A Review on Automatic Testing of Deep Learning Systems

Rijwan Khan
Department of Computer Science and Engineering
ABES Institute of Technology, Ghaziabad, UP, INDIA

Manish Mahajan
Department of Information Technology
Graphic Era Deemed to be University, Dehradun, UK, INDIA

Abstract- The process of testing conventional programs is quiet easy as compared to programs using the Deep Learning approach. The term Deep learning (DL) is used for a novel programming approach that is highly data centric and where the governing rules and logic are primarily dependent on the data used for training. Conventionally, Data Learning models are evaluated by using a test dataset to evaluate their performance against set parameters. The difference in data and logic handling between programs using conventional methods and programs using the DL approach, makes it difficult to apply the traditional approaches of testing directly to DL based programs. The accuracy of the test data is currently the best measure of the adequacy of testing in the DL based systems. This poses a problem because of the difficulty in availability of test data that is of sufficient quality. This in turn restricts the level of confidence that can be established on the adequacy of testing of DL based systems. Unlike conventional applications, using the conventional programming approaches, the lack of quality test data and the lack of interpretability, makes the system analysis and detection of defects a difficult task in DL based systems.

Keywords – Automatic Testing, Deep Learning Systems, Automatic Test Case Generation, Software Testing, Unit Testing.

I. INTRODUCTION

Software testing is a process of delivering error free and reliable software to the customers. Testing of Deep learning program is quite different to the simple programs. Deep Learning Program development is differ with compare to simple programs.

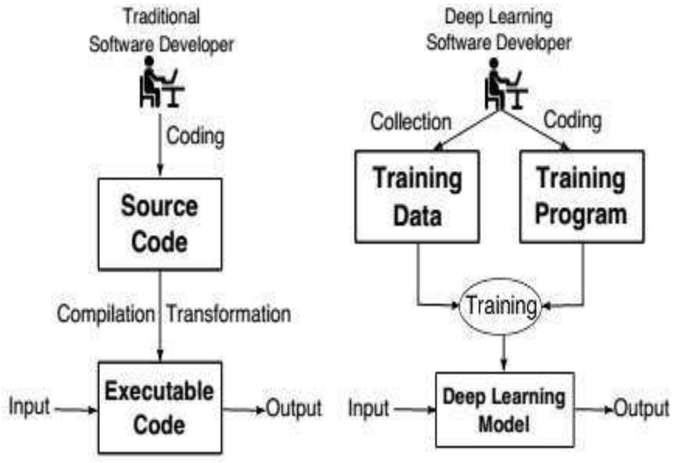

Figure 1. Comparing Conventional and Deep Learning based Software Development Process

In traditional or conventional software applications, the business logic is designed, developed and implemented by software developers using coding techniques with some specific programming language. On the other hand, it is the structure and behavior of the Deep Neural Networks (DNNs) and the respective connection weights used in their inter-connections, which define the behavior of the DL system. The interconnection weights are calculated by executing the pre-defined training programs on the data set provided for training purposes. The structure of the DNN is mainly specified by the logic and code fragments in some High Level language that make up the training programs. This makes both the data set used for training as well as the specific model or program being used for training, the probable causes and sources for errors and defects in the DL based systems [1].

II. SOFTWARE TESTING INRESEARCH

VahidGarousi [10] et al identify how to test embedded software, for this they presented an article on mapping of literature in a systematic manner. They gave an extensively comprehensive survey of the work carried out in the field of testing of embedded software. Their survey reviewed 312 papers to provide extensive details focused on the various aspects of testing, various testing activities, variety of generated test artifacts, and the application areas of the concepts under study. A major proportion of the studies on testing of embedded software focus on automating testing and on model-based testing. Although interest in testing of embedded software is on the rise, most of the contemporary and past researchers have only reported their experiences or proposed solutions. There is a marked paucity of evidence based on empirical research on the effectiveness and efficiency of approaches specifically meant for testing of embedded software.

Ari Takanen [23] et al provided a platform to be used by end users for the purpose of creation of fuzz tests meant for arbitrary protocols. The Fuzzer architecture is time and resource hungry approach to model tests meant for new interfaces. In the event that the framework is not able to offer some ready-made inputs meant for common elements and structures, the process of efficient testing also necessitates the tester to have expertise in the process of designing inputs which can trigger specific faults in the interface being tested.

