Intelligent System to Detect Software Defects in Autonomous Cars

Sudeep Tanwar[1], Smit N. Patel[2], Jil R. Patel[3], Nisarg P. Patel[1], Rajesh Gupta[1], and Sherali Zeadally[4]

[1]Nirma University
[2]Government Engineering College Gandhinagar
[3]Hasmukh Goswami College of Engineering
[4]University of Kentucky

October 31, 2022

Abstract

Autonomous cars have become increasingly popular in the last decade because of their numerous benefits, such as lower travel time, increased safety, and improved fuel economy. Many car manufacturing companies and tech giants are working on this technology to make fully autonomous automobiles or strengthen their existing driver-less cars. These cars use very complex, advanced, and sophisticated hardware technologies. However, the software is an equally important feature because it must operate all functions seamlessly while working in sync with other vehicle components. The software must analyze a large amount of data to make quick real-time decisions, so any vulnerabilities or bugs can be a severe problem to the vehicle and the passengers riding in it. Many researchers have proposed various software defect prediction schemes for different projects and applications, but most of them have focused on specific software issues and excluded others. Thus, their methods cannot be applied to the software of autonomous cars. In this paper, we propose an improved Artificial Neural Network (ANN) model, called Dropout-Artificial Neural Network (D-ANN), to solve this problem of defect prediction in autonomous cars. This inclusive model can consider all the parameters simultaneously for effective bugs prediction. The proposed model can be used for the software of any autonomous cars, and it is trained and evaluated using standard methods. The results obtained show that the proposed model predicts software defects with higher accuracy than other models.

Intelligent System to Detect Software Defects in Autonomous Cars

Sudeep Tanwar, *Senior Member, IEEE*, Smit N. Patel, Jil R. Patel, Nisarg P. Patel, Rajesh Gupta, *Student Member, IEEE,* Sherali Zeadally, *Senior Member, IEEE*

Abstract

Autonomous cars have become increasingly popular in the last decade because of their numerous benefits, such as lower travel time, increased safety, and improved fuel economy. Many car manufacturing companies and tech giants are working on this technology to make fully autonomous automobiles or strengthen their existing driver-less cars. These cars use very complex, advanced, and sophisticated hardware technologies. However, the software is an equally important feature because it must operate all functions seamlessly while working in sync with other vehicle components. The software must analyze a large amount of data to make quick real-time decisions, so any vulnerabilities or bugs can be a severe problem to the vehicle and the passengers riding in it. Many researchers have proposed various software defect prediction schemes for different projects and applications, but most of them have focused on specific software issues and excluded others. Thus, their methods cannot be applied to the software of autonomous cars. In this paper, we propose an improved Artificial Neural Network (ANN) model, called Dropout-Artificial Neural Network (D-ANN), to solve this problem of defect prediction in autonomous cars. This inclusive model can consider all the parameters simultaneously for effective bugs prediction. The proposed model can be used for the software of any autonomous cars, and it is trained and evaluated using standard methods. The results obtained show that the proposed model predicts software defects with higher accuracy than other models.

Index Terms

Artificial Neural Network, Autonomous cars, Bug prediction, Driver-less cars, Modified ANN, Self-driving vehicles, Software defects.

I. Introduction

An autonomous car can be defined as a vehicle that is capable of driving itself to its intended destination without the human interaction [1]. It can sense the real-time traffic, pedestrians' movements, and signboards using different technologies, such as sensors, actuators, and complex algorithms, making an autonomous car reliable, mature, and intelligent. Many researchers from industries, academia, and start-ups are working extensively to make the transport system smart and driverless. For example, Tesla, Uber, Volvo, and Google have advanced and popular autonomous cars, such as Waymo, Tesla model-3, and Volvo XC90 [2]. More than 80 companies, including General Motors, Ford, Mercedes Benz, Volkswagen, Audi, Nissan, Toyota, and BMW, are testing their partially or fully autonomous cars for reliability, efficiency, and accuracy. Around 1400 autonomous cars are running on the roads of the United States, with California being the first state to allow autonomous cars. It is projected that this number will increase to 3 million by 2040 [3]. The total market capitalization of the autonomous car industry is around 27.6 billion dollars in 2021 and is estimated to reach 54 billion dollars in the next five years. These recent trends demonstrate that autonomous cars hold a promising future.

There are several issues associated with autonomous cars because of their complex designs and computations, such as hardware fault tolerance, cooperating with human driven vehicles, and so on. [4] [5]. The components of autonomous cars, such as the Global Positioning System (GPS), Light Detection and Ranging (Lidar), and cameras and many more components, are connected to a centralized system such as the cloud that works in harmony with each other [6]. The centralized system consists of software (a set of instructions), a driving force for autonomous cars that manages operations, hardware interactions, and information processing. A threat to the software components can cause a threat to the entire autonomous car system, which is quite severe and can put public property and many lives in danger. Although the software is designed very precisely and tested many times before publishing its final version, many software defects can occur in software, which can cause severe technical issues if they are not identified early. For example, in 2016, Tesla's autopilot software failed to recognize a white tractor-trailer, which was crossing the highway, because of miscommunication with sensors, and the software ran the car into the trailer, which resulted in the death of the driver [3]. Another accident occurred in Florida in 2018, where the autonomous cars software lost control over the battery and the autonomous car eventually caught fire, which led to the death

S. Tanwar, N. P. Patel, and R. Gupta are with the Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad, Gujarat, India 382481 e-mails: (18bce136@nirmauni.ac.in, 18FTVPHDE31@nirmauni.ac.in, sudeep.tanwar@nirmauni.ac.in).

S. N. Patel is with the Department of Information and Technology, Government Engineering College, Gandhinagar, Gujarat, India (email: smitpatel7815@gecg28.ac.in)

J. R. Patel is with the Department of Information and Technology, Hasmukh Goswami College of Engineering, Vehlal, Gujarat, India (email: jilpatel2812@gmail.com)

S. Zeadally is with the College of Communication and Information, University of Kentucky, Lexington, KY 40506 USA (email: szeadally@uky.edu).

.

of passengers and one injury [7]. Researchers are aware of the high importance of autonomous car software, so they explored various vulnerabilities, faults, and defects that can occur in autonomous cars software. In [8], the authors described some common defects that can arise in software and their possible solutions.

