A Comprehensive Study on Deep Learning Bug Characteristics

Md Johirul Islam
mislam@iastate.edu
Iowa State University
Ames, IA, USA

Giang Nguyen
gnguyen@iastate.edu
Iowa State University
Ames, IA, USA

Rangeet Pan
rangeet@iastate.edu
Iowa State University
Ames, IA, USA

Hridesh Rajan
hridesh@iastate.edu
Iowa State University
Ames, IA, USA

ABSTRACT

Deep learning has gained substantial popularity in recent years. Developers mainly rely on libraries and tools to add deep learning capabilities to their software. What kinds of bugs are frequently found in such software? What are the root causes of such bugs? What impacts do such bugs have? Which stages of deep learning pipeline are more bug prone? Are there any antipatterns? Understanding such characteristics of bugs in deep learning software has the potential to foster the development of better deep learning platforms, debugging mechanisms, development practices, and encourage the development of analysis and verification frameworks. Therefore, we study 2716 high-quality posts from *Stack Overflow* and 500 bug fix commits from *Github* about five popular deep learning libraries *Caffe*, *Keras*, *Tensorflow*, *Theano*, and *Torch* to understand the types of bugs, root causes of bugs, impacts of bugs, bug-prone stage of deep learning pipeline as well as whether there are some common antipatterns found in this buggy software. The key findings of our study include: data bug and logic bug are the most severe bug types in deep learning software appearing more than 48% of the times, major root causes of these bugs are Incorrect Model Parameter (IPS) and Structural Inefficiency (SI) showing up more than 43% of the times. We have also found that the bugs in the usage of deep learning libraries have some common antipatterns.

CCS CONCEPTS

• **Software and its engineering** → **Software defect analysis**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Deep learning software, Q&A forums, Bugs, Deep learning bugs, Empirical Study of Bugs

This work was supported in part by US NSF under grants CNS-15-13263, and CCF-15-18897. All opinions are of the authors and do not reflect the view of sponsors.

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-5572-8/19/08…$15.00
https://doi.org/10.1145/3338906.3338955

ACM Reference Format:
Md Johirul Islam, Giang Nguyen, Rangeet Pan, and Hridesh Rajan. 2019. A Comprehensive Study on Deep Learning Bug Characteristics. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3338906.3338955

1 INTRODUCTION

A class of machine learning algorithms known as *deep learning* has received much attention in both academia and industry. These algorithms use multiple layers of transformation functions to convert input to output, each layer learning successively higher-level of abstractions in the data. The availability of large datasets has made it feasible to train (adjust the weights of) these multiple layers. While the jury is still out on the impact of deep learning on overall understanding of software's behavior, a significant uptick in its usage and applications in wide ranging areas combine to warrant research on software engineering practices in the presence of deep learning. This work focuses on the characteristics of bugs in software that makes use of deep learning libraries.

Previous work on this topic generally fall under two categories: those that have studied bugs in the implementation of machine learning libraries themselves, and those that have studied bugs in the usage of a specific deep learning library. A key work in the first category is Thung *et al.* [24] who studied bugs in the implementation of three machine learning systems Mahout, Lucene, and OpenNLP. In the second category, Zhang *et al.* [27] have studied bugs in software that make use of the *Tensorflow* library. While both categories of approaches have advanced our knowledge of ML systems, we do not yet have a comprehensive understanding of bugs encountered by the class of deep learning libraries.

This work presents a comprehensive study of bugs in the usage of deep learning libraries. We have selected top five popular deep learning libraries *Caffe* [15], *Keras* [7], *Tensorflow* [1], *Theano* [23], and *Torch* [8] based on the user counts from developers Q&A forum *Stack Overflow*. While each of these libraries are for deep learning they have different design goals. For example, *Tensorflow* focuses on providing low-level, highly configurable facilities whereas *Keras* aims to provide high-level abstractions hiding the low-level details. *Theano* and *Torch* are focused on easing the use of GPU computing to make deep learning more scalable. Thus, studying them simultaneously allows us to compare and contrast their design goals *vis-à-vis* bugs in their usage.

Table 1: Summary of the dataset used in the Study

Library	*Stack Overflow*		*Github*	
	# Posts	# Bugs	# Commits	# Bugs
Caffe	183	35	100	26
Keras	567	162	100	348
Tensorflow	1558	166	100	100
Theano	231	27	100	35
Torch	177	25	100	46
Total	2716	415	500	555

We have used two sources of data in our study: posts about these libraries on *Stack Overflow* and also *Github* bug fix commits. The first dataset gives us insights into bugs that developers encounter when building software with deep learning libraries. A number of these bugs would, hopefully, be fixed based on the discussion in Q&A forum. The second dataset gives us insights into bugs that were found and fixed in open source software. Our study focuses on following research questions and compares our findings across the five subject libraries.

RQ1: (Bug Type) What type of bugs are more frequent?
RQ2: (Root cause) What are the root causes of bugs?
RQ3: (Bug Impact) What are the frequent impacts of bugs?
RQ4: (Bug prone stages) Which deep learning pipeline stages are more vulnerable to bugs?
RQ5: (Commonality) Do the bugs follow a common pattern?
RQ6: (Bug evolution) How did the bug pattern change over time?

Findings-at-a-glance. Our study show that most of the deep learning bugs are *Data Bugs* and *Logic Bugs* [5], the primary root causes that cause the bugs are Structural Inefficiency (SI) and Incorrect Model Parameter (IPS) [27], most of the bugs happen in the Data Preparation stage of the deep learning pipeline. Our study also confirms some of the findings of *Tensorflow* conducted by Zhang *et al.* [27]. We have also studied some antipatterns in the bugs to find whether there is any commonality in the code patterns that results in bugs. Our findings show that there is strong correlation among the distribution of bugs as well as in the antipatterns. Finally, we conclude with a discussion on our findings suggesting immediate actions and future research directions based on these findings.

2 METHODOLOGY

2.1 Data Collection

We have used two different data sources for studying the bugs in deep learning software: *Stack Overflow* posts and *Github* bug fix commits. A summary of these datasets is shown in Table 1.

2.1.1 Stack Overflow Data Collection. To study bugs in deep learning software, we have collected data from *Stack Overflow*, a well-known Q&A site for developers to discuss software development problems. The data collection process consists of two steps.

In the first step, we select candidate posts discussing deep learning libraries. We focus on five deep learning libraries: *Caffe*, *Keras*, *Tensorflow*, *Theano*, and *Torch*. These are the five most discussed deep learning libraries on *Stack Overflow*. We did that by searching for posts tagged with *Caffe*, *Keras*, *Tensorflow*, *Theano*, and *Torch*. When posts are about specific libraries, they are more likely to talk about bugs in using deep learning libraries. Using these criteria, we selected all posts about these five libraries. We further filtered the posts that did not contain any source code because posts about bugs usually contain code snippets. Moreover, we reduced the number of posts by selecting the posts whose scores, computed as the difference between the number of its upvotes and the number of its downvotes, were greater than 5 to focus on the high-quality posts and keep the manual effort manageable. After this step, in total, we retrieved 183, 567, 1558, 231, and 177 posts for *Caffe*, *Keras*, *Tensorflow*, *Theano*, and *Torch*, respectively for further study.