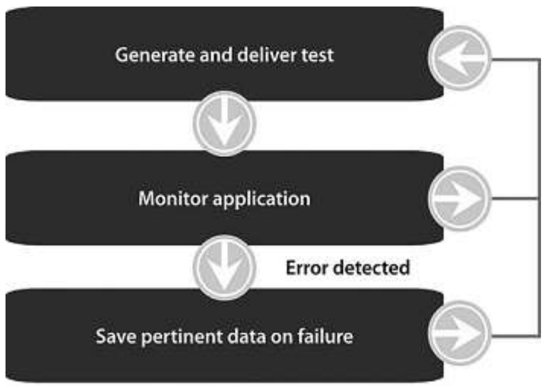

Figure 2. Fuzzing process: A combination of test case generation and monitoring of system

Fredrik Asplund [24] et al studied emphasized on the importance of the context pertaining to the individual tester towards issues that are safety-related. While organizations engaged primarily in developing software may find test specialization to be attractive, the results obtained by the authors suggested that important changes to the processes involved and organizational structures are necessary to maintain efficiency of testing throughout the life time of the projects. There is a need of delaying the rotation of responsibilities for test cases, and there is also a need of active management intervention to avoid asymmetries introduced in information flows by strengthening the presence of test teams for the entire project life time throughout the firm.

Martin Monperrus [25] et al provided an annotated bibliography focusing on automation of software repair approaches. This research area can be considered to be both new and old. Authors have reviewed techniques pertaining to automation of repair in papers on fault-tolerance systems from as early as 70s. It may be considered new, since the approach to change the code automatically, a method known as behavioral repair, has been a focus of research only since the late 2000.

Michael Felderer [26] et al suggested that extensive research to be carried out in the area of software engineering is required in order to understand fully the role played by testing in case of software applications which are data-intensive (for instance big cyber-physical systems) and also in order to provide various approaches and specific frameworks so as to address this crucial issue properly. Their research was targeted towards contributions in this domain by means of providing the basic terminologies, in addition to specifying the future directions that could be taken by the research in the area of so as to develop methodologies for testing the software applications and systems that are data-intensive. Towards this end, authors also provided a clear and concise definition of data-intensive software systems which helped to clearly identify them apart from the other various types of software systems and also helped to identify the typical cyber physical production systems as being data-intensive systems. The definition they provided was also instrumental in deriving the major factors which are used to characterize such type of systems and also were crucial in driving the further investigation. They also further discussed the state of the art and new possible and probable frontiers on testing of the data in these type of software systems.

Yunho Kim [27] et al presented a novel automated test case generation technique called DEMINER. Their approach helps to realize the invasive paradigm of software testing. It does so by using the information obtained from multiple mutant executions. The authors demonstrated that DEMINER was able to increase coverage of testing effectively by testing it on three real world programs written in C. They used their automated test case generation technique on the oncotic testing to showcase its efficiency and effectiveness.

Gomaa [28], el al proposed that some specific aspects in testing of real-time systems show no variation from their counterparts for non-real-time systems. Most of the differences which are there, are due to the interface of the software systems to external devices or from their executing concurrent tasks.

Lilian P. Scatalon [29] el al presented a review on the gaps that are in the knowledge in the areas of software testing from an industry point of view. They approached the knowledge gaps from two directions: conceptual gaps and gaps in practical activities. They observed marked negative gaps in areas like web applications testing, functionality testing and the utilization of client requirements for the purpose of test case generation. They also observed positive gaps in areas like aspect oriented software testing and in some techniques used for test case generation.

Imke Drave [30] et al presented an efficient and systematic method which can be utilised in order to derive the test cases which can be used for automotive software using the specifications which were already existent in the SMArDT method of BMW Group. Since these specifications are represented in form of state charts which moreover have activity diagrams conceptually embedded in them, they are able to be homogenized within the UML/P activity based diagrams. The resulting modified activity diagrams can then be optimized so as to be transformed into a XML-based tabular format which provides effectively the corresponding test actions individually and additionally the valuations of the variable and it represents them in an executable format.

Mustafa Al-Hajjaji [31] et al presented an analysis of time taken by testing of product lines. There is a demand of increasing the fault detection probability especially at an early stage for the product line being evaluated. There have been various approaches that researchers have proposed for the purpose of prioritizing the products. The similarity-based prioritization approach attempts to prioritize the various products on the basis of similarities in the selection and the de-selection of their features. The approach uses sampling algorithms in order to generate a subset out of all the valid possible configurations. The authors have evaluated the similarity-based prioritization approach against the approaches following randomized orders, orders based on interactions, and default orders. The results obtained by them demonstrated that the rate and probability of early detection of faults while using the similarity-based prioritization approach is significantly higher than that while using random orders approach. Since the similarity-based approach is primarily a greedy approach, applying appropriate optimization algorithms might prove to be helpful by avoiding being trapped in vicinity of local optima.