- *Performance defects*: These defects are related to parameters such as speed, responsiveness, and stability. They generally slow down the processing speed and increase the response time. These scenarios can be fatal for autonomous car software and can be resolved by conducting various quantitative tests such as robustness and performance reviews under a specific workload.
- *Functional defects*: These defects are related to the behavior of the software which does not function appropriately for given inputs or may generate wrong outputs. They can be identified by testing software on multiple test cases in real-world scenarios.
- *Security defects*: These are the vulnerabilities in a software system that an unauthorized person can exploit to get control of the entire system. Some of the attacks include Structure Query Language (SQL) injections and cross-site scripting. Various resilient solutions and Blockchain-based security schemes [9] can be used to mitigate potential threats.
- *Compatibility defects*: These defects are related to the consistent performance of software and occur with specific types of systems, hardware components, and older versions of particular equipment. It can be identified by testing the software on various platforms under a controlled environment [10].
- *Logical defects*: These are the defects, which generally occur due to poorly written and less optimized code. It can cause hanging problems, unnatural halts or crashes, and infinite loops. They are relatively easy to counter and can be fixed by software updates.

In a nutshell, these software defects can be addressed based on their priority and severity of the autonomous car system.

Researchers worldwide presented their works on the prioritization of different bugs based on their impact and timing. For example, Tahir *et al.* [11] proposed an automated model to prioritize the bugs in software components and assign bug reports to the specialized developer. It is based on previous knowledge and past experiences. They preferred the LSTM model to prioritize defect reports and then used a content-based filtering technique for assignment purposes. This model has been pre-processed with the help of the Natural Language Processing (NLP) method and uses Senti4SD [11] for classifying bug reports based on different emotions.

Sometimes the software design can also cause blob, data class, lazy class, feature envy defect, and anti-patterns. Madden *et al.* [12] proposed an Iterative Dichotomiser three decision tree model to detect software design defects. They have considered Unified Modeling Language (UML) diagrams instead of text codes and assigned one decision tree for each design defect. This model can detect many designing defects, as mentioned above. They have applied their proposed model on five different software projects and evaluated their performance using a performance metric such as recall.

All these approaches would be very effective for software defect prediction. However, they focus on a few parameters at a particular instance, appropriate for small or simple software. In contrast, autonomous cars' software is complex and has to deal with many real-time attributes for their proper functioning. Consequently, the existing models are unreliable for such complex software that detects only certain software defects and malfunctioning (faulty design or the flaws in updates). Omitting certain features and only focusing on few parameters can lead to disastrous results in the case of autonomous cars. Therefore, autonomous cars needs a practical, reliable, and effective software defect prediction system that can focus on all real-time data points, attributes, and parameters.

Motivated from the aforementioned discussions, this paper proposes a model that can effectively predict autonomous car software defects. It is based on DL algorithms and can use all the features mentioned in a dataset. It is an enhanced ANN model with dropout layers, called D-ANN. We discussed possible software defects and their solutions and we present a comprehensive study of existing approaches and compared it with our proposed model. We compared the performance of the proposed model with different classical ML models, such as decision tree, Gaussian Naive Bayes, SVM, and random forest. The proposed model outperforms all ML models, and we used binary graphs to evaluate the performance of various models. The proposed model can be directly applicable to autonomous cars because it considers all aspects of software and effectively detects faults or bugs.

A. Research contributions of this work

We summarize the research contributions of this work as follows:

The primary contribution of this research article is to identify software defects in the early development stages of an autonomous car and not normal car (as they are not entirely configured with software). Currently, automobile manufacturing companies and research institutions are highly focused on two main challenges when it comes to autonomous cars. The first one is to increase and optimize the car's battery capacity, and the second is to improve the car's communication with other vehicles on the road. There have been very few attempts made to address the problem of software defects for the autonomous car which have resulted in fatal crashes and loss of lives.

The ultimate goal of an autonomous car is to make every ride in it safe and reliable. The results of this paper will be

useful to many communities and sectors. For instance, the research community can use this as a foundation and expand its functionalities by adding new features, such as an enhanced software debugging system.

The main scientific contribution of the paper is that we have proposed a DL-based D-ANN model that is used to process the feature space of a standard software defect dataset of an autonomous car. We described the model in this paper with more than 20 mathematical equations and their detailed explanations. Thus, this model can be used not only for software defects but also for any prediction problem in the domain of DL. Moreover, the proposed D-ANN model can be enhanced according to the problem by changing the number of neurons, changing the order of the different dense and dropout layers, and integrating more advanced neural network-based algorithms and activation functions.

The proposed D-ANN model is also compatible with industry applications with little modifications. For instance, any autonomous car manufacturer can integrate our D-ANN model (as a pre-trained model) in their cars by modifying certain parameters according to the system and functionalities of the car.

Some other research contributions of this work include:

- We present a comprehensive review of existing software defect forecasting techniques, covering different types of defects, such as design, functional, and logical defects, along with their results, merits, and shortcomings. Although there is very limited material available publicly on this relatively new topic, several research articles have been published from 2004 to 2022 as we discuss in the related research work section and Table I.
- We propose a novel DL approach, i.e., D-ANN model to predict defects in autonomous cars. It predicts the possible present software defects based on different 21 software features for the autonomous car. Further, it forecasts the possible current software defects based on 21 software features of self-driving car software. Here, we choose the D-ANN model because of its efficiency in predicting the software defects in autonomous cars. We have also evaluated ML models such as KNN, Decision Tree, and SVM, and then we considered the auto ML model and simple ANN model. However, out of all these models, we selected D-ANN because it achieved the highest accuracy metrics.
- We have conducted a performance evaluation of the proposed D-ANN model and compared it with the classical ML and the auto-ML models using performance metrics, such as precision, recall, and F1-score performance metrics.

B. Organization

We organize the rest of the paper as follows. In Section II, we survey existing methodologies that predict possible software defects present in autonomous cars. Section III describes the system architecture and formulates the problem. Section IV describes our proposed solution. In Section V, we evaluate the performance of our proposed model. Finally, Section VI concludes the paper.

II. Related Research Work

Over the last few years, researchers have been exploring various approaches that can predict software defects accurately using different AI models, such as Artificial Neural Networks (ANN) and Long Short-Term Memory (LSTM). For example, Anam *et al.* [19] surveyed Fault Detection and Diagnosis (FDD) techniques, such as signal processing-based FDD and artificial intelligence-based FDD. Zhang *et al.* [20] proposed a deep learning-based fault diagnosis technique and reviewed all the traditional fault diagnosis methods such as model-based, signal-based, and knowledge-based. WonHaeng *et al.* [21] propose a C-ITS security structure for autonomous vehicle security in a 5G environment, and also analyzed the latest security technologies for autonomous cars. Prabha *et al.* [16] used several information-mining techniques with classical Machine Learning (ML) models such as Naive Bayes, random forest, Support Vector Machine (SVM), and ANN to predict various types of bugs, defects, and malfunctions in a software system. They have proposed a novel Principal Component Analysis (PCA) scheme for feature reduction. They combined the current PCA method with the ANN algorithm to get better results and achieve 98.70% prediction accuracy. Iago *et al.* [22] developed a system using the Dynamic Bayesian Network use for health monitoring and fault detection, diagnosis, and prognosis in autonomous vehicles. The proposed Dynamic Bayesian Network gives good performance results with different ML metrics.

