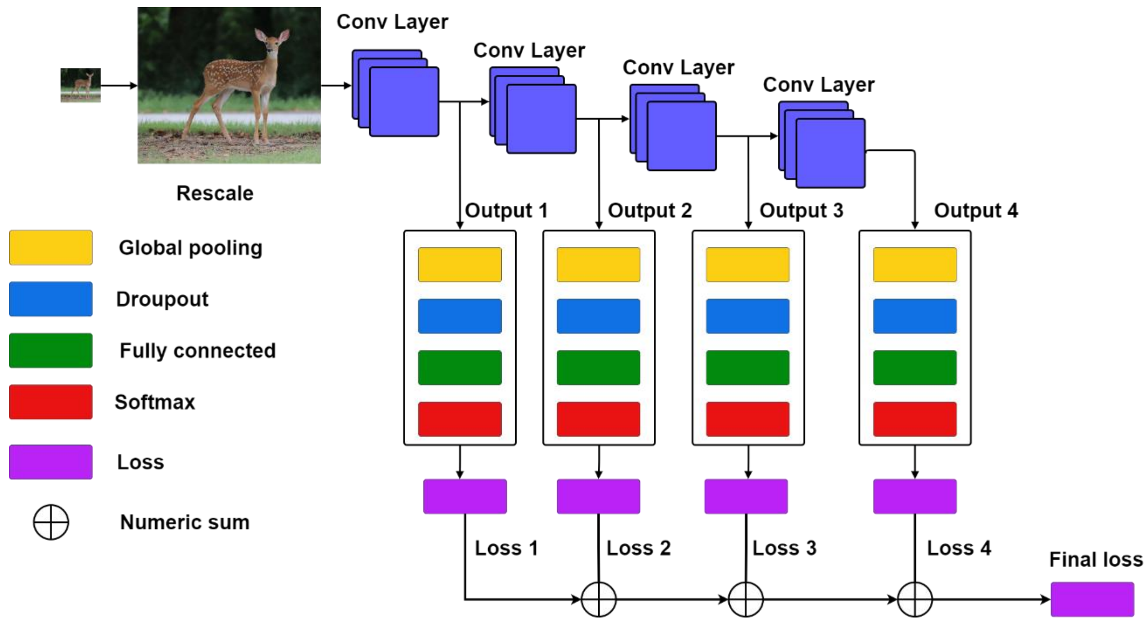


Fig 3.12: ElasticNet Architecture

Fig 3.12 represents the architecture of ElasticNet. CNN feature extraction relies on the convolutional layer to filter incoming data to find patterns, edges, and textures. The output layer predicts these properties using fully connected layers. The loss function measures the gap between expected outputs and target values, driving network learning. Image or segmentation regression tasks use Mean Squared Error (MSE) as a loss function. Final loss is the cumulative value of the specified loss function utilizing anticipated outputs and ground truth values.

RandomForestRegressor is a strong ML model from the ensemble learning family. The RandomForestRegressor algorithm combines the predictions of different decision tree models [32]. Each decision tree is trained using a random subset of the training data and features. When compared to individual decision trees, the randomness and variety in the training process make the model more resilient and less prone to overfitting.

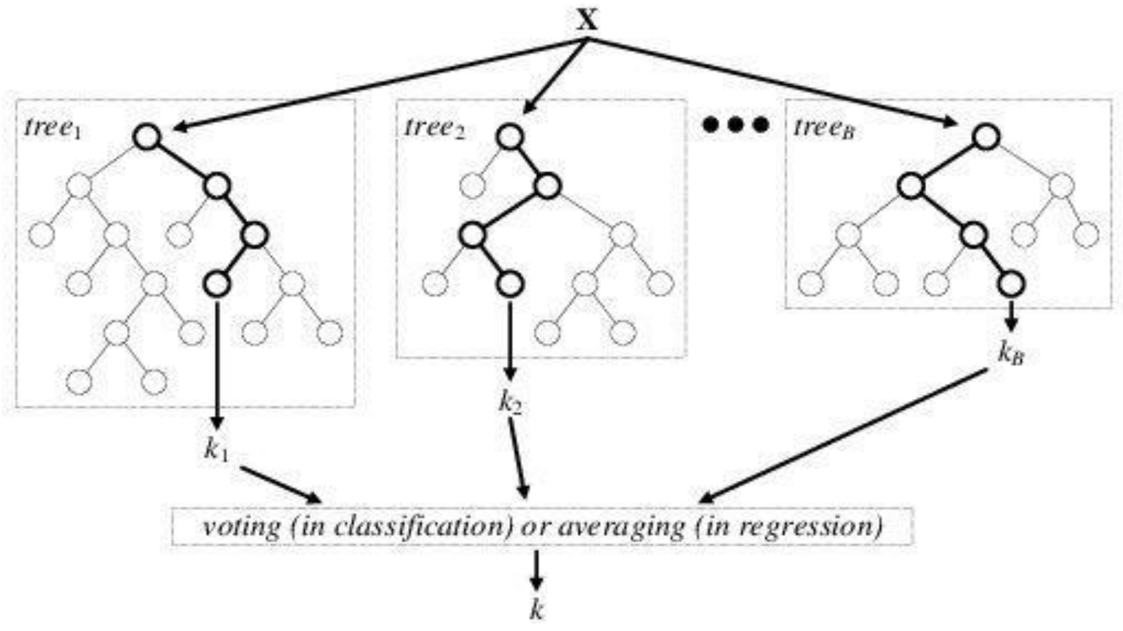


Fig 3.13: RandomForestRegressor Architecture

In the above Fig 3.13 it shows the RandomForestRegressor is a strong ML model from the ensemble learning family. The RandomForestRegressor algorithm combines the predictions of different decision tree models. Each decision tree is trained using a random subset of the training data and features.

DecisionTreeRegressor is a regression-based supervised machine learning model [33]. It is a decision tree method, which means that it divides the feature space into regions and makes predictions based on the average or mean of the target variable within each zone.

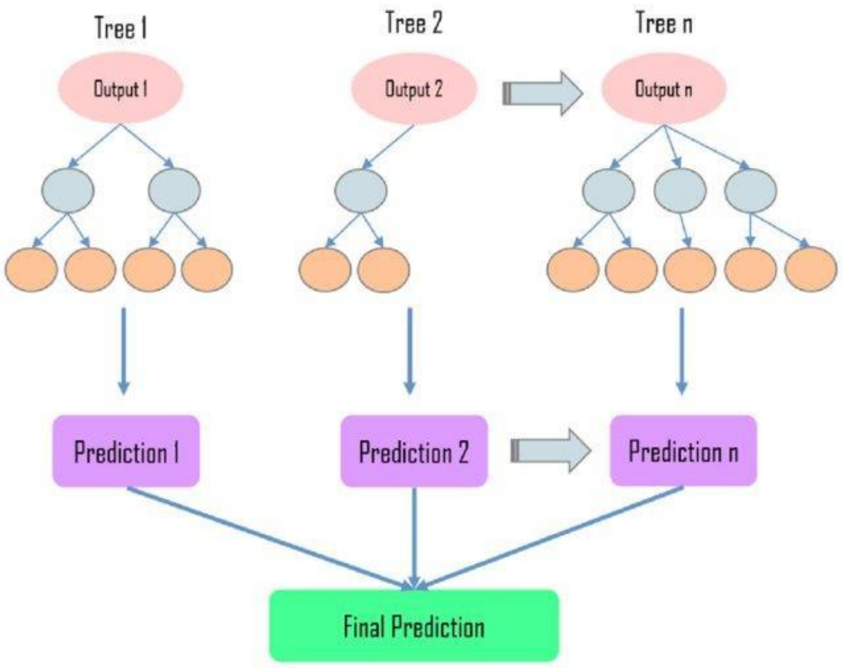


Fig 3.14: DecisionTreeRegressor Architecture

In this Fig 3.14 it shows the DecisionTreeRegressor is a regression-based supervised machine learning model. It is a decision tree method, which means that it divides the feature space into regions and makes predictions based on the average or mean of the target variable within each zone. Here the output of two tree predictions is given by n number of trees which will be the final prediction.

3.2.2 Experiment Setup

We used multiple models in this project. and a few models used for image and other with numeric value. To get the best accuracy out of the models we fine-tune hyperparameters. Below, Table 3.4 shows the configuration used for black box testing's model training.

Table 3.4: Configuration of Black Box Testing

Hyperparameters	**Value(s)**	
Image Size	224 x 224 x 3	
Class Mode	Categorical	
Transfer Learning Weights	ImageNet	
Validation Split	20%	
Pooling	Max-Pooling	
Activation	Hidden Layers	ReLu
	Output Layer	Softmax
Optimizer	Adam	
Loss	CategoricalCrossentropy	
Epoch	100	
Batch size	32	

Fig 3.15 below shows the model configuration we used for all the models we trained and tested for black box testing.