TSONG YUEH CHEN [32] et al have discussed their results with Metamorphic testing (MT). The MT approach has successfully demonstrated detection of various faults while testing applications in multiple domains. Advanced MT techniques have been proposed by its integration with other appropriate software engineering methodologies. These amalgamations are intended to address the oracle problem faced by other areas. MT has also found applications in non-testing areas, including validation, assessment of quality, debugging, localization of fault, fault tolerance, and automatic program repair. The authors have focused on some of the most important as well as influential studies on MT, thereby providing a considerably more in-depth discussion. They have, in addition, tried to clear the common

misconceptions of MT. In addition, they have given a higher-level vision of research in the field of MT and its application.
TOMASZ KUCHTA [33], et al have presented the shadow symbolic execution technique. This is a novel technique used for generation of inputs that help trigger the modified behaviors after the introduction of software patches. The shadow symbolic execution approach is based on the running both patched and unpatched versions in the same common symbolic execution instance while systematically testing any code-level divergences that might come up. This technique helps to unify the two program versions, patched and unpatched, via the change annotations, thereby maximizing the sharing that exists between the corresponding symbolic stores of the two versions. The approach then focuses on those identified paths that have triggered the divergences. The authors have implemented their technique in a software tool called Shadow. The tool has been used in generation of inputs used for exposing bugs and marking changes intended in complex Coreutils patches.

Ahmed M. Alghamdi [34] et al, discussed testing parallel systems that use heterogeneous programming models has become increasingly important, and the movement to Exascale systems makes it even more important to avoid errors that could affect the system requirements, not only errors that can be detected by compilers, but also more critical errors that occur after the compilation process and therefore cannot be further detected by the existing compilers. As a result, testing tools have been built, and different testing techniques have been used to detect static and run-time errors in parallel systems. These tools and techniques are targeting systems built by several types and levels of programming models. Authors studied more than 50 testing tools and classified them according to the testing techniques employed in the tools and also the targeted programming models as well as the run-time errors. They tried to discover the limitations and open areas for the researchers in testing parallel systems, which can yield the opportunity to focus on those areas. To conclude, there has been good effort in testing parallel systems to detect run-time errors, but that cannot be considered enough, especially when considering systems using heterogeneous programming models, as well as the dual- and the more advanced tri-level programming models. The integration of different programming models in the same system will need new testing methodologies in order to catch the various run-time errors encountered in heterogeneous parallel computing systems, which will be addressed in our future work. We believe that to achieve good systems that can be used in Exascale supercomputers, should focus on testing those systems because of their massively parallel natures as well as their huge size, which add more difficulties and issues. Also, these testing tools should integrate more than one testing techniques and perform concurrently in order to detect the run-time faults or errors. This can be done by creating threads for testing, their number depending on the application threads. Therefore, use of hybrid parallel techniques promises to enhance testing efficiency by reducing time and also promises to uncover a wider range of possible run-time errors.

Huayao Wu [37] et al provided an empirical comparison of three of the more popular testing techniques: CT, RT and ART, and they did so by utilizing different test scenarios. The three techniques were compared on the basis of their ability to detect fault and their computational cost and this comparison was done using a variety of choices for proportion of their parameters (para), proportion of the various constraints (cons) available in the model, and the model's fault failure rates (rate).

Stefan Kriebel [38] et al demonstrated that shortening the development processes and modifying requirements frequently has an adverse effect on the balance existing between the time to market and standards of quality. In order to maintain the required high quality standards, a comprehensive testing strategy is crucial especially in automotive software engineering. This is hampered by the fact that till date majority of test case creation process are still manual. The authors had carried out a comprehensive survey in order to investigate the possible benefits of model-based testing in the automotive sector.

Arie van Deursen [39] et al, discussed that test cases can target different levels, ranging from units (individual methods) to the complete system. Authors started it by looking at a system-level test that tries to exercise the system from end-to-end. Authors called this one a smoke test. A Smoke Test is a simple system test that exercises minimal system behavior. If we just start the system, and we "see smoke" (i.e., not even the smoke test passes), there is no point in moving to the next step of the software development cycle.

Caroline Lemieux [40] et al, demonstrated that in case of benchmarks chosen which have a non-similar input formats and used for accomplishing different tasks, the results obtained from their evaluation need not necessarily generalize to other programs. The major limitation seen while using rare branches for the purpose of testing target in Fair Fuzz is that the branches never encountered by any AFL input are not targeted using this method.

Mark Harman [41] et al, reviewed the work being carried out on Search Based Software Testing (SBST). They have presented evidence to the fact that the range and variety of non-functional properties which are being focused using SBST is on the rise, but at the same time, the proportion of research in this field is falling, which indicates a troubling trend, especially with the ever increasing significance of non-functional properties to developers and testers. The authors specifically highlighted the lack of contemporary research work being done in the field of Search Based Energy Testing (SBET), additionally outlining the various energy measurement techniques which might be reused in the form of fitness functions.