Hammouri *et al.* [14] used supervised ML algorithms such as Naive Bayes, decision tree, and ANN to forecast the faults in software using historical data. They evaluated the performance of their model and compared it with other prediction models such as the power model and Linear Autoregressive model. Deep Learning (DL) algorithms can be used to identify bugs in software. Later, Wang *et al.* [18] proposed a software defect prediction model using Gated Hierarchical LSTMs (GH-LSTM). It combines semantic features, which are extracted using Abstract Syntax Trees, and traditional features, which are described in the PROMISE repository [23]. They applied their model on ten different software projects and evaluated their accuracy using three different techniques using metrics for non-effort-aware evaluation, win/tie/loss indicator, and effort-aware assessment.

Some defects can appear while updating the older versions of software. Sikic *et al.* [17] used aggregated change metrics to store changes and updates between two versions, the latest version and older versions, of software in a chronological order. This model is helpful in the software development cycle and can detect any bugs during the updating process. It can be applied to Java programs, and the authors have evaluated its accuracy using Area Under the Curve (AUC), Matthews Correlation Coefficient (MCC), and F1-Score.

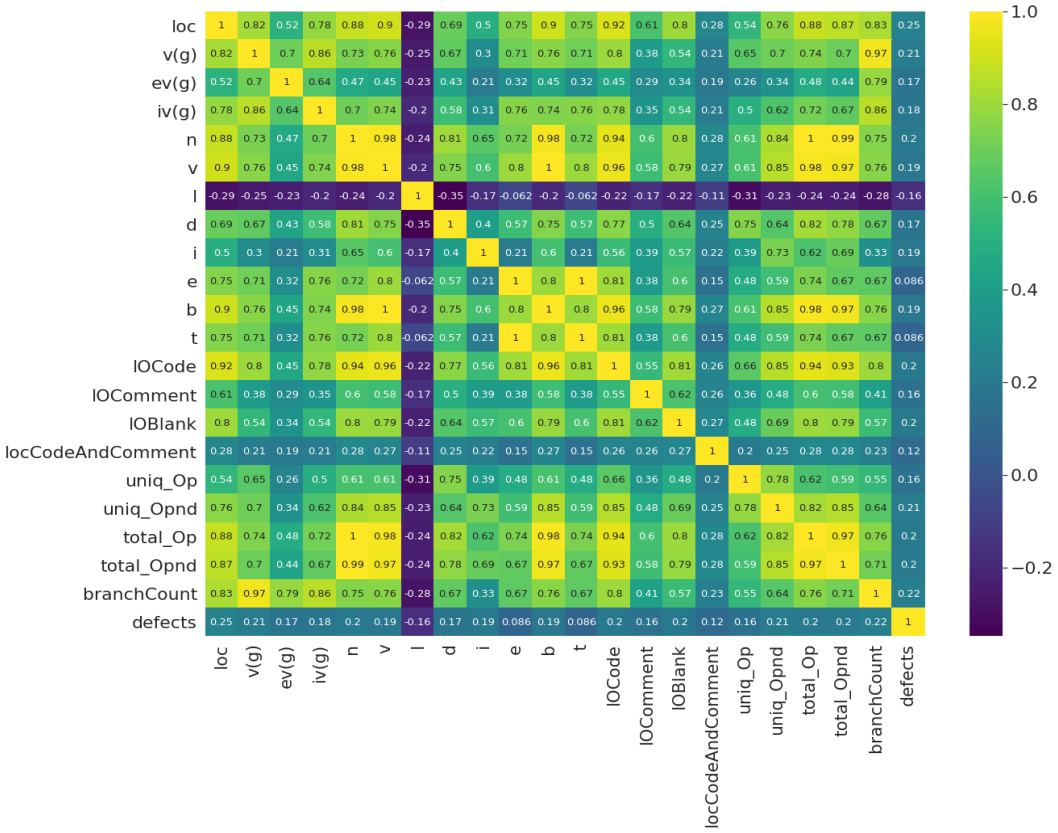

Fig. 1: Heatmap of the 21 features of the software in an autonomous car.

Table I presents a comparative analysis of existing software defect prediction approaches along with the proposed scheme. From Table I, we note that the DL-based model performs better than the classical ML-based models. Another finding from the literature survey is that no previous works have considered all the software features while training their proposed model.

III. System Model and Problem Formulation

This section describes the dataset used, system model and problem formulation for the proposed system that considers all features in predicting the software defect in autonomous cars to improve prediction accuracy.

A. Dataset Description

The work to detect software defects for an autonomous vehicle is based on the Kaggle dataset [24]. The dataset consists of 21 features such as a line count of code, cyclomatic complexity, essential complexity, design complexity, total operators and operands, volume, program length, difficulty, intelligence, time estimator, total operand, total operator, unique operand, unique operator, line count of blank lines, line count of comments, and flow graph of essential features. The target attributes consist of binary values True and False, which have an approximate ratio of 80:20 in 10885 samples out of which 5 samples were removed. These samples have some missing values. The NASA Metrics Data Program made this data publicly for further work.

B. System Model

The dataset is taken from Kaggle [24] to identify the defects present in the autonomous cars' software components as we have discussed earlier. Next, the normalization technique (i.e., standard-scalar normalization) has been applied to the dataset to normalize the data to zero using the equation below.

$$nf = \frac{1}{\sigma}(f - \mu) \tag{1}$$

where, f is a feature, μ is the mean value of that feature for the entire dataset, σ is the standard deviation of that feature, and nf is the normalized feature value. This normalization is applied to all the twenty-one features present in the dataset.

Fig. 1 shows the heat map of all twenty-one features. The normalized data is divided into testing (25%) and training data

TABLE I: Comparative analysis of different existing techniques for software defect prediction.

