An Empirical Study of Fault Localisation Techniques for Deep Learning

Anonymous Author(s)

***Abstract*—Dummy abstract**
***Index Terms*—deep learning, real faults, fault localisation**

I. Introduction

Fault localisation (FL) in DNNs is a rapidly evolving area of DL testing [1]–[5]. The decision logic of traditional software systems is encoded in their source code. Correspondingly, fault localisation for such systems consists of identifying the parts of code that are most likely responsible for the encountered misbehaviours. Unlike traditional software systems, the decision logic of DL systems depends on many components such as the model structure, selected hyper-parameters, training dataset and the framework used to perform the training process. Moreover, DL systems are stochastic in nature, as a retraining with the exactly same parameters might lead to a slightly different final model and performance. These distinctive characteristics make the mapping of a misbehaviour (e.g., poor classification accuracy) to a specific fault type a highly challenging task.

Existing works [1], [2], [4], [6], [7] that focus on the problem of fault localisation for DL systems rely on patterns of inefficient model structure design, as well as a set of predefined rules about the values of internal indicators measured during the DL training process. This makes the effectiveness of these approaches highly dependent on the identified set of rules and on the threshold values selected to discriminate the values of the internal indicators of a fault.

To understand whether these tools effectively generalise to a diverse set of fault types and DL systems, and thus, are effective for the real-world usage, we performed an empirical evaluation on a benchmark of carefully selected subjects. In this benchmark we combined faults obtained by the artificial injection of defects into otherwise well-performing DL models and a set of reproduced real DL faults. This way we ensure that our evaluation involves model of different structures and complexity that solve problems from different domains.

Our results show that existing DNN FL techniques produce stable results in a relatively small amount of time. However, accuracy of fault localisation techniques should still be the focus of future research.

II. Background

Most of the proposed approaches focus on analysing the run-time behaviour during the model training. According to the collected information and some predefined rules, these approaches decide whether they can spot any abnormalities and report them [1], [2], [5].

During the training of a DNN, DeepLocalize (DL) [1] collects various performance indicators such as loss values, performance metrics, weights, gradients and neuron activation values. The main idea behind this approach is that the historic trends in the performance evaluation or the values propagated between layers can serve as an indicator of a fault's presence. To allow the collection of the necessary data, a developer should insert a custom callback provided by the tool into the source code regulating the training process. A callback is a mechanism that is invoked at different stages of the model training (e.g. at the start or end of an epoch, before or after a single batch [8]) to perform a set of desired actions - store historical data reflecting the dynamics of the training process in our case. Both tools then compare the analysed values with a list of pre-defined failure symptoms and root causes, which the authors have collected from the existing literature. Based on the performed analysis, DeepLocalize either claims that the model is *correct* or outputs an error message listing the detected misbehaviours. The final output of DeepLocalize contains the (1) fault type, (2) the layer and the phase (feed forward or backward propagation) in which the DL program has a problem, and (3) the iteration in which learning is stopped. The faults that the tool is able to detect include the following: "Error Before/After Activation", "Error in Loss Function", "Error Backward in Weight/Δ Weight", and "Model Doest not Learn" that suggests an incorrectly selected learning rate.

DeepDiagnosis (DD) [2] was built on the basis of DeepLocalize and improve the latter by enlarging the list of detected symptoms and connecting them to a set of actionable suggestions. It detects 10 types of faults: "Numerical Errors", "Exploding Tensor", "Unchanged Weight", "Saturated Activation", "Dead Node", "Activation Function's Output Out of Range", "Loss Not Decreasing", "Invalid Loss", "Invalid Accuracy", "Accuracy Not Increasing" and "Vanishing Gradient". Depending on the symptom, the actionable messages provided by Diagnosis suggest to change either the loss or optimisation function, layer number, initialisation of weights, learning rate, or indicate that training data is "improper". As DeepLocalize does not provide an output that can be translated into a specific fault affecting the model, we only use DeepDiagnosis in the empirical comparison of fault localisation tools.

Similarly to DeepLocalize and DeepDiagnosis, UMLAUT (UM) [5] operates through a callback that is attached to the model training call. This tool combines dynamic monitoring of the model behaviour during the training with heuristic static checks of the model structure and its parameters. As an output, it provides the results of the checks along with

best practices and suggestions on how to deal with the detected faults. The tool comprises of 10 heuristics for which the authors found mentions in diverse set of sources such as API documentation and existing literature, lecture notes and textbooks, courses, blogs and non-scientific articles. The heuristics are divided by the area of application into "Data Preparation", "Model Architecture" and "Parameter Tuning". Data preparation heuristics are dynamic and check if the training data contains NaNs, has invalid shape, is not normalised or if the validation accuracy is higher than 95% after the third epoch of the training. On the other hand, all model architecture heuristics are static and are focused on the usage of correct activation functions. Parameter tuning category combines both dynamic and static rules that aim to detect overfitting and control the values of the learning and dropout rates. As an output, the tool returns a list of heuristics that were violated.

Nikanjam *et al.* [4] propose NEURALINT (NL), a model-based fault detection approach that uses meta-modelling and graph transformations. The technique starts with building a meta-model for DL programs that consists of their base structure and fundamental properties. It then performs a verification process that applies 23 pre-defined rules, implemented as graph transformations, to the meta-model, to check for any potential inefficiencies. The rules are classified into 4 high-level root causes as suggested by Zhang *et al.* [9]: "Incorrect Model Parameter or Structure" (5 rules), "Unaligned Tensor" (4 rules), "API Misuse" (5 rules), and "Structure Inefficiency" (9 rules). One example of "Unaligned Tensor" rule is a check whether consecutive layers in a model are compatible or whether the reshape of data did not lead to the loss of any elements. "API Misuse" includes a rule that ensures that optimiser is correctly defined and connected to the computational graph. Another rule in this category inspects the parameters initialisation to detect the cases when initialisation is performed more than once or after the training has already started. The "Incorrect Model Parameter or Structure" rule controls that weights and biases are initialised with appropriate values and that suitable activation functions are used for specific layer types. "Structure Inefficiency" is responsible to detect flaws in the design and structure of DNN that can result in the drop of model performance. Among others, there are rules in this category that check if the number of neurons in fully connected layers is decreasing when moving from the input to the output layer or rules that check that pooling layers are not used after each applied convolution not to loose too much information about an input.

DEEPFD [3] (DFD) is a learning-based framework that leverages mutation testing and popular ML algorithms to construct a tool capable of labelling a given DL program as correct or faulty according to a list of common fault types the tool has learned to detect. To train the classifiers that lie in the core of the technique, the authors prepare a set of correct and faulty models. Faulty models are obtained through the artificial injection of up to 5 mutations to each correct program used. The mutations that are used to inject faults are changing loss or optimisation function, changing learning rate, decreasing number of epochs, and changing activation functions. Consequently, these fault types correspond to the fault localisation capabilities of the tool. To construct the training dataset, all of the generated mutants and the original models are trained while run-time data of the same kind as for DEEPLOCALIZE, DEEPDIAGNOSIS, UMLAUT are collected. The authors then extract 160 features from these data using statistical operations (e.g. calculating skewness, variance or standard deviation). As the next step, three popular ML algorithms (K-Nearest Neighbors [10], Decision Tree [11] and Random Forest [12]) are trained on the created dataset. A union of the prediction results of these classifiers is used for fault identification and localisation in a given program under test. DEEPFD outputs a list of detected faults along with the code lines affected by each fault type.

The tools described in this section are built using a limited set of rules and best practices, fixed thresholds or training data, which creates an urge to explore the generalisability of these approaches to diverse programs and architectures.

III. EMPIRICAL STUDY

A. Research Questions

The *aim* of this empirical study is to compare existing DL fault localisation approaches and to explore their generalisability to different subjects represented by our benchmark of artificial and real faults. To cover these objectives, we define the following research questions:

- **RQ1. Effectiveness**: *Can existing FL approaches identify and locate defects in faulty DL models? Which FL tool produces the most accurate and actionable result?*
- **RQ2. Stability**: *Is the outcome of fault identification analysis stable across several runs?*
- **RQ3. Efficiency**: *How costly are FL tools when compared to each other?*

B. Benchmark

Nargiz ►*describe, cite repair paper*◄

C. Experimental Settings & Evaluation Metrics

For the comparison we use publicly available versions of all considered tools [13]–[16]. However, we had to limit the artificial faults to those obtained using CIFAR10, MNIST, and Reuters as DEEPDIAGNOSIS is not applicable to other subjects.

The authors of DEEPFD adopted the notion of statistical mutation killing [17] in their tool. They run each of the models used to train the classifier as well as the model under test 20 times to collect the run-time features. For the fault localisation using DEEPFD, we adopt an ensemble of already trained classifiers provided in the tool's replication package. Similar to the authors, for each faulty model in our benchmark, we collect the run-time behavioural features from 20 retrainings of the model. NEURALINT is based on static checks that do not require any training and thus, are not prone to randomness. We run each of the remaining tools 20 times to account for the randomness in training process and report the most frequently observed result.

To calculate the similarity between the ground truth provided for each fault in our benchmark and the fault localisation results, we adopt the Asymmetric Jaccard metric that was used in the empirical study of repair tools (see Section **??**). In this case, the metric measures the percentage of the fault types in the list of localised faults (OP_{loc}) that are also found in the ground truth (OP_{gt}):

$$AJ = \frac{|OP_{loc} \cap OP_{gt}|}{|OP_{gt}|} \quad (1)$$

IV. Results

A. Results

1) RQ1 (Effectiveness): Tables I, II, III, and IV present the output of the application of fault localisation tools to our benchmark. The column *'GT'* stands for *'Ground Truth'* and provides the list of fault types affecting the model. For artificial faulty models that were obtained by applying DeepCrime to subject models, we denote their fault types by the corresponding mutation operators (see Table **??**). Columns named *'tool_name-output'* show the fault list produced by each FL tool. As the fault types that affect real faulty models from our benchmark are also covered by DeepCrime, their ground truth is also presented as the corresponding mutation operators.

Please note that to simplify, we name 'ARM' faults as 'ACH' as both operators change the activation function of a layer. In addition to the mutation operators mentioned in Table **??**, 'HBS' is an operator that changes batch size to produce faulty models with wrong batch size, 'LRM' and 'LAD' simulate cases where there is a negative effect on the performance when a model is missing a layer or has a redundant one, respectively. Another operator, 'LCN', changes the number of neurons in a layer to an improper one, while 'LCF' corrupts the filter size of a convolutional layer. 'BCI' and 'CPP', however, are not part of DeepCrime's operators; they stand for 'Change bias initialisation' and 'Change Training Data Preprocessing', respectively. For some faults that affect not the whole model but only selected layers, in round brackets we specify the indexes of the faulty layers, for ground truth and for fault localisation results if this information is provided. Moreover, '-' means that an FL tool was not able to either find any fault in a program or to associate a detected symptom with a fault. 'N/A' means that the tool was not applicable to the fault type in question or crashed on it. For example, Neuralint only accepts optimisers that are defined as strings (e.g. 'sgd'), which in turn implies that the default learning rate as defined by the framework is used. This makes it not possible to provide an optimiser with modified learning rate. Symbol '|' separates distinct faults, while separation by '/' means that the faults are alternative to each other.

For each fault from the ground truth list, column *'tool_name-match-GT'* specifies whether or not it was detected by the corresponding tool (0 if not detected and 1 otherwise). For each row (issue) we underline the best result.

Interestingly, in the majority of cases UMLAUT (20 out of 22) and DeepDiagnosis (15 out of 22) suggest changing the activation function of the last layer to 'softmax' even if in 73 % of these cases for UMLAUT and 67% for DeepDiagnosis, the activation function already equals to 'softmax'. This also happens once to Neuralint. We exclude such misleading suggestions from the tools' output. Moreover, sometimes UMLAUT mentions that overfitting is possible. Since it is just a possibility and such a message does not point to a specific fault, we also exclude it from the analysis. Full output messages provided by the tools are available in our replication package [18].

Table V reflects the overall evaluation of the effectiveness of the FL tools. Column "GT # faults" shows the number of fault types in the ground truth, while columns 'tool_name # matches' show the number of faults from the ground truth a tool was able to detect. Columns 'tool_name AJ' show the values of the Asymmetric Jaccard metric calculated for each pair of an issue and a tool. We treated the cases when a tool is not applicable to an issue as if the tool has failed to locate any faults affecting the issue. Also, we provide mean values for each tool across artificial and real faults (rows 'Avg.') and across all issues in the benchmark (row 'T.A'), to ease the comparison between the tools. According to these numbers, DeepFD, on average, exhibits the best performance and significantly outperforms other tools on real faults. For artificial faults, however, DeepFD, Neuralint, and UMLAUT achieve similar performance. Overall, based on all measured metrics, DeepFD has the highest values, while all DeepDiagnosis's measurements are noticeably lower than for other tools. Neuralint and UMLAUT show similar performance according to AJ metric, but Neuralint has a higher mean number of matches with ground truth.

It is worth mentioning that, unlike other tools, DeepFD does not provide layer index suggestions. Thus, it is not possible to understand whether a successfully detected fault of 'ACH' type actually points to the correct layer. This is the case for 2 issues out of 22 and if we exclude these issues from the calculation of mean AJ, the result for DeepFD drops from 0.305 to 0.214, which makes it comparable with Neuralint and UMLAUT. If we assume that DeepFD correctly locates this fault with the probability of 50% (the suggested layer is either correct or not), the mean AJ value will be equal to 0.260. Also, for some of the fault types, other tools but DeepFD provide specific suggestions on which activation function (DD, UM) or weights initialisation (NL) to adopt or whether to increase or decrease the learning rate (UM).

RQ1: Our evaluation shows that all FL tools show relatively low AJ results as, for many issues, the tools are not able to successfully identify faults affecting the model. On average, DeepFD shows the best results and DeepDiagnosis the lowest. At the same time, Neuralint and UMLAUT perform quite similarly. Our findings show that this area of testing can benefit from future contributions aimed at improving the accuracy of the identified faults.

2) RQ2 (Stability): As mentioned before (see Section III-C), the authors of DEEPFD account for the instability of the training process and perform 20 retrainings when collecting input features both during the classifier training stage and during fault identification. This way, the output of the tool is calculated from 20 feature sets for each model under test.

NEURALINT does not require any training to be done and is based on static rules that are stable by design. We preformed 20 runs of all other tools to investigate their stability. We found out that outputs are stable across the experiment repetitions for all considered tools.

RQ2: Existing fault localisation tools provide stable results that do not change from execution to execution.

3) RQ3 (Efficiency): In this RQ, we investigated how demanding the evaluated approaches are in terms of execution time. Here we measure only the time required to run an FL tool on a subject without taking into account the time and effort needed to prepare the subject for the tool application. All of the tools require some manual work to be done: for DEEPFD, a user has to create serialised versions of the training dataset and model configuration according to a specific format; for DEEPDIAGNOSIS and UMLAUT, a user has to insert and a tool-specific callback to the code and provide it with a list of arguments; for NEURALINT, there is a number of manual changes to the source code to make the tool runnable.

Table VI shows execution time measured in seconds on 1 run of DEEPFD and NEURALINT, and an average of 20 runs for the remaining tools. Row 'T.A.' shows the average time spent by each tool on fault localisation over the whole benchmark. To allow fair comparison, row 'Avg.' shows the average execution time over the faults where all tools are applicable. Not surprisingly, DEEPFD takes considerably longer to run than the other tools since it trains 20 instances for each issue, while the other tools perform 1 (DEEPDIAGNOSIS, UMLAUT) or no retraining (NEURALINT). In addition, DEEPDIAGNOSIS often terminates the training when a faulty behaviour is observed, which makes its average execution time the shortest on the issues we used. As NEURALINT does not require to train a model to perform fault localisation, its average execution time is also quite low. It can be noted that for some faults that are very fast to train (e.g. C2, D1, D3, 6), a full training performed by UMLAUT takes less time than the static checks of NEURALINT. On average, DEEPDIAGNOSIS is the fastest to run, followed by NEURALINT and UMLAUT, and finally, DEEPFD. Despite the differences, the execution time of all the tools considered is compatible with real-world use.

RQ3: The tools considered in our empirical study operate on the basis of different strategies and require different numbers of retrainings of the same model to attempt fault localisation. Consequently, DEEPDIAGNOSIS, which often interrupts the execution when it detects a fault symptom, is the fastest, while DEEPFD is the slowest as it is designed to take into account the stochasticity in the training process by generating 20 instances of the model under test. NEURALINT performs fault localisation without training a model, making it sometimes faster than UMLAUT. Neither tool requires resources that are prohibitively expensive for practical use.

TABLE I: Ground Truth (GT) and fault localisation outcome generated by DEEPFD (DFD)

ID	GT	DFD-match-GT	DFD-output
M1	WCI(0)	0	HLR | ACH | LCH | HNE
M2	ACH(7)	0	OCH | HLR | HNE
M3	HLR	1	OCH | HLR | LCH
C1	ACH(2)	1	OCH | HLR | ACH | LCH
C2	HNE	0	OCH | ACH | LCH
C3	WCI(2)	0	OCH | ACH | LCH | HNE
R1	RAW(0)	0	HLR | LCH | HNE
R2	ACH(2)	0	OCH | LCH | HNE
R3	HLR	0	OCH | LCH | HNE
R4	LCH	1	ACH | LCH
R5	OCH	1	OCH | ACH | HNE
R6	WCI(0)	0	OCH | ACH| LCH | HNE
R7	ACH(2)	0	OCH | LCH | HNE
D1	ACH(7)	1	ACH
D2	OCH | HNE | HBS	0 | 0 | 0	ACH
D3	OCH | LCH | ACH(0,1) | HNE | HBS	1 | 0 | 0 | 0 | 0	OCH | HLR
D4	ACH(0,1) | LCH | HLR	0 | 0 | 0	OCH
D5	HNE | HBS	0 | 0	OCH | ACH
D6	HLR | HNE | LCH | ACH(1)	1 | 1 | 0 | 0	OCH | HLR | HNE
D7	HLR	0	LCH
D8	OCH | HLR	1 | 1	OCH | HLR | LCH | HNE
D9	CPP | ACH(5,6) | HBS	0 | 0 | 0	N/A

TABLE II: Ground Truth (GT) and fault localisation outcome generated by DEEPDIAGNOSIS (DD)

ID	GT	DD-match-GT	DD-output
M1	WCI(0)	0	HLR
M2	ACH(7)	1	ACH(7)
M3	HLR	0	-
C1	ACH(2)	0	-
C2	HNE	0	-
C3	WCI(2)	0	-
R1	RAW(0)	0	-
R2	ACH(2)	1	ACH(2)
R3	HLR	0	-
R4	LCH	0	LRM/LAD/ACH(0)
R5	OCH	0	-
R6	WCI(0)	0	-
R7	ACH(2)	1	ACH(2)
D1	ACH(7)	0	HLR
D2	OCH | HNE | HBS	0 | 0 | 0	-
D3	OCH | LCH | ACH(0,1) | HNE | HBS	0 | 0 | 0 | 0 | 0	-
D4	ACH(0,1) | LCH | HLR	1 | 0 | 0	ACH(1)
D5	HNE | HBS	0 | 0	-
D6	HLR | HNE | LCH | ACH(1)	0 | 0 | 0 | 0	-
D7	HLR	0	-
D8	OCH | HLR	0 | 0	-
D9	CPP | ACH(5,6) | HBS	0 | 0 | 0	N/A

TABLE III: Ground Truth (GT) and fault localisation outcome generated by NEURALINT (NL)

ID	GT	NL-match-GT	NL-output
M1	WCI(0)	1	WCI(0)
M2	ACH(7)	0	LCH
M3	HLR	0	N/A
C1	ACH(2)	0	-
C2	HNE	0	-
C3	WCI(2)	1	WCI(3)
R1	RAW(0)	0	-
R2	ACH(2)	0	LCH
R3	HLR	0	N/A
R4	LCH	1	LCH
R5	OCH	0	-
R6	WCI(0)	1	WCI(0)
R7	ACH(2)	0	LCH
D1	ACH(7)	0	LCH
D2	OCH | HNE | HBS	0 | 0 | 0	-
D3	OCH | LCH | ACH(0,1) | HNE | HBS	0 | 1 | 1 | 0 | 0	ACH(1) | LCH | LCN(0)
D4	ACH(0,1) | LCH | HLR	1 | 0 | 0	ACH(0) | BCI(0,1)
D5	HNE | HBS	0 | 0	LCF(0)
D6	HLR | HNE | LCH | ACH(1)	0 | 0 | 0 | 0	-
D7	HLR	0	N/A
D8	OCH | HLR	0 | 0	-
D9	CPP | ACH(5,6) | HBS	0 | 0 | 0	ACH(0) | LCN(2,3)

TABLE IV: Ground Truth (GT) and fault localisation outcome generated by UMLAUT (UM)

ID	GT	UM-match-GT	UM-output
M1	WCI(0)	0	HLR
M2	ACH(7)	1	ACH(7) | HLR
M3	HLR	0	-
C1	ACH(2)	0	-
C2	HNE	0	-
C3	WCI(2)	0	-
R1	RAW(0)	0	-
R2	ACH(2)	1	ACH(2)
R3	HLR	1	HLR
R4	LCH	0	-
R5	OCH	0	-
R6	WCI(0)	0	-
R7	ACH(2)	1	ACH(2)
D1	ACH(7)	0	-
D2	OCH | HNE | HBS	0	ACH(7)
D3	OCH | LCH | ACH(0,1) | HNE | HBS	0 | 0 | 0 | 0 | 0	-
D4	ACH(0,1) | LCH | HLR	1 | 0 | 1	ACH(0,1) | HLR
D5	HNE | HBS	0 | 0	-
D6	HLR | HNE | LCH | ACH(1)	0 | 0 | 0 | 0	-
D7	HLR	0	-
D8	OCH | HLR	0 | 0	-
D9	CPP | ACH(5,6) | HBS	0 | 0 | 0	ACH(0,2,4)

TABLE V: Number (#) of Ground Truth (GT) faults, number (#) of FL matches with GT, and AJ values for each FL tool. Avg. shows the average within artifical or real faults. T.A. shows the total average across faults.

ID	GT	DFD		DD		NL		UM	
	# faults	# matches	AJ	# matches	AJ	# matches	AJ	# matches	AJ
M1	1	0	0	0	0	**1**	1	0	0
M2	1	0	0	1	1	0	0	1	1
M3	1	1	1	0	0	0	0	0	0
C1	1	1	1	0	0	0	0	0	0
C2	1	0	0	0	0	0	0	0	0
C3	1	0	0	0	0	1	1	0	0
R1	1	0	0	0	0	0	0	0	0
R2	1	0	0	1	1	0	0	1	1
R3	1	0	0	0	0	0	0	1	1
R4	1	1	1	0	0	1	1	0	0
R5	1	1	1	0	0	0	0	0	0
R6	1	0	0	0	0	1	1	0	0
R7	1	0	0	1	1	0	0	1	1
Avg.	**1**	**0.308**	**0.308**	**0.231**	**0.231**	**0.308**	**0.308**	**0.308**	**0.308**
D1	1	1	1	0	0	0	0	0	0
D2	3	0	0	0	0	0	0	0	0
D3	5	1	0.2	0	0	2	0.4	0	0
D4	3	0	0	1	0.33	1	0.33	2	0.67
D5	2	0	0	0	0	0	0	0	0
D6	4	2	0.5	0	0	0	0	0	0
D7	1	0	0	0	0	0	0	0	0
D8	2	2	1	0	0	0	0	0	0
D9	3	0	0	0	0	0	0	0	0
Avg.	**2.667**	**0.667**	**0.300**	**0.111**	**0.037**	**0.333**	**0.081**	**0.222**	**0.074**
T.A.	**1.681**	**0.455**	**0.305**	**0.182**	**0.152**	**0.318**	**0.215**	**0.273**	**0.212**

TABLE VI: Execution time measurements (in seconds)

ID	DFD	DD	NL	UM
M1	605.30	6.65	7.63	37.62
M2	485.34	6.84	9.95	40.17
C1	316.10	7.34	10.02	163.08
C2	338.45	7.15	9.77	4.77
C3	321.42	7.03	10.02	135.75
R1	124.50	4.75	9.44	6.25
R2	115.12	4.05	9.59	5.89
R4	125.76	3.90	9.59	6.16
R5	126.13	3.58	7.7	5.10
R6	133.23	4.07	9.19	6.02
R7	158.34	3.95	8.99	6.07
D1	54.50	3.30	9.85	2.07
D2	451.67	20.13	9.95	18.98
D3	32.80	1.58	9.50	1.33
D4	797.46	11.66	6.87	324.57
D5	562.46	11.54	7.50	27.43
D6	19.6	1.32	6.88	0.39
D8	109.40	2.36	10.16	4.38
Avg.	**270.98**	**6.18**	**9.03**	**44.22**
M3	798.23	6.86	N/A	38.66
R3	116.34	4.05	N/A	6.00
D7	53.53	166.12	N/A	1.89
D9	N/A	N/A	9.35	57.07
T.A.	**278.37**	**13.73**	**9.05**	**40.89**

V. Threats to Validity

1) Construct: Threats to construct validity are due to the measurement of the effectiveness of the fault localisation tools and the interpretation of the tools' output. We use the simple count of the matches between fault localisation results and the ground truth, along with the AJ metric that calculates the percentage of the matches from the total number of faults in the ground truth.

2) Internal: Threats to the internal validity of the study lie in the selection of evaluated approaches. To the best of our knowledge, we considered all state-of-the-art techniques and adopted their publicly available implementations.

3) External: To address the threats to external validity, we carefully selected faults of both artificial and real nature, covering a set of diverse subjects for the evaluation of FL techniques. Nevertheless, replicating our study on additional subjects would be useful to corroborate our findings.

VI. Related Work

Nargiz ▶*repair paper*◀

VII. Conclusion

We evaluated 4 state-of-the-art techniques in DL fault localisation on a meticulously tailored set of real and artificial faulty models to asses the advances in the area. Our findings show that all of the evaluated approaches are able to locate a certain percentage of faults. However, all are quite far from the best possible results. DeepFD exhibited the highest effectiveness, followed by NeuralInt and UMLAUT. DeepDiagnosis exhibited relatively poor performance. On the positive side, all proposed techniques are stable across multiple runs and do not require substantial resources. According to our findings, future work in the area of DNN fault identification and localisation should focus on improving the accuracy of the proposed techniques. This suggests that it is highly unlikely that existing approaches for DNN tuning and repair might significantly benefit from the combination with the evaluated localisation tools.

Acknowledgement

Jinhan Kim and Shin Yoo have been supported by the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government (MSIT) (NRF-2018R1A5A1059921), NRF Grant (NRF-2020R1A2C1013629), Institute for Information & communications Technology Promotion grant funded by the Korean government (MSIT) (No.2021-0-01001), and Samsung Electronics (Grant No. IO201210-07969-01). This work was partially supported by the H2020 project PRECRIME, funded under the ERC Advanced Grant 2017 Program (ERC Grant Agreement n. 787703).

References

[1] M. Wardat, W. Le, and H. Rajan, "Deeplocalize: Fault localization for deep neural networks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2021, pp. 251–262. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSE43902.2021.00034

[2] M. Wardat, B. D. Cruz, W. Le, and H. Rajan, "DeepDiagnosis: automatically diagnosing faults and recommending actionable fixes in deep learning programs," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 561–572.

[3] J. Cao, M. Li, X. Chen, M. Wen, Y. Tian, B. Wu, and S.-C. Cheung, "Deepfd: Automated fault diagnosis and localization for deep learning programs," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 573–585. [Online]. Available: https://doi.org/10.1145/3510003.3510099

[4] A. Nikanjam, H. B. Braiek, M. M. Morovati, and F. Khomh, "Automatic fault detection for deep learning programs using graph transformations," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 1, pp. 1–27, 2021.

[5] E. Schoop, F. Huang, and B. Hartmann, "Umlaut: Debugging deep learning programs using program structure and model behavior," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–16.

[6] X. Zhang, J. Zhai, S. Ma, and C. Shen, "Autotrainer: An automatic dnn training problem detection and repair system," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 359–371.

[7] W. Baker, M. O'Connor, S. R. Shahamiri, and V. Terragni, "Detect, fix, and verify tensorflow api misuses," in *International Conference on Software Analysis, Evolution and Reengineering*, 2022, pp. 1–5.

[8] "Keras," Available at https://keras.io.

[9] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, and L. Zhang, "An empirical study on tensorflow program bugs," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2018. New York, NY, USA: ACM, 2018, pp. 129–140. [Online]. Available: http://doi.acm.org/10.1145/3213846.3213866

[10] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992. [Online]. Available: http://www.jstor.org/stable/2685209

[11] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[12] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[13] J. Cao, M. Li, X. Chen, M. Wen, Y. Tian, B. Wu, and S.-C. Cheung, "Replication package of deepfd," https://github.com/ArabelaTso/DeepFD, 2021.

[14] E. Schoop, F. Huang, and B. Hartmann, "Replication package of umlaut," Available at https://github.com/BerkeleyHCI/umlaut, 2021.

[15] A. Nikanjam, H. B. Braiek, M. M. Morovati, and F. Khomh, "Replication package of Neuralint," Available at https://github.com/neuralint/neuralint.

[16] M. Wardat, B. D. Cruz, W. Le, and H. Rajan, "Replication package of DeepDiagnosis," Available at https://github.com/deepdiagnosis/icse2022, 2021.

[17] G. Jahangirova and P. Tonella, "An empirical evaluation of mutation operators for deep learning systems," in *IEEE International Conference on Software Testing, Verification and Validation*, ser. ICST'20. IEEE, 2020, p. 12 pages. [Online]. Available: https://doi.org/10.1109/ICST46399.2020.00018

[18] "Empirical Comparison of Fault Localisation Techniques for DNNs (replication package)," Available at https://github.com/dlfaults/dnn-auto-fl-empirical-assesment, 2023.