Ammann, Paul [42] et al, discussed the process of testing as resulting from the abstract models developed for the software including graphs, as these can as conveniently be derived from a either a black-box view or a white-box view. Thus, the unique philosophical approach and structure of their book helped to render these two terms obsolete.
Definition 1.1 Black-box testing: This is the approach where the external descriptions that are available of the software. Theses may include the software specifications, user requirements, and design specifications.
Definition 1.2 White-box testing: This is the approach where tests are derived using the internals of the software which includes the source code, specifically focusing on branches, individual conditions for these, and also the statements.

Claudia P. C. Maciel [43] el al, reported on the results from an analysis related to the software testing professionals perception on the use of KM initiatives in software engineering companies. For this, they adapted and reapplied the survey conducted by Souza et al. From the survey results, they drew the following main conclusions: (i) Test Planning is considered the most important activity for using KM initiatives; (ii) The type and level of Communications existing between the various members of the test team is actually the form of tacit knowledge item most important to be made explicit; and (iii) Test Case is an artifacts that has received more attention by companies in terms of reuse. A strategic planning and the reusing the existing test cases can result in a significant reduction in the software development costs and duration. One initial threat to internal validity in this work could be poor instrumentation, which could affect understanding the questions. For this reason, a pilot study was conducted as well as a survey validation by a KM and software testing researcher, so problems could be identified and corrected. Based on these threat, we intend to conduct interviews in software companies and compare the results. A second threat of this research concerns the sample size and representativeness. The sample had 75 valid answers among 39 software companies and a large concentration in Brazil's south and southeast regions, which affect both conclusion and external validity. The results cannot be generalized to a nationwide and world scale. Therefore, we intend to replicate this survey in other regions of Brazil and also in an international context. A richer investigation with better mechanisms to perception about KM in software companies forms part of future work.

Tao Xie [44] et al, discussed parameterized unit testing a novel approach to unit testing methodology which extends the previously being used industry practices of non-parameterized unit tests. The proposed parameterized unit test (PUT) is a testing method that employs parameters, executes the code being tested, and subsequently states assertions.

Marcel Bohme [45] et al, discussed the automation of software testing in terms of efficiency. The authors have proposed a framework where they demonstrated that taking too long a time may cause even a more effective method of systematic testing to be deemed inefficient compared to a simplistic technique like random testing.

Benjamin S. Clegg [46] et al, have proposed use of the concept from the CODE DEFENDERS game in order to design specific systematic exercises which could help in teaching a variety of testing related concepts. Their model envisages implementing each concept in the form of a series comprising of individual simple games operating on small atomic programs which are being tested, and the players need to improvise a test suite in order to unearth a mutant character depending on a targeted concept related to testing.

Rico Angell, Brittany Johnson [47] et al, have presented ‘Themis’, is an open-source implementation done for an automated test data generation technique used for testing of causal and group discrimination software. The Themis tool employs three optimizations methods, which help to significantly reduce the size of test suite used.

III. DEEP LEARNING

MinkeXiu et all [6] et al gave an empirical comparison covering the informational elements and overlap of models between the three most popularly used model stores and also two common mobile app stores. Based on elements of the store information which are unique to the specific model stores, the authors have derived some specific Software Engineering practices which unique to ML based models:
• Integration
• Release Management
• Quality and Requirements
• Pricing
• Demo:
• API Documentation
• Cross-store support
• Product maturity

Andreas Kamilaris [12] et al, delivered a survey involving deep learning-based research being carried out in the agricultural domain. They compared deep learning methods with other contemporary techniques, with respect to performance. Their results indicate that deep learning methods offers a much better performance and so outperform the other popularly used techniques of image processing.

Konstantinos P. Ferentinos [13] et al, have developed specialized deep learning based models, which makes use of specifically designed convolutional architectures of neural networks, to be used for the purpose of identifying plant diseases by using simple images of leaves from healthy or diseased plants. The authors used a database from the public domain which comprised of 87,848 photographs which could be used to train the model. They used the images which were captured in both in-house laboratory conditions as well as real outdoor conditions in cultivation fields. The results obtained have demonstrated the high potential of their model.

Bethany Lusch [14] et al, helped identify the coordinate transformations which help to make strong nonlinear dynamics into approximately linear ones and the ones that have a potential of enabling the use of linear theory for the purpose of nonlinear prediction, control and estimation.