Ref.	Year	Contribution	Results obtained	Strengths	Weaknesses
[13]	2014	Proposed a DL-based architecture, which uses autoencoder as a building block to forecast the information about traffic flow.	MAE, MRE, and RMSE.	Latent traffic flow feature representation is discovered using the DL architecture model.	In the prediction layer, only logistic regression is used.
[14]	2018	Software bug prediction using different ML algorithms such as ANN, Naive Bayes,and decision tree.	RSME for decision tree is approximately 0.0826.	Multiple datasets are used to achieve better accuracy.	Limited software attributes are considered.
[15]	2019	Developed a centralized root server to reduce traffic congestion in metropolitan cities.	Automatic Balancing of Traffic through the Integration of Smartphones with vehicles (ABATIS route server).	Reduces traffic congestion as well as traffic fluidity.	Data from only one city is used for the experiment.
[16]	2020	Software defect prediction using a hybrid approach of different ML algorithms.	AUC = 98.70 for neural network.	Multiple parameters are used for a better result, such as precision, recall, recognition accuracy.	The failure rate parameter is not considered to determine accurate outcomes.
[17]	2021	Aggregated Change Metrics are used to predict software bugs.	AUC, MCC and F1 score is improved.	Changes made between the software's two versions are considered in chronological order.	Different programming languages are not explored, such as Java.
[11]	2021	Bug prediction and assignment using LSTM and Content-based filtering, respectively.	F1 score is increased by 21% compared to the state-of-the-art technique.	The proposed model is compared with the state-of-the-art bug prioritization techniques to determine the performance.	Other DL approaches such as ANN are not used.
[18]	2021	Software defect prediction using the GH-LSTM approach.	F-measure	Semantic features are used in this proposed model.	Programming languages such as C and C++ are not explored.
[12]	2021	Detection of defects in design using decision tree approach.	F1 score	Multiple defects along with 15 object-oriented features are used.	The correct approach is not included.
Proposed model	2021	*Software defect prediction in autonomous cars using DL-based D-ANN model, which can focus on multiple different software parameters at the same time.*	*Precision, Recall, F1-Score*	*The main advantage of the D-ANN model is it can predict Software defects in autonomous cars better than Classical ML models, such as naive bayes, random forest, and SVM, and the auto ML model.*	

(75%), where the testing data validates the performance of training data. Next, the D-ANN takes this normalized train input data twice, i.e., one for the first ANN and the second for another ANN with False or True target binary values, which shows whether the defect is present or not. Further, the label encoder is used to process the target column of the dataset, i.e., provide appropriate labels to each values of the target column. It convert the values (True/False) into absolute values of 0's and 1's. The objective of this work is to improve the accuracy of the model for better prediction results.

C. Problem Formulation

The proposed model is a hybrid model of two ANN with additional dropout layers. Both ANN will be fed with the same input values separately. After the pre-processing steps, trained data is passed to both the ANNs with the same target values that were previously encoded.

$$f = [f_1, f_2, f_3, ..., f_{21}] \tag{2}$$

$$nf = [nf_1, nf_2, nf_3, ..., nf_{21}] \tag{3}$$

$$final\_ip = [nf, nf] \tag{4}$$

$$final\_op = [p_0, p_1] \tag{5}$$

where, f is a feature list that contains all twenty-one features of the software of autonomous cars. nf is a normalized feature. $final\_ip$ consists of nf twice. $final\_op$ is the list that has the values of probability of not having a defect (p_0) and having a defect (p_1) in the software. In this context, the objective function is accuracy. The main goal is to maximize the accuracy and improves the model's performance to better predict the defects with the input's value of Eq. (4) and output of the category Eq. (5). The D-ANN predicts the probability of the autonomous car's software being faulty or not at the output stage. Next, the probability value of each class is compared with the baseline using the function named argmax, which is provided in the library NumPy and the highest category will be the final prediction.

$$final\_prediction = max(p_0, p_1) \tag{6}$$

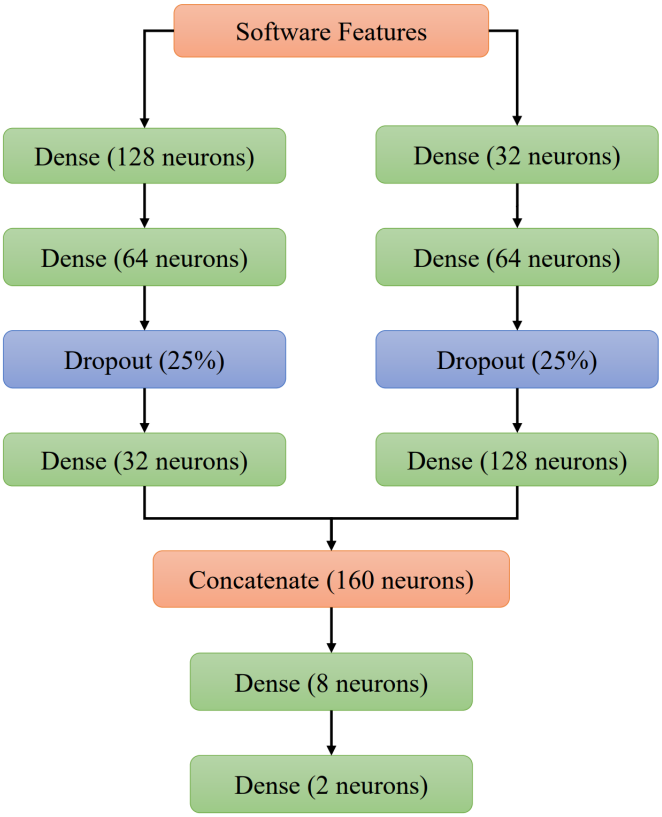

Fig. 2: The proposed autonomous cars software defect prediction model.

where, the $final\_prediction$ has binary values in it based on the higher probability of not having or having a defect in the software. The objective function of the D-ANN:

$$accuracy_{max} = \frac{CP}{TDL} \tag{7}$$

where, CP is the total number of correct predictions for both categories, and TDL is the data length of the test set.

IV. Our Proposed Solution

In this section, we describe the proposed D-ANN model, which can predict the software defects in autonomous cars using the software features dataset. Fig. 2 shows the flowchart that depicts the entire prediction process. It uses the combination of multiple dense layers and dropout layers to generate output from the given input features of the software. The reason for using D-ANN over the classical ANN is that simple ANN cannot find the complex pattern of software defects present in autonomous cars. The prediction accuracy that we got using the traditional ANN was around 81.46%, whereas the D-ANN yielded an accuracy of 82.61%.

In the proposed D-ANN model, the input data has twenty-one software features fed into the proposed model, and it gives the output as a probability for each class being defected or not. The first step is to split the input data into two parts. One part is given to the dense layer, which is on the left-hand side. The main function of the dense layer is to establish a deep connection with the input data. Every neuron in this layer receives input data and performs a matrix-vector multiplication to generate the output, which is then passed to the next connected layer in the model.

The first dense layer has 128 neurons, and there are 21 different inputs in terms of software features.

$$H_1 = A\left[exp1\right] \tag{8}$$

$$exp1 = \left(I_1 W_{I_1 H_1}\right) + \left(I_2 W_{I_2 H_1}\right) + \left(I_3 W_{I_3 H_1}\right) + \cdots + \left(I_n W_{I_n H_1}\right) + b_I$$

$$H_2 = A\left[exp2\right] \tag{9}$$

$$exp2 = \left(I_1 W_{I_1 H_2}\right) + \left(I_2 W_{I_2 H_2}\right) + \left(I_3 W_{I_3 H_2}\right) + \cdots + \left(I_n W_{I_n H_2}\right) + b_I$$

$$H_3 = A\left[exp3\right] \tag{10}$$

$$exp3 = \left(I_1 W_{I_1 H_3}\right) + \left(I_2 W_{I_2 H_3}\right) + \left(I_3 W_{I_3 H_3}\right) + \cdots + \left(I_n W_{I_n H_3}\right) + b_I$$

$$H_{128} = A\left[exp4\right] \tag{11}$$

$$exp4 = \left(I_1 W_{I_1 H_{128}}\right) + \left(I_2 W_{I_2 H_{128}}\right) + \left(I_3 W_{I_3 H_{128}}\right) + \cdots + \left(I_n W_{I_n H_{128}}\right) + b_I$$

Equation (8), (9), (10), and (11) describe the basic formulas of ANNs. W is the weight associated with the edge joining two consequent nodes, b is the bias, and A is the activation function [25] where, $W_{I_\kappa H_\gamma}$ is the weight of the edge running from I_κ to H_γ. We can write all these equations as follows:

$$H_\gamma = A\left(\left(\sum_{\kappa=1}^{21}\left(I_\kappa W_{I_\kappa H_\gamma}\right)\right) + b_I\right) \tag{12}$$
$$\kappa \rightarrow 1-21,$$
$$\gamma \rightarrow 1-128$$

where, κ represents the range of inputs, i.e., from 1→21, and γ represents the number of neurons in the first dense layer, ranging from 1 → 128. b_I is the bias associated with the input layer. This output is then passed to the next dense layer that has 64 neurons. This means that these 128 output points will be associated with 64 neurons of the second dense layer. This step is described in mathematical form as:

$$K_\beta = A\left(\left(\sum_{\gamma=1}^{128}(H_\gamma W_{H_\gamma K_\beta})\right) + b_H\right) \tag{13}$$
$$\gamma \rightarrow 1-128,$$
$$\beta \rightarrow 1-64$$

where, γ represents the range of inputs, i.e., 1→128, and β represents the number of neurons in the second dense layer, which ranges from 1 to 64. b_H is the bias associated with the first dense layer, and A is the activation function. This output is passed to the next layer, which is the dropout layer. The dropout layer has a 25% dropout rate. The main function of this layer is to drop some random samples to prevent the over-fitting problem.

The dropout layer is followed by the third dense layer, which has 32 neurons. The output from the dropout layer is associated with these 32 neurons. The mathematical representation of this step is as follows:

$$L_\alpha = A\left(\left(\sum_{\beta=1}^{64}(K_\beta W_{K_\beta L_\alpha})\right) + b_K\right) \tag{14}$$
$$\beta \rightarrow 1-64,$$
$$\alpha \rightarrow 1-32,$$

where, β represents the range of inputs, i.e., 1→64, and α represents the number of neurons in the third dense layer, which ranges from 1 to 32. b_K is the bias associated with the previous dense layer, and A is the activation function. Meanwhile, the first dense layer on the right-hand side also gets the software features as its inputs in the parallel process. It has 32 neurons, which are connected with every data point to generate the next output. The mathematical representation of this step is as follows:

$$H'_{\gamma'} = A\left(\left(\sum_{\kappa=1}^{21}(I_\kappa W_{I_\kappa H'_{\gamma'}})\right) + b_I\right) \tag{15}$$
$$\kappa \rightarrow 1-21,$$
$$\gamma' \rightarrow 1-32$$

where, κ represents the range of inputs, i.e., 1→21, and γ' represents the number of neurons in the first dense layer, ranging from 1 to 32. b_I is the bias associated with the input layer.

This output is passed to the following dense layer, which has 64 neurons. This step is described in mathematical form as:

$$K'_{\beta'} = A\left(\left(\sum_{\gamma'=1}^{32}(H'_{\gamma'}W_{H'_{\gamma'},K'_{\beta'}})\right) + b_{H'}\right) \quad \gamma' \rightarrow 1-32, \quad \beta' \rightarrow 1-64 \tag{16}$$

where, γ' represents the range of inputs, i.e., 1→32, and β' represents the number of neurons in the second dense layer, which ranges from 1 to 64. $b_{H'}$ is the bias associated with the previous dense layer, and A is the activation function. This output is passed to the next layer, which is the dropout layer that has 25% dropout rate. The dropout layer is followed by the third dense layer, which has 128 neurons. The output from the dropout layer will be associated with these 128 neurons. The mathematical representation of this step is as follows:

$$L'_{\alpha'} = A\left(\left(\sum_{\beta'=1}^{64}(K'_{\beta'}W_{K'_{\beta'},L'_{\alpha'}})\right) + b_{K'}\right) \quad \beta' \rightarrow 1-64, \quad \alpha' \rightarrow 1-128 \tag{17}$$

where, β' represents the range of inputs, i.e., 1→64, and α' represents the number of neurons in the third dense layer ranges from 1 to 128. $b_{K'}$ is the bias associated with the previous dense layer, and A is the activation function. The output generated by the left-hand side layers (L_α), and the output of right-hand side layers ($L'_{\alpha'}$), are both concatenated into a single data unit. This unit has 160 neurons, including 32 from the left and 128 from the right.

$$L_\alpha + L'_{\alpha'} = \left[A\left(\left(\sum_{\beta=1}^{64}(K_\beta W_{K_\beta L_\alpha})\right) + b_K\right) + A\left(\left(\sum_{\beta'=1}^{64}(K'_{\beta'}W_{K'_{\beta'},L'_{\alpha'}})\right) + b_{K'}\right)\right] \quad \beta \rightarrow 1-64, \quad \alpha \rightarrow 1-32, \quad \beta' \rightarrow 1-64, \quad \alpha' \rightarrow 1-128 \tag{18}$$

Next, this data unit is passed to the next dense layer, which has eight neurons. The 160 neurons from the data unit are associated with these eight neurons. The mathematical representation of this step is as follows:

$$m_j = \left[A\left(\left(\left(\sum_{\alpha=1}^{32}(L_\alpha W_{l_\alpha m_j})\right) + b_L\right) + \left(\left(\sum_{\alpha'=1}^{128}(L'_{\alpha'}W_{L'_{\alpha'},m_j})\right) + b_{L'}\right)\right)\right] \quad \alpha \rightarrow 1-32, \quad \alpha' \rightarrow 1-128, \quad j \rightarrow 1-8 \tag{19}$$

where, α and α' represents the ranges of inputs that are 1→32 and 1→128, respectively. j represents the number of neurons in the current dense layer, which ranges from 1 to 8. b_L and $b_{L'}$ are the biases associated with the previous dense layers, and A is the activation function. This output is passed to the final dense layer, which has two neurons. This step is described mathematically as:

$$n_i = \sigma\left(\left(\sum_{j=1}^{8}(m_j W_{m_j n_i})\right) + b_m\right) \quad j \rightarrow 1-8, \quad i \rightarrow 1-2 \tag{20}$$

where, j represents the range of inputs, i.e., 1→8, and i represents the number of neurons in the second dense layer ranges from 1 to 2. b_m is the bias associated with the previous dense layer, and A is the activation function. The output is calculated using Eq. (20) and the activation function, which is the *softmax* function (21). Both these outputs are passed to the *argmax* function, which determines which value is the largest among 0s and 1s. After determining this, the model gives its final output

as equation (6) describes. From the generated output, the model can predict any defects in the software. This model is trained over 40 epochs. After the training, the values are predicted.

$$\sigma(\overrightarrow{Q})_i = \frac{e^{Q_i}}{\sum_{j=1}^{M} e^{Q_j}} \tag{21}$$

where, σ is a *softmax* function. $\overrightarrow{Q}$ is an input vector. e^{Q_i} and e^{Q_j} represent the standard exponential functions for both input and output vectors, respectively. M is the number of classes in a multi-class classifier.

V. Performance Evaluation

In this section, we evaluate the performance of our proposed model. We have compared results obtained using the proposed model with the classical ML models and Auto-ML models in the following section. We have used the following performance metrics: precision, recall, and F1-score. Here, precision is a performance indicator that shows the number of true positive predictions divided by the total number of positive predictions. Precision is the number of true positives divided by the number of true positives and false positives. The recall is a metric that quantifies the number of correct positive predictions made out of all positive predictions that could have been made. The recall is the number of true positives divided by the number of true positives and false negatives. Here, F1-score is the harmonic mean of the precision and recall. F1-score is the total number of true positives divided by true positives plus 0.5*(false positives + false negatives). The support value (the quantity that shows the number of samples of the true responses belonging to a specific class) is also calculated for the proposed Auto-ML models. Moreover, confusion metrics include each quantity's macro average and weighted average. The Receiver Operating Characteristic (ROC) (a graph that shows the performance of a classification model at all classification thresholds in terms of two parameters - true positive rate and false positive rate) is also plotted.

TABLE II: Performance parameters.

Parameter	**Values**
Programming language	Python 3.8.0
Platform	Google colab
Framework	TensorFlow
Total data points	10880
Train data points	8160
Test data points	2720
Batch size	64
Epochs	40
Optimizer	Adam
Loss function	Binary cross-entropy
Metrics	Accuracy
Supporting metrics	precision, recall, F1-score

We have taken a standard Kaggle dataset and pre-processed the data using a standard scalar method. The D-ANN was trained on the pre-processed data using the parameters and values shown in Table II. These parameter include the programming language, the amount of training and testing data points, the framework used, loss function, optimization function, metrics, and other parameters.

A. Classical ML Models

Firstly, the input data consists of twenty-one different features of the autonomous car software is pre-processed and passed to five different ML models: KNN, decision tree, Gaussian Naive Bayes, SVM, and random forest. The input data is divided into training (75%, i.e., 8160 data points) and testing datasets. Then, we optimized the hyperparameters of the autonomous car software data using the *GridSearchCV* function to choose the best parameter combination for the learning model. This step increases the accuracy of the prediction model. For KNN, the value of cross-validator (CV) is 10, which means that the cross-validation will be performed ten times for each selected set of hyperparameters. In the end, the model selects a pair that generates the best output in terms of accuracy of defects prediction in autonomous car software. The KNN algorithm achieved a software defect prediction accuracy of 81.17%, the highest among all five ML algorithms. The decision tree algorithm also generates reliable results with an accuracy of $\approx$ 80%. For the decision tree, the hyperparameters are optimized in the same way as we did in the KNN algorithm by considering the value of CV as 10. Then, Gaussian Naive Bayes achieved a comparatively lower software defect prediction accuracy, i.e., 79.59%. For SVM, a GridSearchCV uses different pairs of gamma and C to produce the best results. A total of four pairs are generated for different values of gamma and C, which are (0.1,1), (0.1,0.1), (1,1), and (1,0.1). SVM achieves the software defect prediction accuracy of 80.03%. The random forest algorithm manages to achieve the prediction accuracy of 80.44% using the same dataset. Fig. 3 shows the comparative analysis of the accuracies achieved by all five algorithms.

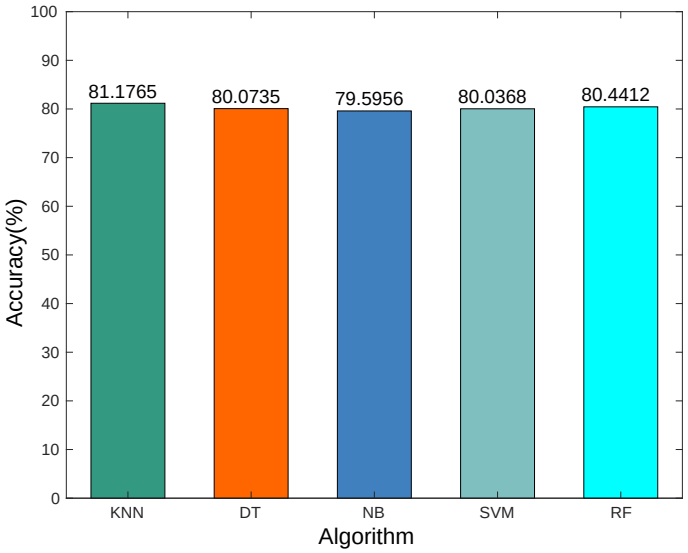

Fig. 3: Comparison of the accuracies of classical ML models.

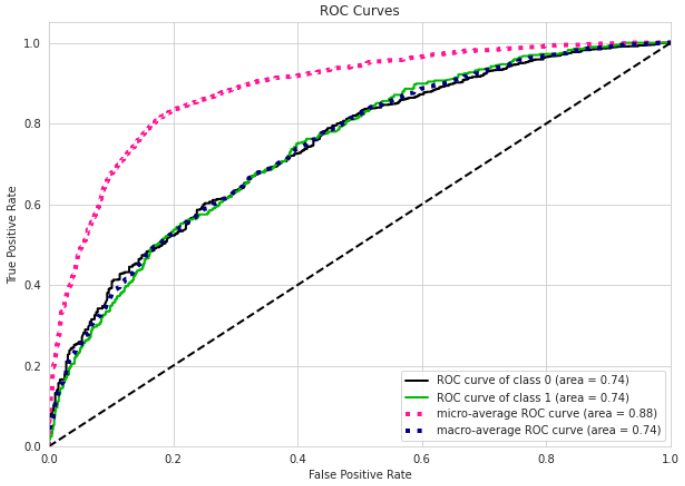

Fig. 4: ROC curve for AutoML.

B. AutoML Model

The AutoML model contains five different classification models: decision tree, random forest, Xgboost, neural network, and LightGBM. The AutoML model uses ensemble ML methods for binary classification and calculates the log loss value for all the algorithms. These values consist of the actual values and their prediction probability. It took around 290 seconds to train it, and it achieved an accuracy of 81.47%, and precision, recall, and f1-Score evaluated the accuracy of defect prediction in autonomous car software. Table III describes the confusion metric for the prediction accuracy. Here, precision determines the number of true positive class predictions, the software defects detected, and recall determines the number of positive class predictions out of all data units in the given dataset, which means the total predicted defects out of all autonomous car software features. F1-score is the harmonic mean of the precision and recall. Support is the quantity that shows the number of samples that belong to a specific class. For 0, the value of support is 2197, and it is 523 for 1. The sum of these two quantities is 2720, representing the total number of testing points taken from the dataset of autonomous car software. The Receiver Operating Characteristic (ROC), described as Fig. 4, shows a visual representation of the evaluation. It shows the performance of our model at various classification thresholds and uses two parameters namely, true-positive rate and false-positive rate.

C. D-ANN Model

The D-ANN model is different from the basic ANN model used to identify the defect from autonomous car's software, as it has around eight dense layers and two dropout layers. D-ANN is trained with 75% of the total dataset with 25% of data points reserved for testing purposes. The D-ANN model is trained for 40 epochs and achieved 82.61% prediction accuracy,

TABLE III: Auto-ML model's confusion metrics.

	Precision	Recall	F1-Score	Support
0	0.84	0.96	0.89	2197
1	0.55	0.21	0.31	523
Accuracy			0.81	2720
Macro average	0.69	0.59	0.60	2720
Weighted average	0.78	0.81	0.78	2720

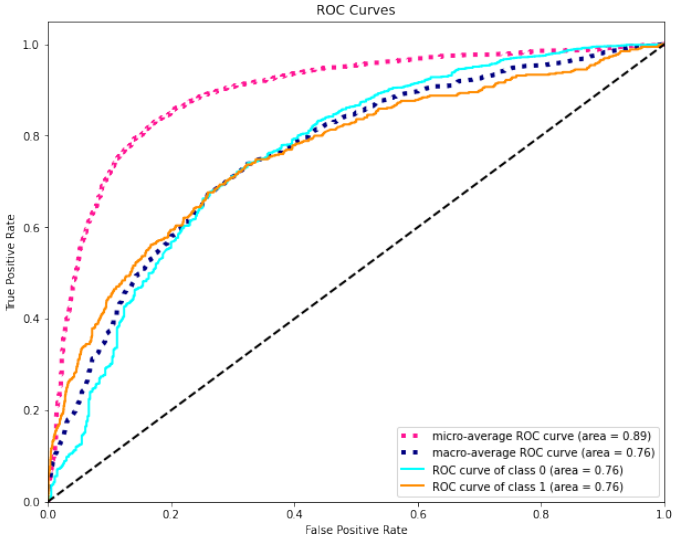

Fig. 5: ROC curve for D-ANN.

outperforming other existing ML models. This means that the D-ANN model is more accurate than classical ML models and the AutoML model in predicting the probable defect present in the software of autonomous cars. We evaluated D-ANN using precision, recall, and F1-score methods with a total of 2720 testing data points. The data point for class 0 is 2232, and for class 1 is 488. Table IV shows the confusion metrics of these quantities. We plotted the ROC curve for these values for different thresholds as Fig. 5 shows. Fig. 6 shows the comparison of precision, recall, and f1-score values for D-ANN and AutoML for the autonomous cars software's defect. Finally, Fig. 7 shows the comparison of accuracies of with existing AI models (i.e., classical ML models, AutoML, and D-ANN) in identifying defect present in the autonomous car software.

TABLE IV: D-ANN's confusion metrics.

	Precision	Recall	F1-Score	Support
0	0.85	0.86	0.90	2232
1	0.54	0.19	0.29	488
Accuracy			0.83	2720
Macro average	0.69	0.58	0.59	2720
Weighted average	0.79	0.83	0.79	2720

VI. Conclusion and Future Work

Software vulnerability detection in autonomous cars is a challenging task for researchers because of the complexity of the software used in autonomous cars'. Various ML models, such as SVM, Naive Bayes, and random forest, are widely used to detect faults in software. Moreover, LSTM-based models have also been used to prioritize defects and early warnings. Many DL-based algorithms have shown promising results in predicting bugs or vulnerabilities, and neural networks have improved the prediction results to some extent. However, most of the existing work focus only on limited aspects of software and cannot respond to all real-time defects. This work addresses this problem by describing it, discussing recently proposed solutions to address the problem, and proposing a viable solution that can be helpful to the industry, the research community, and the scientific domain of deep learning. This paper proposes an improved ANN model called D-ANN, with additional dropout layers for software defect prediction. Our proposed model can utilize all the features of the dataset. We evaluated the model using well-known performance metrics such as precision and recall and f1-score. Our performance results show that our proposed model outperforms existing AI models, such as the classical ML models and the AutoML model. The accuracy achieved using the proposed model is 82.6103%.

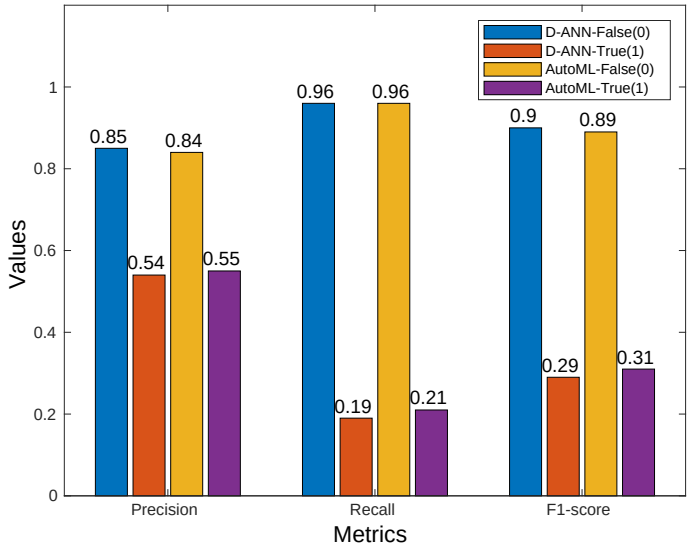

Fig. 6: Comparison of performance metrics for AutoML and D-ANN.

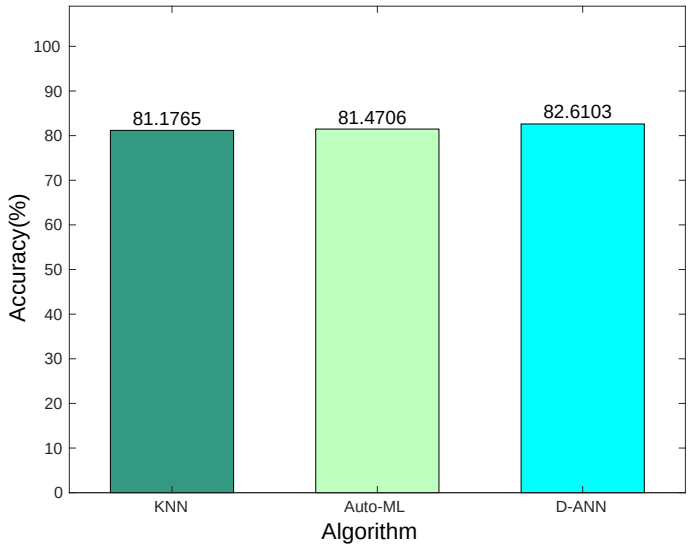

Fig. 7: Final comparison with D-ANN.

In the future, we will include certain hardware features of autonomous cars along with enhanced software debugging techniques. We will integrate this system with traffic regulation and inter-vehicle communication systems to improve it further. This work will help make autonomous cars more reliable and improve decision-making, management operations, and performance. Furthermore, as we are moving toward the 4th industrial revolution, the domain of autonomous vehicles is growing bigger. There are plenty of scopes of this research, and we want to take it to the next level by building a similar scheme for other autonomous vehicles such as trucks and buses.

REFERENCES

[1] T. Litman, *Autonomous vehicle implementation predictions*. Victoria Transport Policy Institute Victoria, Canada, 2017.
[2] S. Betz, "The top 25 self-driving car companies paving the way for an autonomous future." https://builtin.com/transportation-tech/self-driving-car-companies. Online; Accessed: 2021.
[3] A. Kopestinsky, "25 astonishing self-driving car statistics for 2021." https://policyadvice.net/insurance/insights/self-driving-car-statistics/. Online; Accessed: 2021.
[4] R. Hussain and S. Zeadally, "Autonomous cars: Research results, issues, and future challenges," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1275–1313, 2019.
[5] P. Koopman and M. Wagner, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90–96, 2017.
[6] J. Guerrero-Ibáñez, S. Zeadally, and J. Contreras-Castillo, "Sensor technologies for intelligent transportation systems," *Sensors*, vol. 18, no. 4, 2018.
[7] C. Law, "The dangers of driverless cars." https://www.natlawreview.com/article/dangers-driverless-cars. Online; Accessed: 2021.
[8] A. D. Kumar, K. N. R. Chebrolu, S. KP, *et al.*, "A brief survey on autonomous vehicle possible attacks, exploits and vulnerabilities," *arXiv preprint arXiv:1810.04144*, 2018.

[9] R. Gupta, S. Tanwar, N. Kumar, and S. Tyagi, "Blockchain-based security attack resilience schemes for autonomous vehicles in industry 4.0: A systematic review," *Computers & Electrical Engineering*, vol. 86, p. 106717, 2020.

[10] B. Kim, Y. Kashiba, S. Dai, and S. Shiraishi, "Testing autonomous vehicle software in the virtual prototyping environment," *IEEE Embedded Systems Letters*, vol. 9, no. 1, pp. 5–8, 2016.

[11] H. Tahir, S. U. R. Khan, and S. S. Ali, "Lcbpa: An enhanced deep neural network-oriented bug prioritization and assignment technique using content-based filtering," *IEEE Access*, vol. 9, pp. 92798–92814, 2021.

[12] M. Maddeh, S. Ayouni, S. Alyahya, and F. Hajjej, "Decision tree-based design defects detection," *IEEE Access*, vol. 9, pp. 71606–71614, 2021.

[13] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with big data: a deep learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 865–873, 2014.

[14] M. Alnabhan, A. Hammouri, M. Hammad, and F. Alsarayrah, "Software bug prediction using machine learning approach," *Int. J. Adv. Comput. Sci. Appl*, vol. 9, no. 2, 2018.

[15] J. L. Zambrano-Martinez, C. T. Calafate, D. Soler, L.-G. Lemus-Zúñiga, J.-C. Cano, P. Manzoni, and T. Gayraud, "A centralized route-management solution for autonomous vehicles in urban areas," *Electronics*, vol. 8, no. 7, 2019.

[16] C. L. Prabha and N. Shivakumar, "Software defect prediction using machine learning techniques," in *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, pp. 728–733, IEEE, 2020.

[17] L. Šikić, P. Afrić, A. S. Kurdija, and M. ŠIlić, "Improving software defect prediction by aggregated change metrics," *IEEE Access*, vol. 9, pp. 19391–19411, 2021.

[18] H. Wang, W. Zhuang, and X. Zhang, "Software defect prediction based on gated hierarchical lstms," *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 711–727, 2021.

[19] A. Abid, M. T. Khan, and J. Iqbal, "A review on fault detection and diagnosis techniques: basics and beyond," *Artificial Intelligence Review*, vol. 54, pp. 3639–3664, Jun 2021.

[20] J. Ren, M. Green, and X. Huang, "Chapter 8 - from traditional to deep learning: Fault diagnosis for autonomous vehicles," in *Learning Control* (D. Zhang and B. Wei, eds.), pp. 205–219, Elsevier, 2021.

[21] W. Lee, K. Yun, M. Chung, J. Oh, H. Shin, and K. Kwak, "Development of total security platform to protect autonomous car and intelligent traffic system under 5g environment," in *Mobile Internet Security* (I. You, H. Kim, T.-Y. Youn, F. Palmieri, and I. Kotenko, eds.), (Singapore), pp. 379–395, Springer Nature Singapore, 2022.

[22] I. P. Gomes and D. F. Wolf, "Health monitoring system for autonomous vehicles using dynamic bayesian networks for diagnosis and prognosis," *Journal of Intelligent & Robotic Systems*, vol. 101, p. 19, Dec 2020.

[23] J. Sayyad Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases.." School of Information Technology and Engineering, University of Ottawa, Canada, 2005.

[24] Kaggle, "Software defect prediction." https://www.kaggle.com/semustafacevik/software-defect-prediction. Online; Accessed: 2017.

[25] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," *Journal of pharmaceutical and biomedical analysis*, vol. 22, no. 5, pp. 717–727, 2000.