In the second step, we manually read these candidates to identify the ones about bugs. After that, the second and the third authors manually reviewed the candidates. For each post, we read the question and all answers focusing on the best-accepted one. If the best-accepted answer was to fix the usages of the deep learning API(s) in the question, we considered that post as talking about deep learning bugs. After this step, we found 35, 162, 166, 27, and 25 bugs for *Caffe*, *Keras*, *Tensorflow*, *Theano*, and *Torch* respectively.

2.1.2 Github Data Collection. We mine the *Github* commits to study the change in the commits and to check and confirm the bug patterns that we studied from *Stack Overflow*. The data collection process consists of two steps.

First, we collect all the repositories of *Caffe*, *Keras*, *Tensorflow*, *Theano*, and *Torch*. For collecting the repositories that use these libraries, we first find the repositories that contain the keywords related to the libraries. After that, we mine all the commits whose title contains the word `"fix"`. Then, we check the import statements in the program to identify if those repositories truely use deep learning libraries. Next, we randomly select 100 commits for each library from mined commits and classify them.

Secondly, we use the same process that we used for *Stack Overflow*. Specifically, the second and the third authors manually studied the 500 commits and separately label them. After that, these two authors compare their results to fix the conflict in the labeling process. We study each line of change in the commits. Note that some commits may have more than one bugs and some commit may not have bug. Overall, we got 26, 348, 100, 35, and, 46 bugs for the commits of *Caffe*, *Keras*, *Tensorflow*, *Theano*, and *Torch*, respectively.

2.2 Classification

In our classification, we focus on three criteria which are bug types, root causes and effects of bug. The classification scheme used for labeling of the bugs in each of these three criteria discussed in §2.4, §2.5, and §2.6. We have also classified the bugs into different deep learning stages [26].

To label the bug types we followed the classification from an already existing well vetted taxonomy [5] and appended on top of that. The added types were based on the data that we studied following an open coding scheme.

The bugs may have different root causes and effects. A supervised pilot study and open coding schemes were used to identify the effects that are possible through these bugs. We have adapted the classification scheme of root causes and bug effects from [27] and added on top of that as found from the study of the posts. One of the authors with expertise in these libraries studied the posts initially to come up with the classification scheme for bug types, root causes and effects. We followed the open coding scheme and a pilot study was conducted to get agreement on the classification.

We also classified the bugs into different stages of the pipeline to understand which stages are more vulnerable to bugs. Deep learning process can be divided into seven stage pipeline [26]. The stages are data collection, data preparation, choice of model, training, evaluation, hyper parameter tuning and prediction. Among the seven stages, the first one is not related to software development. The other stages are related to software development, and are supported by the deep learning libraries through their APIs. We use these stages to label the bugs into different stages.

2.3 Labeling the Bugs

Once we have all the classification criteria, we used those criteria to label the posts. The second and the third authors independently studied the posts. We measured the inter rater aggrement among the labellers using Cohen's Kappa coefficient [25] when 5%, 10%, 20% , 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100% of the posts were labeled. After 5% labeling, the Cohen's Kappa coefficient was close to 0. Then we conducted a training session among the raters to clarify the labeling and what they mean. After the training session, we conducted another pilot study at 10% including the first 5%. This time the Cohen's Kappa coefficient was 82%. We again discussed the results and find out the reasons for major disagreements. We then discussed those cases further through examples and continued labeling. The Cohen's Kappa coefficient was more than 90% in subsequent pilot studies.

The labeling effort was continuously being monitored with the help of Kappa coefficient to understand the agreement. We conducted reconciling efforts ideally at every 10% interval of the labeling. The posts where there was disagreement between the raters were further discussed in the presence of a supervisor. After discussion and arguments a common label was given. Finally, all the bugs were given a common label.

2.4 Types of Bugs in Deep Learning Software

Developers often encounter different types of bugs while trying to write deep learning software. To understand those bugs and their root causes, we have classified them into different categories. The classification is inspired from [5] and adapted based on all the *Stack Overflow* posts that we have analyzed.

2.4.1 API Bug. This group of bugs is caused by deep learning APIs. Generally, when a developer uses a deep learning API, different bugs associated with that API are inherited automatically without the knowledge of the user. The prime causes for triggering of deep learning API bugs can be because of the change of API definition with different versions, lack of inter-API compatibility and sometimes wrong or confusing documentation.

2.4.2 Coding Bug. These kind of bugs originate due to programming mistakes. This in turn, introduces other types of bugs in the software which lead to either runtime error or incorrect results. A big percentage of the deep learning bugs that we have checked arises from syntactic mistakes that cannot be fixed by changing only some lines of code. This type of bugs are not identified by the programming language compiler resulting in wrong output.

2.4.3 Data Bug. This bug may arise if an input to the deep learning software is not properly formatted or cleaned well before supplying it to the deep learning model. This type of bug occurs before data is fed to the deep learning model. It is not because of the wrong deep learning model, rather it is purely based on the type and structure of training or test data. Similar to coding bugs, data bugs are usually flagged by the compiler, but in some scenarios it can pass unchecked through the compilation process and generate erroneous results.

2.4.4 Structural Bug (SB). A vast majority of the deep learning bugs are occurring due to incorrect definitions of the deep learning model's structure. These include mismatch of dimensions between different layers of deep learning models, the presence of anomaly between the training and test datasets, use of incorrect data structures in implementing a particular function, etc. These type of bugs can be further classified into four subcategories.

Control and Sequence Bug. This subclass of the bug is caused by the wrong structure of control flow. In many scenarios, due to wrong if-else or loop guarding condition, the model does not perform as expected. This type of bug either leads to a crash when a part of deep learning model does not work or, leads to incorrect functionality due to mishandling of data through the layers.

Data Flow Bug. The main difference between the Data Flow Bug and the Data Bug is the place of origin. If a bug occurs due to the type or shape mismatch of input data after it has been fed to the deep learning model, we label it as Data Flow Bug. It includes those scenarios where model layers are not consistent because of different data shape used in consecutive layers. To fix these bugs, developers need to modify the model or reshape the data.

Initialization Bug. In deep learning, Initialization Bug means the parameters or the functions are not initialized properly before they are used. This type of bugs would not necessarily produce runtime error but it will simply make the model perform worse. Here, the definition of functions includes both user-defined and API defined. We also categorize a bug into this category when the API has not been initialized properly.

Logic Bug. In deep learning, the logical understanding of each stage of the pipeline is an integral part of the coding process. With an incorrect logical structure of the deep learning model, the output of a program may result in either a runtime error or a faulty outcome. These bugs are often generated in the absence of proper guarding conditions in the code.