Asheesh Kumar Singh [15] et al, demonstrated that the ML approaches show great promise towards the improvement of speed, achieved accuracy, measured reliability, and the scalability attained in the disease phenotyping required to achieve the diversified programmatic goals. Authors discussed that the rapid progress in, and the easy availability of, ML or DL workflows has made it more feasible to finally develop and implement application-specific models. Thus, the high-throughput phenotyping being done in the field can now be as accessible as the phenotyping being carried out in controlled laboratory environment.

Maziar Raissi [16] et al have considered the traditional coupled problem faced by a freely vibrating cylinder caused by lift forces and have demonstrated how deep learning methods can be employed to infer relevant quantities from data scattered in space-time.

Merima Kulin [17] et al, have presented a systematic introduction to the end-to-end learning approach from spectrum data which is a DL based approach for the realization of different tasks involved in identification of wireless signals, the crucial building blocks used in the various spectrum monitoring systems. Their proposed approach develops around the systematic application of Dl based techniques in order to obtain more accurate wireless signal classifiers in an end-to-end learning pipeline. They discussed means to empower and guide practitioners in the field of machine learning/signal processing and wireless engineers to enable them to design novel innovative research based applications of the end to-end learning approach.

Christopher J. Shallue [18] et al, presented a novel method for the automatic classification of Kepler transiting planet candidates using deep learning based methods. The neural network based model given by them was able to efficiently distinguish the subtle differences existing between the transiting exoplanets and the false positives like eclipsing binaries, instrumental artifacts, and stellar variability.

Samuel G. Finlayson [19] et al discussed the prospect of improvement in healthcare and medicine by the use of deep learning base methods. For medical providers, payers, and policy makers, their practical examples have the capability to motivate the medical providers, payers, and policy makers towards carrying out a meaningful discussion over how to utilize these algorithms by incorporating them into the clinical ecosystem.

Bradly C. Stadie [20] et al discussed the relevance of sampling as applied in Meta reinforcement learning approach. They derived two new algorithms and analyzed their properties.

Maziar Raissi [21] discussed a machine learning based approach for the purpose of extraction of nonlinear dynamical systems out of periodic or time-series data. Their proposed algorithm helps leverage the inherent structure of the various established multi-step time-stepping approaches like Adams-Bashforth, Adams Moulton, and BDF families, to construct efficient algorithms for learning dynamical systems using deep neural networks. A key property of the proposed approach is the use of multiple steps which enables us to incorporate memory effects in learning the temporal dynamics and tackle problems with a nonlinear and non-Markovian dynamical structure. Specifically, the use of M steps allows us to decouple the regression complexity due to several temporal lags, ultimately leading to a simpler D-dimensional regression problem, as opposed to an $(M \times D)$-dimensional problem in the case of a brute force NARMAX or recurrent neural network approaches. Although state-of-the-art results are presented for a diverse collection of benchmark problems, there exist a series of open questions mandating further investigation. How could one handle a variable temporal gap Δt, i.e., irregularly sampled data in time? How would common techniques such as batch normalization, drop out, and L1/L2 regularization enhance the robustness of the proposed algorithm and mitigate the effects of over-fitting? How could one incorporate partial knowledge of the dynamical system in cases where certain interaction terms are already known? In terms of future work, interesting directions include the application of convolutional architectures for mitigating the complexity associated with very high-dimensional inputs, as well as studying possible connections with recent studies linking deep neural networks with numerical methods and dynamical systems.

Yaron Gurovich [22] et al, proposed a framework called Deep Gestalt which was based on facial analysis for the classification of genetic syndrome. The proposed framework helps to leverage the deep learning technologies and also helps in learning the facial representation effectively from the large scale face-recognition dataset provided. This is followed by using the fine-tuning approach towards transfer of knowledge to the genetic syndrome domain.

IV. AUTOMATIC TESTING FOR DEEP LEARNING SYSTEM

Grano, Giovanni [35] et al, discussed that while considering a continuous integration environment, the apriori knowledge of the overage achieved by the tools used to generate the test data might help to facilitate important decisions, like the choice (and the order) of the subset of classes to be used to test, or the provisions of the budget to allocated for them. The authors have taken the early steps towards defining the features and various machine learning approaches which are able to predict the branch coverage being achieved by the tools used to generate the test data. Because of the non-deterministic nature shown by the algorithms used for this purpose, such prediction still are a troublesome task.

Tariq M. King [36] et al, reviewed the industry panel for the 2018 Annual Western Conference on Software Testing Analysis and Review featured a two-session panel on AI for Software Testing (AIST). The conference had held discussions on the future scope, conceptual ideas, research and industry specific strategies, directions, and lessons learned during the course of developing systems using AI which were intended to the purpose of testing the software, applying specific efficient methods used to test the various AI systems, and designing systems which are self-testing.