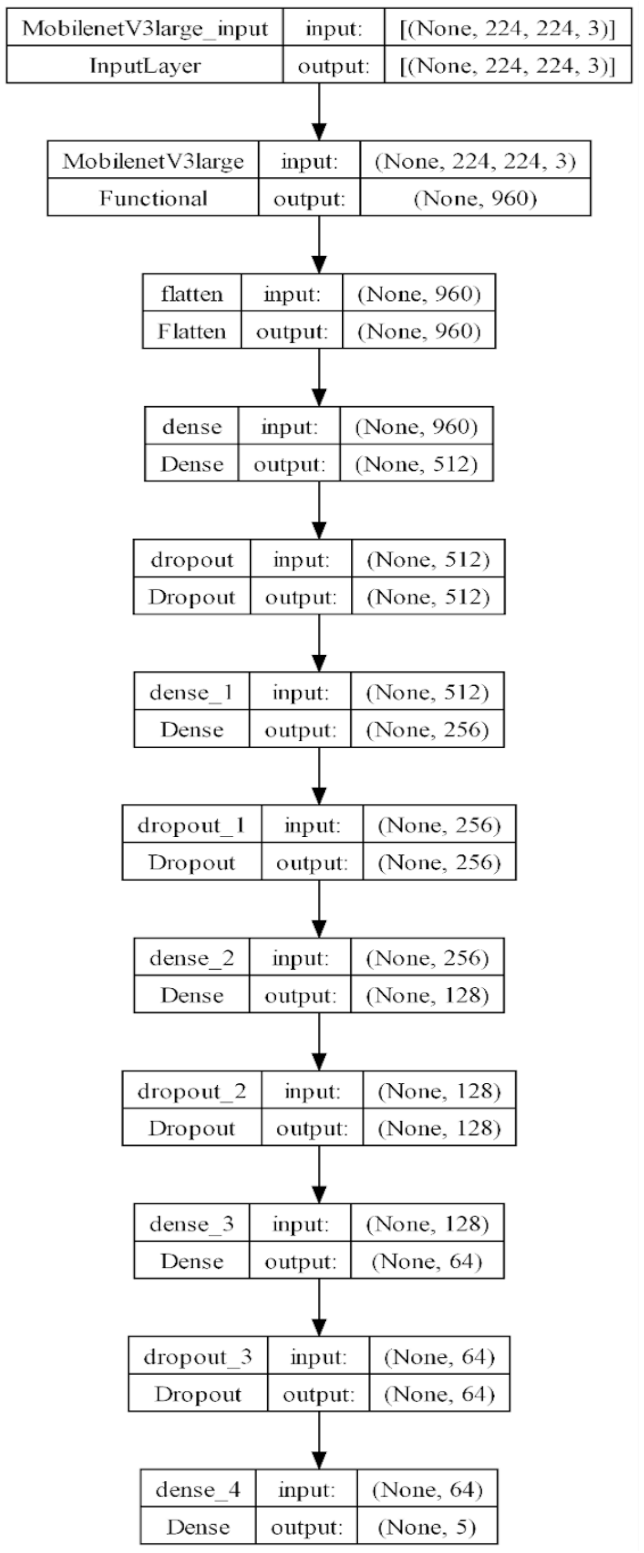

MobilenetV3large_input | input: | [(None, 224, 224, 3)]
InputLayer | output: | [(None, 224, 224, 3)]
MobilenetV3large | input: | (None, 224, 224, 3)
Functional | output: | (None, 960)
flatten | input: | (None, 960)
Flatten | output: | (None, 960)
dense | input: | (None, 960)
Dense | output: | (None, 512)
dropout | input: | (None, 512)
Dropout | output: | (None, 512)
dense_1 | input: | (None, 512)
Dense | output: | (None, 256)
dropout_1 | input: | (None, 256)
Dropout | output: | (None, 256)
dense_2 | input: | (None, 256)
Dense | output: | (None, 128)
dropout_2 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_3 | input: | (None, 128)
Dense | output: | (None, 64)
dropout_3 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_4 | input: | (None, 64)
Dense | output: | (None, 5)

Fig 3.15: Model Configuration of Black Box Testing

And for the white box, test split, max depth and data type are hyperparameters. Here, the test split is 30%, max depth is 8 and data type is float. All of the corresponding models have been trained with the same settings.

3.2.3 Algorithm/Model Formulation

Machine learning and deep learning are part of Artificial intelligence. However, the way both works is different. Machine learning makes decisions about what it learns based on given data. On the other hand, deep learning makes a layer from the given data and then makes its own decision based on that layer. Deep learning is most of the time used for image-type data. Where machine learning works with numeric data, in this project, we have to use both of them because of our dataset. For Whitebox, we had numeric data, so we chose machine learning. Furthermore, for the black box, we had image-type data. Data had to be preprocessed for both black box and white box testing before training the model.

Algorithm: For Blackbox, after experimenting with a few models, we had the best accuracy with MobileNetV3Large. So, we implement the model in our final product. We take the wireframe screenshot and the view type from the user. Then resize the screenshot into a 224 x 224 numpy array. Then predict the dense class through the model. Then we gave suggestions based on view type and the dense class.

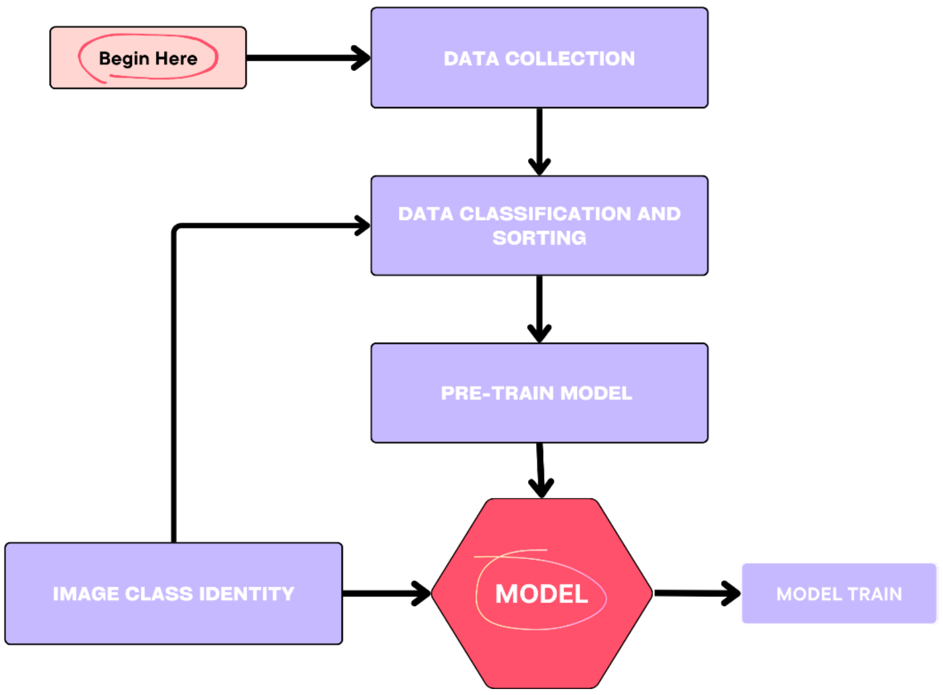


Fig 3.16: Model Creation Flowchart for Black Box Testing

For black box testing, the first step of the model creation is collecting the data. The second step is classifying the data and sorting it according to the classification. The third step is pre-training the model. Image class identity step is essential for both the second step and final step which is model training. Fig 3.16 summarizes all the steps.

We experimented with a few models for the white box too. Then we had the highest R2 score with XGBoostRegressor. So, we used this model in our final product. We take a series of code information alongside which testing tools it may use. Then we convert all the data into a float list. Then through the model, we generate TLR, MTRL, MRTL, TMR, MCR, MMR, RFCR and FCR.

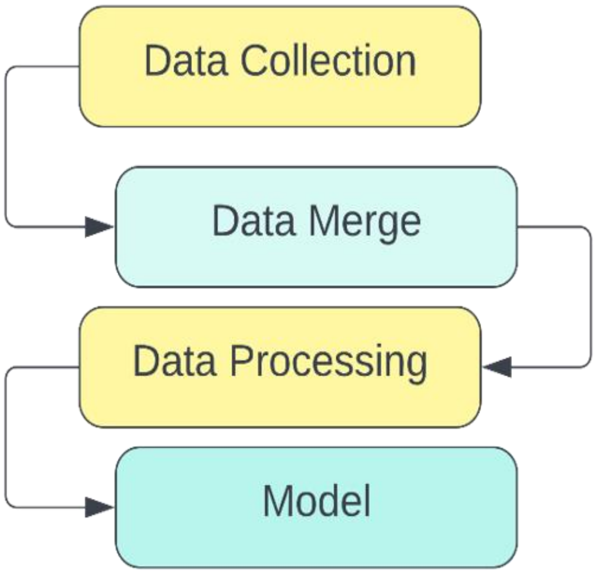


Fig 3.17: Model Creation Flowchart for White Box Testing

For white box testing, the first step is collecting data. The second step is merging all data from six files and assigning each row with its corresponding testing tools. The third step is processing the data after alteration that has been discussed in section 3.1.4 and finally in the last step, training the model with the processed data Fig 3.17 highlights all the steps.

3.2.4 Obtained Results of Models

This study for black box testing shows the comparison among six models, MobileNetV3Large, MobileNetV3Small, EfficientNetB0, EfficientNetB1, EfficientNetV2B0 and ResNET50 models. And for white box testing, comparison between five models, XGBRegressor, LinearRegression, ElasticNet, RandomForestRegressor and DecisionTreeRegressor.

