T.C.
İSTANBUL KÜLTÜR UNIVERSITY
INSTITUTE OF GRADUATE STUDIES

DEEP LEARNING-BASED DEFECT PREDICTION FOR MOBILE APPS

Master of Science Thesis in Computer Engineering

Manzura Jorayeva
1800004575

Department: Computer Engineering
Programme: Computer Engineering

Advisor: Assoc. Prof. Dr. Akhan Akbulut
Co-Advisor: Prof. Dr. Çağatay Çatal

FEBRUARY 2022

T.C.

İSTANBUL KÜLTÜR UNIVERSITY

INSTITUTE OF GRADUATE STUDIES

DEEP LEARNING-BASED DEFECT PREDICTION FOR MOBILE APPS

Master of Science Thesis in Computer Engineering

Manzura Jorayeva

1800004575

Department: Computer Engineering

Programme: Computer Engineering

Advisor: Assoc. Prof. Dr. Akhan Akbulut

Co-Advisor: Prof. Dr. Çağatay Çatal

Members of Examining Committee:

Advisor: Assoc. Prof. Dr. Akhan Akbulut (Istanbul Kültür University)
Assis. Prof. Dr. Görkem Kar (Bahçesehir University)
Assis. Prof.Dr. Öznur Şengel (Istanbul Kültür University)

FEBRUARY 2022

I acknowledge that all information in this document has been received and conferred in accordance with academic rules and ethical conduct. I also acknowledge that, as required by these rules and conduct. I have entirely cited and referenced all material and results that are not unique to this work.

Manzura Jorayeva

ABSTRACT

DEEP LEARNING-BASED DEFECT PREDICTION FOR MOBILE APPS

Jorayeva, Manzura

M.Sc., Department of Computer Engineering

Advisor: Assoc. Prof. Dr. Akhan Akbulut

Advisor: Prof. Dr. Çağatay Çatal

FEBRUARY 2022, 60 Pages

Mobile applications are increasing their popularity every year. However, unrecognized defects within mobile applications can affect businesses due to negative user experience. To avoid this, defects of applications should be reviewed before releases. The well-known methods for defect prevention include Review and Inspection, Walkthrough, Logging and Documentation, and Root Cause Analysis, as well as employing innovative predictive approaches using machine learning. The benefit of these prediction models is that more testing resources can be allocated to fault-prone modules effectively. This study aims to present a defect prediction model for mobile applications. We applied cross-project and used deep learning algorithms including Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Long-Short Term Memory (LSTM) to develop a defect prediction model and applied it to Android applications datasets. SMOTE Oversampling technique is used to balance datasets, accuracy metrics such as precision, recall, F1-score, ROC, and AUC to achieve performance, and model results are evaluated with tenfold cross-validation.

Keywords: Software defect prediction, software fault prediction, mobile application, Android applications, deep learning, machine learning.

ÖZ

MOBİL UYGULAMALAR İÇİN DERİN ÖĞRENME TABANLI HATA TAHMİNLEMESİ

Jorayeva, Manzura

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Danışmanı: Doç. Dr. Akhan Akbulut

Tez Danışmanı: Prof. Dr. Çağatay Çatal

ŞUBAT 2022, 60 Sayfa

Mobil uygulamalar her geçen yıl popülerliğini artırıyor. Ancak, mobil uygulamalar içerisindeki tanınmayan kusurlar olumsuz kullanıcı deneyiminden ötürü işletmeleri etkileyebilmektedir. Bunu önlemek için, uygulama kusurları sürümlerden önce gözden geçirilmelidir. Kusur önleme için en iyi bilinen yöntemler arasında Gözden Geçirme ve İnceleme, Çözüm Yolu, Günlüğe Kaydetme ve Belgeleme ve Kök Neden Analizi ile makine öğrenimini kullanan yenilikçi tahmine dayalı yaklaşımlar yer almaktadır. Bu tahmin modellerinin yararı, hataya açık modüllere etkin bir şekilde daha fazla test kaynağının tahsis edilebilmesidir. Bu çalışma, mobil uygulamalar için bir hata tahmin modeli sunmayı amaçlamaktadır. Bir kusur tahmin modeli geliştirmek için çapraz proje uyguladık ve Yapay Sinir Ağları (ANN), Evrişimsel Sinir Ağları (CNN) ve Uzun-Kısa Süreli Bellek (LSTM) gibi derin öğrenme algoritmalarını kullanarak Android uygulamaları verisetlerine uyguladık. Veri kümelerini dengelemek için SMOTE Aşırı Örnekleme tekniği, performans elde etmek için hassasiyet, geri çağırma, f1-puanı, ROC ve AUC gibi doğruluk ölçütleri kullanılır ve model sonuçları on kat çapraz doğrulama ile değerlendirildi.

Anahtar Kelimeler: Yazılım kusur tahmini, yazılım hata tahmini, mobil uygulama, Android uygulama, derin öğrenme, yapay öğrenme.

ACKNOWLEDGEMENTS

I would like to formulate my gratefulness to my advisor, Assoc. Prof. Dr. Akhan Akbulut for his direction, motivation, and assistance during my thesis study.

I would like to represent my thanks to my co-advisor, Prof. Dr. Cagatay Catal, for his orientation, encouragement, and support during my thesis study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ML -Machine Learning

DL- Deep Learning

SFP – Software Fault Prediction

ANN – Artificial Neural Network

CNN – Convolutional Neural Network

LSTM – Long Short-Term Memory

SMOTE – Synthetic Minority Oversampling Technique

ROC- Receiver Operating Characteristics

AUC- Area Under the Curve

1 INTRODUCTION

Mobile applications are recently playing undeniable roles in many aspects of our lives [1]. Our new generation is growing with new technology such as mobile phones, tablets, and laptops. We are connected by smartphones and use them for many different purposes in daily life frequently, and as such, mobile applications are getting more and more crucial in our lives. Nowadays, we download mobile applications to access digital information, play games, learn languages, and communicate with each other [2]. Many applications are available with free and paid versions. However, the main advantage of mobile applications is their effectiveness and working with nearly zero defects. Currently, we download mobile applications from digital markets such as App Store and Android Market to access information, play games, learn languages, and communicate with others. Not only the complexities of these applications but also the user expectations are increasing dramatically. Many people are progressively affected by mobile application downloads, and users do not waste time using the mobile app once they observe a functional or non-functional problem. Thus, it is a threat for mobile application developers and businesses to deploy software even with some minor bugs. These faults or bugs affect the effectiveness of mobile app and can cause unpredictable crashes [3]. When users download these problematic versions of applications, they may cause serious problems [4]. Diagnosing mobile applications crashes can give a chance to improve faults before new releases [5].

A defect can be an internal failure of the application and cause the system to shut down [6]. Similarly, defect in an element or case of not meeting correct planned operation requirements. Not automatic tests can predict 35 to 60% of faults, and automatic tools developed by artificial neural networks can predict 70% of faults [7]. The most obtained and usually used approach is analyzing an improved set of metrics and using them in machine learning methods [8]. The accuracy and efficiency of detection models depend on the type of the dataset, metrics selection methods, and the ML techniques. The task is to complete applications without errors. From the 1990s until now, software defect prediction models were analyzed to raise operating system performance, and defective parts were selected before system tests by using these parts. Software defect prediction methods use past software metrics and defects data to predict defective parts for the following software versions. When a failure is listed at the time of system tests, that part of defective data is stated as one for modeling expectation.

The aim of this study is to develop a defect prediction model for mobile applications. We focused on Android mobile applications because it is open source available. In our research we identified that Android applications datasets is more open. In mobile application defect prediction model development stage, we reviewed software defect prediction articles that specialized in applications. We evaluated them from numerous perspectives. The studies analyzed in this study show that what was done in the mobile applications defect prediction area. We analyzed risks of defective mobile application releases and developed our Mobile Applications Defect Prediction Model and applied it to 14 Android applications datasets to show the performance of our model. The most studies used web applications only a few studies focused on mobile application defect prediction. The thesis sections are organized as follows: Section 2 provides the background and related work. Section 3 defines the selected research methodology. The systematic literature review introduced in Section 4. Systematic literature review. Responses to research

questions are presented in section 5. Model development process is described in section 6. Finally, Section 7 Results, Discussion presented in Section 8 and the Section 9 is Conclusions and Future work.

2 BACKGROUND AND RELATED WORK

First study on number of defects on software was presented by [9]. It is practiced on web and desktop applications until the 1973s first mobile phone was produced; the mobile phone market has undergone a huge evolution. Mobile applications are also integrated into mobile phones and increasing in popularity every year. However, there are unrecognized defects of mobile applications that can affect businesses from negative user experience. To avoid this, defects of applications should be reviewed before releases. The benefit of these prediction models is that more testing resources can be allocated to fault-prone modules effectively. Software defect prediction and machine learning identified Section 2.1 and in Section 2.2 we present the related studies and compare our study with them.

2.1 SOFTWARE DEFECT PREDICTION

In software development developers first test functionality of code in local side the compare actual and expected results. When they find difference between actual and expected code it become a defect. Software defect affects software quality. When test engineer is tested code and find difference between actual and expected results it is a bug. Error is mistakes in the code, while writing code developer not able to run or compile code. Failure is when program is ready and customer facing issue in production. There are a few types of software defects: errors of commissions, errors of omissions, errors of clarity, error of speed and capacity. Error of commissions means the error which in command. Error of omission means we forget to close bracket or left or excluded. Error of clarity means misunderstanding between developer and customer. Error of speed or capacity means when program works but not in allowed time.

2.2 MACHINE LEARNING

Machine learning research aims to identify data patterns and discover exciting knowledge from a large amount of data. Since the 1990s, software defect prediction tasks have been using machine learning algorithms—various metrics with erroneous data used to identify fault-prone classes. While software metrics are calculated based on the collected data from software repositories, erroneous data is retrieved from issue tracking systems. There are different software tools such as Understand to calculate software metrics automatically from software projects; however, automation of fault data is more challenging. Machine Learning is also used for (i.e., regression task) and the fault-prone classes (i.e., binary class classification). Machine learning has the following learning types: In Supervised Learning, practiced labeled data to detect traits. Unsupervised learning evaluates not labeled data to identify hidden structures through determining the correlations of features. The algorithm can gain from a limited number of labeled

text documents for clustering and dimensionality reduction. Semi-Supervised learning is used when data points are unlabeled (e.g., 80% - 90%), and only a tiny percentage of data is labeled. The last category is reinforcement learning that uses software agents to learn the environment by using a trial-and-error basis and, applies feedback mechanism.

2.3 RELATED WORK

All related articles published in the latest thirty years on software defect prediction and mobile applications defect prediction is investigations were extensively studied. Nevertheless exceptionally, articles were published after the first decade of our century. Catal and Diri analyzed published defect prediction articles with a unique target of metrics, datasets, and approaches. Their review described those method-level metrics used for machine learning defect prediction [10]. Malhotra reviewed publications from 1991s to 2013 on machine learning methods for defect prediction. He presented a systematic literature review of identified sixty-four articles also machine learning methods. The report proves that machine learning approaches can be implemented to predict the faults of classes [11]. Fahle et al. presented a literature review of machine learning applications in the industrial sector. He reviewed that machine learning applications are compelling on production, training, transportation, support, process administration, system, management, personnel, testing, monitoring, and analysis [12]. Radjenovic et al. analyzed software metrics practice and presented the study to evaluate metrics on defect prediction. In review included hundred-six reports published from 1991s to 2011. Traditional machine learning methods and object-oriented metrics are applied for fault detection problems [13].

Hall et al. analyzed defect prediction accuracy in Software Engineering and identified 208 articles and presented a review. They included studies from the 2000s to 2010s that describe that machine learning methods such as Logistic Regression and Naive Bayes are practiced diagnosing errors in the source code [14]. Misirli et al. analyzed thirty-eight publications on software quality prediction using machine learning methods and presented a systematic mapping study on software quality detection. Machine Learning algorithms such as Bayesian networks were used for 70% of studies [15]. Morera et al. analyzed studies on software defect prediction from 2002s to 2014. They applied quality criteria to identify papers, selected forty studies, and presented a systematic mapping study on software defect prediction. They described the performance of Machine Learning methods like Random Forest, Naive Bayes, Logistic Regression, Decision Tree and OO metrics, Halstead, McCabe, and LOC metrics [16]. Carvalho et al. analyzed prediction on unexpected errors in equipment using machine learning methods and presented a review directed on two digital platforms.

Different machine learning classifiers such as Support Vector Machines, Random Forest, Artificial Neural Networks, Deep Learning, and k-means are used for PdM applications [17]. Ozakıncı et al. reviewed publications on the success and performance from the 2000s to 2016s on ESDP models and selected fifty-two publications. They investigated the aim, development, progress, advantages, components of models and presented a systematic mapping and systematic literature review [18]. Son et al. performed a systematic mapping study of software defect prediction. They analyzed 156 articles, and eventual mapping was aimed based on these articles. Among that, very few studies described cross-project DeP, and several studies have worked on defects [19]. Najm et al. analyzed studies from 1985s to 2017 on DT-based software development and presented a study.

The selected publications are categorized on publication platform, analysis model, research strategy approaches applied in organizations with machine learning methods [20]. Alsolai et al. analyzed publications focused on defect prediction of OO Software systems applied machine learning methods and presented a systematic literature review. They reported that the authors used private datasets and affairs on publicly available datasets. They applied to model k-fold cross-validation approaches and named regression tasks. Also, various studies implemented unique models [21]. Auch et al. analyzed studies from 2002s to 2019 of applications similarity-based analysis and selected 136 articles. They applied inclusion and exclusion criteria to related studies, identified applications' similarities, and presented a systematic literature review on similarity organization and analysis of software applications [22]. Degu analyzed thirty-one studies on the lack of android applications memory and energy discharge expansion. The presented a review to classify and design the research results formed in android application memory and energy work, resource leaks, and performance testing approaches and threats described [23]. Kaur et al. (2018) presented a mobile application evolution and testing progress study. They analyzed and correlated existing test evaluation methods for conventional mobile and desktop applications [24].

Del Carpio et al. analyzed using deep learning approaches in software engineering and presented a study. They described that Deep Learning methods like Convolutional Neural Networks, Long-Short Term Memory, Recurrent Neural Networks are used for fault detection, analyzing images, demands, diagnose errors on the monitoring stage. Also, they identified the usage of deep learning for defect prediction, classification problems, visualization, test, and analysis software requirements [25]. Kaur et al. presented a review on code smells and QA relations. They reported that various code smells could have differing results against other software quality attributes [26].

However, in mobile defect prediction we identified following studies: Kaya et al. presented a study and defect prediction model on machine learning and design-level metrics used with data sampling methods. They described the point of data sampling methods for imbalanced datasets to improve defect prediction performance. They mentioned the Adaboost algorithm as the best for defect prediction [27]. Kaur et al. analyzed process metrics to predict mobile applications defects and performed a study using publicly available mobile applications datasets. They focused on regression problem on machine learning and applied process and code metrics models to datasets. Process metrics-based machine learning model named as the best predictor [28]. Zhao et al. presented an imbalanced deep learning-based model for Android applications defect prediction. They applied to 12 applications datasets loss function for imbalance class problem and deep learning model for defect prediction. They pointed that IDL model performance is better than other models [29]. Sewak et al, analyzed different types of LSTM architectures and presented a study on malware detection using LSTM networks [30]. The most of these models used traditional machine learning algorithms. Only a few studies used deep learning algorithms for model development. In our model we use an Artificial Neural Networks, Convolutional Neural Networks and Long-Short Term Memory to solve a defect prediction for android applications.

3 RESEARCH METHODOLOGY

In this section we present research methodology of this study. We identified nine research questions and applicable papers were reclaimed from digital platforms. Study selection criteria were applied to the reclaimed papers, and a subgroup was settled for quality assessment. Each paper was delicately reviewed based on the quality assessment questions. First, we reviewed previous studies on software defect prediction and categorized them into three groups. Web applications, mobile android applications and

windows phone applications defect prediction. On web applications defect prediction many studies are completed, for windows phone applications are limited resource and for android mobile applications defect prediction a few studies published.

We carefully studied each publication and performed a systematic literature review on mobile application defect prediction using machine learning methods. From obtained results in literature review we decided to develop a model for Android applications defect prediction using deep learning techniques because precious studies on this a very limited. For model development process we first identified three research questions. We reviewed previous studies to identify models developed for Android applications defect predictions. We recognized that only a few deep learning algorithms applied. After we collected android applications datasets, completed data processing, and developed a model using deep learning algorithms. We answer to nine research question in systematic literature review and four research questions after model development process completed. Table 1 shows research questions used in this study. Figure 1 shows methodology of this study.