V. RELATED WORK

Lei Ma [1] et al studied the relevance and applicability of the mutation testing approach towards DL based systems. Authors have initially proposed a source level mutation testing approach which works with the training data and the specific training programs and then the authors helped design a set of source-level novel mutation operators in order to inject faults which mimic those faults that could be potentially introduced during the process of development of DL based systems. In addition, the authors also proposed a model-level methodology for mutation testing and have designed another set of novel mutation operators that can be used to inject faults directly into DL based models. Furthermore, the authors have proposed the specific mutation testing metrics to be used to measure the quality of test data being used. This paper performs an initial exploratory attempt to demonstrate the usefulness of mutation testing for deep learning based systems.

Lei Ma [2] et al discussed the threat resulting from the extensive use of DL based systems, especially in many safety-critical areas, towards their quality and generalization property. In order to measure the testing adequacy effectively and to lay down the foundation for designing effective DL based testing techniques, they have then proposed a set of testing criteria for DNNs. They carried out experiments on two common well-known datasets, as well as five DNNs having diverse complexity, and additionally four state-of-the-art adversarial testing techniques to demonstrate that the tests generated by employing the adversarial techniques markedly how increases in the coverage with respect to the metrics which were defined in the paper

Lei Ma [3] et al discussed the efficacy of combinatorial testing as a successful and efficient technique being employed in the testing of traditional software systems. They initiated a comprehensive empirical survey based on the applicability and usefulness of CT in the area of testing DL based systems. Their evaluation results helped demonstrate the usability and effectiveness of CT for testing of DL based systems. We again emphasize that our research is the first to adapt the concept of CT, provide CT coverage guided test generation, and perform an empirical study toward evaluating the robustness of DL based systems using CT. We thus hope that applying CT to DL based systems may constitute a seminal foresight built on our evaluation results, and in parallel shed light on the construction of more CT coverage criteria and scalable test generation techniques towards achieving robust real-world DL applications.

Anurag Dwarakanath [4] et al investigated the problem of identification of implementation bugs present in applications based on machine learning. The current approaches being used to test the ML based applications are limited to performing ‘validation’, in which the tester mainly acts as a human-oracle. This is due to the fact that ML based applications are inherently complex and so the verification of their output w.r.t the application specification proves to be extremely challenging. The proposed approach was based on Metamorphic Testing. This involves building of multiple relations between the subsequent obtained outputs from a program in order to effectively verify and validate the appropriateness of the implementation. The authors demonstrated their results by developing such relations using two applications used for image classification - one using a classifier based on Support Vector Machine (SVM) and the another using Convolutional Neural Network using Deep Learning approach. Their experimental results demonstrated that, their approach was successful in capturing on average, 71% of the implementation bugs.

Jinhan Kim [5] et al proposed a novel surprise adequacy framework for DL based systems (SADL) which can help to evaluate the relative value of surprise for each input in a quantitative manner with respect to the available training data The value thus obtained was called the Surprise Adequacy (SA). Using this value of SA, Surprise Coverage (SC) can be developed, which can be used to measures the range of discretized input surprise adequacy values. This provides a better insight as compared to the one obtained from measuring the count of neurons having the pre-determined specific activation traits. The survey and resulting empirical evaluation carried out by them shows that the SA and SC measures are able to measure the surprise element of inputs much more accurately and therefore are better indicators to ensure how DL based systems are going to react to hitherto unknown inputs. The SA measure correlates to the level of difficulty faced by the DL system in processing a given input, and can therefore be employed to classify accurately various adversarial examples. The SC measure may be employed to guide the selection of inputs to be used for retraining of the DL based systems effectively both for adversarial examples and inputs synthesised by Deep Xplore.

Jie M [7] et al have provided a extensive analysis and review of contemporary research work that has been carried on in the field of ML testing. Their survey helped to present the terminologies and state of art in research in the field of properties, components, and workflow of ML testing. It also helped to summarize the various datasets which were used for the purpose of experiments and also the various open-source testing tools and frameworks available,. They also helped to analyze the trends being followed by research, the research directions, and also the challenges in ML testing. Their survey paved way for researchers streamline their work and contributions towards the pressing problems of ML testing.

Siwakorn Srisakaokul [8] et al have proposed an approach involving testing of learning software using supervised learning methods by using multiple implementations. The results obtained by them on two of the more popular ML methods viz., , k-Nearest Neighbor (KNN) algorithm and Naive Bayes (NB) algorithm, clearly indicate that the majority-voted oracle they use, which is obtained as a result of using multi-implementation testing, can act as an effective and efficient proxy for a given test oracle. The specific majority-voted oracle used by them exhibits low levels of false positives and can therefore detect a larger number of real faults as compared to the benchmark-listed oracle. Specifically, the approach used by them helped to detect a total of 13 real defects and also detected 1 potential fault when applied to a set of 19 KNN implementations. Their approach effectively identified 16 real faults when applied to a set of 7 Naïve Bayes implementations. Moreover, the approach used by them was able to detect 7 real faults and 1 potential fault from the three more popularly employed open-source Machine Learning projects viz., KNIME, Weka, and Rapid Miner.