Table 3.5: Results of the Models of Black Box Testing

Name	**Test**		**Validation Accuracy**	**Train Accuracy**
	Loss	**Accuracy**		
MobileNetV3Large	**0.898**	**0.746**	**0.735**	**0.778**
MobileNetV3Small	0.980	0.687	0.681	0.661
EfficientNetB0	0.923	0.712	0.711	0.688
EfficientNetB1	1.003	0.627	0.628	0.611
EfficientNetV2B0	0.942	0.688	0.715	0.642

ResNET50	1.081	0.624	0.596	0.584

In Table 3.5, the ResNet50 model demonstrated the greatest test loss, while the MobileNetV3Large model attained the lowest test loss. The RestNet50 model had the lowest test precision, while the MobileNetV3Large model achieved the highest. ResNet50 performed poorly on the dataset compared to other models evaluated, while MobileNetV3Large performed the best. Despite the fact that all models were trained on the ImageNet dataset, disparities in architecture led to varying data outcomes.

Table 3.6: Results of the Models of White Box Testing

Name	**MSE**	**R2 Score**
XGBRegressor	**0.212**	**0.958**
ElasticNet	0.910	0.123
LinearRegression	0.881	0.192
RandomForestRegressor	0.396	0.873
DecisionTreeRegressor	0.514	0.837

In Table 3.6, the R2 score of XGBRegressor, RandomForestRegressor and DecisionTreeRegressor models performed better than ElasticNet and LinearRegression model as we can see there is a huge difference in the scores. Also, all three regressor models have lower MSE compared to the other two models. So, XGBRegressor model's overall performance is better than both RandomForestRegressor and DecisionTreeRegressor.

3.2.5 Analysis of Models

In this study, every model was trained with various configurations, classes, and datasets, and for the final evaluation, a consistent design, classes, and datasets were used to effectively evaluate all models. 10,000 data points out of the available 60,600 semantic annotations were used to validate the model.

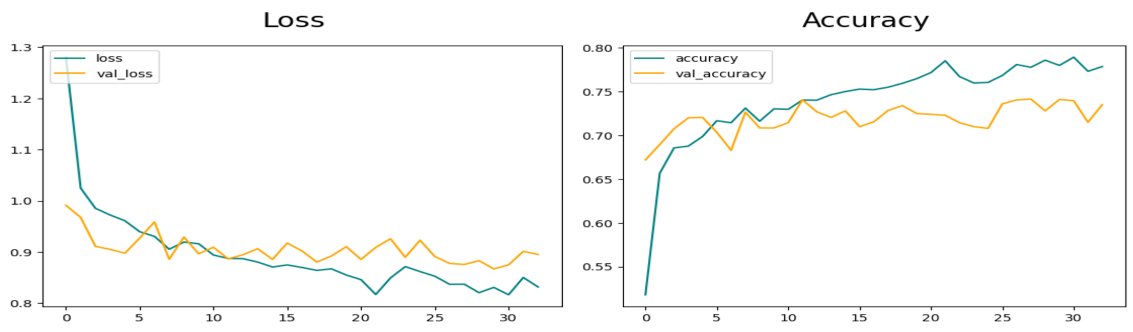


Fig 3.18: MobileNetV3Large Results

Fig 3.18 shows the average training and validation curves for loss and accuracy for MobileNetV3Large. The model performs well on testing and validation data, as we can see no difference between training loss and validation loss.

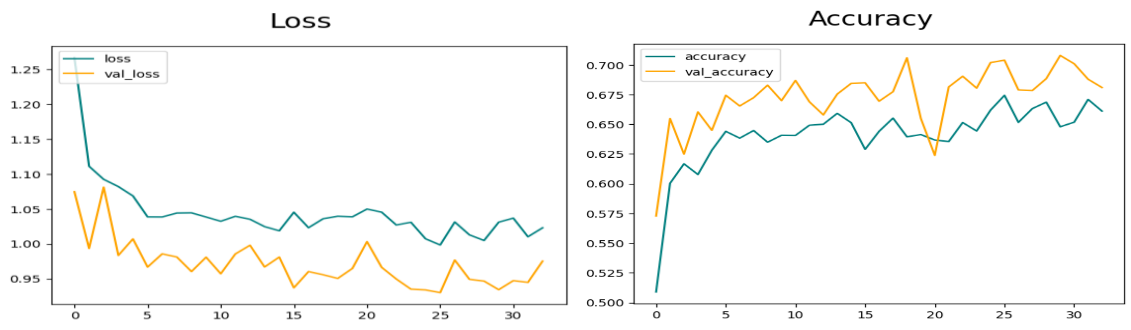


Fig 3.19: MobileNetV3Small Results

Fig 3.19 shows the average training and validation curves for loss and accuracy for MobileNetV3Small. This model's performance is poor as there is a big gap between the training and validation loss.

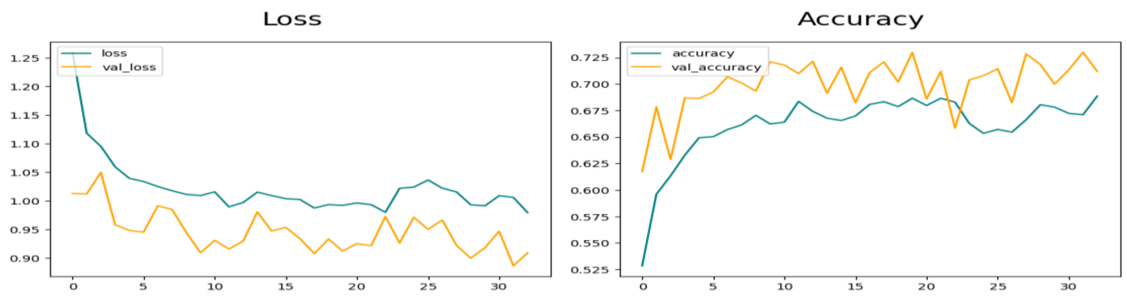


Fig 3.20: EfficientNetB0 Results

Fig 3.20 shows the average training and validation curves for accuracy and loss for EffecientNetB0. The model performs better than MobileNetV3Small as the big gap between the training and validation loss is lesser than the MobileNetV3Small.

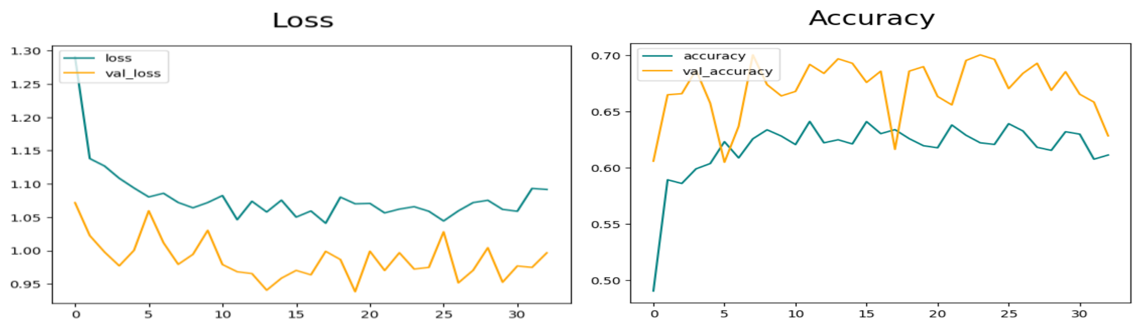


Fig 3.21: EfficientNetB1 Results

Fig 3.21 shows the average training and validation curves for accuracy and loss for EffecientNetB1. The model performs poorer than EfficientNetB0 as the big gap between the training and validation loss is wider than the EfficientNetB0.

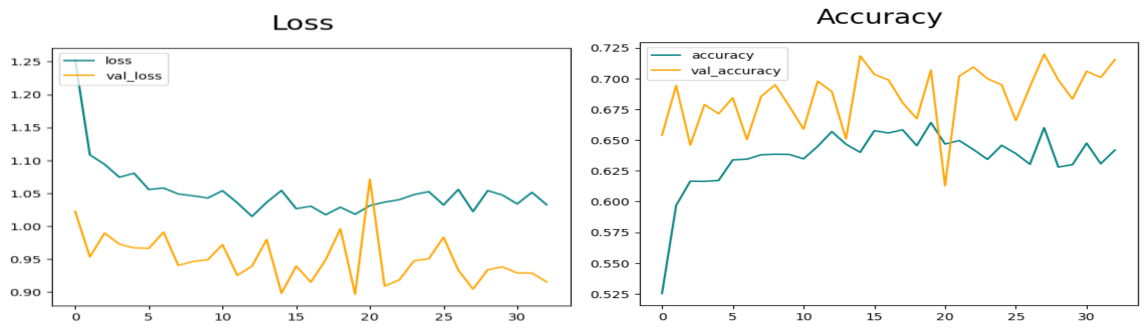


Fig 3.22: EfficientNetV2B0 Results

Fig 3.22 shows the average training and validation curves for accuracy and loss for EffecientNetB1. The model performs better than EfficientNetB1 as the big gap between the training and validation loss is lesser than the EfficientNetB1.

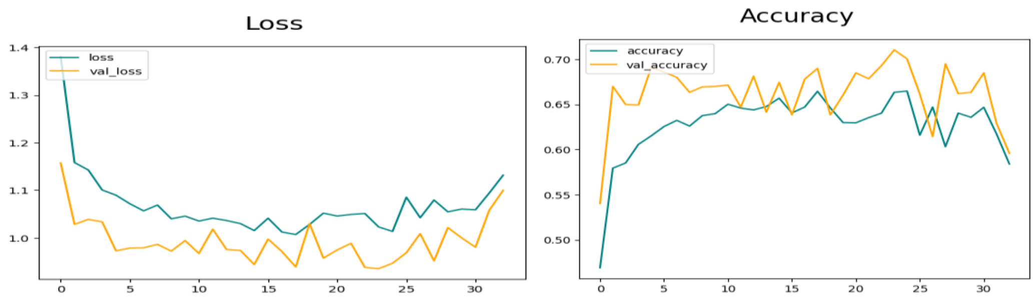


Fig 3.23: ResNET50 Results

Fig 3.23 shows the average training and validation curve for loss and accuracy for ResNET50. The model didn't perform well on validation data from the loss curve, as there is a huge gap between training and validation loss.