Table 1 research questions in this study

ID	Research Questions
RQ1	Which platforms are addressed for software defect prediction?
RQ2	Which datasets are used for software defect prediction?
RQ3	Which Machine Learning types are used?
RQ4	Which Machine Learning algorithms are applied?
RQ5	Which evaluation metrics are used?
RQ6	Which validation were approaches used?
RQ7	Which software metrics were adopted?
RQ8	Which Machine learning algorithm works best for defect prediction?
RQ9	What are the challenges and research gaps?
RQ10	Can we predict the mobile apps defect using cross-project for deep learning methods?
RQ11	Can we improve the performance of deep learning using data balancing approaches?
RQ12	Can we improve the performance of our deep learning model using feature selection techniques?
RQ13	To what extent LSTM-based deep learning predict software faults for mobile applications?

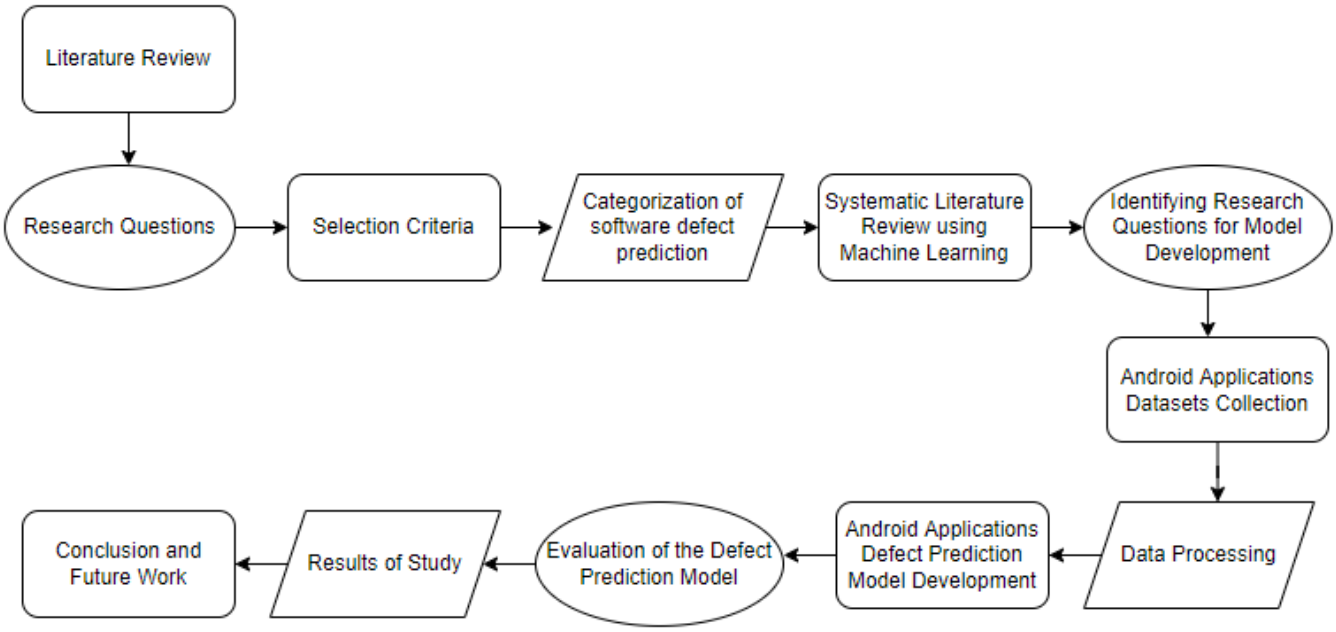

Figure 1 Methodology of study

4 SYSTEMATIC LITERATURE REVIEW

In this section we present systematic literature review. In section 3 described Research methodology of study and SLR research methodology presented in section 4. The following databases were used to retrieve relevant papers: IEEE Xplore, Science Direct, ACM Digital Library, Wiley Online Library, Springer Link, and Google Scholar. The search spanned the last 10 years to identify up-to-date papers. Table 3 shows exclusion criteria used in this study. The following search criteria were applied: (("machine learning" OR "artificial intelligence") AND "mobile software" AND ("fault prediction" OR "defect prediction" OR "software quality")). Table 2 shows the distribution of papers per database and the number of papers at each stage (i.e., after initial query, after exclusion criteria, and after quality assessment).

Table 2 The distribution of papers per database

Database	Number of papers after query search	Number of papers after exclusion criteria	After quality assessment
1. Science Direct	16	9	5
2. IEEE Xplore	381	24	13
3. ACM Digital	46	9	7
4. Wiley	113	1	1
5. Springer Link	98	7	6
6. Google Scholar	67	48	15
Total	721	98	47

Table 3 Exclusion criteria

ID	Exclusion criteria
1.	The full text is not accessible, and only the abstract is provided
2.	The paper is not written in English
3.	The article is not a primary study paper
4.	The content does not provide any experimental results
5.	The study does not describe in detail how machine learning is applied

We analyzed electronic databases and collected previous studies on Mobile Application Defect Prediction using Machine Learning Algorithms. We prepared exclusion criteria and applied to all selected papers. In the end we had a final subgroup of papers presented in Figure 2. Science Direct, IEEE Xplore, ACM Digital Library, Wiley Online Library, Springer Link, Google Scholar is the most popular databases. The most papers were retrieved from the IEEE Xplore database and Google Scholar also included similar number of papers in the final selection.

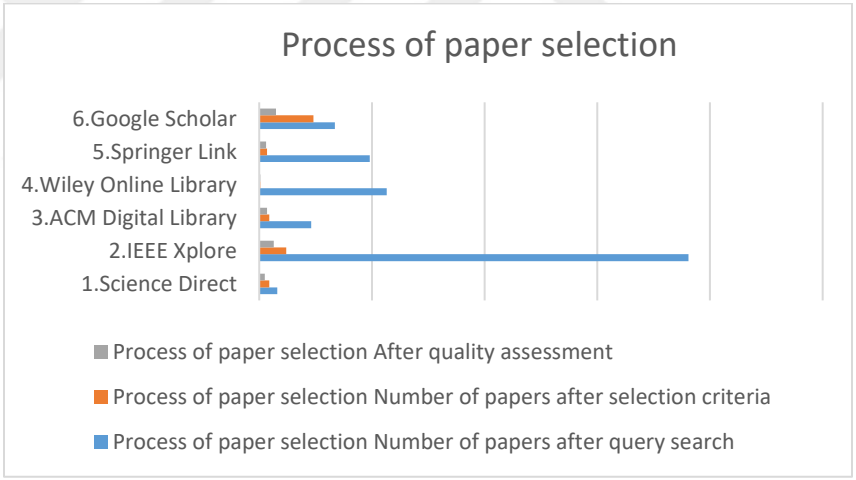

Figure 2 Distribution of the selected papers

After the exclusion criteria were applied, we graded papers for quality assessment using the approach proposed by [31]. Table 4 shows quality evaluation questions. Papers having lower scores than 10 were excluded from the list. Figure 2 shows the quality distribution of papers. If the answer is 'Yes' for the question, the paper gets 2 points, the 'partial' response gets 1 points, and 'no' answer receives 0 point.

Table 4 Quality evaluation questions. Yes, scores 2; Partial scores 1; No scores 0.

ID	Questions
Q1	Are the aims of the study clearly declared?
Q2	Are the scope and context of the study clearly defined?
Q3	Is the proposed solution clearly explained and validated by an empirical study?
Q4	Are the variables used in the study likely to be valid and reliable?
Q5	Is the research process documented adequately?
Q6	Are all study questions answered?
Q7	Are the negative findings presented?
Q8	Are the main findings stated clearly in terms of credibility, validity, and reliability?

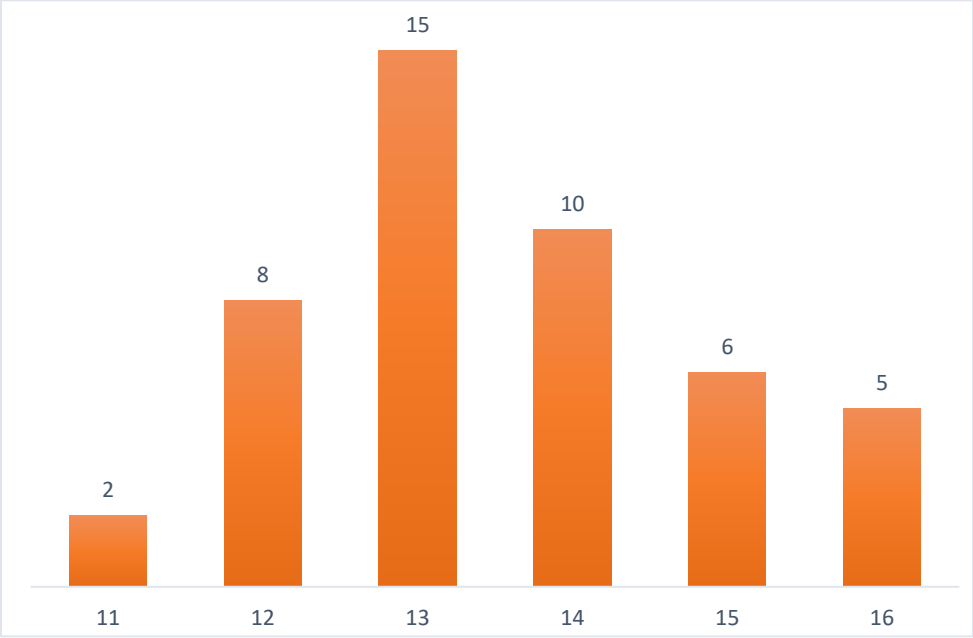

Figure 3 Quality score distribution of selected papers (x axis: paper score, y-axis: number of papers)

After quality assessment questions were applied, publications were synthesized. In Figure 4, the distribution of the selected publications per year is shown. As shown in this figure, within the last five years, more papers were published on this topic and this field is still active.

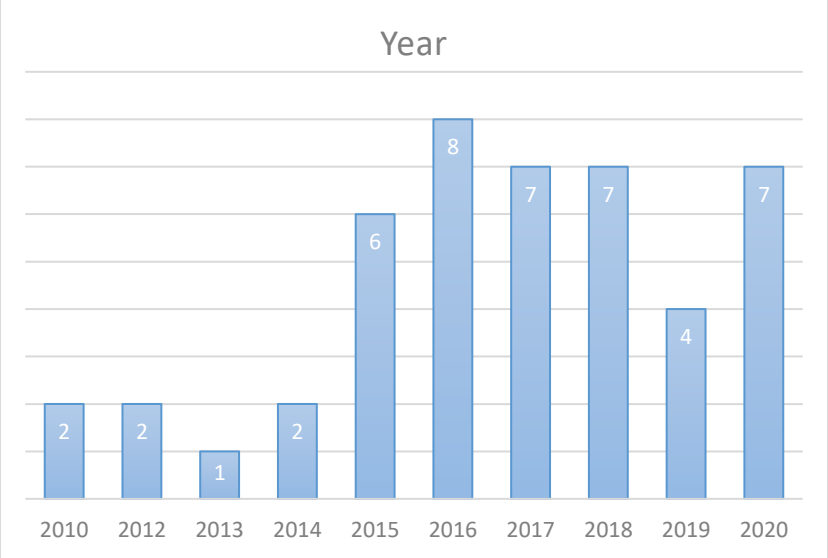

Figure 4 Selected publications per year

Figure 5 presents the representation of the type of publications. Nearly half of the papers are journals papers, and the rest are conference proceedings. This indicates that some researchers prefer publishing this type of paper in conferences; however, enough journal articles are evaluated in this SLR paper.

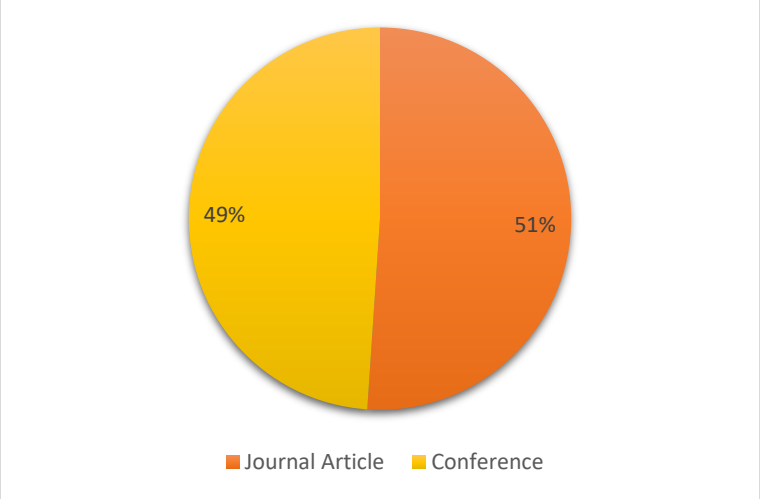

Figure 5 Distribution of type of publications

In total, we reclaimed 721 publications from digital platforms. Later implementing selection criteria and quality assessment, 47 publications were selected for additional review. Within this section, we provide our acknowledgments to the examination issues of this article.

4.1 RQ-1: PLATFORMS

This section provides the details of the platforms for mobile applications used in the primary studies. We named mobile software types and classified them inside three columns. As shown in Table 4, the most used mobile software is the Android platform's twenty-one publications. Windows Mobile software is used only in one publication. Web applications used in twenty publications.

Table 4 Platforms

Platforms	Total
Android	21
Windows Phone	1
Web Applications	20

4.2 RQ-2: DATASETS

The datasets related to software defect prediction subjects are openly available in repositories. Datasets are used from different repositories, and Figure 4 shows the summary of each dataset used in primary studies. For mobile applications, datasets were limited. Table 5 presents the report on the name of datasets, repositories, and web addresses.

Table 5 Repositories

Repositories	Datasets	Web address
GIT Repository	Contact, MMS, Bluetooth, Email, Calendar, Gallery2, and Telephony	https://android. googlesource.com,
Sharejar and Source Forge	Contact, MMS, Bluetooth, Email, Calendar, Gallery2, and Telephony	https://sourceforge.net/p/bitweaver/bugs/., http://securityfocus.com, https://sourceforge.net/p/Webcalendar/bugs.
GitHub Repository	Bootstrap, Avaya Communicator, The K-9 Mail client, Space Blaster game, K-9 issue report.	https://github.com/bpellin/keepassdroid, https://github.com/adrian/upm-android, http://android.scap.org.cn/.,http://cve.mitre.org/, https://github.com/breezedong/DNN-based SFP, https://github.com/JesusFreke/smali/wiki, https://github.com/tapjdey/release_qual_model, https://github.com/geometer/FBReaderJ., https://github.com.,
Other	Connectbot, Boardgame Geek, AnkiDroid, Android Wallpaper, Quiksearchbox	https://techterms.com/definition/repository., http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html, https://source.android.com/devices/tech/dalvik/dalvikbytecode, http://searchoracle.techtarget.com/definition/repository., http://code.google.com/p/sipdroid/., https://forum.xda-developers.com/showthread.php?t=1800090, f-droid.org.

Git Repository datasets used in one, Sharejar, and Source Forge datasets used in four publications. Datasets from GitHub Repository used in 12 publications. Other datasets were used in 11 publications.

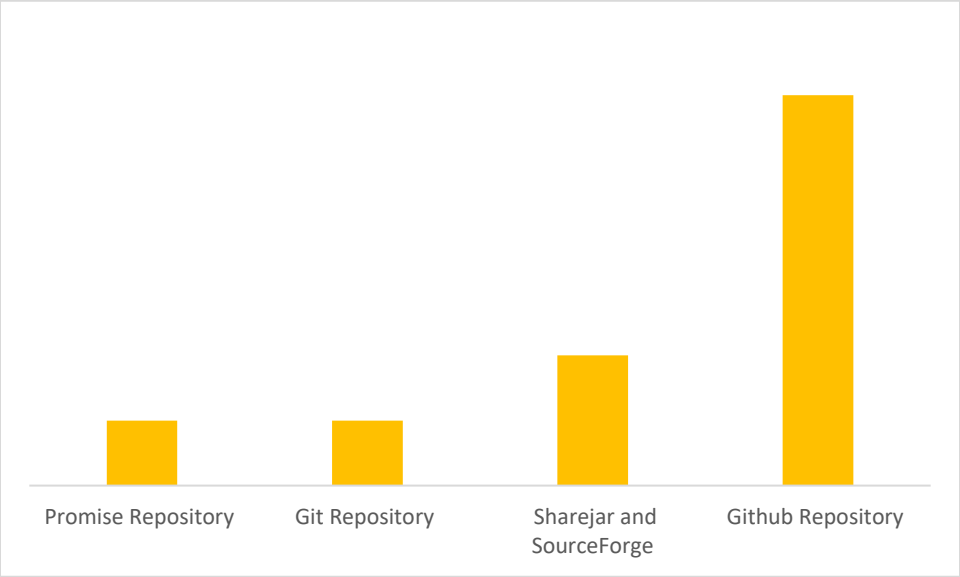

Figure 6 Repositories

4.3 RQ-3: MACHINE LEARNING TYPES

Inside that part, we include the collected result by questioning software fault prediction using machine learning approaches. We implement the outcomes of Machine Learning approaches valued in 47 initially selected articles. Supervised learning practiced in forty-three examinations. The other types functioned in two reports. Figure 7 shows ML types used in selected publications. That indicates that most of the researchers preferred supervised learning approaches when developing models for mobile applications. However, the literature applies unsupervised fault prediction models [32] and semi-supervised fault learning models [33]. Also, noisy instances can be removed from the datasets to improve the overall performance of the supervised models [34].

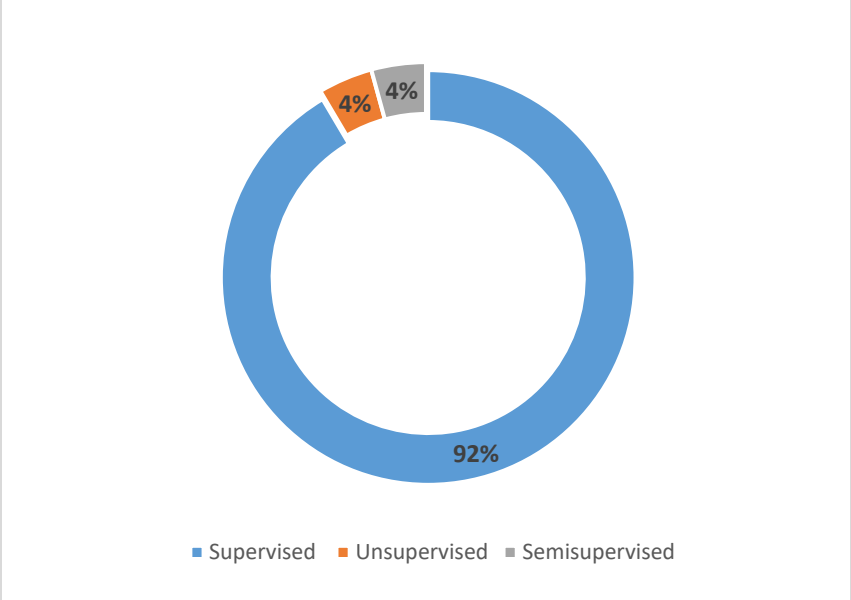

Figure 7 Machine Learning Types

4.4 RQ-4: MACHINE LEARNING ALGORITHMS

In selected studies, eighteen types of machine learning methods were used. The most used algorithms and their categorization are as follows: Naive Bayes (NB) was used in 26 studies, Support Vector Machines (SVM) were used in 22 studies, Logistic Regression (LR) were used in 19 studies, Neural Network (NN) was used in 18 studies, Decision Tree (DT) were used in 18 studies, Random Forest (RF) were used in 15 studies, K-nearest Neighbors (KNN) were used in 6 studies, Alpha-beta pruning (AB), K-means, Bayesian Network (BN) were used in 5 studies, Bootstrap aggregating (Bag), Multilayer Perceptron (MLP) were used in 3 studies, Voting Future Intervals (VFI), DTNB, Non-Nested Generalization (NNge), Logistic model tree (LMT) were used in 2 studies. Artificial Neural Network (ANN), An adaptive genetic algorithm (AGA) used in 1 study. Figure 8 explains machine learning algorithms used in publications.

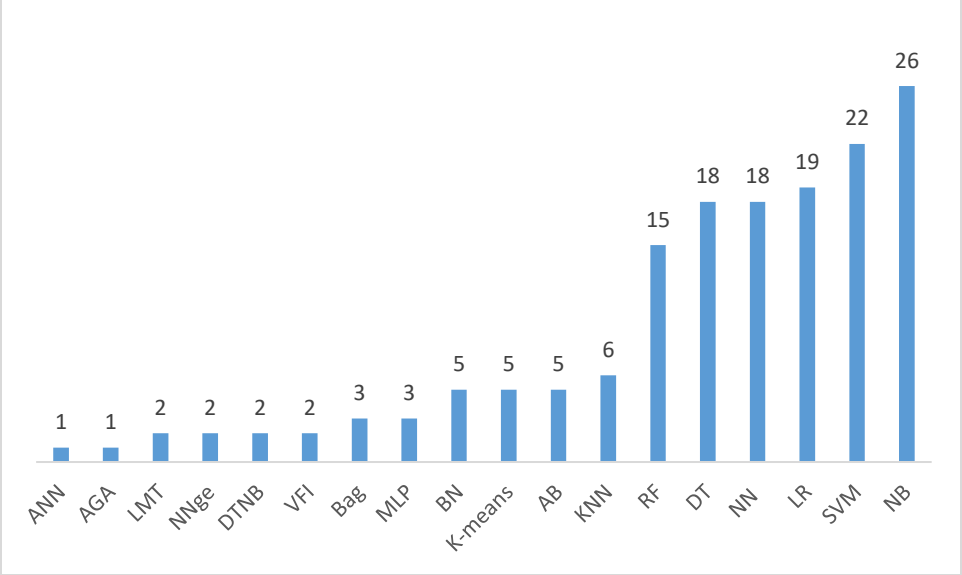

Figure 8 Machine Learning Algorithms

4.5 RQ-5: EVALUATION METRICS

We identified nineteen evaluation metrics in selected forty-seven articles. Precision, Recall, Accuracy metrics set practiced in thirty-one articles. Seventeen articles used ROC Curve, AUC metrics. F-measure used in ten publications. F1 scores were used in four publications. Mean Absolute Errors used in 4 publications. The probability of detection used in 4 publications. T-test, U-test, Statistical tests, PSA analysis, and DF Beta were used in 1 publication. Evaluation metrics are shown in Fig 9

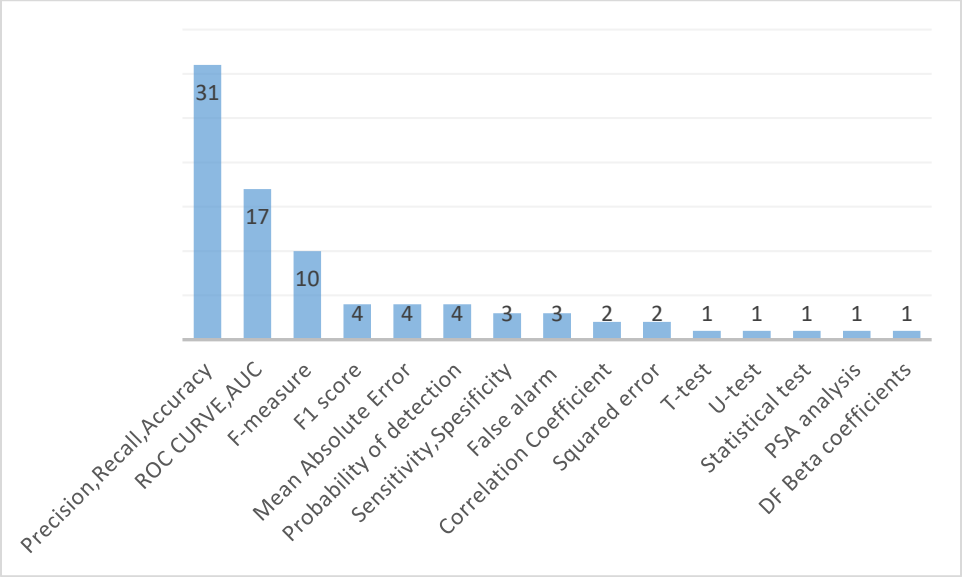

Figure 9 Evaluation Metrics

4.6 RQ-6: VALIDATION APPROACHES

In machine learning, validation techniques are used to increase the performance of the dataset before defect prediction. They are a few types of validation methods. *Leave one out cross-validation*; in this method, all the data except one record used for training and one used for testing. In k fold, cross-validation data have n splits and, in every split, selected following test data. From obtained results in selected studies, we classified the two most used types of validation techniques. 88% of studies used k-fold cross-validation. Leave-one-out validation methods were applied in 12% of studies. Fig 10 represents applied validation approaches in the pie chart.

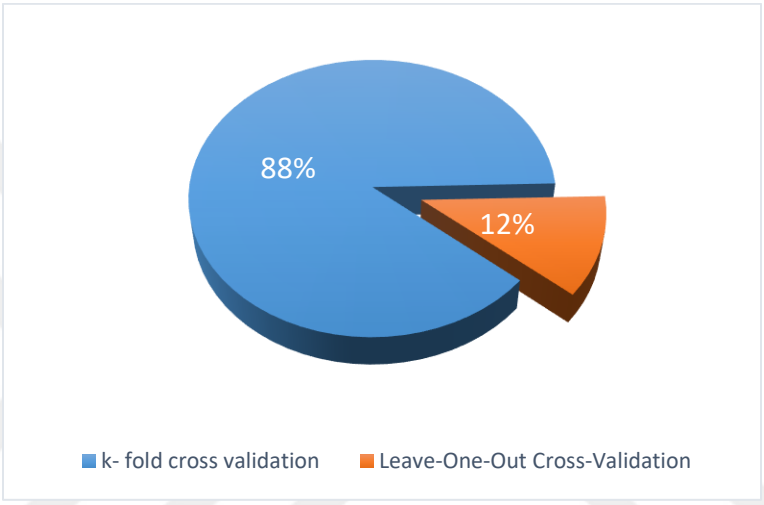

Figure 10 Validation Approaches

4.7 RQ-7: SOFTWARE METRICS

We extracted all the metrics used in primary studies. As seen in selected publications, many metric types have been used. Therefore, we decided to categorize metrics used in publications. Accordingly, we voted to classify metrics used in papers. Object-oriented metrics were used in 24 publications. Procedural metrics, Code churn, Change-related metrics, developer-related metrics, organization metrics, Change-process metrics, Halstead Complexity measures, McCabe's Cyclomatic Complexity in 11 publications. Web metrics (Table, Link, Graphic, List, Division, Word count, Page Title Length, Body Text Words, Page Size used in 2 publications. Process metrics MFA, MOA, DAM, IC, CAM, RFC, NPM, Statement metrics, Control flow graph metrics, call graph metrics, used in 2 publications. Performance metrics RT, AVL were used in 2 publications. Fig 11 shows categorized metric types and publications.

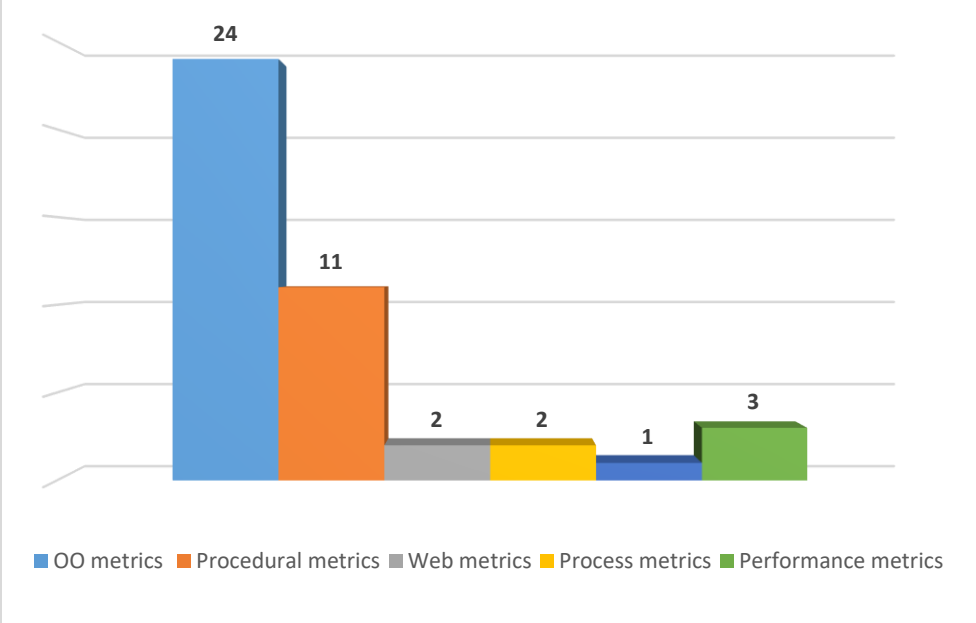

Figure 11 Software Metrics

4.8 RQ-8: THE BEST ALGORITHM

We categorized algorithms into two traditional Machine Learning and Deep Learning. Selected publications used many algorithms. Algorithms may work differently for different datasets, but we collected all and summarized them in this review. In primary studies, many algorithms were used, and as best, one author identified after a performance. If they used one algorithm, they performed only one of the best algorithms. Fig 12 shows the best Machine Learning algorithms. Support Vector Machines (SVM) and Decision Tree (DT) were identified four times as the best algorithm in publications. Random Forest (RF), Multilayer Perceptron (MLP), in three publications identified as the best algorithm. Bayesian Network (BN), Naive Bayes (NB), Logistic Regression (LR) were used 2 times as the best algorithm. The other algorithms Multiple Linear Regression (MLR), Multinominal Naïve Bayes (MNB), Co-trained Random Forest (Co Forest), Adaptive Boosting (AdaBoost), Gradient Boosting (GBDT), J48, Bootstrap aggregating (Bag), K-means++ (K-means clustering), KNN (K-nearest neighbors), Artificial Neural Network (ANN), An adaptive genetic algorithm (AGA) is used in one publication as the best algorithm used each one in one publication only as of the best algorithm.

Selected publications used many algorithms. Algorithms may be working differently for different datasets, but we collected all and summarized them in this review. In primary studies, many algorithms were used, and as best, one author identified after a performance. If all applied one algorithm, they produced particularly one of the most practical algorithms. Fig 12 shows the best Machine Learning classifiers. The other algorithms, Multiple Linear Regression (MLR), Multinominal Naïve Bayes (MNB), Co-trained Random Forest (Co Forest), K-means++ (K-means clustering), used in one publication as the best algorithm used each one in one publication only as of the best algorithm.

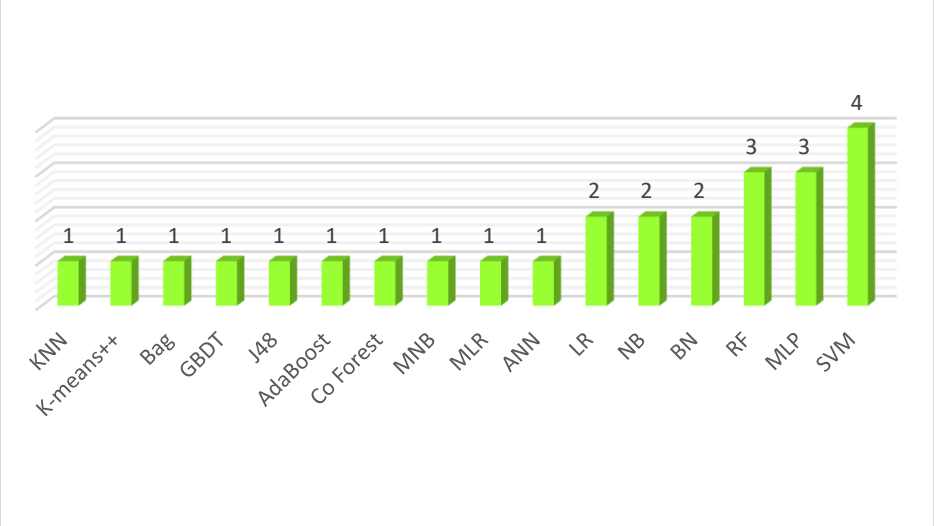

Figure 12 Machine Learning Algorithms Performance

Fig 13 shows the well-performed Deep Learning algorithms in records. Long Short-Term Memory was named in two publications as the fittest algorithm.

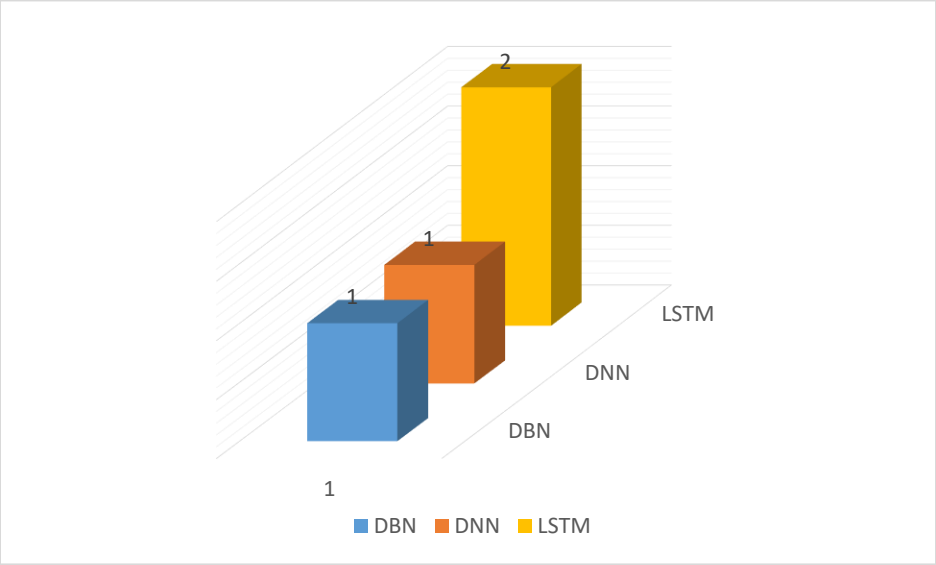

Figure 13 Deep Learning Algorithms

4.9 RQ-9: CHALLENGES

Inside that part, we show papers investigated in which hold gaps and weaknesses. Most academic articles discuss their study and results. This can be assumed in the latest definition of mobile applications performance tests to develop new methods for unseen issues, in Table 6 presented identified challenges and proposed solutions for selected studies.

Table 6 Challenges and possible solution

Challenges	Proposed Solutions	Reference
Metric selection limitations for mobile software	Limited code and process metrics	3,18
Faults in and Android data	Defects removed	9
Limited android mobile app repository	Open-source repository	11,27,28,35
Repeated data/code in the project	Domain Adaptation	26
Small data set problem.	Not mentioned	22
Different programming language problem	Defect prediction only GIT open-source Android, Java, and C++ uncertain	8,26,32,10
Modeling problem (Algorithms)	Not mentioned	30,4,11
Different platforms and languages	Not mentioned	18,21
Extensive data sets defect prediction problem.	Not mentioned	16
	Not mentioned	
Not fully automated	Manually code, log, bug, and review control	11
Imbalance Class problem	Sampling methods, Under sampling methods	7,12,22
Manual Feature Engineering	Not mentioned	26

5 RESPONSES TO RESEARCH QUESTIONS

This study aims to collect, synthesize, and evaluate mobile application defect prediction publications using machine learning techniques. To the best of our knowledge, there is no similar SLR paper published on this topic yet. Therefore, we performed this SLR study and aimed to answer some research questions that we defined at the beginning of this research. We believe that the observations and suggestions will pave the way for further research and help both practitioners and researchers in this field. Responses to research questions are briefly discussed as follows:

RQ1 – We noticed that most papers addressed Android platform, but Windows mobile operating system was discussed only in 2% of the studies. We also did not see any paper that focused on iOS platform. The reason is probably related to the open-source code bases of Android-based applications, which supported the researchers in a way that they were able to calculate the software metrics and collect the defect information from different publicly available repositories. Since many mobile platforms have Android operating system, many researchers prefer to perform experiments on this platform.

RQ2 –Many datasets have been stored in Github or Git repositories for defect prediction. These repositories are widely used by practitioners and researchers; therefore, the available datasets are mostly located in these platforms. This is also related to the open-source nature of Android applications; they are mostly hosted in these platforms. There were also a few datasets that used other platforms; however, their percentage was lower compared to the use of Github-related repositories.

RQ3 – Most of the studies used supervised learning approaches, there were limited number of papers that used unsupervised and semi-supervised learning techniques. The reason is that most researchers were probably able to get the available defect information from the publicly available repositories and therefore, they aimed to build supervised learning models instead of unsupervised or semi-supervised learning models. However, it is also possible to do some experiments in the context of available defect information by simulating different scenarios. There is still some room for further research on the use of these less preferred machine learning types.

RQ4 – Naive Bayes (NB), Support Vector Machines (SVM), and Logistic Regression algorithms are the most preferred algorithms. The reason is that most probably researchers preferred the widely used machine learning algorithms such as SVM and NB in their experiments. Previously, it has been also demonstrated that NB provides high performance in software defect prediction, and therefore, it might have been preferred in the mobile application defect prediction studies as well.

RQ5 – Most of the papers used precision, recall, and accuracy metrics to evaluate the performance of the models and also, Area under ROC curve (AUC) metric was preferred by researchers. These metrics are widely used in machine learning studies and therefore, researchers probably selected these metrics. Accuracy is not a good metric for defect prediction studies because these datasets are imbalanced and the accuracy metric cannot be used alone to judge the performance of the models, it must be used together with other metrics such as precision and recall.

RQ6 – Most studies used k-fold cross-validation strategy for evaluation of the model performance. This is also the widely used evaluation approach among machine learning approaches and therefore, researchers might have preferred to use this strategy. There are also other alternatives such as leave-one out technique, however, k-fold cross-validation was applied by most researchers. There is also possibility to perform k-fold n times, which can be called k*n cross-validation, however, the use of this strategy in these papers was quite limited although this approach can present more statistically sound results.

RQ7– Most of the studies used object-oriented metrics. This is probably due to the widely adoption of object-oriented programming paradigm in software industry. However, new metrics can be proposed and evaluated by researchers for mobile applications. This might be a potential research topic for researchers.

RQ8 – Support Vector Machines (SVM), Random Forest (RF), Multilayer Perceptron (MLP) were among the best performing algorithms. For deep learning algorithms, LSTM provided the best performance. When the dataset is not large, the use of deep learning algorithms is not needed. Therefore, the researchers

and practitioners should first consider the scale of the dataset before building the datasets. Highly accurate models can be built using SVM, RF, and MLP algorithms if the dataset is not large, therefore, the complexity of deep learning algorithms is not needed in such cases.

RQ9 – Several challenges were mentioned to answer this research question. We extracted these challenges from the papers if they were mentioned. However, there is a possibility that authors might not have discussed the challenges in the paper explicitly. In such cases, we were unable to add those challenges. If the challenge has not been experienced by researchers and mentioned as future work, they were also not included. There might be additional challenges that are missing in this paper, however, we aimed to collect them from the available literature.

5.1 POTENTIAL THREATS TO VALIDITY

In this part, we will describe the validity considerations of this systematic literature review. We selected publications from six digital platforms (Sec. 3) from 2007 till 2020. We performed research questions in 2020 and extracted the first keywords to ensure that all relevant papers till 2020 are covered. We conducted a manual search on all digital databases on the published papers in the past thirteen years. The article concludes that the activity is functioned by rejection and addition rules. It was completed in two steps: the paper's first author applied the standards and refined out not related publications second authors double-checked in the selection process. We performed quality assessment questions and applied them to the selected publications to include only initial studies. Quality Assessment questions are assumed from the study of [35] were practiced, for each question score was given. If yes, the highest rate is 2, and partial rates are 1; if no rates, then 0. For each publication, prepared the ID number and, after reviewing all publications, summarized results. If that overall score of the paper was less than four, the paper was excluded. We summarized our findings and reported that the impact of the defect on software might miss in-depth analysis.

All research questions and answers, data synthesis, and correctness of extraction were checked and discussed among the authors. Disagreements settled during data extraction between us. Data extraction was performed manually by the author and validated by research fellows. To remove publications, we agreed on during meetings. We assume we diminished the leading threats to validity for this review with the stated parts approximately probable. We selected publications from six digital platforms using our search criteria and also conducted a snowballing process. Authors held several meetings to minimize the researcher bias. However, there might be some papers in some electronic databases that we have missed in this research. Also, new papers are also published very frequently and therefore, we might have missed some new papers published recently. Another threat is the use of the search criteria. There might be more synonyms that could have been used in this research and we have missed some papers due to this issue.

6 MODEL DEVELOPMENT

The goal of this study is to develop model for mobile applications defect prediction using deep learning algorithms. We used Python data science libraries, NumPy for numerical processing, pandas for data analysis, matplotlib for visualization, scikit learn for processing machine learning datasets. Machine learning is the practice of using algorithms to analyze data, learn from that data, and then decide or

prediction about new data. In machine learning we can use different algorithms to classify and predict the desired output. But machine learning algorithms do not consider the context of the word as a sequence. It works based on the number of times the word is repeated a probability is derived out of it and then performs a classification task. Deep Learning learns a pattern of a word or a sequence and then tries to predict the desired outcome of the task. For deep learning we use Keras API Classes. To use Estimator API, we define a list of feature columns, create the Estimator model, create a data input function, call train, evaluate, and predict methods on the estimator object.

6.1 DATASETS

We used open source available 14 Android applications datasets from previous studies [36]. The datasets are available on COMMIT GURU platform [37]. The datasets are numerical, different size and include 6 columns and 34042 lines. Table 5 shows name of datasets, web address and downloads.

Table 7 Datasets

Datasets	Repository	Lines	Downloads
Afwall	https://github.com/ukanth/afwall	1025	500 000
Alfresco	https://github.com/Alfresco/alfresco-android-app	1004	50 000
androidSync	https://figshare.com/s/9a075be3e1fb64f76b48	209	100 000
androidWalpaper	https://github.com/olivergeith/android_wallpaperDesigner	588	5 000 000
anySoftKeyboard	https://github.com/AnySoftKeyboard/AnySoftKeyboard	2971	25 271
Apg	https://github.com/thialfihar/apg	3780	N/A
atmosphere	https://github.com/Atmosphere/atmosphere	5474	1 000 000
chatSecure	https://github.com/guardianproject/ChatSecureAndroid	2579	N/A
facebook	https://github.com/facebook/facebook-android-sdk	548	5 000 000 000
flutter	https://github.com/flutter/flutter	10405	100 000
kiwis	https://github.com/kiwix/kiwix-android	1373	1 000 000
owncloudandroid	https://github.com/owncloud/android	3700	100 000
Pageturner	https://github.com/NightWhistler/PageTurner	164	50 000
reddit	https://github.com/emmaguy/wear-notify-for-reddit	222	50 00 000

6.2 DATA PROCESSING

We controlled number of examples belongs to true or false class to identify dataset type. Dataset is imbalanced (False class is more than True class) shown in Figure 14. Then we searched null values on

datasets if datasets contain nonvalues, it will create a problem. We changed null values belongs to the feature columns than we filled with zero values (zero means this feature has no effects on prediction) and False for Contain bug folder. Then we split dataset X-feature columns, Y-class columns, data X-input, data Y-output. Machine understand 0,1 but we want to predict true and false.

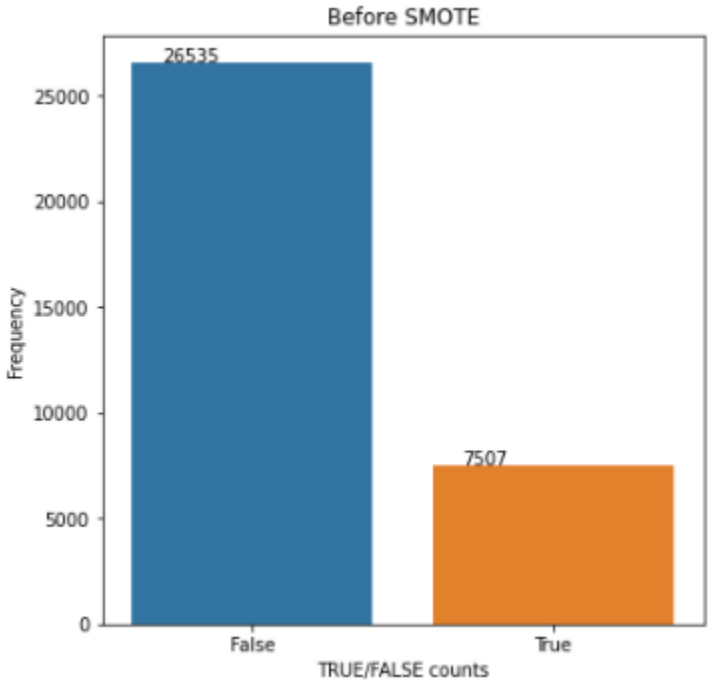

Figure 14 Dataset classes distribution

We must use label to encoder to encode this label to a specific number (0,1,2) and sklearn preprocessing technique to transform 0 -false ,1 – true. After this we split our dataset into train and testing and in every loop, we combine all datasets except one which will be used for testing. X_train, X_test, Y_train, Y_state, X features and Y classes. We used random state as parameter of test split, every time when run cell, if we have 1000 example in dataset random state choose randomly 80% of training and test dataset. We checked X_train.shape, X_test. Shapes to see how many examples we have; we split and get size. Correlation between every feature show Figure 15.

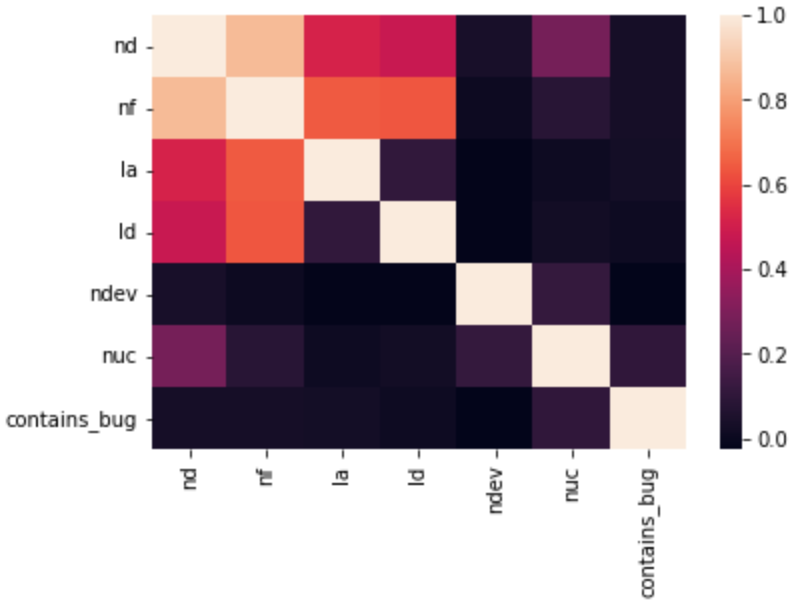

Figure 15 Core between features

6.3 TEN-FOLD CROSS-VALIDATION

We applied 10-fold cross-validation to dataset to improve performance of our model. Accuracy of random state might be not true. But with cross validation we can resolve this problem. We split dataset into 10 folds and cross-validation uses 10 blocks to train the methods and then last block to test the method and keeps the track how of how well the method did with the test data. From training set we get a best learning result, from testing test best validation result. We believe that assessment of our model will be more accurate using 10-fold cross-validation.

6.4 FEATURE SELECTION

We use supervised machine learning to predict features in the dataset. Features have two critical properties: units and magnitude computed with them. When we have a different feature in the dataset, it will be calculated by various units and magnitude. To run data into a neural network, we need to scale it—two techniques, normalization, and standardization, used for data scaling. Normalization scales down data between 0 to 1. standardization helps to scale down data based on a standard normal distribution. We need to scale down values between zero to one in deep learning. In Convolutional Neural Network images, pixels are between 0 to 1. Artificial Neural Network, which uses TensorFlow, Keras libraries except for input between 0 to 1, will help learn weights quickly. To do that, we can use MinMaxScaler. We create an instance of a MinMaxScaler after this; we fit our data. Fitting data allows the model to know each column's minimum and maximum values.

6.5 DATA BALANCING

In this study, we are searching for solutions for mobile applications defects. We are studying Android applications numerical datasets. In binary classification, we can end up with one class, which is essential for this study. However, we do not have enough data. In our datasets, a majority class is more significant than a minority class, and we have a class imbalance problem. Our minority class is a focus class. There are a few techniques to create fake data and balance datasets: Random Oversampling – from imblearn library, makes a new sample for minority class by sampling with replacement. We discovered that in literature studies, Random OverSampler reported that their model did not provide the best performance, and we dropped out of this method. Random Under-sampling picks random samples of the majority class, and after sampling, the majority class and minority class have the same data points. Near Miss Under-sampling method uses KNN to do under-sampling. There are a few types of this method, near-miss selects positive samples to the nearest distance, to farthest distance and negative samples kept, and positive chosen samples to the average of N closest and most significant. We focus on SMOTE because of its performance and use a Python imblearn library, and oversampling method SMOTE to balance train dataset and increase minority class of dataset [38]. SMOTE takes each minority sample and introduces synthetic data points connecting the minority sample and its nearest neighbors. Neighbors from the K-nearest neighbors were chosen randomly, same random oversampling.

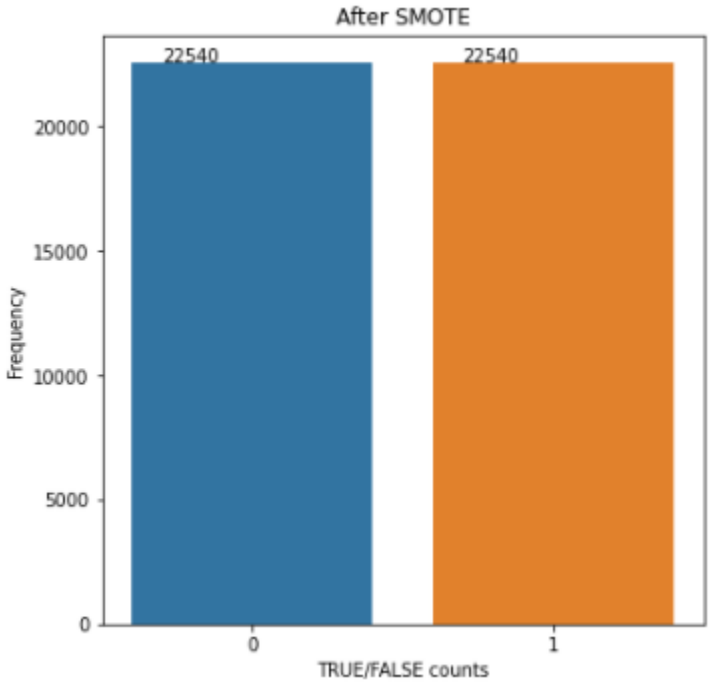

Figure 16 Dataset classes after SMOTE

6.6 DEEP LEARNING ALGORITHMS

This section identifies algorithms used for the defect prediction model development of Android applications. We analyzed previous studies and identified most studies used traditional Machine learning methods and focused on web and desktop applications. Deep learning algorithms were used in a few studies, for mobile applications were limited research. We developed our model with deep learning algorithms and believe that the performance of our model will be best.

Artificial Neural Networks are computing systems inspired by the brain's neural networks. These networks are based on a collection of connected units called artificial neurons. Each connection between neurons can transmit a signal from one neuron to another [39]. The receiving neuron processes the signal and signals downstream neurons connected to it. The structure of Artificial Neural Networks has: Neurons are organized in layers, Input layer (absolute values from the data), Hidden layers (layers between input and output, three or more hidden layer is "deep network"), Output layer (final estimate of the output). Artificial Neural Networks are typically organized in layers. Different types of layers include Dense (or fully connected) Layers, Convolutional layers, Pooling layers, Recurrent layers. Additional layers may perform various transformations on their inputs. The convolutional layer is most likely used with image data, the Recurrent layer will be used for time-series data, and the Dense layer is a layer that connects each input to each output within its layer. We use the Keras library to build our neural network. Our model is sequential; we have an input layer, three hidden layers, and an output layer—our Artificial Neural Network parameters are shown in Fig 10. We use kernel initializer for weights and activation functions such as Relu, SoftMax, and Sigmoid; we also used adam optimizer and binary_crossentropy. We fit with X_train, y_train, batch_size= 10, epochs=100 and run our model presented in Figure 17.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 16)                112

 dense_11 (Dense)            (None, 12)                204

 dense_12 (Dense)            (None, 10)                130

 dense_13 (Dense)            (None, 3)                 33

 dense_14 (Dense)            (None, 1)                 4

=================================================================
Total params: 483
Trainable params: 483
Non-trainable params: 0
```

Figure 17 ANN parameters

We started building our Artificial Neural Network model with an input layer with six nodes because we have six columns in our dataset. Then we add hidden layers 12, 10, 3 nodes and the output layers are one node. Our model trained for 100 epochs and presented in Figure 18. Epoch is when all datasets pass

neural network. Then 1,2,3. We need to learn from the dataset and update the model. Batch size is how many % data we want to process in a loop step by step. First ten samples, then Artificial Neural Network gives another ten until the end with randomly given samples.

```
Epoch 183/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4862 - accuracy: 0.7628
Epoch 184/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4865 - accuracy: 0.7616
Epoch 185/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4870 - accuracy: 0.7612
Epoch 186/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4875 - accuracy: 0.7604
Epoch 187/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4861 - accuracy: 0.7626
Epoch 188/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4858 - accuracy: 0.7621
Epoch 189/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4856 - accuracy: 0.7618
Epoch 190/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4864 - accuracy: 0.7609
Epoch 191/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4855 - accuracy: 0.7623
Epoch 192/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4855 - accuracy: 0.7607
Epoch 193/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4848 - accuracy: 0.7625
Epoch 194/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4853 - accuracy: 0.7614
Epoch 195/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4843 - accuracy: 0.7637
Epoch 196/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4830 - accuracy: 0.7636
Epoch 197/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4847 - accuracy: 0.7599
Epoch 198/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4836 - accuracy: 0.7637
Epoch 199/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4838 - accuracy: 0.7638
Epoch 200/200
2605/2605 [==============================] - 11s 4ms/step - loss: 0.4824 - accuracy: 0.7644
<keras.callbacks.History at 0x7efcd10b2890>
```

Figure 18 ANN epochs

All data processed in training and the background of the model, we will calculate the loss function from the actual class, prediction class we will update model, if the model profound very well the model stopped learning. The architecture of our Artificial Neural Networks is shown in Figure 19. We need the test dataset to predict accuracy. We will evaluate the model with 10-fold cross-validation and calculate ROC AUC, confusion matrix, accuracy metrics.

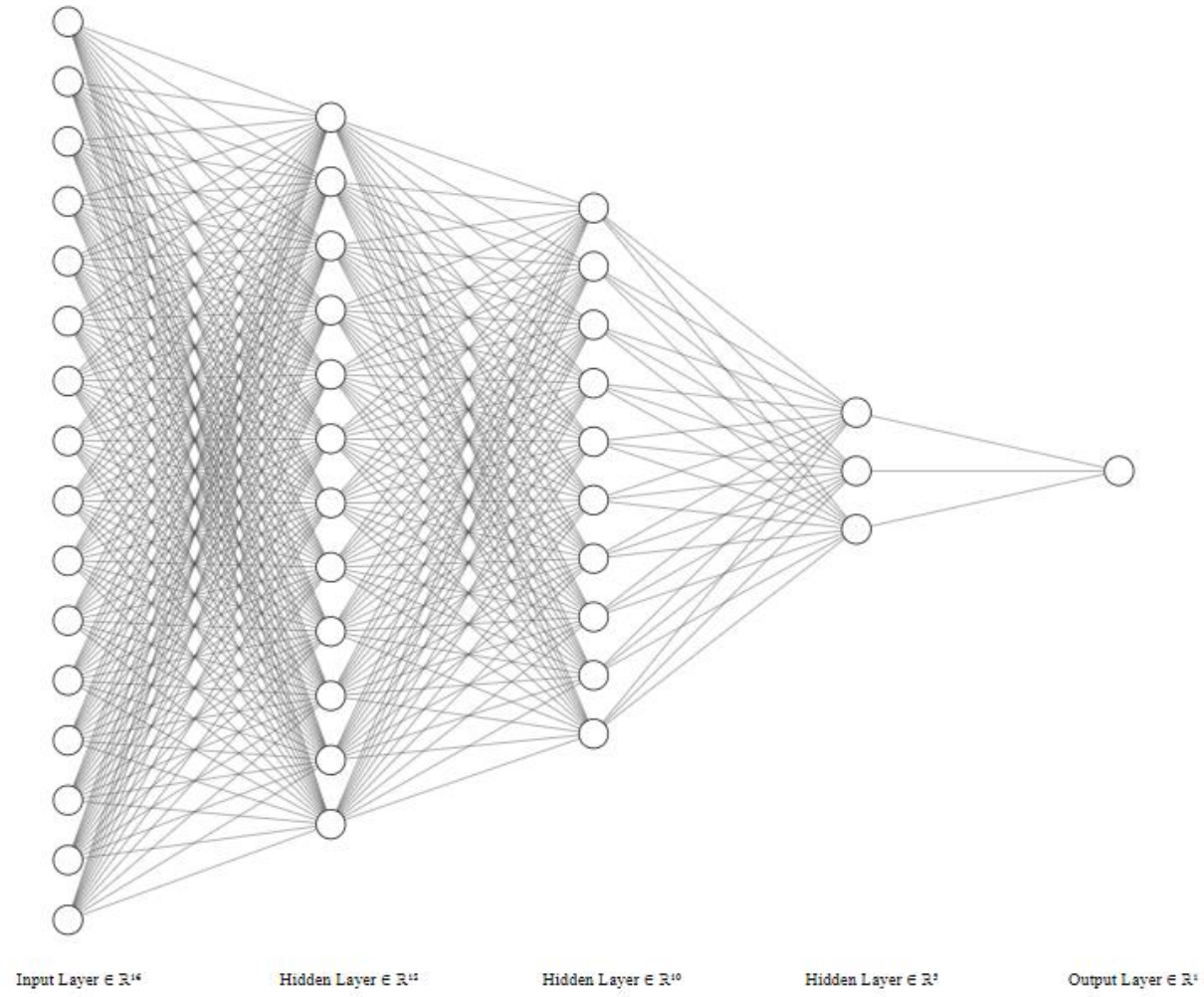

Figure 19 Arhitecture of ANN

Convolutional Neural Network (CNN) - is an Artificial Neural Network that is most used for analyzing images. They can also be used for other data analysis or data classification. Most generally, we can think of CNN as an Artificial Neural Network specializing in picking out or detecting patterns or making sense of them. This pattern detection is what makes CNN useful for image analysis. Convolutional Neural Network is just some form of an Artificial Neural Network with different shapes from a standard MLP. CNN has hidden layers called convolutional layers, and this layer precisely makes CNN as CNN. CNN has other non-convolutional layers, but the basis of a CNN is the convolutional layers. The convolutional layer receives input and then transforms and then outputs the transformed inputs layer. With a convolutional layer, this transform is a convolutional operation. Recall that Tensors are N-Dimensional Arrays that we build up to: Scaler -3, Vector - [3,4.5]**,** Matrix - [[3,4], [5,6], [7,8]]**, Tensor** - [[[1,2], [3,4]], [[5,6], [7,8]]]. Tensors make it very convenient to feed in sets of images into our model - (I, H, W, C)**.** I: Images**,** H: Height of Image in Pixels**,** W: Width of Image in Pixels**,** C: Color Channels: 1-Grayscale, 3-RGB. The difference between a Densely Connected Neural Network and a Convolutional Neural Network, recall that we have already been able to create Deep Neural Network with tf. estimator API. In a Densely Connected layer, every neuron in one layer is directly connected to every other neuron to

another layer. For the Convolutional layer, we have a different approach. Each unit is connected to a smaller number of nearby units in the next layer.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 1, 6, 64)          128

 max_pooling2d (MaxPooling2D  (None, 1, 1, 64)          0
 )

 conv2d_1 (Conv2D)            (None, 1, 1, 32)          2080

 max_pooling2d_1 (MaxPooling  (None, 1, 1, 32)          0
 2D)

 dropout (Dropout)            (None, 1, 1, 32)          0

 flatten (Flatten)            (None, 32)                0

 dense_5 (Dense)              (None, 250)               8250

 dropout_1 (Dropout)          (None, 250)               0

 dense_6 (Dense)              (None, 1)                 251

=================================================================
Total params: 10,709
Trainable params: 10,709
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 20 Convolutional Neural Networks parameters

Convolutions also have a significant advantage for image processing, where pixels nearby are much more correlated to each other for image detection. Each Convolution Neural Networks layer looks at an increasingly more significant image part. Having units only connected to nearby units also helps with regularization, limiting the search of weights to the size of the convolution. Convolutional Neural Networks did not practice before for mobile application defect prediction. We decided to build a sequential model using Convolutional Neural Networks too. We used Keras library, NumPy, Scikit learn libraries, Conv2D, MaxPooling2D, dropout, flatten, dense layers, activation functions such as relu, SoftMax, sigmoid. We reshaped our train and test data to make it easy to process our model efficiently. We compiled a model to measure performance with adam optimizer and used loss functions such as binary cross entropy. Convolutional Neural Network model parameters are presented in Figure 21.

```
Epoch 83/100
1384/1384 [==============================] - 3s 3ms/step - loss: 0.5020 - accuracy: 0.7422
Epoch 84/100
1384/1384 [==============================] - 3s 2ms/step - loss: 0.5004 - accuracy: 0.7427
Epoch 85/100
1384/1384 [==============================] - 4s 3ms/step - loss: 0.5007 - accuracy: 0.7440
Epoch 86/100
1384/1384 [==============================] - 3s 2ms/step - loss: 0.5007 - accuracy: 0.7425
Epoch 87/100
1384/1384 [==============================] - 3s 2ms/step - loss: 0.4983 - accuracy: 0.7463
Epoch 88/100
1384/1384 [==============================] - 3s 2ms/step - loss: 0.4983 - accuracy: 0.7441
Epoch 89/100
1384/1384 [==============================] - 3s 2ms/step - loss: 0.5003 - accuracy: 0.7437
Epoch 90/100
1384/1384 [==============================] - 3s 2ms/step - loss: 0.4985 - accuracy: 0.7450
Epoch 91/100
1384/1384 [==============================] - 3s 2ms/step - loss: 0.4971 - accuracy: 0.7465
Epoch 92/100
1384/1384 [==============================] - 4s 3ms/step - loss: 0.4977 - accuracy: 0.7458
Epoch 93/100
1384/1384 [==============================] - 4s 3ms/step - loss: 0.4983 - accuracy: 0.7425
Epoch 94/100
1384/1384 [==============================] - 3s 2ms/step - loss: 0.4990 - accuracy: 0.7441
Epoch 95/100
1384/1384 [==============================] - 4s 3ms/step - loss: 0.4981 - accuracy: 0.7436
Epoch 96/100
1384/1384 [==============================] - 4s 3ms/step - loss: 0.4972 - accuracy: 0.7477
Epoch 97/100
1384/1384 [==============================] - 4s 3ms/step - loss: 0.4976 - accuracy: 0.7448
Epoch 98/100
1384/1384 [==============================] - 4s 3ms/step - loss: 0.4973 - accuracy: 0.7457
Epoch 99/100
1384/1384 [==============================] - 3s 2ms/step - loss: 0.4966 - accuracy: 0.7446
Epoch 100/100
1384/1384 [==============================] - 3s 3ms/step - loss: 0.4962 - accuracy: 0.7469
<keras.callbacks.History at 0x7fc815454790>
```

Figure 21 Convolutional Neural Network epochs

We started building our Convolutional Neural Network with the sequential method and convolution layer. The convolution means core between blocks of CNN. We applied a 2D Convolutional layer with 64 filters and one kernel size; we defined Relu (Rectified Linear Unit activation function). Input data multiplied by weights and the point of training model is by adjusting weights and biases to find the appropriate values representing the training data. Neuron gets a bunch of signals and outputs the signals. The activation function decides output or not signals. There are a few types of activation functions. Sigmoid is used to predict probability of output. Relu is used most in convolutional neural networks and other classifications. As the second layer, we applied the Max Pooling 2D layer with (1,6) filters in our model. Max Pooling 2D operation takes the higher value from the convolutional layer and replaces it with the output. There are also different types of pooling layers, such as the Mean Pooling layer, which takes the low values from the convolutional layer output and replaces it in the Mean Pooling layer output.

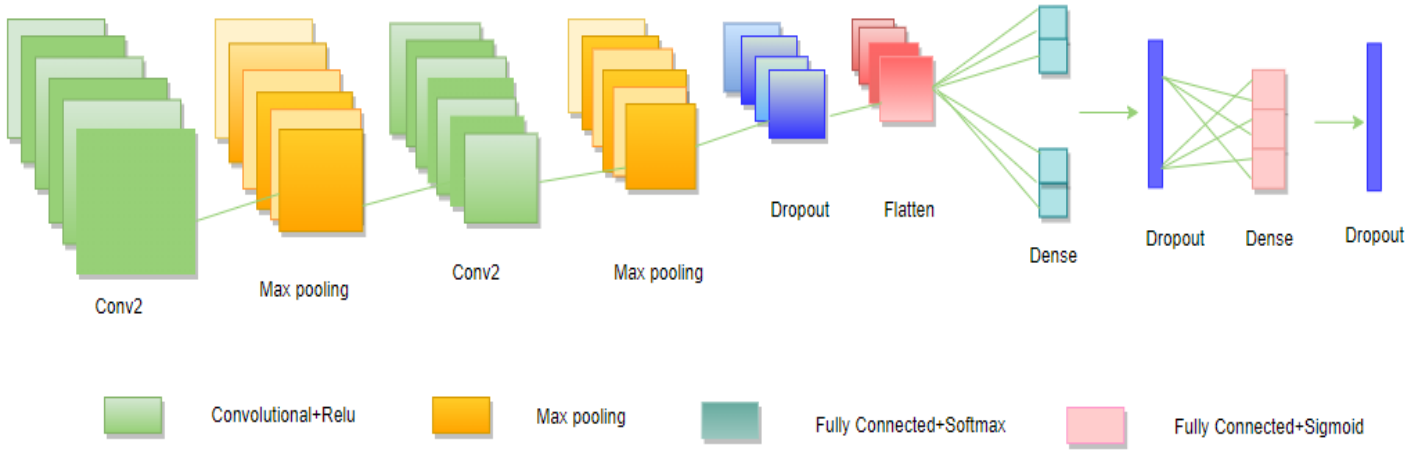

Figure 22 Convolutional Neural Network Architecture

Third layer of our model is the Convolutional 2D layer with 32 filters and one kernel size, and the activation function is Relu. Then we add Max Pooling 2D layer with (1,1) filters. We use Drop Out layer to reduce the overfitting problem. Then we applied to Flatten to our model because the last layer is dense, and Artificial Neural Network requires values in a one-dimensional feature vector. Flattening gets the output of the Convolutional or Max Pooling layer to Flatten all its structures to create a single feature vector that the Dense layer can use for final classification. We added a Dense layer with 250 nodes and activation function SoftMax because it gives the best performance and then Drop Out layer. Final and output layer with one node and activation function Sigmoid. We compile the model with Adam Optimizer based of datasets, and for deciding parameters, we use loss function Binary Cross Entropy. We trained Convolutional Neural Network with 100 epochs and 10 batch size. Figure 24 shows the architecture of a Convolutional Neural Network.

Long short-term memory (LSTM) - is an Artificial Recurrent Neural Networks (RNN) architectures type and it is used in deep learning, and it allows Neural Networks to remember data and forget not applicable data. We can say that Long Short-Term Memory is a type of Recurrent Neural Network. There are a few types of Recurrent Neural Networks such as one to one used for image classification, one to many images captioning, many to one sentiment analysis, many to many languages' translation. Recurrent Neural Networks are good for short context classification. In Recurrent Neural Network, processed input data after output is provided to the input in the next step and new input data. It allows Recurrent Neural Network to remember previous steps in the sequence.

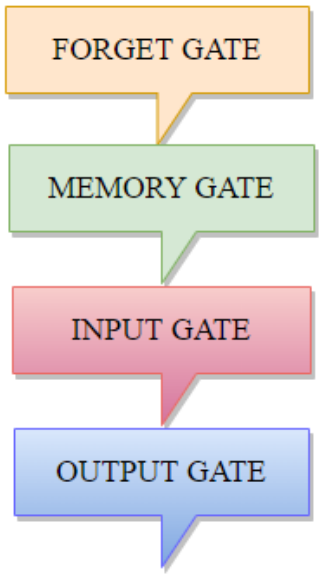

Figure 23 Long Short-Term Memory Cells

However, in Recurrent Neural Network, more data is a problem. It makes not effective in learning new data. Long Short-Term Memory designed for those issues of Recurrent Neural Network. Long Short-Term Memory provides a solution for this problem; it is named as a state. The state is a cell consisting of four parts and each part a gate: forget gate, memory gate, input gate, output fate. Forget gate says the sort of state in which data stored in the internal state can be forgotten, and it is no longer relevant. Input gate says what new data we should add or update into working storage data. Output gate says all the data stored in that state which part should be output in this instance. Numbers between 0-1 can assign these gates; 0 means the gates are effectively closed, one means the gate is wide open. We can make regularizations on it, add, or add a little bit. LSTM cells are shown in Figure 24.

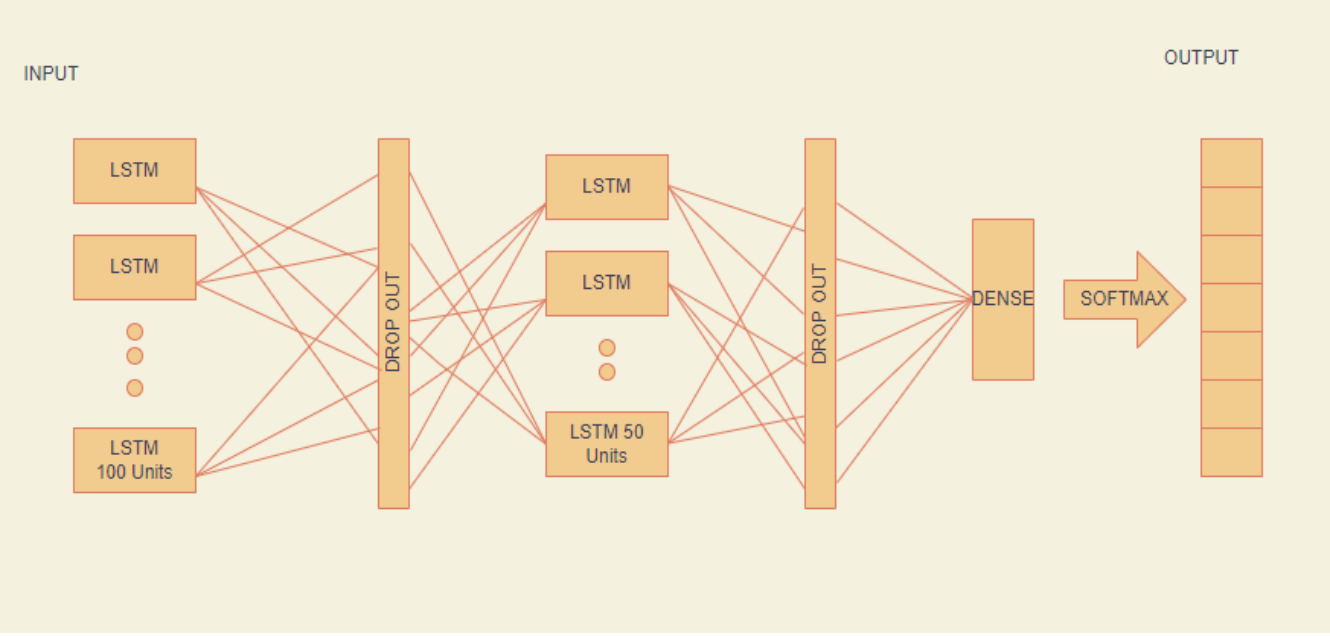

Figure 24 Long Short-Term Memory Architecture

We used Long Short-Term Memory model for Android applications defect prediction. We imported Keras, TensorFlow libraries, because LSTM expects data in a specific shape, we reshaped our data. We defined num_of_samples, timesteps, nb_features, nb_out. We started building our model with sequential method and added first LSTM layer with 100 units. We used return sequences true because we want each output signal is returns to an input. The second layer used Drop Out to reduce overfitting. Third layer is LSTM with 50 units and return sequences false, then Drop Out layer used. Last layer is Dense with activation function Sigmoid. We compiled our model with Binary Cross Entropy and Adam Optimizer and Accuracy metrics. In Figure 25 presented Long Short-Term Memory model parameters and in Figure 24 shown LSTM architecture.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm (LSTM)                  (None, 1, 100)            42800

 dropout_2 (Dropout)          (None, 1, 100)            0

 lstm_1 (LSTM)                (None, 50)                30200

 dropout_3 (Dropout)          (None, 50)                0

 dense_7 (Dense)              (None, 1)                 51

=================================================================
Total params: 73,051
Trainable params: 73,051
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 25 LSTM parameters

We trained our model and evaluated results with tenfold cross-validation. Figure 26 shows the Long Short-Term Memory model training. Model loss and accuracy curve are presented in Figure 27, 28.

```
4208/4208 - 18s - loss: 0.5915 - accuracy: 0.6952 - val_loss: 0.6716 - val_accuracy: 0.7287
Epoch 9/100
4208/4208 - 18s - loss: 0.5906 - accuracy: 0.6970 - val_loss: 0.5739 - val_accuracy: 0.7869
Epoch 10/100
4208/4208 - 18s - loss: 0.5894 - accuracy: 0.6967 - val_loss: 0.6091 - val_accuracy: 0.7648
Epoch 11/100
4208/4208 - 18s - loss: 0.5892 - accuracy: 0.6971 - val_loss: 0.7419 - val_accuracy: 0.7219
Epoch 12/100
4208/4208 - 18s - loss: 0.5886 - accuracy: 0.6976 - val_loss: 0.5961 - val_accuracy: 0.7702
Epoch 13/100
4208/4208 - 18s - loss: 0.5881 - accuracy: 0.6986 - val_loss: 0.6367 - val_accuracy: 0.7625
Epoch 14/100
4208/4208 - 18s - loss: 0.5879 - accuracy: 0.6977 - val_loss: 0.5196 - val_accuracy: 0.8163
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_retur
INFO:tensorflow:Assets written to: ../Trained_model/assets
INFO:tensorflow:Assets written to: ../Trained_model/assets
Epoch 15/100
4208/4208 - 19s - loss: 0.5874 - accuracy: 0.6994 - val_loss: 0.6356 - val_accuracy: 0.7625
Epoch 16/100
4208/4208 - 18s - loss: 0.5872 - accuracy: 0.6988 - val_loss: 0.5949 - val_accuracy: 0.7887
Epoch 17/100
4208/4208 - 18s - loss: 0.5859 - accuracy: 0.7003 - val_loss: 0.6347 - val_accuracy: 0.7738
Epoch 18/100
4208/4208 - 19s - loss: 0.5862 - accuracy: 0.7005 - val_loss: 0.5886 - val_accuracy: 0.7806
Epoch 19/100
4208/4208 - 18s - loss: 0.5865 - accuracy: 0.7012 - val_loss: 0.5336 - val_accuracy: 0.8099
Epoch 20/100
4208/4208 - 18s - loss: 0.5862 - accuracy: 0.6998 - val_loss: 0.6069 - val_accuracy: 0.7688
Epoch 21/100
4208/4208 - 18s - loss: 0.5848 - accuracy: 0.7005 - val_loss: 0.5682 - val_accuracy: 0.7819
Epoch 22/100
4208/4208 - 18s - loss: 0.5849 - accuracy: 0.7014 - val_loss: 0.6294 - val_accuracy: 0.7725
Epoch 23/100
4208/4208 - 17s - loss: 0.5853 - accuracy: 0.7020 - val_loss: 0.5636 - val_accuracy: 0.7928
Epoch 24/100
4208/4208 - 18s - loss: 0.5852 - accuracy: 0.7006 - val_loss: 0.5951 - val_accuracy: 0.7747
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Figure 26 Long Short-Term Memory Epochs

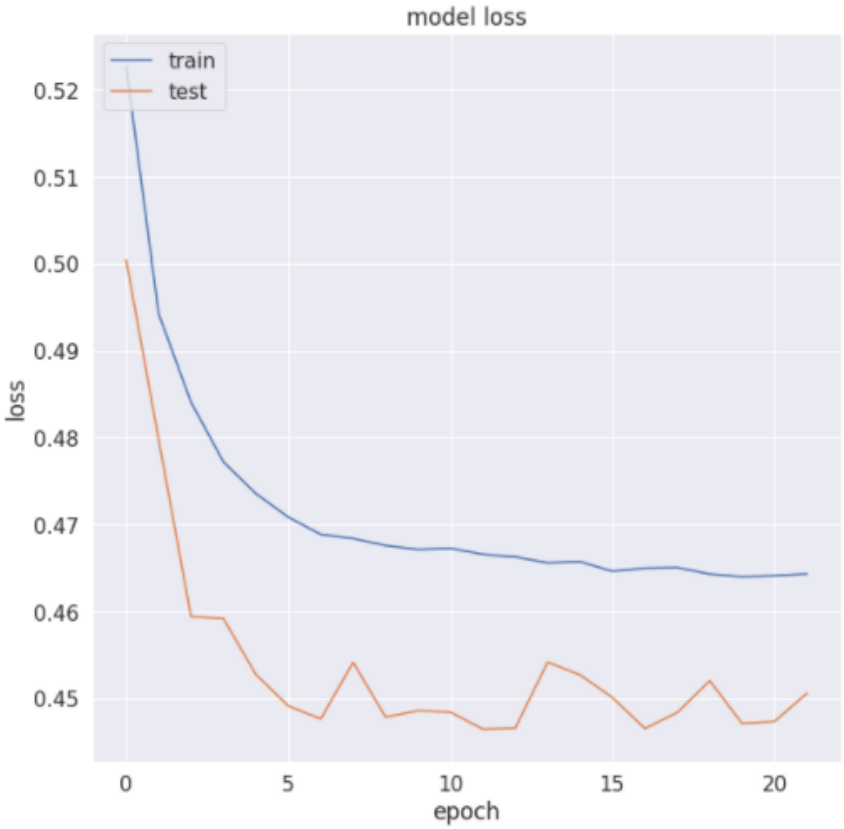

Figure 27 Long Short-Term Memory Model Loss

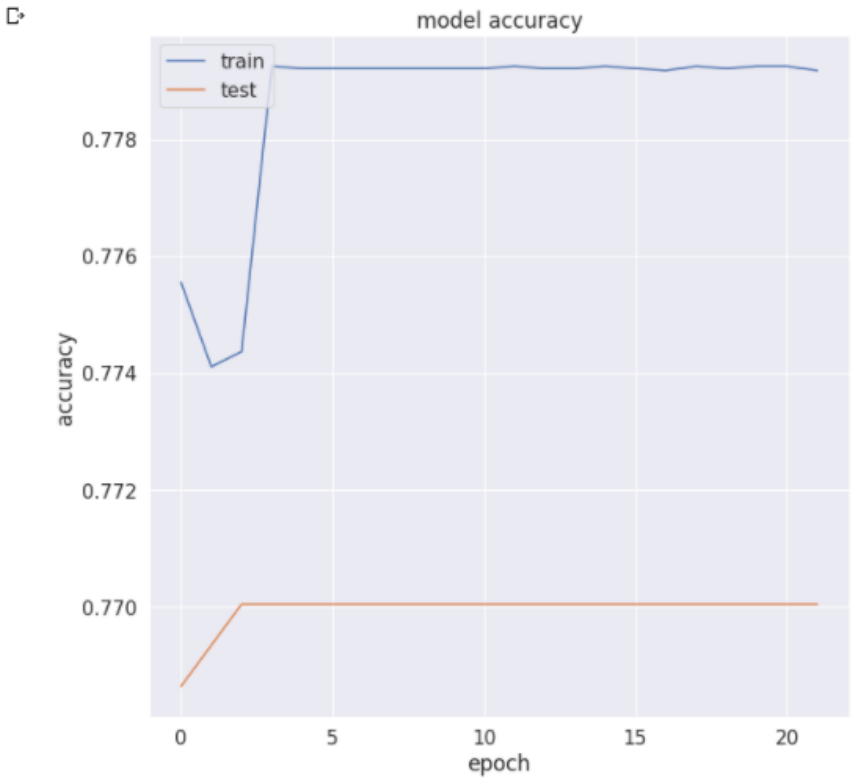

Figure 28 Long Short-Term Memory Model Accuracy

6.6 ACCURACY METRICS

In this section we explain about accuracy metrics used in this study. We trained our model in training data and tested with testing data. Now we need to summarize our predicted results. One way is to do this is a Confusion Matrix method. We have two categories faulty and non-faulty in the last column we have a True Positives – this is faulty classes identified by algorithm. True Negatives – this is non-faulty classes correctly identified by algorithm, False Negatives -it is faulty classes, but algorithms say non-faulty classes, False Positives -it is non-faulty classes, but algorithm says it faulty.

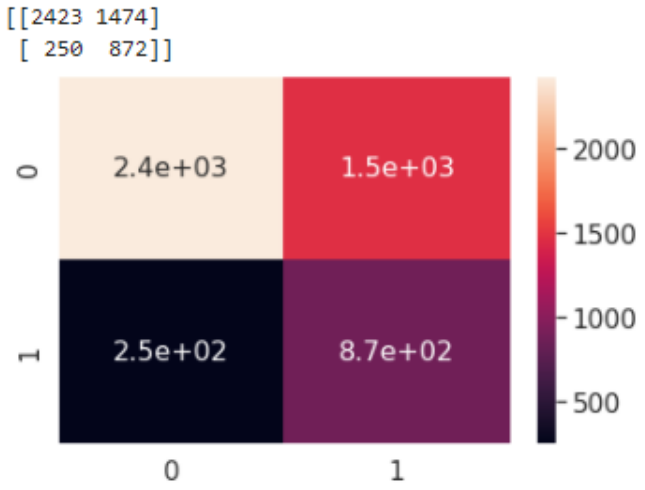

Figure 2929 Confusion Matrix ANN

We use a binary classification because of datasets type. In classification there are two statements: class labels and probability. We applied Confusion Matrix to see how many predicted classes were correctly predicted and how wany were not. X axis is predicted, and Y axis is the class label. In balanced dataset we can calculate the accuracy from confusion matrix and get results but in imbalanced dataset values in categories are very different one category can be maximum and another minimum.

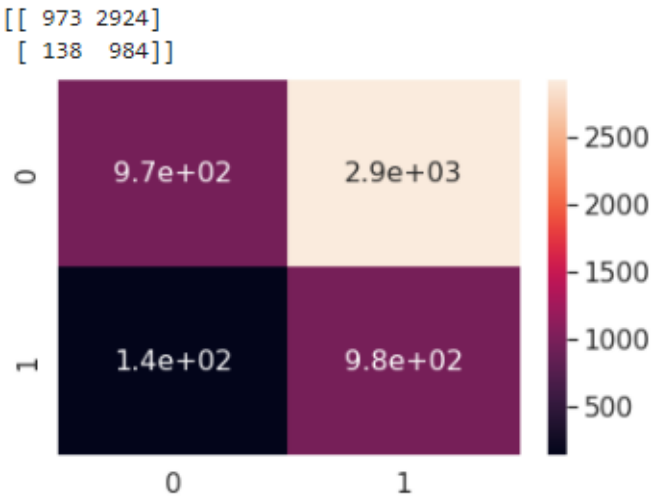

Figure 30 Confusion Matrix CNN

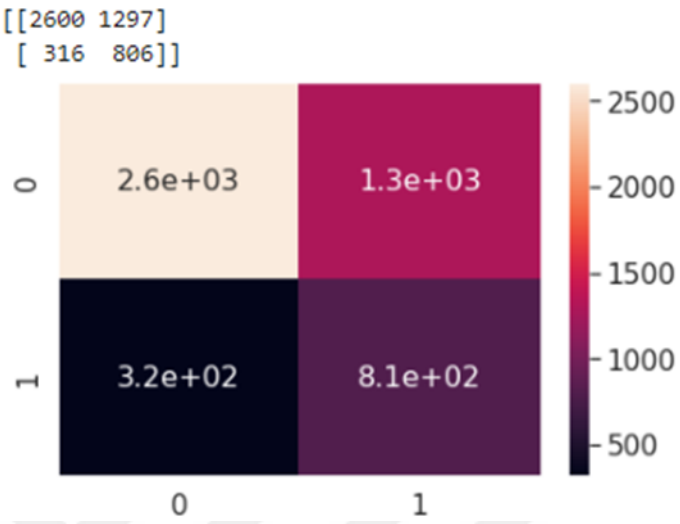

Figure 31 Confusion Matrix LSTM

Accuracy calculated with these values will give a bad meaning of a model. Our Confusion Matrix results are presented in Figures 29,30,31. Instead of our datasets, results shows that true positives are more than false positives. If we calculate accuracy with formula (1):

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

it will be not the best results. We focused for Recall and Precision because these metrics are used for imbalance datasets. In Recall, out of the total actual positive values how many values did we correctly predicted positively. Recall also named ad true positive rate or sensitivity. Precision, out of the total predicted results how many results are positive. Precision also named as positive prediction value. Formula of Precision and Recall shown as follow as:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

We applied F-beta score because, all values are important from actual and predicted class. If we consider our beta value as 1 our F-beta becomes an F1-Score. The formula of F-beta show as below:

$$F_\beta = (1 + \beta^2) \frac{Precision \times Recall}{\beta^2 \times Precision + Recall} \quad (4)$$

We combined Precision and Recall selecting right metrics. In a binary classification also used ROC (Receiver Operating Characteristics) and AUC (Area Under Curve) performance metrics. ROC represents probability of curve and AUC represents measurement of separability. If the area under curve higher it means model is better at predicting classes. It is mostly used in binary classification such as, fraud detection, spam detection, software defect prediction. When AUC is 1 it means model performing well, when model AUC is 0 or close to 0 then model is performing worst separability [40]. ROC Curve of Artificial Neural Network is shown in Figure 32.

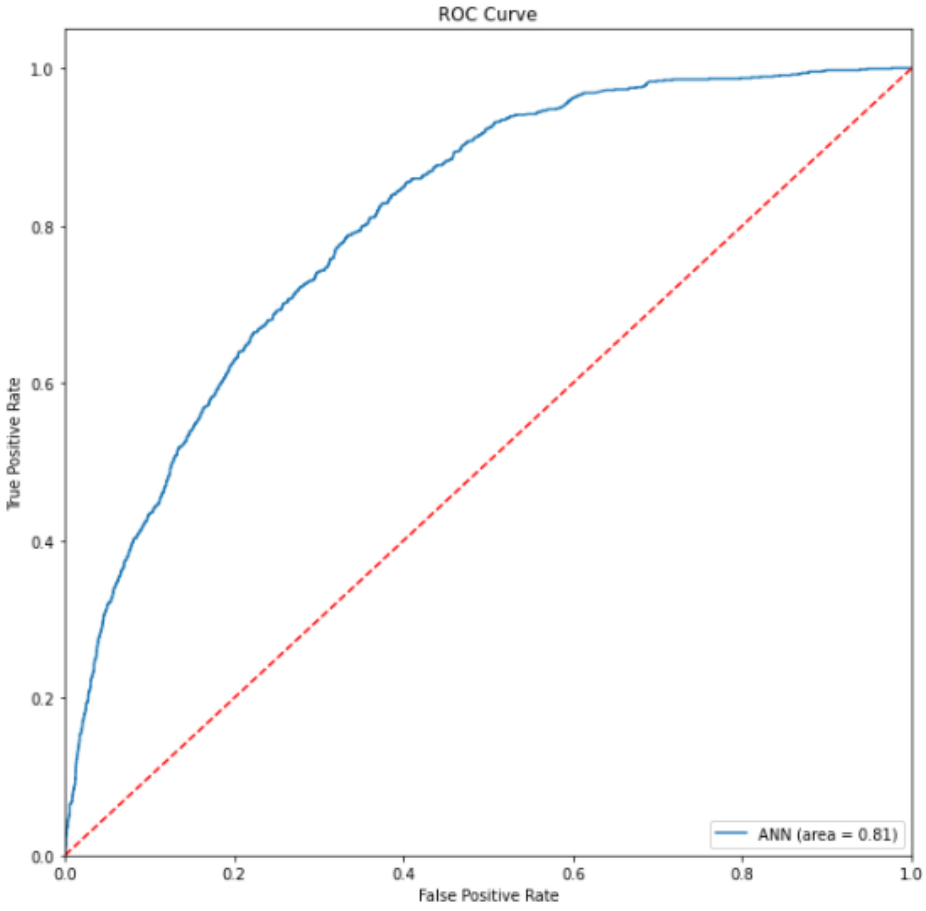

Figure 32 ROC and AUC Curve

7 RESULTS

In this section we summarize obtained results for an Android applications software defect prediction model. We used Artificial Neural Networks, Convolutional Neural Networks and Long Short -Term Memory. We applied cross-project to our model and combined all datasets for train and for test each time we chose different dataset and loop our experiment 14 times. We used deep learning algorithms because previous studies applied mostly machine learning classifiers. For each algorithm we summarized obtained results and presented in this section. Experimental results of Artificial Neural Networks show Table 8, Convolutional Neural Network Table 9, Long Short-Term Memory in Table 9.

Table 8 Artificial Neural Networks Defect Prediction Results

Datasets	**Accuracy**	**Precision**	**Recall**	**F-1 score**	**ROC AUC**	**Average**
aFall	**0.65%**	**0.73%**	**0.69%**	**0.80%**	**0.91%**	**0.76%**
Alfresco	**0.68%**	**0.74%**	**0.70%**	**0.77%**	**0.92%**	**0.76%**
androidSync	**0.70%**	**0.70%**	**0.70%**	**0.83%**	**0.96%**	**0.78%**
androidWalpaper	**0.84%**	**0.71%**	**0.84%**	**0.77%**	**0.94%**	**0.82%**
anySoftKeyboard	**0.75%**	**0.93%**	**0.80%**	**0.79%**	**0.90%**	**0.83%**
Apg	**0.70%**	**0.76%**	**0.83%**	**0.75%**	**0.89%**	**0.79%**
atmosphere	**0.71%**	**0.73%**	**0.72%**	**0.81%**	**0.93%**	**0.78%**
chatSecure	**0.69%**	**0.82%**	**0.68%**	**0.73%**	**0.91%**	**0.77%**
facebook	**0.72%**	**0.74%**	**0.70%**	**0.70%**	**0.84%**	**0.74%**
flutter	**0.70%**	**0.80%**	**0.76%**	**0.72%**	**0.90%**	**0.78%**
kiwis	**0.67%**	**0.71%**	**0.90%**	**0.69%**	**0.93%**	**0.78%**
owncloudandroid	**0.70%**	**0.72%**	**0.80%**	**0.71%**	**0.96%**	**0.78%**

Pageturner	**0.68%**	**0.73%**	**0.76%**	**0.73%**	**0.92%**	**0.76%**
reddit	**0.72%**	**0.70%**	**0.71%**	**0.77%**	**0.90%**	**0.76%**
Average	**0.71%**	**0.75%**	**0.76%**	**0.76%**	**0.92%**	**0.78%**

In Table 8 we presented defect prediction results of Artificial Neural networks. We used android mobile applications imbalanced datasets. We evaluated results with Accuracy metrics such as Precision, Recall, F-1 score, ROC AUC. We used 14 datasets with different size and scope, but ROC AUC metrics results is the best.

Table 9 Convolutional Neural Network Defect Prediction Results

Datasets	**Accuracy**	**Precision**	**Recall**	**F-1 score**	**ROC AUC**	**Average**
aFall	**0.67%**	**0.73%**	**0.70%**	**0.69%**	**0.95%**	**0.75%**
Alfresco	**0.70%**	**0.67%**	**0.80%**	**0.71%**	**0.96%**	**0.77%**
androidSync	**0.64%**	**0.94%**	**0.70%**	**0.65%**	**0.94%**	**0.77%**
androidWalpaper	**0.66%**	**0.82%**	**0.66%**	**0.70%**	**0.96%**	**0.76%**
anySoftKeyboard	**0.72%**	**0.75%**	**0.70%**	**0.83%**	**0.90%**	**0.78%**
Apg	**0.69%**	**0.75%**	**0.77%**	**0.70%**	**0.93%**	**0.77%**
atmosphere	**0.70%**	**0.68%**	**0.90%**	**0.80%**	**0.91%**	**0.80%**
chatSecure	**0.67%**	**0.70%**	**0.75%**	**0.73%**	**0.96%**	**0.76%**
facebook	**0.71%**	**0.75%**	**0.69%**	**0.81%**	**0.90%**	**0.77%**
flutter	**0.68%**	**0.84%**	**0.70%**	**0.72%**	**0.94%**	**0.78%**

kiwis	**0.73%**	**0.76%**	**0.74%**	**0.69%**	**0.92%**	**0.77%**
owncloudandroid	**0.70%**	**0.80%**	**0.72%**	**0.69%**	**0.96%**	**0.77%**
Pageturner	**0.69%**	**0.75%**	**0.82%**	**0.70%**	**0.90%**	**0.77%**
reddit	**0.70%**	**0.73%**	**0.69%**	**0.85%**	**0.93%**	**0.78%**
Average	**0.69%**	**0.76%**	**0.74%**	**0.73%**	**0.93%**	**0.77%**

Convolutional Neural Network results showed in Table 9. We know that Convolutional Neural Network used for images. But we used CNN for binary classification because this algorithm is well known to predict pixels. We used accuracy metrics such as Recall, Precision, F-1 score, ROC AUC. To achieve best results of CNN we tried different variations of filters and activation functions.

Table 10 Long Short-Term Memory Defect Prediction Results

Datasets	**Accuracy**	**Precision**	**Recall**	**F-1 score**	**Roc Auc**	**Average**
aFall	**0.69**	**0.77%**	**0.80%**	**0.69%**	**0.95%**	**0.78%**
Alfresco	**0.70%**	**0.71%**	**0.77%**	**0.82%**	**0.93%**	**0.79%**
androidSync	**0.73%**	**0.83%**	**0.79%**	**0.65%**	**0.86%**	**0.77%**
androidWalpaper	**0.68%**	**0.80%**	**0.93%**	**0.77%**	**0.94%**	**0.82%**
anySoftKeyboard	**0.71%**	**0.73%**	**0.69%**	**0.80%**	**0.91%**	**0.77%**
Apg	**0.72%**	**0.74%**	**0.80%**	**0.77%**	**0.90%**	**0.79%**
atmosphere	**0.69%**	**0.70%**	**0.72%**	**0.71%**	**0.89%**	**0.74%**
chatSecure	**0.73%**	**0.80%**	**0.75%**	**0.72%**	**0.95%**	**0.79%**
facebook	**0.70%**	**0.72%**	**0.74%**	**0.83%**	**0.90%**	**0.78%**

flutter	**0.67%**	**0.70**	**0.76%**	**0.74%**	**0.92%**	**0.76%**
kiwis	**0.70%**	**0.71%**	**0.90%**	**0.73%**	**0.96%**	**0.80%**
owncloudandroid	**0.72%**	**0.83%**	**0.70%**	**0.69%**	**0.90%**	**0.77%**
Pageturner	**0.69%**	**0.75%**	**0.71%**	**0.70%**	**0.93%**	**0.76%**
reddit	**0.70%**	**0.73%**	**0.80%**	**0.77%**	**0.90%**	**0.78%**
Average	**0.70%**	**0.75%**	**0.78%**	**0.74%**	**0.92%**	**0.78%**

In Table 11 presented Long Short-Term Memory results. We know from literature and previous studies that LSTM good for data classification. We used LSTM with 100 and 50 units, reshaped for our datasets. In Table 11 we compared our defect prediction results with dataset and Artificial Neural Networks is the best algorithm.

Table 11 Algorithm's performance on datasets

Projects	**ANN**	**CNN**	**LSTM**
aFall	**0.65%**	**0.67%**	**0.69**
Alfresco	**0.68%**	**0.70%**	**0.70%**
androidSync	**0.70%**	**0.64%**	**0.73%**
androidWalpaper	**0.84%**	**0.66%**	**0.68%**
anySoftKeyboard	**0.75%**	**0.72%**	**0.71%**
Apg	**0.70%**	**0.69%**	**0.72%**
atmosphere	**0.71%**	**0.70%**	**0.69%**
chatSecure	**0.69%**	**0.67%**	**0.73%**

facebook	**0.72%**	**0.71%**	**0.70%**
flutter	**0.70%**	**0.68%**	**0.67%**
kiwis	**0.67%**	**0.73%**	**0.70%**
owncloudandroid	**0.70%**	**0.70%**	**0.72%**
Pageturner	**0.68%**	**0.69%**	**0.69%**
reddit	**0.72%**	**0.70%**	**0.70%**
Average	**0.71%**	**0.69%**	**0.70%**

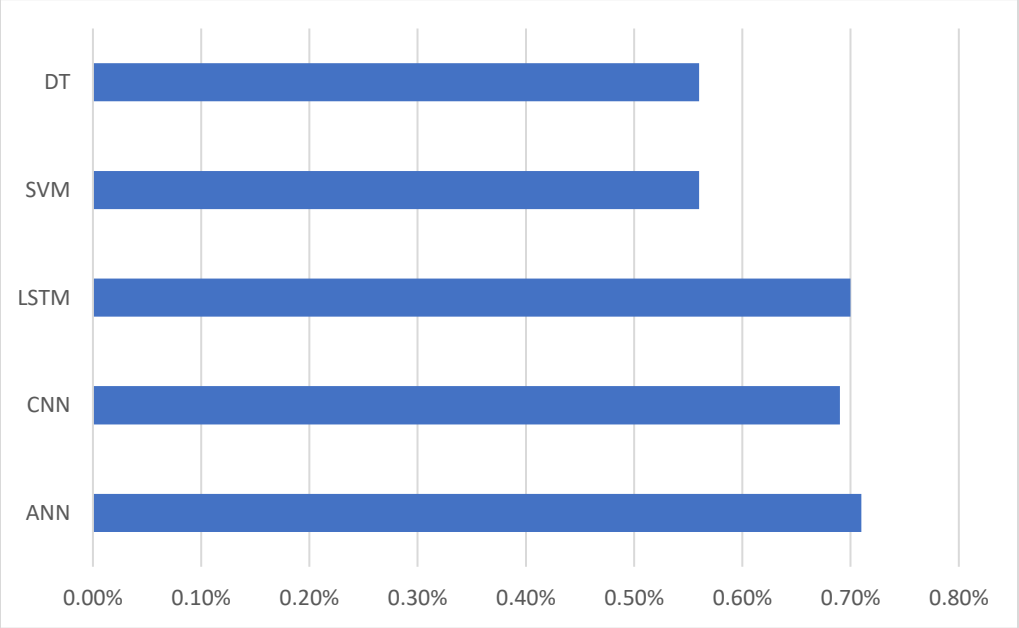

Figure 33 Algorithm's performance on datasets

8 DISCUSSION

In this section we discuss our software defect prediction for android applications model performance. We evaluated three dee learning defect prediction models. We used fourteen Android applications projects from COMMIT GURU platform. We used tenfold cross-validation to find performance of each algorithm. We implemented SMOTE oversampling techniques to balance our datasets. We evaluated results with accuracy metrics such as Recall, Precision, F1-score, ROC and AUC. Artificial Neural Networks, Convolutional Neural Networks and Long Short-Term Memory experimental results are given in section 7. The model performance is different for each dataset. We identified 13 Research Questions for this study

and answered to nine of them in systematic literature review. The goal of this study is to identify defects of mobile applications before releases. The defect prediction accuracy of the model is trained with different classifiers and methods. We evaluated the following research questions:

RQ10. To what extent can we predict the software faults using cross-project for mobile apps using deep learning approaches?
With this research question we analyzed deep learning algorithms performance. The goal is to understand how deep learning algorithms performs for mobile applications.

RQ11. Can we improve the performance of deep learning models using data balancing approaches?
The aim of this research question is to achieve best results with data balancing techniques. For imbalanced mobile application datasets, we used SMOTE Over-sampling technique to balance and improve effectiveness of datasets.

RQ12. Can we improve the performance of deep learning-based fault prediction models using feature selection techniques?
To improve the performance of deep learning models we applied data scaling technique to from input variables to an output variable. We used MinMaxScaler because our datasets are numerical, and we practiced binary classification. MinMaxScaler used for 0,1 and for deep learning input variable expected in an 0,1.

RQ13 To what extent can LSTM-based deep learning architectures predict software faults for mobile apps?
To answer this question we studied previous studies, LSTM is performing well to predict sequences, and we practiced LSTM to predict mobile applications defect prediction.

9 CONCLUSION AND FUTURE WORK

Mobile applications are become a part of our lives. The goal of businesses to propose to the users' applications with zero defects and make things easier. Mobile applications tested manually and there might be unseen defects. Also, nowadays increasing type of mobile phones in the market and mobile applications should be adapted for this. To recognize defects before testing stage, before in literature proposed defect prediction models for software and used machine learning techniques. For mobile applications were limited studies and we proposed a study on software defect prediction model for mobile applications using deep learning methods. We used open-source fourteen Android applications datasets with different size and platforms. We applied Artificial Neural Networks, Convolutional Neural Networks and Long Short-Term Memory. In Figure 33 we compared our results with accuracy ANN 0.71%, CNN 0.69%, LSTM 0.70% with open-source machine learning algorithms such as Support Vector Machines and Decision Tree 56% and our models has a better result. As the best algorithm we identify Artificial Neural Networks for our datasets. For future work we would explore defect prediction for iOS mobile applications using deep learning algorithms.

REFERENCES

[1] Zhao, K., Xu, Z., Yan, M., Xue, L., Li, W., & Catolino, G. (2021). A compositional model for effort-aware Just-In-Time defect prediction on android apps. IET Software.

[2] Kaur, A., Kaur, K., & Kaur, H. (2015, September). An investigation of the accuracy of code and process metrics for defect prediction of mobile applications. In *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions)* (pp. 1-6). IEEE.

[3] Khalid, H., Shihab, E., Nagappan, M., & Hassan, A. E. (2014). What do mobile app users complain about? *IEEE software*, *32*(3), 70-77.

[4] Harman, M., Jia, Y., & Zhang, Y. (2012, June). App store mining and analysis: MSR for app stores. In *2012 9th IEEE working conference on mining software repositories (MSR)* (pp. 108-111). IEEE.

[5] Xia, X., Shihab, E., Kamei, Y., Lo, D., & Wang, X. (2016, September). Predicting crashing releases of mobile applications. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1-10).

[6] Avizienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, *1*(1), 11-33.

[7] Malhotra, R. (2016). An empirical framework for defect prediction using machine learning techniques with Android software. *Applied Soft Computing*, *49*, 1034-1050.

[8] Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2011). A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, *38*(6), 1276-1304.

[9] Akiyama, F. (1971, August). An Example of Software System Debugging. In *IFIP congress (1)* (Vol. 71, pp. 353-359).

[10] Catal, C., & Diri, B. (2009). A systematic review of software fault prediction studies. *Expert systems with applications*, *36*(4), 7346-7354.

[11] Malhotra, R. (2015). A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, *27*, 504-518.

[12] Fahle, S., Prinz, C., & Kuhlenkötter, B. (2020). Systematic review on machine learning (ML) methods for manufacturing processes–Identifying artificial intelligence (AI) methods for field application. *Procedia CIRP*, *93*, 413-418.

[13] Radjenović, D., Heričko, M., Torkar, R., & Živkovič, A. (2013). Software fault prediction metrics: A systematic literature review. *Information and software technology*, *55*(8), 1397-1418.

[14] Hall et al., T. (2011). A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering, 38(6), 1276-1304

[15] Misirli, A. T., & Bener, A. B. (2014, June). A mapping study on bayesian networks for software quality prediction. In *Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering* (pp. 7-11).

[16] Murillo-Morera, J., Quesada-López, C., & Jenkins, M. (2015). Software Fault Prediction: A Systematic Mapping Study. In *CIbSE* (p. 446).

[17] Carvalho, T. P., Soares, F. A., Vita, R., Francisco, R. D. P., Basto, J. P., & Alcalá, S. G. (2019). A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, *137*, 106024.

[18] Özakıncı, R., & Tarhan, A. (2018). Early software defect prediction: A systematic map and review. *Journal of Systems and Software*, *144*, 216-239.

[19] Son, L. H., Pritam, N., Khari, M., Kumar, R., Phuong, P. T. M., & Thong, P. H. (2019). Empirical study of software defect prediction: a systematic mapping. *Symmetry*, *11*(2), 212.

[20] Najm, A., Zakrani, A., & Marzak, A. (2019, July). Decision tree based software development effort estimation: A systematic mapping study. In *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)* (pp. 1-6). IEEE.

[21] Alsolai, H., & Roper, M. (2020). A systematic literature review of machine learning techniques for software maintainability prediction. *Information and Software Technology*, *119*, 106214.

[22] Auch, M., Weber, M., Mandl, P., & Wolff, C. (2020). Similarity-based analyses on software applications: A systematic literature review. *Journal of Systems and Software*, *168*, 110669.

[23] Degu, A. (2019). 'Android application memory and energy performance: Systematic literature review. *IOSR J. Comput. Eng*, *21*(3), 20-32.

[24] Kaur, A., & Kaur, K. (2018). Systematic literature review of mobile application development and testing effort estimation. *Journal of King Saud University-Computer and Information Sciences*.

[25] Del Carpio, A. F., & Angarita, L. B. (2020, August). Trends in Software Engineering Processes using Deep Learning: A Systematic Literature Review. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 445-454). IEEE.

[26] Kaur, A. (2020). A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes. *Archives of Computational Methods in Engineering*, *27*(4), 1267-1296.

[27] Kaya, A., Keçeli, A. S., Catal, C., & Tekinerdogan, B. (2020). Model analytics for defect prediction based on design-level metrics and sampling techniques. In *Model Management and Analytics for Large Scale Systems* (pp. 125-139). Academic Press.

[28] Kaur, A., Kaur, K., & Kaur, H. (2016). Application of machine learning on process metrics for defect prediction in mobile application. In *Information Systems Design and Intelligent Applications* (pp. 81-98). Springer, New Delhi.

[29] Zhao, K., Xu, Z., Yan, M., Tang, Y., Fan, M., & Catolino, G. (2021, March). Just-in-time defect prediction for Android apps via imbalanced deep learning model. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing* (pp. 1447-1454).

[30] Sewak, M., Sahay, S. K., & Rathore, H. (2020, November). Assessment of the Relative Importance of different hyper-parameters of LSTM for an IDS. In *2020 IEEE REGION 10 CONFERENCE (TENCON)* (pp. 414-419). IEEE.

[31] Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, *51*(1), 7-15.

[32] Catal, C., Sevim, U., & Diri, B. (2010). Metrics-driven software quality prediction without prior fault data. In Electronic Engineering and Computing Technology (pp. 189-199). Springer, Dordrecht.

[33] Catal, C. (2014). A comparison of semi-supervised classification approaches for software defect prediction. *Journal of Intelligent Systems*, *23*(1), 75-82.

[34] Alan, O., & Catal, C. (2011). Thresholds based outlier detection approach for mining class outliers: An empirical case study on software measurement datasets. *Expert Systems with Applications*, *38*(4), 3440-3445.

[35] Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, *51*(1), 7-15.

[36] Rosen, C., Grawi, B., & Shihab, E. (2015, August). Commit guru: analytics and risk prediction of software commits. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering* (pp. 966-969).

[37] Catolino, G., Di Nucci, D., & Ferrucci, F. (2019, May). Cross-project just-in-time bug prediction for mobile apps: an empirical assessment. In *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)* (pp. 99-110). IEEE.

[38] Fernández, A., Garcia, S., Herrera, F., & Chawla, N. V. (2018). SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research*, *61*, 863-905.

[39] Brownlee, J. (2016). *Machine learning algorithms from scratch with Python*. Machine Learning Mastery.

[40] Flach, P. A. (2003). The geometry of ROC space: understanding machine learning metrics through ROC isometrics. In *Proceedings of the 20th international conference on machine learning (ICML-03)* (pp. 194-201).

[41] Gezici, B., Tarhan, A., & Chouseinoglou, O. (2019). Internal and external quality in the evolution of mobile software: An exploratory study in open-source market. *Information and Software Technology*, *112*, 178-200.

[42] Malhotra, R., & Khurana, A. (2017, September). Analysis of evolutionary algorithms to improve software defect prediction. In *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)* (pp. 301-305). IEEE.

[43] Malhotra, R., & Sharma, A. (2013, August). A neuro-fuzzy classifier for website quality prediction. In *2013 International Conference on Advances in Computing, Communications, and Informatics (ICACCI)* (pp. 1274-1279). IEEE.

[44] Mohamed, A., Ruan, H., Abdelwahab, M. H. H., Dorneanu, B., Xiao, P., Arellano-Garcia, H., ... & Tafazolli, R. (2020, June). An Inter-Disciplinary Modelling Approach in Industrial 5G/6G and Machine Learning Era. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)* (pp. 1-6). IEEE.

[45] Kaur, A., Kaur, K., & Kaur, H. (2015, September). An investigation of the accuracy of code and process metrics for defect prediction of mobile applications. In *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)* (pp. 1-6). IEEE.

[46] Catal, C., Akbulut, A., Ekenoglu, E., & Alemdaroglu, M. (2017, May). Development of a software vulnerability prediction web service based on artificial neural networks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 59-67). Springer, Cham.

[47] Xia, X., Shihab, E., Kamei, Y., Lo, D., & Wang, X. (2016, September). Predicting crashing releases of mobile applications. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1-10).

[48] Arévalo, F., Oestanto, E., & Schwung, A. (2018, July). Development of a Mobile App for Fault Detection Assessment based on Information Fusion. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)* (pp. 635-640). IEEE.

[49] Biçer, M. S., & Diri, B. (2015, October). Predicting defect-prone modules in web applications. In *International Conference on Information and Software Technologies* (pp. 577-591). Springer, Cham.

[50] Malhotra, R., & Sharma, A. (2019). An empirical assessment of feature selection techniques in defect prediction models using web applications. *Journal of Intelligent & Fuzzy Systems*, *36*(6), 6567-6578.

[51] Habibi, P. A., Amrizal, V., & Bahaweres, R. B. (2018, May). Cross-project defect prediction for web application using naive Bayes (case study: Petstore web application). In *2018 International Workshop on Big Data and Information Security (IWBIS)* (pp. 13-18). IEEE.

[52] Öztürk, M. M., Cavusoglu, U., & Zengin, A. (2015). A novel defect prediction method for web pages using k-means++. *Expert Systems with Applications*, *42*(19), 6496-6506.

[53] Ramakrishnan, R., & Kaur, A. (2020). An empirical comparison of predictive models for web page performance. *Information and Software Technology*, *123*, 106307.

[54] Biçer, M. S., & Diri, B. (2016). Defect prediction for cascading style sheets. *Applied Soft Computing*, *49*, 1078-1084.

[55] Pang, Y., Xue, X., & Wang, H. (2017, June). Predicting vulnerable software components through the deep neural network. In *Proceedings of the 2017 International Conference on Deep Learning Technologies* (pp. 6-10).

[56] Malhotra, R., & Sharma, A. (2018). Analyzing Machine Learning Techniques for Fault Prediction Using Web Applications. *JIPS*, *14*(3), 751-770.

[57] Ouni, A., Daagi, M., Kessentini, M., Bouktif, S., & Gammoudi, M. M. (2017, June). A machine learning-based approach to detect web service design defects. In *2017 IEEE International Conference on Web Services (ICWS)* (pp. 532-539). IEEE.

[58] Cui, J., Wang, L., Zhao, X., & Zhang, H. (2020). Towards the predictive analysis of android vulnerability using statistical codes and machine learning for IoT applications. *Computer Communications*, *155*, 125-131.

[59] Dong, F., Wang, J., Li, Q., Xu, G., & Zhang, S. (2018). Defect prediction in android binary executables using a deep neural network. *Wireless Personal Communications*, *102*(3), 2261-2285.

[60] Padhy, N., Satapathy, S. C., Mohanty, J. R., & Panigrahi, R. (2018). Software reusability metrics prediction by using evolutionary algorithms: The interactive mobile learning application RozGaar. *International Journal of Knowledge-based and Intelligent Engineering Systems*, *22*(4), 261-276.

[61] Dam, H. K., Tran, T., Pham, T., Ng, S. W., Grundy, J., & Ghose, A. (2017). Automatic feature learning for vulnerability prediction. *arXiv preprint arXiv:1708.02368*.

[62] Scandariato, R., Walden, J., Hovsepyan, A., & Joosen, W. (2014). Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering*, *40*(10), 993-1006.

[63] Dehkordi, M. R., Seifzadeh, H., Beydoun, G., & Nadimi-Shahraki, M. H. (2020). Success prediction of android applications in a novel repository using neural networks. *Complex & Intelligent Systems*, *6*, 573-590.

[64] Jogarah, K. K., Soopaul, K., Beeharry, Y., & Hurbungs, V. (2018). Hybrid machine learning algorithms for fault detection in android smartphones. *Transactions on Emerging Telecommunications Technologies*, *29*(2), e3272.

[65] Dey, T., & Mockus, A. (2018, October). Modeling relationship between post-release faults and usage in mobile software. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering* (pp. 56-65).

[66] Bhandari, G. P., Gupta, R., & Upadhyay, S. K. (2019). An approach for fault prediction in SOA-based systems using machine learning techniques. *Data Technologies and Applications*.

[67] Dam, H. K., Tran, T., Pham, T. T. M., Ng, S. W., Grundy, J., & Ghose, A. (2018). Automatic feature learning for predicting vulnerable software components. *IEEE Transactions on Software Engineering*.

[68] Marcek, S., & Drozda, M. (2016). Predicting system failures on mobile devices. In *Proceedings of the Mediterranean Conference on Information & Communication Technologies 2015* (pp. 499-508). Springer, Cham.

[69] Liu, H., Shen, M., Jin, J., & Jiang, Y. (2020, July). Automated classification of actions in bug reports of mobile apps. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 128-140).

[70] Syer, M. D., Nagappan, M., Adams, B., & Hassan, A. E. (2015). Studying the relationship between source code quality and mobile platform dependence. *Software Quality Journal*, *23*(3), 485-508.

[71] Tosun, A., Bener, A., & Kale, R. (2010, July). Ai-based software defect predictors: Applications and benefits in a case study. In *Twenty-Second IAAI Conference*.

[72] Abunadi, I., & Alenezi, M. (2015, September). Towards cross-project vulnerability prediction in open-source web applications. In *Proceedings of The International Conference on Engineering & MIS 2015* (pp. 1-5).

[73] Linares-Vásquez, M., McMillan, C., Poshyvanyk, D., & Grechanik, M. (2014). On using machine learning to classify software applications into domain categories automatically. *Empirical Software Engineering*, *19*(3), 582-618.

[74] Liu, Y., Ma, C., Dong, Z., Zhang, T., Cheng, J., & Zhang, J. (2020, April). Research on Defect Priority Classification of Crowdsourcing Testing for Mobile Applications. In *Journal of Physics: Conference Series* (Vol. 1518, No. 1, p. 012008). IOP Publishing.

[75] Moran, K., Linares-Vásquez, M., Bernal-Cárdenas, C., & Poshyvanyk, D. (2015, August). Auto-completing bug reports for android applications. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (pp. 673-686).

[76] Gómez, M., Rouvoy, R., Adams, B., & Seinturier, L. (2016, May). Mining test repositories for automatic detection of UI performance regressions in Android apps. In *Proceedings of the 13th International Conference on Mining Software Repositories* (pp. 13-24).

[77] Shar, L. K., Briand, L. C., & Tan, H. B. K. (2014). Web application vulnerability prediction using hybrid program analysis and machine learning. *IEEE Transactions on dependable and secure computing*, *12*(6), 688-707.

[78] Shar, L. K., & Tan, H. B. K. (2012, September). Predicting common web application vulnerabilities from input validation and sanitization code patterns. In *2012 Proceedings of the 27th IEEE/ACM international conference on automated software engineering* (pp. 310-313). IEEE.

[79] Padhy, N., Singh, R. P., & Satapathy, S. C. (2019). Cost-effective and fault-resilient reusability prediction model by using adaptive genetic algorithm-based neural network for web-of-service applications. *Cluster Computing*, *22*(6), 14559-14581.

[80] Kumar, A., Chugh, R., Girdhar, R., & Aggarwal, S. (2017, March). Classification of faults in Web applications using machine learning. In *Proceedings of the 2017 International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence* (pp. 62-67).

[81] Kang, D., & Bae, D. H. (2010, July). Software fault prediction models for web applications. In *2010 IEEE 34th Annual Computer Software and Applications Conference Workshops* (pp. 51-56). IEEE.

[82] Catolino, G. (2017, May). Just-in-time bug prediction in mobile applications: the domain matters! In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)* (pp. 201-202). IEEE.

[83] Catolino, G., Di Nucci, D., & Ferrucci, F. (2019, May). Cross-project just-in-time bug prediction for mobile apps: An empirical assessment. In *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)* (pp. 99-110). IEEE.

[84] Ricky, M. Y., Purnomo, F., & Yulianto, B. (2016, March). Mobile application software defect prediction. In *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)* (pp. 307-313). IEEE.

[85] Tsakiltsidis, S., Miranskyy, A., & Mazzawi, E. (2016, October). On automatic detection of performance bugs. In *2016 IEEE international symposium on software reliability engineering workshops (ISSREW)* (pp. 132-139). IEEE.

[86] Fan, Y., Cao, X., Xu, J., Xu, S., & Yang, H. (2018, July). High-Frequency Keywords to Predict Defects for Android Applications. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* (Vol. 2, pp. 442-447). IEEE.