Youcheng Sun [9] et al proposed some new criteria for testing of DNNs. They have experimented extensively using a wide variety of datasets and also many different methods for generation of test cases and have obtained promising results. Their results have supported the efficiency, effectiveness and suitability of the test criteria proposed by them. The metrics developed for evaluating test coverage. These mainly focus on establishing confidence by obtaining sufficient evidence to support adversarial robustness. The metrics also help the experts of the business domain while deciding on the suitability of particular dataset for applicability in the application

VI. CONCLUSION

The ubiquitous applicability of DL based systems, primarily in non-fault tolerant and mission-critical areas, has resulted in the emergence of a serious threat to the quality of these systems and also to their generalization property. It has become very difficult to measure the testing adequacy of such systems efficiently and effectively and to develop effective techniques and methodologies for the testing of DL based systems. The problem essentially lies in the identification of bugs and defects in the implementation of machine learning based systems. Currently, the testing of ML applications is limited to “validation”, with the tester acting as the human-oracle. The task is made more challenging by the complexity of ML or DL based applications, which makes it very difficult to verifying their output against the specifications. The survey presented by the authors tries to cover the major terminologies and definitions and also to provide an idea of the status of current research being carried out in the field of ML application testing, properties of these Testing methods, components involved in them, and testing workflows.

REFERENCES

[1] Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., & Wang, Y. (2018, October). Deepmutation: Mutation testing of deep learning systems. In 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE), pp. 100-111.

[2] Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., & Zhao, J. (2018, September). Deepgauge: Multi-granularity testing criteria for deep learning systems. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 120-131.

[3] Ma, L., Zhang, F., Xue, M., Li, B., Liu, Y., Zhao, J., & Wang, Y. (2018). Combinatorial testing for deep learning systems. arXiv preprint arXiv:1806.07723.

[4] Dwarakanath, A., Ahuja, M., Sikand, S., Rao, R. M., Bose, R. J. C., Dubash, N., & Podder, S. (2018, July). Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 118-128.

[5] Kim, J., Feldt, R., & Yoo, S. (2019, May). Guiding deep learning system testing using surprise adequacy. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 1039-1049.

[6] Xiu, M., Ming, Z., & Adams, B. (2019). An Exploratory Study on Machine Learning Model Stores. arXiv preprint arXiv:1905.10677.

[7] Zhang, J. M., Harman, M., Ma, L., & Liu, Y. (2019). Machine learning testing: Survey, landscapes and horizons. arXiv preprint arXiv:1906.10742.

[8] Srisakaokul, S., Wu, Z., Astorga, A., Alebiosu, O., & Xie, T. (2018, June). Multiple-implementation testing of supervised learning software. In Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence.

[9] Sun, Y., Huang, X., & Kroening, D. (2018). Testing deep neural networks. arXiv preprint arXiv:1803.04792.

[10] Garousi, V., Felderer, M., Karapıçak, Ç. M., & Yılmaz, U. (2018). Testing embedded software: A survey of the literature. Information and Software Technology, 104, 14-45.

[11] Marcus, G. (2018). Deep learning: A critical appraisal. arXiv preprint arXiv:1801.00631.

[12] Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. Computers and electronics in agriculture, 147, 70-90.

[13] Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. Computers and Electronics in Agriculture, 145, 311-318.

[14] Lusch, B., Kutz, J. N., & Brunton, S. L. (2018). Deep learning for universal linear embeddings of nonlinear dynamics. Nature communications, 9(1), 1-10.

[15] Singh, A. K., Ganapathy subramanian, B., Sarkar, S., & Singh, A. (2018). Deep learning for plant stress phenotyping: trends and future perspectives. Trends in plant science, 23(10), 883-898.

[16] Raissi, M., Wang, Z., Triantafyllou, M. S., & Karniadakis, G. E. (2019). Deep learning of vortex-induced vibrations. Journal of Fluid Mechanics, 861, 119-137.

[17] Kulin, M., Kazaz, T., Moerman, I., & De Poorter, E. (2018). End-to-end learning from spectrum data: A deep learning approach for wireless signal identification in spectrum monitoring applications. IEEE Access, 6, 18484-18501.

[18] Shallue, C. J., & Vanderburg, A. (2018). Identifying exoplanets with deep learning: A five-planet resonant chain around kepler-80 and an eighth planet around kepler-90. The Astronomical Journal, 155(2), 94.