Processing Bug. One of the most important decisions in the deep learning model structure is to choose the correct algorithm for the learning process. In fact, different deep learning algorithms can lead to different performance and output [14]. Also, to make different layers be compatible with each other, the data types of each layer need to follow a contract between them. Processing Bugs happen due to the violation of these contracts.

2.4.5 Non Model Structural Bug (NMSB). Unlike SB, NMSB occur outside the modeling stage. In other words, this bug can happen in any deep learning stage except the modeling stage such as the training stage or the prediction stage. NMSB has similar subcategories as SB. The subcategories of NMSB are Control and Sequence Bug, Logic Bug, Processing Bug, and Initialization Bug. We do not define Non Model Structural Data Flow Bug like Structural Data

Flow Bug because Data Bug already covers the meaning of Non Model Structural Data Flow Bug.

Control and Sequence Bug. This subclass is similar to Control and Sequence Bug in SB. The bug is caused by an incorrect structure of control flow like wrong if-else condition; however, this kind of bug happens outside modeling stage.

Initialization Bug. This subclass is similar to Initialization Bug in SB. The bug is caused by incorrect initialization of a parameter or a function prior to its use.

Logic Bug. This subclass is similar to Logic Bug in SB. The bug is caused by misunderstanding the behavior of case statements and logical operators.

Processing Bug. This subclass is similar to Processing Bug in SB. The bug is caused by an incorrect choice of algorithm.

2.5 Classification of Root Causes of Bugs

2.5.1 Absence of Inter API Compatibility. The main reason for these bugs is the inconsistency of the combination of two different kinds of libraries. For example, a user cannot directly use `Numpy` function in *Keras* because neither *Tensorflow* backend nor *Theano* backend of *Keras* has the implementation of `Numpy` functions.

2.5.2 Absence of Type Checking. This kind of bugs involves a type mismatch problem when calling API methods. These bugs are usually mistakes related to the use of wrong type of parameters in an API.

2.5.3 API Change. The reason for these bugs is the release of the new versions of deep learning libraries with incompatible APIs. In other words, the bug happens when the new API version is not backward compatible with its previous version. For example, a user updates the new version of a deep learning library which has new API syntax; however, the user does not modify his/her code to fit with the new version, which leads to the API change bug.

2.5.4 API Misuse. This kind of bugs often arises when users use a deep learning API without fully understanding. Missing conditions can be one kind of API misuse, and this bug occurs when a usage does not follow the API usage constraints to ensure certain required conditions. Crash is the main effect of these bugs.

2.5.5 Confusion with Computation Model. These bugs happen when a user gets confused about the function of deep learning API, which leads to the misuse of the computation model assumed by the deep learning library. For instance, a user gets confused between the graph construction and the evaluation phase.

2.5.6 Incorrect Model Parameter or Structure (IPS). IPS causes problems with constructing the deep learning model, e.g. incorrect model structures or using inappropriate parameters. IPS is a common bug in the deep learning software because of both the lack of deep learning knowledge among the users and the incomprehensibilty of deep learning models. This kind of bugs causes the functional incorrectness; thus, the effect of this bug is a crash.

2.5.7 Others. These bugs are not related to deep learninng software. In other words, these bugs are mostly related to mistakes in the development process like incorrect syntax.

2.5.8 Structure Inefficiency (SI). SI causes problems related to modeling stage in deep learning software like IPS; however, SI leads to bad performance of the deep learning software while IPS leads to a crash.

2.5.9 Unaligned Tensor (UT). These bugs often occur in the computation graph construction phase. When a user builds the computation graph in deep learning process, they have to provide correct input data that satisfies input specifications of the deep learning API; however, many users do not know the API specifications, or they misunderstand API signature leading to UT bugs.

2.5.10 Wrong Documentation. Incorrect information in library documentation leads to these bugs. Deep learning library users may face this kind of bugs when they read an incorrect definition or an incorrect usage of a deep learning API from documentation.

2.6 Classification of Effects of Bugs

2.6.1 Bad Performance. Bad performance or poor performance is one of common kind of effect in deep learning software. Furthermore, the major root causes of this effect are SI or CCM that are related to the model construction. Even though developers can use deep learning libraries correctly, they still face model construction problems because APIs in these libraries are abstract.

2.6.2 Crash. Crash is the most frequent effect in deep learning. In fact, any kind of bugs can lead to Crash. A symptom of crash is that the software stops running and prints out an error message.

2.6.3 Data Corruption. This bug happens when the data is corrupted as it flows through the network. This effect is a consequence of misunderstanding the deep learning algorithms or APIs. When Data Corruption occurs, a user will receive unexpected outputs.

2.6.4 Hang. Hang effect is caused when a deep learning software ceases to respond to inputs. Either slow hardware or inappropriate deep learning algorithm can lead to Hang. A symptom of Hang is that the software runs for a long period of time without providing the desired output.

2.6.5 Incorrect Functionality. This effect occurs when the software behaves in an unexpeced way without any runtime or compile-time error/warning. This includes the incorrect output format, model layers not working desirably, etc.

2.6.6 Memory Out of Bound. Deep learning software often halts due to unavailability of the memory resources. This can be caused by, either the wrong model structure or, not having enough computing resources to train a particular model.

3 FREQUENT BUG TYPES

In this section, we explore the answer to **RQ1** through a statistical analysis of the labeled data. The normalized distribution of bug types in *Stack Overflow* data is shown in Figure 1. The distribution of bugs shown in Figure 1 and the *Stack Overflow* and *Github* data in Table 2 shows the presence of different kinds of bugs in both *Stack Overflow* and *Github* for the deep learning libraries we have studied. We present some of the key findings related to bug types in the following subsections.

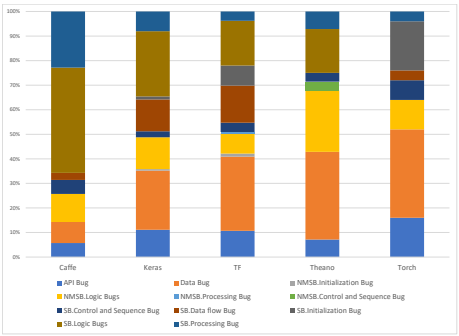


Figure 1: Distribution of Bug Types in *Stack Overflow*

3.1 Data Bugs

Finding 1: Data Bugs appear more than 26% of the times

From Figure 1 we see that among the bug types the Data Bugs frequently appear (26%) in all the libraries. In the studied *Stack Overflow* data, we have seen 30% of the posts in *Tensorflow*, 24% posts in *Keras*, 36% posts in *Torch*, 35% posts in *Theano*, and 9% posts in *Caffe* have Data Bugs. Data bugs mostly appear due to the absence of data pre-processing facilities like feature engineering, data validation, data shuffling, etc. For example, a developer is trying to read some image files using the following method[1].

```
def _read32(bytestream):
  dt = numpy.dtype(numpy.uint32).newbyteorder('>')
  return numpy.frombuffer(bytestream.read(4), dtype=dt)
```

The developer eventually got stuck with the following error while trying to train the model using the data returned by the previous library call.

```
TypeError: only integer scalar arrays can be converted to a scalar
      index
```

An expert suggested an answer to change the last return statement with the following, which solved the problem and was accepted.

```
return numpy.frombuffer(bytestream.read(4), dtype=dt)[0]
```

The bug is hard to fix by just looking at the error message. It is difficult to identify the exact reason of bug which led the developer to post a question on *Stack Overflow* and the question was upvoted by other fellow developers as a qualified post.

The large percentage of Data Bugs indicate data pre-processing related difficulties are quite common in deep learning software. These bugs could be addressed by development and refinement of data verification tools. Support for modern abstract data types like *DataFrame* and the properties of the model in data verification tool would help the deep learning community.

3.2 Structural Logic Bugs

Finding 2: *Caffe* has 43% Structural Logic Bugs

The second major bug type is Structural Logic Bug in *Stack Overflow* that was expected from our initial hypothesis based on a pilot study. *Caffe* has more Structural Logic Bugs in *Stack Overflow* compared to other libraries. Other libraries also have significant portion of Structural Logic Bugs ranging from 0% - 27%.

[1] https://tinyurl.com/y3v9o7pu

Table 2: Statistics of Bug Types in *Stack Overflow* and *Github*

	Caffe		Keras		TF		Theano		Torch		P value
	SO	GitHub	SO	GitHub	SO	GitHub	SO	GitHub	SO	GitHub	
API Bug	6%	0%	11%	57%	11%	72%	7%	3%	16%	2%	0.3207
Data Bug	9%	49%	24%	8%	30%	0%	35%	17%	36%	15%	0.3901
NMSB.Control and Sequence Bug	0%	8%	0%	0%	0%	0%	4%	0%	0%	7%	0.3056
NMSB.Initialization Bug	0%	0%	1%	0%	1%	0%	0%	3%	0%	0%	0.7655
NMSB.Logic Bugs	11%	0%	13%	2%	8%	0%	25%	6%	12%	7%	0.0109
NMSB.Processing Bug	0%	0%	0%	0%	1%	0%	0%	3%	0%	7%	0.2323
SB.Control and Sequence Bug	6%	12%	2%	0%	4%	0%	4%	3%	8 %	9%	1.0000
SB.Data flow Bug	3%	8%	13%	26%	15%	0%	0%	14%	4%	16%	0.2873
SB.Initialization Bug	0%	0%	1%	0%	8%	1%	0%	23%	20%	11%	0.8446
SB.Logic Bugs	42%	15%	27%	3%	18%	23%	18%	14%	0%	13%	0.3442
SB.Processing Bug	23%	8%	8%	4%	4%	4%	7%	14%	4%	13%	0.8535

3.3 API Bugs

Finding 3: *Torch*, *Keras*, *Tensorflow* have 16%, 11% and 11% API bugs respectively

In deep learning libraries API changes sometimes break the entire production code. The implicit dependencies between libraries cause problems when one library has some major changes. For example, when Numpy is updated *Tensorflow*, *Keras* software may fail. *Keras* often uses *Tensorflow* or *Theano* as backend and hence update of *Tensorflow* or *Theano* can cause the software developed using *Keras* to crash. API bugs arise more often in *Keras* and *Tensorflow* as shown in Figure 1. More than 81% of the API bugs are from *Keras* and *Tensorflow*. An example of such bug is shown in the code snippet below. The bug in the code below arises because the keyword names in the API signature of *Keras* has changed.

```
model.fit(tX, tY, epochs=100, batch_size=1, verbose=2)
```

The developer will get the error because epochs keyword does not exist in version 2+ of *Keras*.

```
model.fit(tX, tY, batch_size=1, verbose=2, epochs = 100) File
"keras/models.py", line 612, in fit str(kwargs))
Exception: Received unknown keyword arguments: {'epochs': 100}
```

To fix this error, the developer needs to change the keyword parameter from epochs to nb_epoch.

```
model.fit(tX, tY, nb_epoch=100, batch_size=1, verbose=2)
```

3.4 Bugs in *Github* Projects

We have also analyzed the distributions of bugs in some *Github* bug fix commits. The distribution of bugs across different libraries in *Github* data is shown in Table 2. We computed the P value using t-test where one distribution is bug type in *Github* for all the libraries and the other distribution is bug type for all the libraries in *Stack Overflow*.

Finding 4: All the bug types have a similar pattern in *Github* and *Stack Overflow* for all the libraries

We analyze the *Stack Overflow* and *Github* result using the t-test to find whether the distributions differ significantly. We use 95% significant level to find the difference beween *Stack Overflow* and *Github* results for each of the bug type In our analysis the null hypothesis is: H_0***: The distributions are same***. If we fail to reject

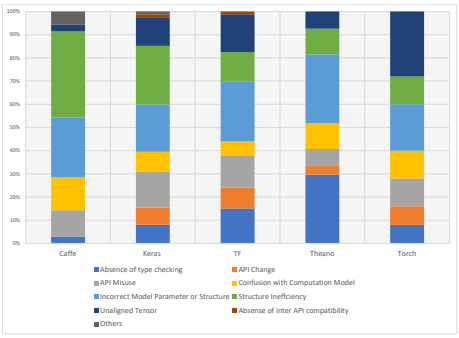


Figure 2: Stack Overflow Root Cause Classification

this null hypothesis using the t-test then we can say the distributions follow the same pattern in both *Stack Overflow* and *Github* data.

We see that for all the bug types except Non Model Structural Logic Bug the P value is greater than 5% indicating they have a similar pattern as we fail to reject the null hypothesis.

4 ROOT CAUSE

In this section, we present the analyses and findings to answer **RQ2** identifying major root causes of bugs in deep learning software. The normalized distribution of root causes in *Stack Overflow* code snippets is shown in Figure 2. The data in Table 3 shows the presence of different categories of root causes in both *Stack Overflow* and *Github* for the deep learning libraries and presents P value showing the similarity of distributions using t-test. We discuss the significant root causes in the following subsections.

4.1 Incorrect Model Parameter (IPS)

Finding 5: IPS is the most common root cause resulting in average 24% of the bugs across the libraries

IPS results in bugs that causes the program to crash at runtime and the execution does not succeed. In *Tensorflow* and *Theano* IPS leads other root causes in causing bugs having 26% and 26% of the total share of root causes, respectively.

4.2 Structural Inefficiency (SI)

Finding 6: *Keras*, *Caffe* have 25% and 37% bugs that arise from SI

SI bugs do not cause the program to crash. These bugs often yield suboptimal performance of the deep learning model. These bugs have more relation to QoS or non-functional requirements. For example, a programmer is trying to train a model to recognize handwritten digits but the accuracy does not improve and stays constant from epochs 2 - 10.[2]

```
Epoch 1/10
2394/2394 [==============================] - 0s - loss: 0.6898 -
    acc: 0.5455 - val_loss: 0.6835 - val_acc: 0.5716
Epoch 2/10
2394/2394 [==============================] - 0s - loss: 0.6879 -
    acc: 0.5522 - val_loss: 0.6901 - val_acc: 0.5716
.........
```

[2] https://stackoverflow.com/questions/37213388/keras-accuracy-does-not-change

Table 3: Statistics of the Root Causes of Bugs

	Caffe		Keras		TF		Theano		Torch		P value
	SO	GitHub	SO	GitHub	SO	GitHub	SO	GitHub	SO	GitHub	
Absense of inter API compatibility	0%	0%	1%	0%	1%	0%	0%	0%	0%	0%	0.1411
Absence of type checking	3%	12%	8%	3%	15%	15%	30%	20%	8%	13%	0.9717
API Change	0%	0%	7%	51%	9%	58%	4%	0%	8%	2%	0.2485
API Misuse	11%	0%	15%	4%	14%	0%	7%	3%	12%	2%	0.0003
Confusion with Computation Model	14%	28%	9%	1%	6%	10%	11%	3%	12%	4%	0.7839
Incorrect Model Parameter or Structure	26%	31%	21%	30%	26%	16%	30%	14%	20%	19%	0.5040
Others	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0.3466
Structure Ineffciency	37%	12%	26%	5%	13%	1%	11%	26%	12%	38%	0.7170
Unaligned Tensor	3%	19%	12%	5%	16%	0%	7%	34%	28%	20%	0.7541
Wrong Documentation	6%	0%	1%	1%	0%	0%	0%	0%	0%	0%	0.3402

```
Epoch 10/10
2394/2394 [==============================] - 0s - loss: 0.6877 -
    acc: 0.5522 - val_loss: 0.6849 - val_acc: 0.5716
1027/1027 [==============================] - 0s
```

The problem that was pointed out by an expert, which solved the performance degradation bug is following:

```
#In summary, replace this line:
model.compile(loss = "categorical_crossentropy", optimizer = "adam
    ")
#with this:
from keras.optimizers import SGD
opt = SGD(lr=0.01)
model.compile(loss = "categorical_crossentropy", optimizer = opt)
```

The answer suggested to change `optimizer` for enhancing the performance.

4.3 Unaligned Tensor (UT)

Finding 7: *Torch* has 28% of the bugs due to UT

In deep learning, tensor dimensions are important for successful construction of the model. *Tensorflow*, *Keras*, *Torch*, *Theano*, *Caffe* have 16%, 12%, 28%, 7% and 3% of bugs due to UT respectively. In *Torch* UT is the leading root cause of bugs.

4.4 Absence of Type Checking

Finding 8: *Theano* has 30% of the bugs due to the absence of type checking

Most of the deep learning libraries are written in Python. Due to the dynamic nature of Python, the problem of the absence of type checking is felt strongly in these libraries. The absence of type checking leads to 30% of the bugs in *Theano*, 8% of the bugs in *Keras* and 15% of the bugs in *Tensorflow*.

4.5 API Change

Finding 9: *Tensorflow* and *Keras* have 9% and 7% bugs due to API change

In deep learning libraries, API change tends to have a drastic effect. These libraries are interdependent. So, API change in one library breaks other libraries.

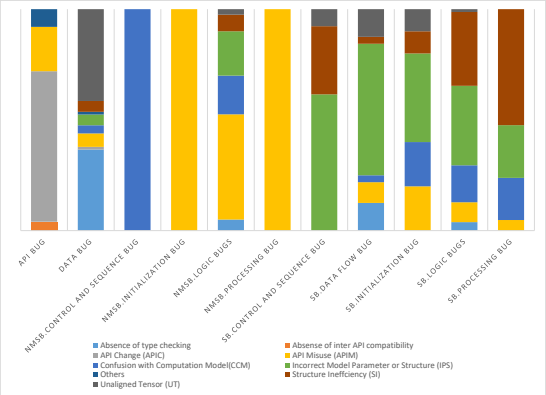


Figure 3: Relation between Root Causes and Types of Bugs

4.6 Root Causes in *Github* Data

Finding 10: Except API Misuse all other root causes have similar patterns in both *Github* and *Stack Overflow* root causes of bugs

We computed the P value at 95% significant level for both the *Stack Overflow* and *Github* data for all the root causes in the five libraries. We see that, P value for API Misuse root cause is much less than 5% indicating API Misuse in *Stack Overflow* and *Github* has different distribution compared to other root causes as we reject the null hypothesis. The other root causes are similar for both *Stack Overflow* and *Github* data as their P value is greater than 5%.

4.7 Relation of Root Cause with Bug Type

Finding 11: SI contributes 3% - 53% and IPS contributes 24% - 62% of the bugs related to model

We have seen from Figure 3 that most of the non model related bugs are caused by API Misuse (6% - 100%). Non Model Structural Initialization Bugs and Non Model Structural Processing Bugs are caused by API Misuse in 100% of the time in our studied data. Interestingly in API Bug API Change plays the vital role (68%) compared to API Misuse (20%); however, the model related bugs are more vulnerable to IPS and SI root causes. We see from Figure 3 that Structural Control and Sequence Bug, Structaral Data Flow Bug, Structural Initialization Bug, Structural Logic Bug, Structural Processing Bug which are related to model are caused by SI 31%, 3%, 10%, 33% and 53% of the times respectively and caused by IPS 62%, 59%, 40%, 36%, 24% of the times respectively.

5 IMPACTS FROM BUGS

In this section, we explore the answer to **RQ3** to understand the major effects of bugs in deep learning software. The normalized distribution of effects of *Stack Overflow* is shown in Fig. 4. The data in Table 4 shows the presence of different kinds of effects in both *Stack Overflow* and *Github* for the deep learning libraries. We discuss some of the major effects of bugs in deep learning software in the rest of this section.

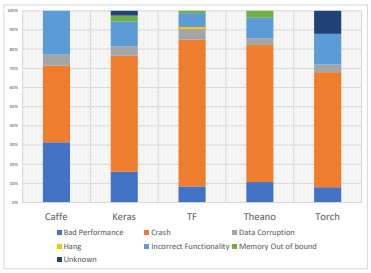


Figure 4: Distribution of Bug Effects in *Stack Overflow*

5.1 Crash

Finding 12: More than 66% of the bugs cause crash.

Our analysis reveals that, the most severe effect of bugs is Crash. In deep learning, the bugs mostly cause total failure of the program. In all the libraries Crash is the top impact ranging from 40% - 77% as shown in Figure 4.

5.2 Bad Performance

Finding 13: In *Caffe*, *Keras*, *Tensorflow*, *Theano*, *Torch* 31%, 16%, 8%, 11%, and 8% bugs lead to bad performance respectively

Bad performance is often a concern for deep learning software developers. Even though the model trains successfully, during the evaluation or prediction phase the model may give very poor accuracy in classifying the target classes.

For example, in the following code snippet the user had low accuracy after traning because of the use of incorrect value of parameter `nb_words` that is the value of the maximum size of the vocabulary of the dataset. The developer should use `nb_words + 1` instead of `nb_words` as answered by an expert [3]. If the developer uses `nb_words` instead of `nb_words + 1`, the model will not train on the last word, which can lead to the bad performance effect.

```
embedded = Embedding(nb_words, output_dim=hidden, input_length=
    maxlen)(sequence)
```

5.3 Incorrect Functionality

Finding 14: 12% of the bugs cause Incorrect Functionality

Incorrect functionality happens when the runtime behavior of the software leads to some unexplainable outcome that is not expected from the logical organization of the model or from previous experience of the developer.

For example, in the following code snippet the user wants to convert the image to a $28 * 28$ `Numpy` array; however, the output is a black image.[4]

```
with tf.Session() as sess:
    first_image = mnist.train.images[0]
    first_image = np.array(first_image, dtype='uint8')
    pixels = first_image.reshape((28, 28))
    plt.imshow(pixels, cmap='gray')
```

[3] https://stackoverflow.com/questions/37817588/masking-for-keras-blstm
[4] https://stackoverflow.com/questions/42353676/display-mnist-image-using-matplotlib

Table 4: Effects of Bugs in *Stack Overflow* and *Github*

	Caffe		Keras		TF		Theano		Torch		P value
	SO	GitHub	SO	GitHub	SO	GitHub	SO	GitHub	SO	GitHub	
Bad Performance	31%	19%	16%	14%	8%	8%	11%	6%	8%	24%	0.9152
Crash	40%	69%	61%	86%	77%	92%	70%	20%	60%	16%	0.7812
Data Corruption	6%	4%	5%	0%	6%	0%	4%	6%	4%	16%	0.948
Hang	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0.3466
Incorrect Functionality	23%	8%	13%	0%	7%	0%	11%	59%	16%	42%	0.5418
Memory Out of bound	0%	0%	3%	0%	1%	0%	4%	0%	0%	0%	0.0844
Unknown	0%	0%	2%	0%	0%	0%	0%	9%	12%	2%	0.8419

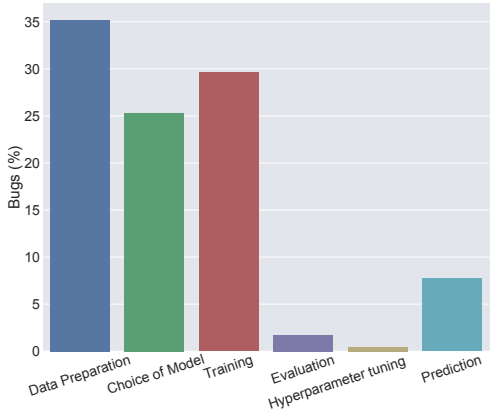


Figure 5: Bugs across stages of the Deep Learning pipeline

The user got incorrect output because of casting a float array to uint8, which will convert all the pixels to 0 if they are less than 1. To fix the problem, the user can multiply the array with 255 as suggested by an answer. *Theano* has a higher percentage of posts about incorrect functionality problems compared to bad performance.

5.4 Effects of Bugs in *Github*

Finding 15: For all the libraries the P value for *Stack Overflow* and *Github* bug effects reject the null hypothesis to confirm that the bugs have similar effects from *Stack Overflow* as well as *Github* bugs

The P value is shown in Table 4 shows that Bad Performance in *Stack Overflow* and *Github* have 79% of P value which indicates that they are very similar. Crash has P value of 50% in *Stack Overflow* and *Github* indicating they also can not reject the null hypothesis with strong confidence. None of the impacts reject the null hypothesis at 95% significance level.

6 DIFFICULT DEEP LEARNING STAGES

In this section, we answer **RQ4** by studying the bugs arising at the different stage of the deep learning pipeline. We use the categorization of the posts about deep learning stages to analyze **RQ4**.

6.1 Data Preparation

Finding 16: 32% of the bugs are in the data preparation stage

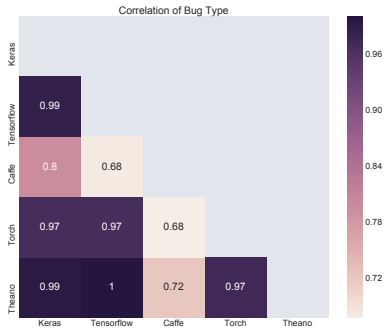


Figure 6: Correlation of Bug Types among the libraries

From Figure 5 we see, most of the bugs in deep learning programming happen at the data preparation stage.

6.2 Training Stage

Finding 17: 27% of the bugs are seen during the training stage

The next bug prone stage is the Training stage which is as expected. Most bugs related to IPS and SI arise in the training stage.

6.3 Choice of Model

Finding 18: Choice of model stage shows 23% of the bugs

Choice of model is the third bug prone stage. In choice of model stage, we construct the model and chose the right algorithm. Major root causes of bugs in this stage are IPS, SI, and UT.

7 COMMONALITY OF BUG

In this section, we explore the answer to **RQ5** to identify whether there is any relationship among the bugs in different deep learning libraries. Our primary hypothesis was that the libraries will be strongly correlated based on the distribution of bugs as they are performing similar tasks.

Our analysis confirms that hypothesis as shown in Figure 6. We see that the libraries have a strong correlation coefficient close to 1. Surprisingly *Caffe* has shown very weak correlation with other libraries in terms of bug type. We then randomly studied 30 *Stack Overflow* posts for each of the libraries to see whether we notice any common antipatterns that can lead to this strong correlation of bug type.

Finding 19: *Tensorflow* and *Keras* have a similar distribution of antipatterns while *Torch* has different distributions of antipatterns

We have identified the antipatterns through deeper analysis of the *Stack Overflow* buggy codes for further investigating the strong correlation of *Tensorflow* and *Keras* as well as the weak correlation of *Torch* and *Caffe*. The antipatterns found are **Continuous Obsolescence, Cut-and-Paste Programming, Dead Code, Golden Hammer, Input Kludge, Mushroom Management, Spaghetti Code**. This classification is taken from [2]. The distribution of different antipatterns across the libraries is shown in Figure 7. We see that in *Tensorflow* and *Keras* 40% of the antipatterns are Input

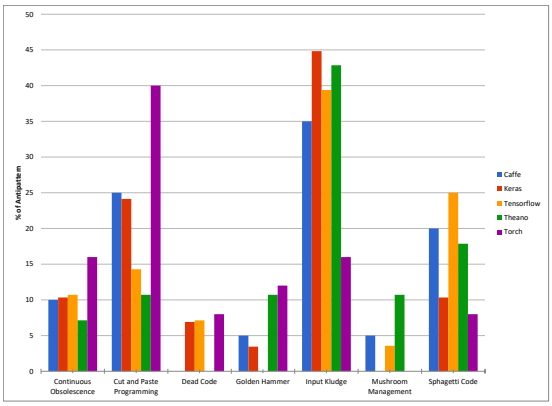


Figure 7: Distribution of different antipatterns

Kludge. On the other hand, in *Torch* 40% of the bugs arise due to the Cut-and-Paste Programming antipattern. *Tensorflow* and *Keras* have almost same distribution in Continuous Obsolescence and Dead Code as well. This shows that the strong correlation between the distribution of bugs in *Tensorflow* and *Keras* can be explained from the similarity of common antipatterns for these two libraries. The weak correlation between the distribution of *Torch* and *Caffe* bugs can be the result of a dissimilar distribution of antipatterns between these two libraries. For example, we see *Stack Overflow* code snippets of Input Kludge antipatterns from *Tensorflow* and *Keras* in the example shown in Figure 8. Both of these programs can be easily broken by user input and the program does not perform sanity check on the inputs.

8 EVOLUTION OF BUGS

In this section, we explore the answer to **RQ6** to understand how the bug patterns have changed over time.

8.1 Structural Logic Bugs Are Increasing

Finding 20: In *Keras*, *Caffe*, *Tensorflow* Structural logic bugs are showing increasing trend

From 2015 - 2018 Structural logic bugs in *Caffe* are respectively 30%, 32%, 67%, 100% indicating structural logic bugs are being discussed more by the developers since 2015. It is expected as deep learning started gaining increasing attention since 2015 and more developers started to use deep learning libraries to write software.

8.2 Data Bugs Are Decreasing

Finding 21: Data Bugs slowly decreased since 2015 except *Torch*

In *Torch* Data Bugs stayed almost consistent maintaining close to 50% of the bugs in discussed in 2016-2018. In *Keras* Data Bugs slowly decreased from 27% - 15% since 2015. In *Tensorflow* Data Bugs slowly decreased from 30% - 10% since 2015 - 2018. In the other two libraries also, the Data Bugs slowly decreased reaching close to 0. The possible reason for this trend is the development of popular specialized data libraries like pandas that enable exploratory data analysis to understand the properties of data better. Besides, the use of Tensor data type having type and shape information helps to get rid of some of the Data Bugs. Still more verification support in these libraries will help to get rid of these bugs.

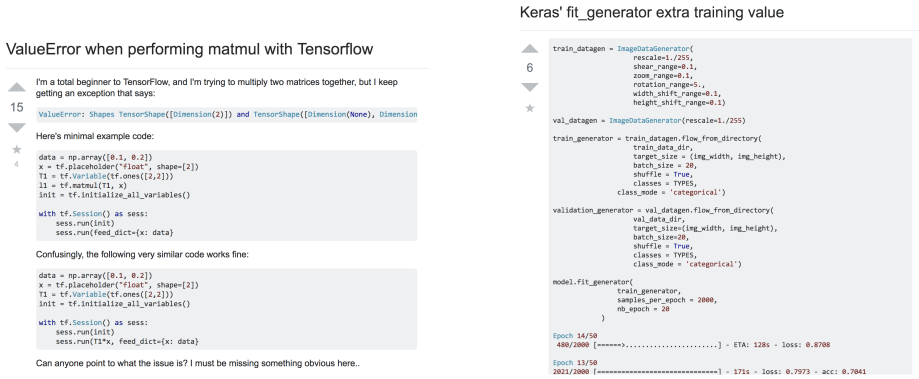
9 THREATS TO VALIDITY

Internal threat. One internal threat to the validity of our results could be our classification of the bugs. We used the classification scheme from a vetted taxonomy [5, 27] to classify the bugs. We also followed open coding scheme to add more types if needed. One PhD student was initially dedicated to go over all the posts to come up with additional classification scheme, if necessary. This whole process was monitored using pilot study. Another possible source of the threat is that the labeling of the data can be biased. To mitigate this threat two trained Ph.D. students independently studied the misuse posts to label them. The inter-rater agreements was measured using Cohen's Kappa coefficient and the disagreements were reconciled under the monitoring of an expert. We conducted pilot study to continuously monitor the labeling process and conducted further training at 5% and 10% of the labeling where the Kappa coefficient was close to 0% and 80%.

External threat. An external threat can be the trustworthiness of the dataset we collected. To avoid low-quality posts we only collected the posts that have score of at least 5. A score of 5 can be a good metric to trust the post as a good discussion topic among the programmer community that cannot merely be solved using some Google search. The reputation of the users asking question about deep learning can be another reason to question the quality of the posts. To alleviate this threat we have only studied top scored posts which are from users with different range of reputations (1 - 150K+). This indicates that the posts are from users ranging from newbie to experts. The dataset is unbalanced in terms of frequency of bugs studied for each library; however, to confirm the distribution of the bugs, we have performed ANOVA test on the bug types, root causes, and impacts for each library. We have found that $F(0.99) < F\text{-}critical(2.55)$. This implies that the means of the five libraries population are not significantly different. This suggests that even though the dataset seems unbalanced in term of frequency, the bug distribution is not.

10 DISCUSSION

We have seen in the analysis of RQ1 that most of the bugs in deep learning programming are Data Bugs. These type of Bugs can have drastic effect causing the program to crash as well as leading to bad performance. In general, we see the programmers have very limited or no access to data verification tools. It is often confusing whether the data is in right format needed by the model, whether the variables are properly encoded or not, whether there are missing data that can cause the model to fail, whether the train test split is good enough, whether the data is shuffled properly to avoid training bias etc. This finding suggests that development of **data verification tools** can help programmers solve a large number of data bugs. As deep learning models are strongly coupled with data, **model analysis** tool to explore whether a particular model is the right fit for the data in hand can help to resolve these strong coupling of data and model related problems.



(a) Tensorflow Example of Input Kludge

(b) Keras Example of Input Kludge

Figure 8: Example of similar antipattern in *Tensorflow* and *Keras*

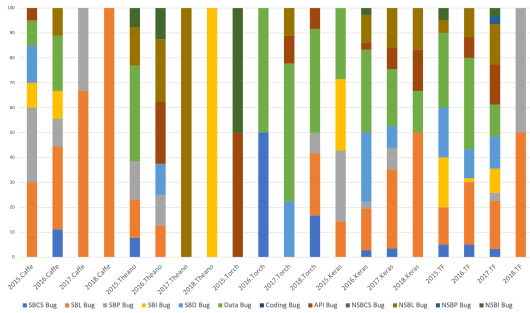


Figure 9: Timeline of Evolution of Bugs

We have also seen while exploring RQ1 that structural logic bugs are the second major type of bugs. This happens due to wrong logical organization of the model, hidden layers, using wrong codes, etc. These kind of problems can be solved by some **automated model and parameter recommendation** tools. How to develop these kind of tools need further research. A methodology could be to mine large scale open source code repositories [10–12] using Python dataset [6] and identify the common code patterns and suggest examples from common code patterns.

11 RELATED WORKS

The closest related work is by Zhang *et al.* [27] who have investigated bugs from deep learning applications built on top of Tensorflow. They collected 500 *Stack Overflow* posts and filtered them to study 87 posts and selected 11 Github projects to include 82 commits with 88 bugs using keywords e.g., bug, fix, wrong, etc. In contrast, we studied a cross-section of five deep learning libraries with different design constraints, using 555 bugs from GitHub and 415 *Stack Overflow* posts, that allowed us to draw interlibrary observations. Zhang *et al.* have studied the bugs and have categorized them into 7 types of bug/root causes and 4 types of impacts/symptoms. Our work expanded the study to include bug types from literature and categorizes the bugs into 11 bug types, 10 root causes and, 7 impacts. In term of results, our study both confirms what was known as a small scale and produces new knowledge e.g., correlating antipatterns with bugs [22].

Thung *et al.* [24] studied three machine learning systems, Apache Mahout, Lucene, and OpenNLP and manually categorize the bugs into different categories. They focused on bug frequencies, bug types, severity of the bug, bug-fixing duration, bug-fixing effort, and bug impact. Different from them, we focus on bug types, bug root causes, and bug impact of five deep learning libraries which are Tensorflow, Keras, Torch, Caffe, and Theano.

There are some empirical studies focused on specific types of bugs. Lu *et. al.* [20] studied real-world concurrency bug characteristics. Gao *et. al.* [13] conducted an empirical study on recovery bugs in large-scale distributed systems. API changes problems was studied by [4, 9, 17, 18]. Our work focuses on the bugs in the usage of deep learning libraries.

Other prior work that have studied *Stack Overflow* posts, e.g. [3, 16, 19, 21], have not focused on deep learning software.

12 CONCLUSION AND FUTURE WORK

Although deep learning has gained much popularity and strong developer community in recent years, developing software using existing deep learning libraries can be error-prone. In this paper, we have presented an empirical study to explore the bugs in software using deep learning libraries. In our study we have studied `2716` qualified *Stack Overflow* posts and `500` *Github* bug fix commits to identify the bug types, root causes of bugs, effects of bugs in usage of deep learning. We have also performed an inter-stage analysis to identify the stages of deep learning pipeline that are more vulnerable to bugs. We have also studied the buggy codes in *Stack Overflow* to find antipatterns leading to bugs to understand the strong correlation of the bug types in deep learning libraries. Our study found that data bug and logic bug are the most severe bug types in deep learning software appearing more than 50% of the times. Major root causes of these bugs are Incorrect Model Parameter (IPS) and Structural Inefficiency (SI). Last but not least, bugs in the usage of deep learning libraries are strongly correlated. This work opens multiple avenues for exploration. For instance, while we have studied bugs, we haven't yet examined the fix strategies that programmers use. This study is also on a relatively modest dataset and could be repeated on a much larger dataset. Finally, repair strategies could be developed for deep learning programs.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[2] Alexander Shvets. 2017. Software Development AntiPatterns. https://sourcemaking.com/antipatterns/software-development-antipatterns.

[3] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. 2014. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering* 19, 3 (2014), 619–654.

[4] Gabriele Bavota, Mario Linares-Vasquez, Carlos Eduardo Bernal-Cardenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2015. The impact of api change-and fault-proneness on the user ratings of android apps. *IEEE Transactions on Software Engineering* 41, 4 (2015), 384–407.

[5] Boris Beizer. 1984. *Software system testing and quality assurance*. Van Nostrand Reinhold Co.

[6] Sumon Biswas, Md Johirul Islam, Yijia Huang, and Hridesh Rajan. 2019. Boa Meets Python: A Boa Dataset of Data Science Software in Python Language. In *MSR'19: 16th International Conference on Mining Software Repositories*.

[7] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.

[8] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. 2002. *Torch: a modular machine learning software library*. Technical Report. Idiap.

[9] Danny Dig and Ralph Johnson. 2006. How do APIs evolve? A story of refactoring. *Journal of software maintenance and evolution: Research and Practice* 18, 2 (2006), 83–107.

[10] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2013. Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 422–431.

[11] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2015. Boa: an Enabling Language and Infrastructure for Ultra-large Scale MSR Studies. *The Art and Science of Analyzing Software Data* (2015), 593–621.

[12] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2015. Boa: Ultra-Large-Scale Software Repository and Source-Code Mining. *ACM Trans. Softw. Eng. Methodol.* 25, 1, Article 7 (2015), 7:1–7:34 pages.

[13] Yu Gao, Wensheng Dou, Feng Qin, Chushu Gao, Dong Wang, Jun Wei, Ruirui Huang, Li Zhou, and Yongming Wu. 2018. An empirical study on crash recovery bugs in large-scale distributed systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 539–550.

[14] David Gómez and Alfonso Rojas. 2016. An empirical overview of the no free lunch theorem and its effect on real-world machine learning classification. *Neural computation* 28, 1 (2016), 216–228.

[15] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 675–678.

[16] David Kavaler, Daryl Posnett, Clint Gibler, Hao Chen, Premkumar Devanbu, and Vladimir Filkov. 2013. Using and asking: Apis used in the android market and asked about in stackoverflow. In *International Conference on Social Informatics*. Springer, 405–418.

[17] Raula Gaikovina Kula, Ali Ouni, Daniel M German, and Katsuro Inoue. 2018. An empirical study on the impact of refactoring activities on evolving client-used APIs. *Information and Software Technology* 93 (2018), 186–199.

[18] Jun Li, Yingfei Xiong, Xuanzhe Liu, and Lu Zhang. 2013. How does web service API evolution affect clients?. In *2013 IEEE 20th International Conference on Web Services*. IEEE, 300–307.

[19] Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2014. How do api changes trigger stack overflow discussions? a study on the android sdk. In *proceedings of the 22nd International Conference on Program Comprehension*. ACM, 83–94.

[20] Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. 2008. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *ACM SIGARCH Computer Architecture News*, Vol. 36. ACM, 329–339.

[21] Sarah Meldrum, Sherlock A Licorish, and Bastin Tony Roy Savarimuthu. 2017. Crowdsourced Knowledge on Stack Overflow: A Systematic Mapping Study. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. ACM, 180–185.

[22] Seyyed Ehsan Salamati Taba, Foutse Khomh, Ying Zou, Ahmed E Hassan, and Meiyappan Nagappan. 2013. Predicting bugs using antipatterns. In *2013 IEEE International Conference on Software Maintenance*. IEEE, 270–279.

[23] The Theano Development Team, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, et al. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688* (2016).

[24] Ferdian Thung, Shaowei Wang, David Lo, and Lingxiao Jiang. 2012. An empirical study of bugs in machine learning systems. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering*. IEEE, 271–280.

[25] Anthony J Viera, Joanne M Garrett, et al. 2005. Understanding interobserver agreement: the kappa statistic. *Fam med* 37, 5 (2005), 360–363.

[26] Yufeng Guo. 2017. The 7 Steps of Machine Learning. https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e.

[27] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An empirical study on TensorFlow program bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 129–140.