3.2.6 Performance Analysis of Models

Multiple deep-learning models were experimented with, including variations that were significantly different from each other. All models were trained using the same configuration, classes, and data. The results from the experiments can be observed in the provided Table or diagram.

The highest test loss was observed in the ResNet50 model, while the lowest test loss was achieved by the MobileNetV3Large model. The RestNet50 model had the most insufficient test accuracy, whereas the MobileNetV3Large model reached the highest. Among the tested models, ResNet50 performed poorly on the dataset, while MobileNetV3Large emerged as the best-performing model. Although all models were trained on the ImageNet dataset, variations in architecture led to differences in data outcomes which can be seen in Fig 3.24.

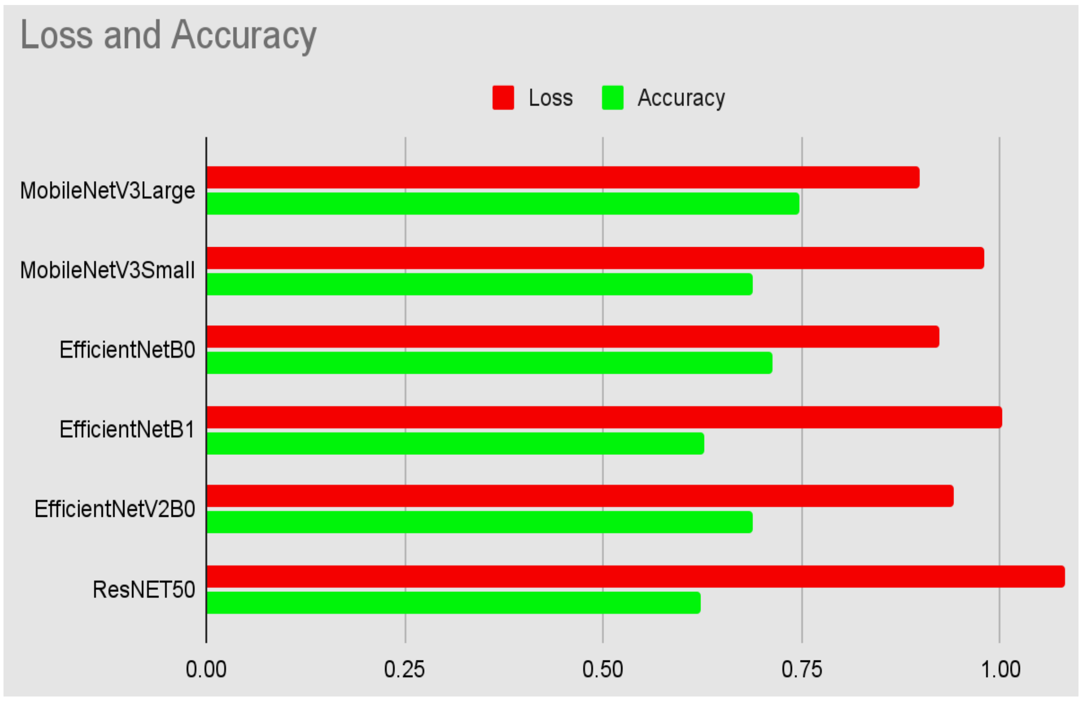


Fig 3.24: Result Comparison of Black Box Testing

In white box testing, we can see that among XGBRegressor, LinearRegression, ElasticNet, RandomForestRegressor, DecisionTreeRegressor; XGBRegressor, RandomForestRegressor, and DecisionTreeRegressor models performed the best because their R2 score is higher and MSE score is lower than other models. The LinearRegression model did not perform well because the dataset and the output are not linear. As, XGBRegressor model has the highest R2 score and lowest MSE, it is the best fit out of them all which can be seen in Fig 3.25.

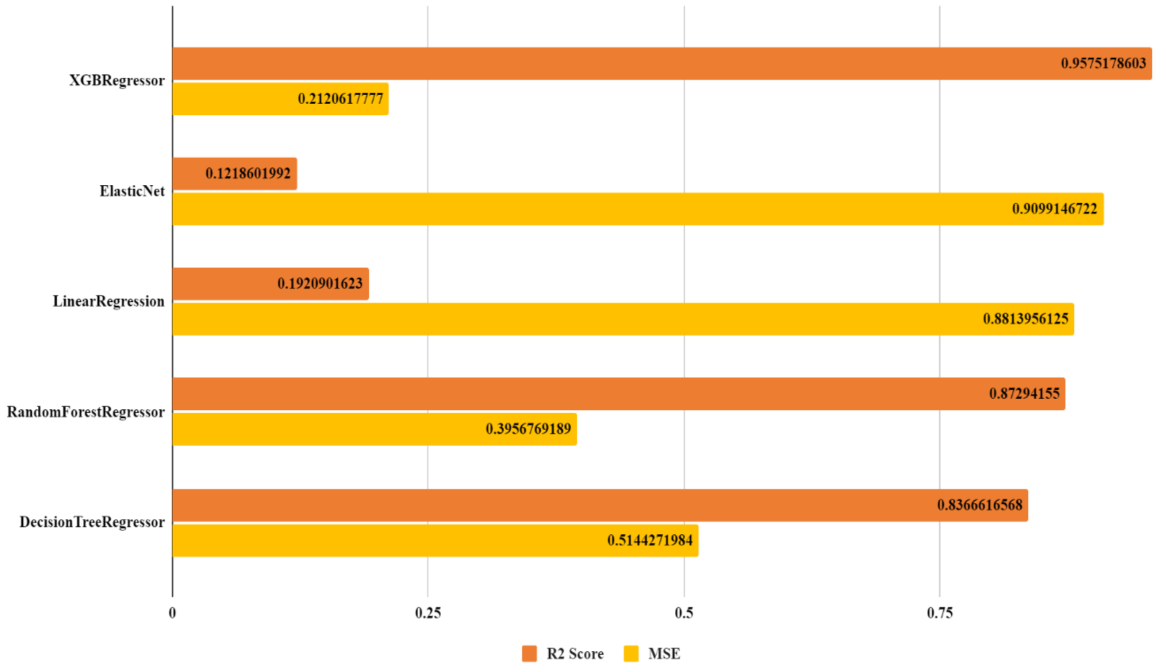


Fig 3.25: Result Comparison of White Box Testing

3.2.7 Design/Framework

In this project, there are two-part Whitebox and Blackbox. For Blackbox, we upload images and then select the view type. Then the image goes through the trained mobileNetV3Large model and classifies its denseness. After classifying, we manually suggest according to the view type and dense class. Fig 3.26 gives a visual representation of the workflow of it.

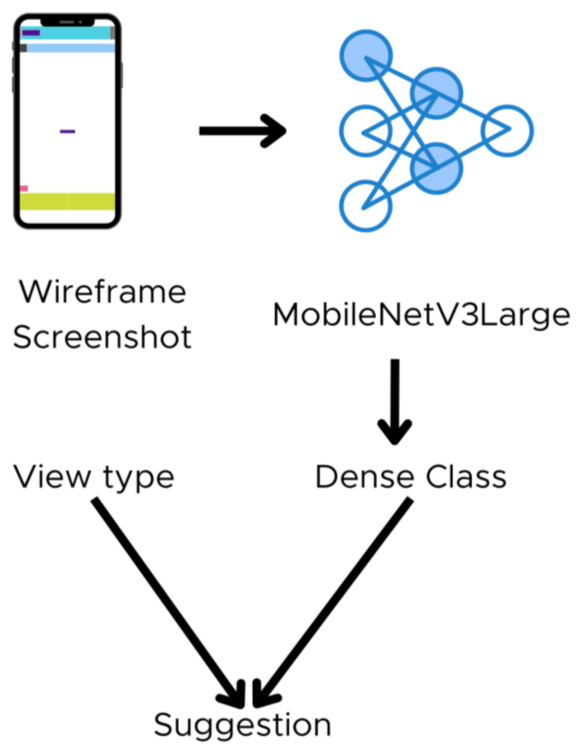


Fig 3.26: Workflow of the Black Box Testing

For the Whitebox, we take the code information and then send that information to the XGBoostRegressor to process. After processing data, the model produces metrics value according to the tool which has been represented in Fig 3.27.

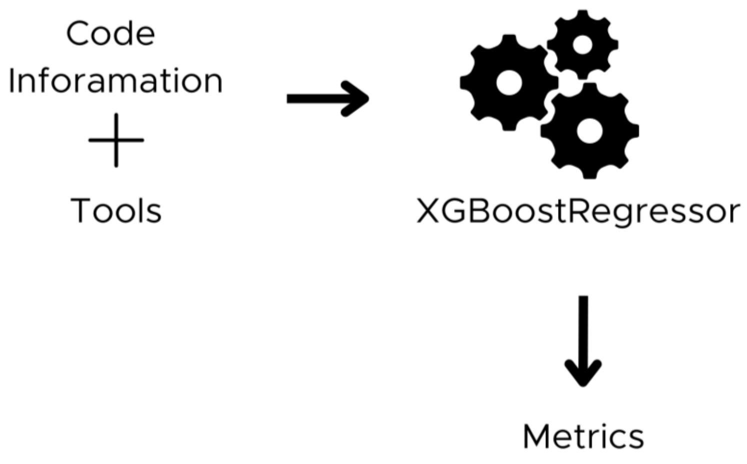


Fig 3.27: Workflow of the White Box Testing

3.3 Summary

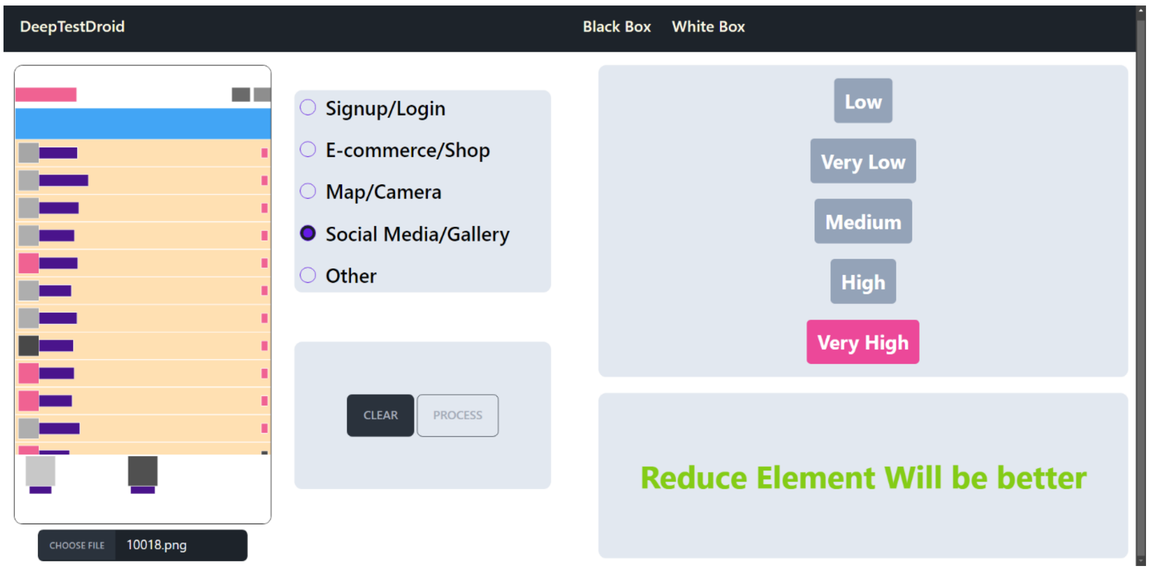
Rico, a dataset for mining Android applications produced through a combination of programmed and manual investigation, is utilized in this study. The infrastructure for app mining in Rico does not require access to the source code, and crowd workers download applications from the Google Play Store and access them via a web interface. The dataset contains images and hierarchies with semantic annotations for Android application user interfaces. Whitebox testing consists of 22 classes with 17713 numeric data, gathered from scripted GUI testing of Android open-source applications. Data is divided into six files, and JSON layout classification files are generated. The final sifting technique was selected due to its dependability. The Deep learning-based "DeepTestDroid" model was designed to address these issues and conduct UI testing. The performance of the model is determined by MobileNetV3, ResNet50, EfficientNet_b0, EfficientNet_b1, and EfficientNetV2_b0 neural networks and Python. For white box testing, the XGBRegressor, LinearRegression, ElasticNet, RandomForestRegressor, and DecisionTreeRegressor models were trained and evaluated. ResNet50 had the highest test loss and lowest test accuracy, while **MobileNetV3Large** had the best. Deep learning black box testing and white box testing are used to test app density UI screenshots. The **XGBRegressor** model provided the best fit due to its higher R2 score and lower MSE, indicating its superior overall efficacy. These models are trained on ImageNet, but architecture affects data results.

Chapter 4

Results and Discussion

4.1 Project Prototype

The prototype is an early version of a final product. We created a website as our prototype to prove the motivation of this project [34]. We used Django as a web framework. We chose Django because it utilizes Python for its instruction. Our model was created using Python, giving much more flexibility than many other frameworks. We also used sklearn, tensorflow, keras, numpy, pandas and many more libraries for this project. We use Tailwind and daisyUI for the front end and Ajax for passing the value between the front and back end. Our website has two sections, a black box, and a white box. In the black box section, we have an image input field and a view type radio button to select the view type. After selecting the image and type, input data will pass to the backend if the user presses the process button. Then the suggestion and the message were sent to the frontend from the backend. For the white box tab, we have a dropdown of the testing tools and 14 number input field. After submitting the field, the generated value will be on the window's right side. Fig 4.1 and Fig 4.2 is the UI of the website which has been built for user to perform the black box as well as the white box testing in the same platform. The UI has been designed to make it as user friendly as possible.



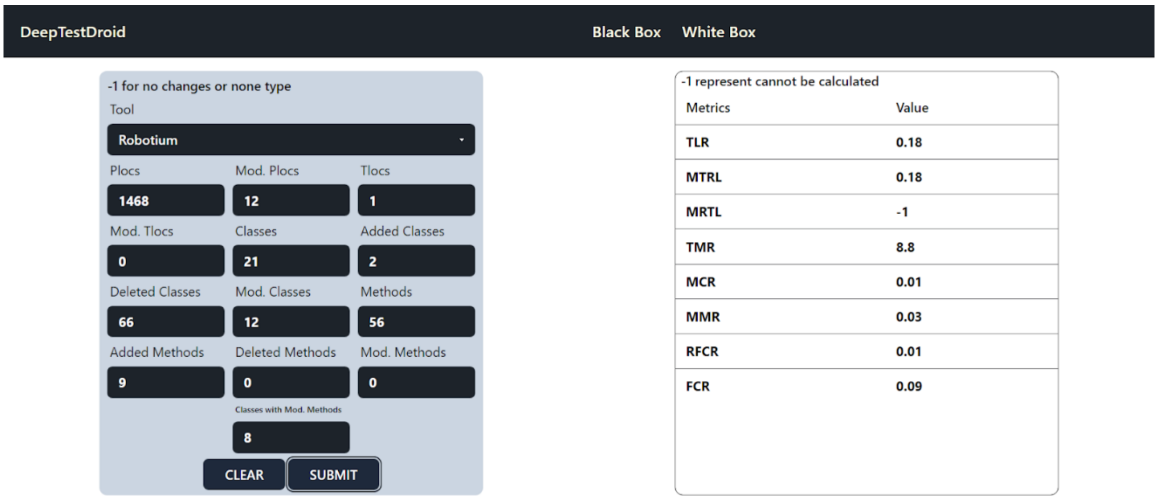
10018.png

Signup/Login
E-commerce/Shop
Map/Camera
[✓ Social Media/Gallery]
Other

Very High

Reduce Element Will be better

Fig 4.1: Frontend of the Black Box Testing



-1 for no changes or none type

Tool: Robotium		
Plocs: 1468	Mod. Plocs: 12	Tlocs: 1
Mod. Tlocs: 0	Classes: 21	Added Classes: 2
Deleted Classes: 66	Mod. Classes: 12	Methods: 56
Added Methods: 9	Deleted Methods: 0	Mod. Methods: 0
	Classes with Mod. Methods: 8	

-1 represent cannot be calculated

Metrics	Value
TLR	0.18
MTRL	0.18
MRTL	-1
TMR	8.8
MCR	0.01
MMR	0.03
RFCR	0.01
FCR	0.09

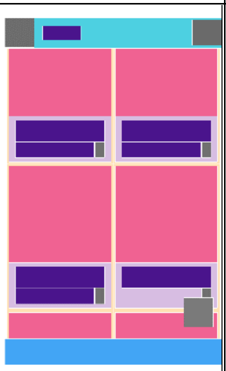
Fig 4.2: Frontend of the White Box Testing

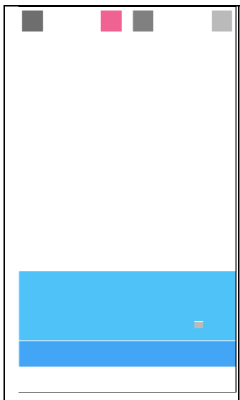
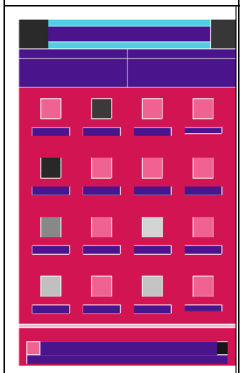
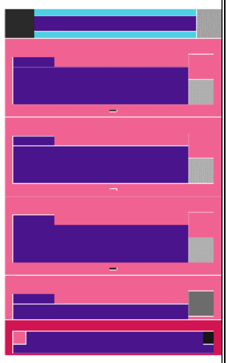
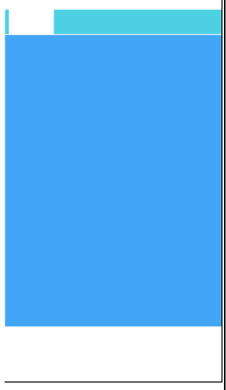
4.2 Obtained Results

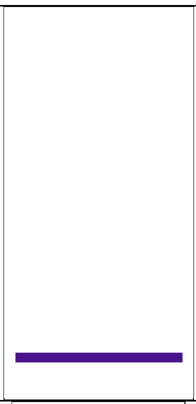
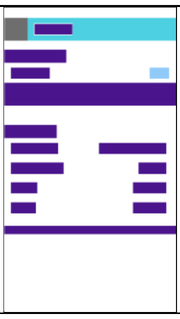
To test the prototype, both black box and white box testing have been performed. For black box testing, the MobileNetV3Large model and the XGBRegressor model have been chosen based on their performance, which has been elaborated in Chapter 3.

In Table 4.1 every screenshot of wireframe is the input. The same input images were evaluated against multiple category types which is Signup/Login, E-commerce/Shop, Map/Camera, Social Media/Gallery and Other which will be selected by the user. The output of each input image has been represented from the second to the last column. The second column represents the density class, which can be very low, low, medium, high, or very high, as determined by our model. Then different suggestions for each signup or login, e-commerce or shop, map or camera, social media or gallery, and other categories have been generated manually, which are shown from the third column to the last for each input image.

Table 4.1: Results of Black Box Testing

Input Images	**Density Class**	**Signup/Login Suggestion**	**E-commerce/Shop Suggestion**	**Map/Camera Suggestion**	**Social Media/Gallery Suggestion**	**Other Suggestion**
	Very High	Reducing Element Will be better	View is Fine	Reducing Element Will be better	Reducing Element Will be better	Reducing Element Will be better

	Low	View is Great	You can Add more elements	View is Great	View is Fine	If the purpose of the view is showing data, then adding more elements will be good. otherwise , fine.
	Very High	Reducing Element Will be better	View is Fine	Reducing Element Will be better	Reducing Element Will be better	Reducing Element Will be better
	High	Reducing Element Will be better	View is Fine	Reducing Element Will be better	View is Fine	If it's a landing page, then reducing the element will be better. Otherwise , fine
	Low	View is Great	You can Add more element	View is Great	View is Fine	If the purpose of the view is showing data, then adding more elements will be good. otherwise , fine.

	Very Low	View is Great	You can Add more elements	View is Fine	You can add more element	If the View has lots of text, then reduce it. adding more element will be good.
	Medium	View is Fine	View is Great	View is Fine	View is Great	View is fine

For white box testing, we checked the same input for all testing tools. In Table 4.2, the first column represents all 13-input metrics. Five inputs have been put to the test with selected existing tools such as Espresso, Robolectric, Robotium, and Uiautomator.

Table 4.2: Inputs of White Box Testing

Metrics	**A**	**B**	**C**	**D**	**E**
Plocs	1899	67267	20159	20159	3575
Mod. Plocs	58	7121	5021	5021	-1
Tlocs	471	1265	1746	1746	0
Mod. Tlocs	6	77	100	100	-1
Classes	1	11	11	11	-1
Added Classes	0	5	0	0	-1
Deleted Classes	0	0	0	0	-1
Mod. Classes	1	4	3	3	-1
Methods	17	62	98	98	-1
Added Methods	0	27	1	1	-1
Deleted Methods	0	0	0	0	-1
Mod. Methods	3	2	12	12	-1
Classes with Mod. Methods	1	2	2	2	-1

In Table 4.3 and 4.4, for every input with each selected tool, there are different outputs of eight different metrics, which are predicted by the XGBRegressor model.

Table 4.3: Tool Specified Outputs of White Box Testing

	Espresso				Robolectric			
A	TLR	0.25	MTRL	0.01	TLR	0.25	MTRL	0.01
	MRTL	0.11	TMR	0.19	MRTL	0.11	TMR	0.19
	MCR	1	MMR	0.17	MCR	1	MMR	0.17
	RFCR	1	FCR	1	RFCR	1	FCR	1
B	TLR	0.25	MTRL	0.01	TLR	0.25	MTRL	0.01
	MRTL	0.11	TMR	0.19	MRTL	0.11	TMR	0.19
	MCR	1	MMR	0.17	MCR	1	MMR	0.17
	RFCR	1	FCR	1	RFCR	1	FCR	1
C	TLR	0.25	MTRL	0.01	TLR	0.25	MTRL	0.01
	MRTL	0.11	TMR	0.19	MRTL	0.11	TMR	0.19
	MCR	1	MMR	0.17	MCR	1	MMR	0.17
	RFCR	1	FCR	1	RFCR	1	FCR	1
D	TLR	0.25	MTRL	0.01	TLR	0.25	MTRL	0.01
	MRTL	0.11	TMR	0.19	MRTL	0.11	TMR	0.19
	MCR	1	MMR	0.17	MCR	1	MMR	0.17
	RFCR	1	FCR	1	RFCR	1	FCR	1
E	TLR	0.25	MTRL	0.01	TLR	0.25	MTRL	0.01
	MRTL	0.11	TMR	0.19	MRTL	0.11	TMR	0.19
	MCR	1	MMR	0.17	MCR	1	MMR	0.17
	RFCR	1	FCR	1	RFCR	1	FCR	1

Table 4.4: Tool Specified Outputs of White Box Testing

	Robotium				Uiautomator			
A	TLR	0.25	MTRL	0.01	TLR	0.25	MTRL	0.01
	MRTL	0.11	TMR	0.19	MRTL	0.11	TMR	0.19
	MCR	1	MMR	0.17	MCR	1	MMR	0.17
	RFCR	1	FCR	1	RFCR	1	FCR	1
B	TLR	0.25	MTRL	0.01	TLR	0.25	MTRL	0.01
	MRTL	0.11	TMR	0.19	MRTL	0.11	TMR	0.19
	MCR	1	MMR	0.17	MCR	1	MMR	0.17
	RFCR	1	FCR	1	RFCR	1	FCR	1
C	TLR	0.25	MTRL	0.01	TLR	0.25	MTRL	0.01
	MRTL	0.11	TMR	0.19	MRTL	0.11	TMR	0.19
	MCR	1	MMR	0.17	MCR	1	MMR	0.17
	RFCR	1	FCR	1	RFCR	1	FCR	1
D	TLR	0.25	MTRL	0.01	TLR	0.25	MTRL	0.01
	MRTL	0.11	TMR	0.19	MRTL	0.11	TMR	0.19
	MCR	1	MMR	0.17	MCR	1	MMR	0.17
	RFCR	1	FCR	1	RFCR	1	FCR	1
E	TLR	0.25	MTRL	0.01	TLR	0.25	MTRL	0.01
	MRTL	0.11	TMR	0.19	MRTL	0.11	TMR	0.19
	MCR	1	MMR	0.17	MCR	1	MMR	0.17
	RFCR	1	FCR	1	RFCR	1	FCR	1

4.3 Discussion

This work is based on black box and white box testing using deep learning method. The work has come to a solution of testing the UI screenshots of apps density using black box testing. The accuracy of the models for black box testing is far better than expected but it can be more developed by training the models again. The white box testing model can eliminate the need for other testing tool as it can predict the values of the existing tool within the DeepTestDroid platform.

4.4 Summary

Black box and white box tests were done on the prototype. Chapter 3 describes the performance of the MobileNetV3Large and XGBRegressor models, which were chosen for black box testing. In Table 4.1, every wireframe screenshot is input. The same input photographs were tested against Signup/Login, E-commerce/Shop, Map/Camera, social media/Gallery, and Other, which the user will choose. Each input image's output is shown in the second to last column. Our model determines the class in the second column: very low, low, medium, high, or very high. Then manually created choices for signup or login, e-commerce or shop, map or camera, social media or gallery, and other categories are shown from the third column to the last for each input image. All testing tools checked the same input for white box testing. The first column of Table 4.2 lists all 13 input metrics. Five inputs were tested with Espresso, Robolectric, Robotium, and Uiautomator. The XGBRegressor model predicts eight metrics for each input with each tool in Table 4.3. Deep learning is used for black box and white box testing in this work. The work uses black box testing to test app density UI screenshots. The black box testing models' accuracy is better than expected, but training them again can improve it. The white box testing model is capable of predicting the values of existing testing tools on DeepTestDroid, eliminating the need for extra tools.

Chapter 5

Conclusion

5.1 Overall Contributions

Black box testing is a strong testing method since it tests a system from beginning to end. A tester can replicate user action and check to see if the system fulfills its promises, just as end users don't care how a system is written or designed and expect to get a suitable response to their requests. A black box test assesses every relevant subsystem along the route, including the UI/UX, database, dependencies, and integrated systems, as well as the web server or application server. Deep learning-based app testing solutions can help development teams perform both black-box and white-box testing with more efficiency during the design and analysis phases.

This study is based on deep learning to develop a deep learning-based system to forecast exam results. This paper uses an innovative technique by combining deep insight and black-box testing. Six black boxes and test data must be produced for a deep learning algorithm. The goal is to use the deep learning algorithm to make the most accurate forecast possible, ensuring the dependability, performance, and longevity of Android applications. Based on the quantity of components in the UI layout, five categories must be created for automated UI testing. These categories must be used to order the data. The sorted data can be used to train a model. Finally, a framework for UI testing of Android applications will be created, classifying the layouts into the appropriate groups. such as EfficientNetB3, ResNet50, and MobileNetV3, are used so the model continuously improves. This project's model will generate the values of existing testing tools for white box testing, thereby reducing the need for multiple testing tools. In order to determine the most effective model, the XGBRegressor, LinearRegression, ElasticNet, RandomForestRegressor, and DecisionTreeRegressor models have been evaluated.

This study provides quite an acceptable accuracy rate for black-box testing UI by training the models accordingly. Testing manually is time-consuming, so with the deep learning method, it decreases a lot of hassle. Therefore, this study will benefit the developers who work on testing the apps with UI screenshots as well as reduce the need for using other testing tools to perform white box testing.

5.2 Limitations and Future Works

The limited dataset is one of the study's primary limitations. We experimented with multiple deep learning models. Some of the variations of themselves Some of them are completely different from each other. Here, every single model ran with the same configuration, classes, and data. From the top table, or diagram, we can see multiple things. Our highest test loss was in ResNet50. And the lowest was MobileNetV3Large. Our lowest test accuracy was RestNet50. And the highest test accuracy was MobileNetV3Large. For our dataset, ResNet50 was the worst model. But our **best model was MobileNetV3Large**. even though they are all pre-train models on the ImageNet dataset. But they still produce variations of data because of their architecture.

We have approximately 60,600 datasets, but for our model we have used 10,000, which were categorized by the layouts of the apps and divided into 4 categories of layout density and labeled as density 1 to 4. We sorted these datasets according to our categorized UI screenshots of the apps. This data sorting technique is called the crowdsourcing technique. The quality of the images is another drawback of our research. Since the photographs are pulled from many web sources, their quality is considered when choosing them. Another limitation of our study is the quality of the images. The majority of the photos were taken in close proximity and with lots of light. Users who provide images that were taken in low light or at a great distance might not provide an accurate prediction. In that instance, the user's capacity to capture an accurate image of the skin will be crucial to testing.

Finally, as part of the future work for this experiment, we want to add more data to the dataset and categorize more photographs from Android applications. With photographs taken from various perspectives and lighting conditions, we hope to produce a dataset that is more adaptable. Furthermore, training models often take a lot of time. As more photographs are added to the collection, the processing time will increase noticeably. In this case, model training can be accelerated by using distributed file systems. We intend to integrate white-box testing using deep learning in the future because the development is currently on hold owing to a lack of datasets and available time. More architectures, including InceptionResNetV2 and InceptionV3, will be included in the future to confirm and contrast our current findings and offer a better solution.

Bibliography

1. D. Lì, G. Advisors, Rubèn, T. Liesa, and X. Gù Ardia Latorre, “A Deep Learning Based Approach to Automated App Testing.”
2. I. Banerjee, B. Nguyen, V. Garousi, and A. Memon, “Graphical user interface (GUI) testing: Systematic mapping and repository,” *Information and Software Technology*, vol. 55, no. 10. pp. 1679–1694, Oct. 2013. doi: 10.1016/j.infsof.2013.03.004.
3. E. Alégroth, R. Feldt, and L. Ryrholm, “Visual GUI testing in practice: challenges, problemsand limitations,” *Empir Softw Eng*, vol. 20, no. 3, pp. 694–744, Jun. 2015, doi: 10.1007/s10664-013-9293-5.
4. R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek and A. Stavrou, "A whitebox approach for automated security testing of Android applications on the cloud," *2012 7th International Workshop on Automation of Software Test (AST*), Zurich, Switzerland, 2012, pp. 22-28, doi: 10.1109/IWAST.2012.6228986.
5. Y. Li, Z. Yang, Y. Guo and X. Chen, "Humanoid: A Deep Learning-Based Approach to Automated Black-box Android App Testing," 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 2019, pp. 1070-1073, doi: 10.1109/ASE.2019.00104.
6. D. Lì, G. Advisors, Rubèn, T. Liesa, and X. Gù Ardia Latorre, “A Deep Learning Based Approach to Automated App Testing.”
7. A. Vilhunen, “User interface test automation for an Android application,” *User interface test automation for an Android application*, May 16, 2022.
8. C. Hu and I. Neamtiu, "Automating GUI testing for Android applications," in Proceedings of the 6th International Workshop on Automation of Software Test (AST '11), New York, NY, USA, 2011, pp. 77-83, DOI: 10.1145/1982595.1982612.
9. A. Rosenfeld, O. Kardashov, and O. Zang, “Automation of Android Applications Testing Using Machine Learning Activities Classification,” *arXiv.org*, Sep. 04, 2017.
10. N. Paul and R. Tommy, "An Approach of Automated Testing on Web Based Platform Using Machine Learning and Selenium," *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, Coimbatore, India, 2018, pp. 851-856, DOI: 10.1109/ICIRCA.2018.8597297.
11. F. Ricca, A. Marchetto and A. Stocco, "AI-based Test Automation: A Grey Literature Analysis," *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Porto de Galinhas, Brazil, 2021, pp. 263-270, DOI: 10.1109/ICSTW52544.2021.00051.
12. A. Méndez-Porras, J. Alfaro-Velasco, and A. Martínez, “Evaluation of an Automated Testing Framework: A Case Study,” *Evaluation of an Automated Testing Framework: A Case Study*.
13. K. Moran, M. Linares-Vasquez, C. Bernal-Cardenas, C. Vendome and D. Poshyvanyk, "CrashScope: A Practical Tool for Automated Testing of Android Applications," *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, Buenos Aires, Argentina, 2017, pp. 15-18, doi: 10.1109/ICSE-C.2017.16.
14. I. K. Villains, E. A. B. Costa, and A. C. Dias-Neto, “Automated Mobile Testing as a Service (AM-TaaS),” in *Proceedings - 2015 IEEE World Congress on Services, SERVICES 2015*, Aug. 2015, pp. 79–86. DOI: 10.1109/SERVICES.2015.20.
15. B. Mishra et al., “Privacy Protection Framework for Android,” *IEEE Access*, vol. 10, pp. 7973–7988, 2022, DOI: 10.1109/ACCESS.2022.3142345.
16. S. Khan, Z. Jiangbin, and A. Wahab, “Design and Development of Android Performance Testing Tool,” in *2020 IEEE Conference on Big Data and Analytics, ICBDA 2020, Nov. 2020, pp. 57–60. DOI: 10.1109/ICBDA50157.2020.9289714.*
17. T. Takala, M. Katara, and J. Harty, “Experiences of system-level model-based GUI testing of an android application,” in *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST 2011*, 2011, pp. 377–386. DOI:

10.1109/ICST.2011.11.
18. A. Rosenfeld, O. Kardashov, and O. Zang, “Automation of Android Applications Functional Testing Using Machine Learning Activities Classification,” in *Proceedings - International Conference on Software Engineering*, May 2018, pp. 122–132. DOI: 10.1145/3197231.3197241.
19. S. Fatima, B. Mansoor, L. Ovais, S. A. Sadruddin, and S. A. Hashmi, “Automated Testing with Machine Learning Frameworks: A Critical Analysis,” *MDPI*, Jul. 28, 2022.
20. J. Kaasila, D. Ferreira, V. Kostakos, and T. Ojala, “Testdroid: Automated remote UI testing on android,” in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, MUM 2012, 2012. DOI: 10.1145/2406367.2406402.
21. Scripted GUI testing of Android open-source apps: evolution of test code and fragility causes, doi.org/10.1007/s10664-019-09722-9
22. H. Zhang, H. Wu, and A. Rountev, “Automated test generation for detection of leaks in Android applications,” in Proceedings - 11th International Workshop on Automation of Software Test, AST 2016, May 2016, pp. 64–70. DOI: 10.1145/2896921.2896932.
23. B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, R. S. Kumar, "Rico: A Mobile App Dataset for Building Data-Driven Design Applications," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, 2017, pp. 4237-4249.
24. Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, Hartwig Adam*; Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1314-1324
25. Mo, N., Yan, L., Zhu, R., & Xie, H. (2019, January 30). Class-Specific Anchor Based and Context-Guided Multi-Class Object Detection in High-Resolution Remote Sensing Imagery with a Convolutional Neural Network. *Remote Sensing, 11(3)*, 272, doi: 10.3390/rs11030272
26. Makanapura, N., Sujatha, C., Patil, P. R., & Desai, P. (2022, January 1). Classification of plant seedlings using deep convolutional neural network architectures. *Journal of Physics: Conference Series*, 2161(1), 012006, DOI: 10.1088/1742-6596/2161/1/012006
27. Jie, Y., Ji, X., Yue, A., Chen, J., Deng, Y., Chen, J., & Zhang, Y. (2020, December 21). Combined Multi-Layer Feature Fusion and Edge Detection Method for Distributed Photovoltaic Power Station Identification. *Energies, 13*(24), 6742.
28. V. -T. Hoang and K. -H. Jo, "Practical Analysis on Architecture of EfficientNet," 2021 14th International Conference on Human System Interaction (HSI), Gdańsk, Poland, 2021, pp. 1-4, DOI: 10.1109/HSI52170.2021.9538782.
29. Y. Wang, Z. Pan, J. Zheng, L. Qian, and M. Li, “A hybrid ensemble method for pulsar candidate classification - Astrophysics and Space Science,” *SpringerLink*, Aug. 29, 2019.
30. D.-H. Min and H.-K. Yoon, “Suggestion for a new deterministic model coupled with machine learning techniques for landslide susceptibility mapping - Scientific Reports,” *Nature*, Mar. 23, 2021.
31. Y. Zhou, Y. Bai, S. S. Bhattacharyya and H. Huttunen, "Elastic Neural Networks for Classification," *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Hsinchu, Taiwan, 2019, pp. 251-255, doi: 10.1109/AICAS.2019.8771475.
32. A. Verikas, E. Vaiciukynas, A. Gelzinis, J. Parker, and M. Olsson, “Electromyographic Patterns during Golf Swing: Activation Sequence Profiling and Prediction of Shot Effectiveness,” *Sensors*, vol. 16, no. 4, p. 592, Apr. 2016, doi: 10.3390/s16040592.
33. E. Jumin, N. Zaini, A. N. Ahmed, S. Abdullah, M. Ismail, M. Sherif, A. Sefelnasr, and A. El-Shafie, "Machine learning versus linear regression modelling approach for accurate ozone concentrations prediction," *Environmental Science and Pollution Research*, vol. 27, no. 1, pp. 713-725, May 2020. DOI: 10.1080/19942060.2020.1758792.
34. https://github.com/434huzaifa/DeepTestDroid?fbclid=IwAR1bYQ1KV2PM0J5IRJWWxDArs5EAzYdLGCmcuQl6Kae3fd4PjAwumNVvPkU

Appendix A

Mapping of Course and Program Outcomes

CSE400-A

Program Outcomes:

PO1 (Engineering Knowledge): To ensure the accuracy of the testing for black box and white box using deep learning.

PO4 (Investigation): Appium is an open-source tool for automating native, mobile web, and hybrid applications on iOS mobile, Android mobile, and Windows desktop platforms.

CO	Details	Knowledge Profile (K)	Engineering problem (EP)
CO1	To ensure the accuracy of the testing for black box and white box using deep learning.	**(i) Background [K1, K2, K3]** **K1:** To ensure accuracy. of the testing for black box and white box using deep learning. **K2:** Appium is an open-source tool for automating native, mobile web, and hybrid applications on iOS mobile, Android mobile, and Windows desktop platforms. **K3:** Automated android application testing using deep learning.	**i) Background [EP1]** **K3:** Appium is an open-source tool for automating native, mobile web, and hybrid applications on iOS mobile, Android mobile, and Windows desktop platforms. **K4:** Testing architecture followed by Appium – Junit and JaCoCo. **K5:** Test cases design and automation. **K8:** Literature Paper review on black box and white box testing using deep learning.

			ii) Research questions/problem statements [EP6] Test case generation- · Approach · Address ·
CO2	Lorem Ipsum is simply dummy text of the printing and typesetting industry.	**(i) Related works [K8]** ● Literature paper review on current automated testing models for black box and white box ● Paper review on deep learning methods to automate the generation of test cases. ● Literature review in the	**i) Related works [EP1]** **K5:** Flowchart Design. **K6:** Automation using deep learning. **ii) Objectives [EP2, EP6, EP7]** ***EP2:*** Implementing a deep learning model to prioritize test inputs according to the importance of the user's perspective. ***EP6:*** Test case generation- · Approach · Address · Effectiveness ***EP7:*** · Application · Android drive · Test data · Trained data · Deep learning · Test label data

		platform for automated android application testing using deep learning.	· Expected result · Appium · Test result **iii) Planned Methodology [EP2, EP6]** ***EP2:*** Implementing a deep learning model to prioritize test inputs according to the importance of the user's perspective. ***EP6:*** Test case generation- · Approach · Address · Effectiveness

CSE400-B

Program Outcomes:

PO2 (Problem Analysis): Analyze various aspects of the objectives for designing a solution for the capstone project.

PO3 (Design/Development of Solutions): **Design** and **develop** solutions for the capstone project that meet public health and safety, cultural, societal, and environmental considerations.

PO5 (Modern Tool Usage): Identify and **apply** modern engineering and IT tools for the design and development of the capstone project.

PO6 (The Engineer and Society): Assess and **address** societal, health, safety, legal, and cultural aspects related to the implementation of the capstone project considering the relevant professional and engineering practices and solutions.

CO	**Details**	**Knowledge Profile (K)**	**Engineering Problem (EP)**
CO3	**Analyze** various aspects of the objectives for designing a solution for the capstone project.	**(i) Problem Analysis [K1, K2, K3, K4]** **K1:** To ensure the testing is accurate. **K2:** python, vscode, tensorflow, kaggle for training the model. **K3:** Mobile Application and software scheduling. **K4:** Testing architecture followed by consensus mechanism like image dataset of mobile applications UI.	**(i) Problem Analysis [EP1, EP2, EP3, EP6, EP7]** **EP1:** **K5:** Model training, image datasets, GUI testing process. **K6:** Testing through code and deep learning method. **EP2:** The Black Box and White Box techniques are broad ones that are not only for AI. In addition to their many other applications, they are used to create AI models in addition to developing and testing traditional software.
CO4	Lorem Ipsum is simply dummy text of the printing and typesetting industry.	**(i) Design and Implementation [K5]** We designed a UI for sorting the datasets easily.	**(i)Design and Implementation [EP1, EP2, EP4, EP5, EP6, EP7]**

			Implementing a deep learning model to prioritize test inputs according to the importance of the user’s perspective. ***EP6:*** Test case generation- · Approach · Address · Effectiveness
CO5	Lorem Ipsum is simply dummy text of the printing and typesetting industry. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged.	**(i) Materials and Devices [K6]** **K6: Engineering Practice (technology):** TensorFlow, VS code, Python, and Rico, the mobile app datasets for building data-driven design applications.	**(i) Materials and Devices [EP1, EP2, EP4, EP5]** TensorFlow is an open-source framework developed by Google for deep learning applications, while Python is used for developing websites, software, task automation, data analysis, and data visualization. Kaggle is an online community of data scientists and machine learning.
CO6	**Assess** and **address** societal, health,	**(i) Social and Environmental Impact**	**(i) Social and Environmental**

	safety, legal, and cultural aspects related to the implementation of the capstone project considering the relevant professional and engineering practices and solutions. specimen book.	**of Engineering [K7]** **K7: Comprehension of engineering in society:** This will replace manual testing with AI-powered automation, making GUI testing simpler.	**Impact of Engineering [EP2, EP5, EP6]** Visual GUI testing (VGT) is more flexible and resilient to GUI modifications than earlier high-level (GUI) test automation techniques. VGT is useful, adaptable, and cost-effective, with 58 distinct CPLs and 26 categories.

CSE400-C

Program Outcomes:

PO7 (Environment and Sustainability): Analyze various aspects of the objectives for designing a solution for the capstone project.

PO8 (Ethics): Design and **develop** solutions for the capstone project that meet public health and safety, cultural, societal, and environmental considerations.

PO9 (Individual Work and Teamwork): Assess and **address** societal, health, safety, legal, and cultural aspects related to the implementation of the capstone project considering the relevant professional and engineering practices and solutions.

P10 (Communication): The main approach for this project would be an online gathering via Google Meet. There was no communication breakdown because every project participant was close to one another.

P11 (Project Management and Finance): We were always working on this project under the direction of our supervisor. We have continued to use the Work Breakdown Structure for project management. Each task had a time limit, and we completed them all by that time. This is how we were able to complete our assignment on schedule. The project had no significant costs. The project's participants self-funded any costs that were necessary.

P12 (Life-Long Learning): We have put the ideas we acquired in our prior classes into practice in this project. We picked up a few more ideas while working on the project. Along with these, we also learned some fundamental skills like problem-solving, critical thinking, and communication that will be useful in the future.

CO	**Details**	**Knowledge Profile (K)**	**Engineering Problem (EP)**
CO7	**Identify** and **apply** modern engineering and IT tools for the design and development of the capstone project.	**(i) Societal and environmental contexts [K7]** **K7: Comprehension of engineering in society**: This will replace manual testing with AI-powered automation, making GUI testing simpler.	**(i) Societal and environmental contexts [EP2, EP5, EP6]** **EP2: Range of conflicting requirements:** Social and Environmental Impact of Engineering [EP2, EP5, EP6] Visual GUI testing (VGT) is more flexible and resilient to GUI modifications than earlier high-level (GUI) test automation techniques. VGT is useful, adaptable, and cost-effective, with 58 distinct CPLs and 26 categories.

CO8	**Assess** and **address** societal, health, safety, legal, and cultural aspects related to the implementation of the capstone project considering the relevant professional and engineering practices and solutions.	**(i) Ethical principle and practices [K7]** **K7: Comprehension of engineering in society:** We designed a UI for sorting the datasets easily. ring in society:	Materials and Devices [EP1, EP2, EP4, EP5] TensorFlow is an open-source framework developed by Google for deep learning applications, while Python is used for developing websites, software, task automation, data analysis, and data visualization. Kaggle is an online community of data scientists and machine learning engineers,
C09	**Identify** and **apply** modern engineering and IT tools for the design and development of the capstone project.	**Materials and Devices [K6]** TensorFlow, VS code, Python, and Rico, the mobile app datasets for building data-driven design applications.	
CO10	We have an effective report on the capstone project. In the design		

	and implementation section of the report, we went into great detail on the specific design and implementation aspects of our project.		
CO11	Given the size of the project, it took a long time to complete. Our project work had been scheduled. The ability to keep on schedule and submit the project on time was one of the most crucial aspects of the project, which we successfully accomplished thanks to our strong teamwork.		
CO12	We had to acquire new concepts and apply them in order to employ the concepts we used in this experiment. For this project, we were able		

	to combine the exploration of fresh ideas with more established ones. Users will be able to identify the inflammatory skin diseases with the aid of this practical knowledge application in the finished software program.		