[19] Finlayson, S. G., Chung, H. W., Kohane, I. S., & Beam, A. L. (2018). Adversarial attacks against medical deep learning systems. arXiv preprint arXiv:1804.05296.

[20] Stadie, B. C., Yang, G., Houthooft, R., Chen, X., Duan, Y., Wu, Y., & Sutskever, I. (2018). Some considerations on learning to explore via meta-reinforcement learning. arXiv preprint arXiv:1803.01118.

[21] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2018). Multistep neural networks for data-driven discovery of nonlinear dynamical systems. arXiv preprint arXiv:1801.01236.

[22] Gurovich, Y., Hanani, Y., Bar, O., Nadav, G., Fleischer, N., Gelbman, D., & Bird, L. M. (2019). Identifying facial phenotypes of genetic disorders using deep learning. Nature medicine, 25(1), 60-64..

[23] Takanen, A., Demott, J. D., Miller, C., & Kettunen, A. (2018). Fuzzing for software security testing and quality assurance. Artech House.

[24] Asplund, F. (2019). Exploratory testing: Do contextual factors influence software fault identification? Information and Software Technology, 107, 101-111.

[25] Monperrus, M. (2018). Automatic software repair: a bibliography. ACM Computing Surveys (CSUR), 51(1), 1-24.

[26] Felderer, M., Russo, B., & Auer, F. (2019). On Testing Data-Intensive Software Systems. In Security and Quality in Cyber-Physical Systems Engineering, pp. 129-148. Springer, Cham.

[27] Kim, Y., Hong, S., Ko, B., Phan, D. L., & Kim, M. (2018, April). Invasive software testing: Mutating target programs to diversify test exploration for high test coverage. In 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), pp. 239-249.

[28] Budgen, D., & Tomayko, J. E. (2003, March). Norm Gibbs and his contribution to software engineering education through the SEI curriculum modules. In Proceedings 16th Conference on Software Engineering Education and Training, 2003.(CSEE&T 2003), pp. 3-13.

[29] Scatalon, L. P., Fioravanti, M. L., Prates, J. M., Garcia, R. E., & Barbosa, E. F. (2018, October). A survey on graduates' curriculum-based knowledge gaps in software testing. In 2018 IEEE Frontiers in Education Conference (FIE), pp. 1-8.

[30] Drave, I., Hillemacher, S., Greifenberg, T., Rumpe, B., Wortmann, A., Markthaler, M., & Kriebel, S. (2018, August). Model-Based Testing of Software-Based System Functions. In 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 146-153.

[31] Al-Hajjaji, M., Thüm, T., Lochau, M., Meinicke, J., & Saake, G. (2019). Effective product-line testing using similarity-based product prioritization. Software & Systems Modeling, 18(1), 499-521.

[32] Chen, T. Y., Kuo, F. C., Liu, H., Poon, P. L., Towey, D., Tse, T. H., & Zhou, Z. Q. (2018). Metamorphic testing: A review of challenges and opportunities. ACM Computing Surveys (CSUR), 51(1), 1-27.

[33] Kuchta, T., Palikareva, H., & Cadar, C. (2018). Shadow symbolic execution for testing software patches. ACM Transactions on Software Engineering and Methodology (TOSEM), 27(3), 1-32.

[34] Alghamdi, A. M., & Eassa, F. E. (2019). Software testing techniques for parallel systems: A survey. Int. J. Comput. Sci. Netw. Secur., 19(4), 176-186.

[35] Grano, G., Titov, T. V., Panichella, S., & Gall, H. C. (2018, March). How high will it be? using machine learning models to predict branch coverage in automated testing. In 2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), pp. 19-24.

[36] King, T. M., Arbon, J., Santiago, D., Adamo, D., Chin, W., & Shanmugam, R. (2019, April). AI for testing today and tomorrow: industry perspectives. In 2019 IEEE International Conference On Artificial Intelligence Testing (AITest), pp. 81-88.

[37] Wu, H., Petke, J., Jia, Y., & Harman, M. (2018). An empirical comparison of combinatorial testing, random testing and adaptive random testing. IEEE Transactions on Software Engineering.

[38] Markthaler, M., Kriebel, S., Salman, K. S., Greifenberg, T., Hillemacher, S., Rumpe, B., & Richenhagen, J. (2018, May). Improving model-based testing in automotive software engineering. In 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), pp. 172-180.

[39] Van Deursen, A., Aniche, M., Boone, C., Cunha, M. L., & Nadeem, A. (2019). Software Quality and Testing. Delft University of Technology.

[40] Lemieux, C., & Sen, K. (2018, September). Fairfuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 475-485.

[41] Harman, M., