Software Bug Prediction using ResNet Neural Networks

Deepesh Tank
Computer Science and Engineering
Indian Institute of Technology
Varanasi, (IIT BHU)
Varanasi, UP, India
deepeshtank.cse18@itbhu.ac.in

***Abstract*—Software Bug Prediction Task is an important issue in software development and maintenance processes, which concerns with the overall of software successes. This is because predicting the software faults in earlier phase improves the software quality, reliability, efficiency and reduces the software cost. However, developing robust bug prediction model is a challenging task and many techniques have been proposed in the literature. This paper presents a software bug prediction model based on ResNet Neural Network using a binary-class classification approach. The evaluation process showed that Deep Learning Model can be used effectively with high accuracy rate. Furthermore, we will compare the model performance on different software modules provided in NASA Dataset.**

Keywords—ResNet Neural Network, Software Bug Prediction, NASA dataset, ResNet50, ResNet50V2, Artificial Neural Networks (*ANNs****)***

I. Introduction

The existence of software bugs affects dramatically on software reliability, quality, and maintenance cost. Achieving bug-free software also is hard work, even the software applied carefully because most time there is hidden bugs. In addition to, developing software bug prediction model which could predict the faulty modules in the early phase is a real challenge in software engineering. Software bug prediction is an essential activity in software development. This is because predicting the buggy modules before software deployment achieves the user satisfaction, improves the overall software performance. Moreover, predicting the software bug early improves software adaptation to different environments and increases the resource utilization. Various techniques have been proposed to tackle Software Bug Prediction (SBP) problem. The most known techniques are Machine Learning (ML) techniques. The ML techniques are used extensively in SBP to predict the buggy modules based on historical fault data, essential metrics and different software computing techniques.[1]

In this paper, we have used NASA software Repository to predict bugs in the software [4]. We built our model using untrained different variants of ResNet Neural architecture and then trained it using labeled data [3].

The paper is organized as follows: in section II Dataset; in section III ResNet Neural Network; in section IV Methodology; in section V Evaluations Parameters; the main results are presented in section VI, with conclusions to follow in section VII.The last section contains references.

II. Dataset

The dataset we have is NASA dataset which is a software defect dataset created by NASA in 2004.

Data comes from McCabe and Halstead features extractors of source code.[5][6] These features were defined in the 70s in an attempt to objectively characterize code features that are associated with software quality.

Dataset consists of 21 features and 1 label that is (True or False). Features are independent variables and label is the dependent variable which is linear combination of independent variables.

Features includes attributes like line of code, number of operator, cyclomatic complexity, essential complexity etc. Combined Dataset is also created concatenating all the csv files (containing same number of attributes i.e., 21). There are a total of 8 different files in the dataset which consists of data from 8 different programs.[4]

III. ResNet Neural Network

It is a type of deep neural network. For solving a complex problem, we stack some additional layers in the Deep Neural Networks which results in improved accuracy and performance. Adding more layers means layers learning more complex features. Deep learning network Like CNN faces with vanishing gradient problems.

Using ResNet solves the vanishing gradient problem by allowing an alternate shortcut path for the gradient to flow through. and improve the accuracy of the model.[3]

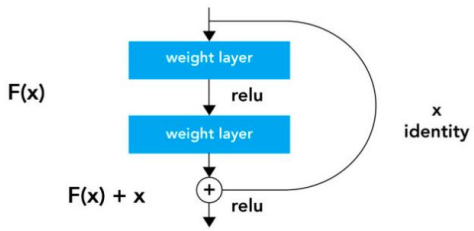


Fig 1. ResNet Residual Block

IV. Methodology

We follow the approach of binary classification using ResNet Neural Network. Where two classes represent if there is a bug or not. Detailed methodology is described using following sub-sections:

A. Preprocessing

It can be described in following steps.

1) First, we scale the dataset using Z score Normalisation

2) To remove class imbalance, we use Smote technique.

3) We split the dataset in both ratios of 70:30 and 80:20 for getting experimental results.

4)Finally we compare the model performance on 2 types of ResNet Variants. (ResNet50 and ResNet50V2)

B. Model Architecture

We use Keras Sequential API for creating our Model Architecture [2]. Input is taken in form of the image of size (64*64*3) therefore we first add a dense layer to convert input data to the image of mentioned dimension. Then we add untrained ResNet Variant Models to the architecture [3]. As we know the output of ResNet Variants is of dimension 2048. Therefore, we reduce the number of dimensions using multiple dense layers with 'relu' activation. The last layer of the architecture has a Sigmoid layer to predict the binary class [3]. The dataset taken for training and testing purposes is the NASA dataset [4].

C. Loss Function and Optimizer

The binary Cross-Entropy function is used to calculate the loss for backward propagation. SGD (Stochastic Gradient Descent) Optimizer is used for updating the weights of the model. Learning Rate is set to 0.0001 for the SGD optimizer.

V. EVALUATION PARAMETERS

Following evaluation parameters are used to examine the performance of the deep learning model.[7]

A. Recall

Recall is the fraction of relevant instances that were retrieved.

Recall = True Positive / (True Positive + False Negative)

B. Accuracy

The accuracy of a classifier is given as the percentage of total correct predictions divided by the total number of instances.

Accuracy = (True Positive + True Negative)/(Total)

C. Precision

Precision is the fraction of relevant instances among the retrieved instances.

Precision = True Positive / (True Positive + False Positive)

D. F1 Score

It's a measure of the test's accuracy and calculated from precision and recall

F1 Score = (2 * Precision* Recall)/ (Precision + Recall)

E. ROC Curve

A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: True Positive Rate. False Positive Rate

F. AUC (Area Under Curve)

AUC measures the entire two-dimensional area underneath the entire ROC curve [7].

VI. RESULTS

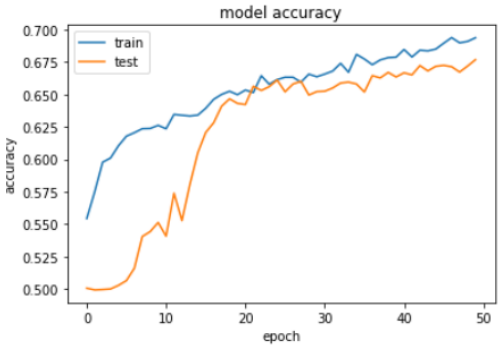
The following results are obtained when the training is done on 50 epochs and batch size is variable depending on the number of data points. We obtained our results on 2 variants of the ResNet Model (ResNet50 and ResNet50V2) and dataset divided in 80:20 and 70:30 Ratio (Train to test ratio).

	Ratio	Accuracy	Recall	Precision	F1 Score	AUC
CM1	**80:20**	0.7374	0.8023	0.6969	0.7459	0.7945
	70:30	0.7026	0.6815	0.7132	0.6969	0.7598
JM1	**80:20**	0.6333	0.5328	0.6967	0.6038	0.6884
	70:30	0.6663	0.5739	0.7016	0.6314	0.7278
KC1	**80:20**	0.7152	0.7272	0.7052	0.7160	0.8001
	70:30	0.7193	0.6846	0.7524	0.7169	0.8169
KC2	**80:20**	0.7090	0.6666	0.7200	0.6923	0.7782
	70:30	0.7338	0.5615	0.8902	0.6886	0.8472
KC3	**80:20**	0.6825	0.6470	0.8461	0.7333	0.7915
	70:30	0.6702	0.6818	0.6382	0.6593	0.7315
MC1	**80:20**	0.8737	0.8311	0.9026	0.8653	0.9543
	70:30	0.8660	0.8080	0.9173	0.8594	0.9586
MC2	**80:20**	0.6969	0.7500	0.7500	0.7500	0.7480
	70:30	0.6327	1.0000	0.5714	0.7272	0.7092
PC1	**80:20**	0.8252	0.9073	0.7782	0.8378	0.8816
	70:30	0.7783	0.7500	0.7986	0.7735	0.8684
Full Data set	**80:20**	0.7329	0.7357	0.7325	0.7341	0.8037
	70:30	0.6768	0.6475	0.6884	0.6673	0.7419

Table 1. Results on **ResNet50** Model

	Ratio	Accuracy	Recall	Precision	F1 Score	AUC
CM1	**80:20**	0.7932	0.8333	0.7731	0.8021	0.8876
	70:30	0.7769	0.7786	0.7669	0.7727	0.8765
JM1	**80:20**	0.6906	0.5723	0.7854	0.6621	0.7264
	70:30	0.6712	0.5910	0.7526	0.6551	0.7324
KC1	**80:20**	0.7938	0.8309	0.7620	0.7949	0.8634
	70:30	0.7820	0.8220	0.7671	0.7936	0.8717
KC2	**80:20**	0.8303	0.8000	0.8607	0.8292	0.8780
	70:30	0.7903	0.6960	0.8613	0.7699	0.8411
KC3	**80:20**	0.7460	0.5757	0.9047	0.7037	0.8232
	70:30	0.6808	0.5400	0.7941	0.6428	0.7981
MC1	**80:20**	0.9664	0.9840	0.9485	0.9659	0.9954
	70:30	0.9245	0.9152	0.9087	0.9119	0.9764
MC2	**80:20**	0.6667	0.7727	0.7391	0.7555	0.6777
	70:30	0.6122	0.9615	0.5814	0.7199	0.6062
PC1	**80:20**	0.8859	0.9402	0.8437	0.8894	0.9518
	70:30	0.8317	0.8494	0.8230	0.8359	0.9154
Full Data set	**80:20**	0.7451	0.7475	0.7526	0.7499	0.8128
	70:30	0.7326	0.7419	0.7347	0.7383	0.8064

Table 2. Results on **ResNet50V2** Model



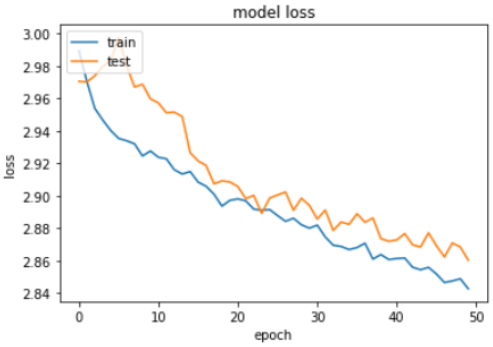


Fig 2. ResNet50 (70:30) model [accuracy/loss v/s epochs]

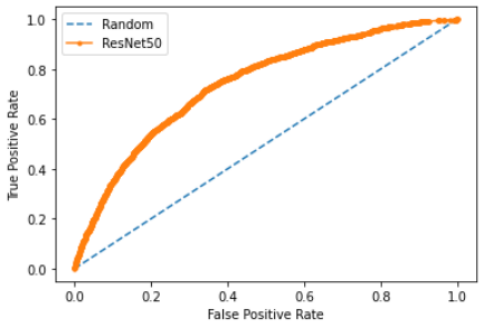
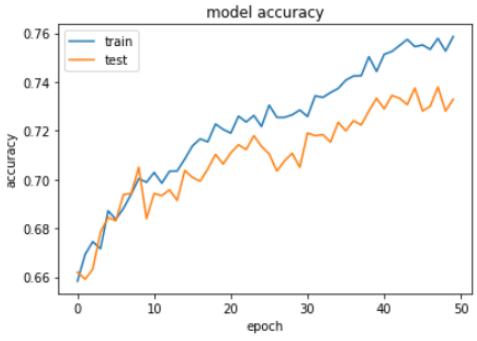


Fig 3. ROC Curve for ResNet50 (70:30) model



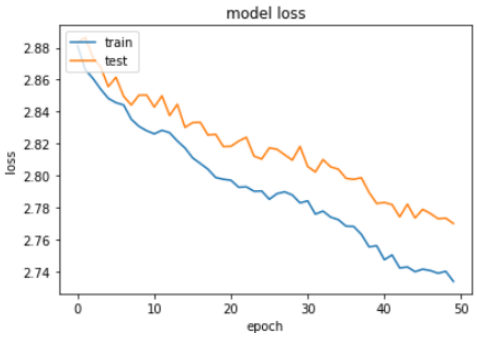


Fig 4. ResNet50 (80:20) model [accuracy/loss v/s epochs]

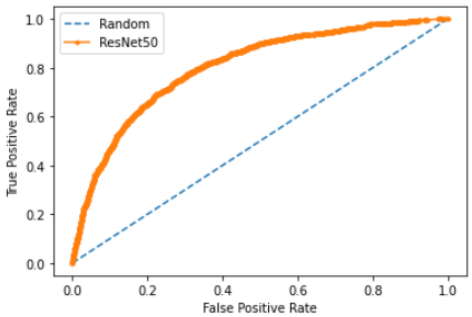
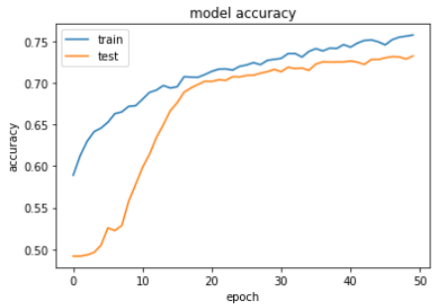


Fig 5. ROC Curve for ResNet50 (80:20) model



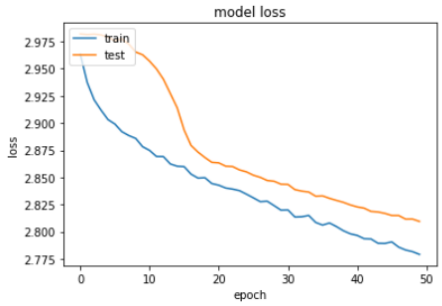


Fig 6. ResNet50V2 (70:30) model [accuracy/loss v/s epochs]

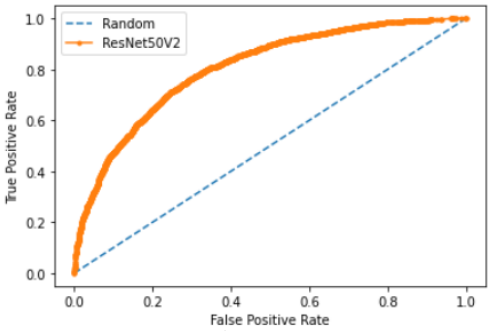


Fig 7. ROC Curve for ResNet50V2 (70:30) model

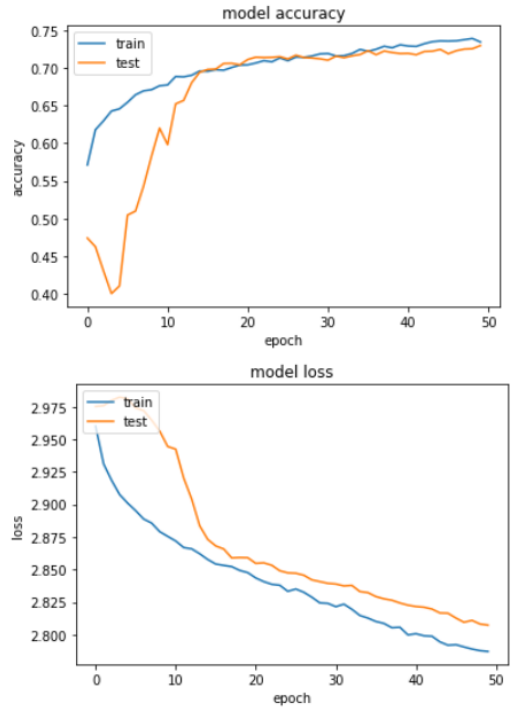


Fig 8. ResNet50V2 (80:20) model [accuracy/loss v/s epochs]

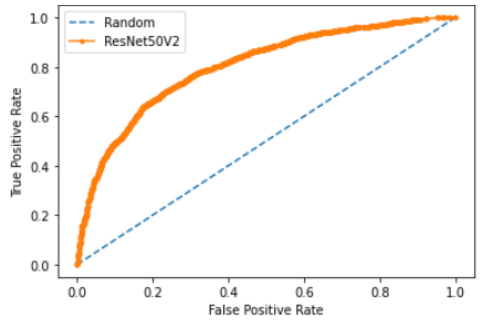


Fig 9. ROC Curve for ResNet50V2 (80:20) model

VII. Conclusion

The following Conclusions can be derived from implementing the paper:

1. The value of hyperparameters (batch size) varies with the choice of training dataset file.
2. A noticeable difference between the accuracy of 70:30 and 80:20 ratio of MC2 in both model variants due to a very small number of data points.
3. Our model requires sufficient data points to obtain good results therefore small dataset files like KC3 and MC2 have less accuracy than the remaining dataset files.
4. Best Results are obtained on MC1 dataset file on both variants and on both ratios due to large numbers of data points resulting in good training of the models. Accuracy of 0.9664 is achieved on ResNet50V2 (80:20) Model.
5. In terms of Ratio 80:20 yields better results than 70:30 training to testing ratio.
6. In terms of Model, ResNet50V2 variant yields better results than ResNet50 variant.
7. An best accuracy 0.7451 is obtained for combined dataset on ResNet50V2 (80:20) model. Therefore, proving that our model is good in predicting the occurrence of bugs in the software.

REFERENCES

[1] Hammouri, Awni & Hammad, Mustafa & Alnabhan, Mohammad & Alsarayrah, Fatima. (2018). Software Bug Prediction using Machine Learning Approach. International Journal of Advanced Computer Science and Applications. 9. 10.14569/IJACSA.2018.090212.

[2] Gulli, A. & Pal, S., 2017. Deep learning with Keras, Packt Publishing Ltd.

[3] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[4] Sayyad Shirabad, J. and Menzies, T.J. (2005) The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada. Available: http://promise.site.uottawa.ca/SERepository/datasets-page.html

[5] T. J. McCabe, "A Complexity Measure," in IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308-320, Dec. 1976, doi: 10.1109/TSE.1976.233837.

[6] Maurice H. Halstead. 1977. Elements of Software Science (Operating and programming systems series). Elsevier Science Inc., USA.

[7] Goutte, Cyril & Gaussier, Eric. (2005). A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. Lecture Notes in Computer Science. 3408. 345-359. 10.1007/978-3-540-31865-1_25.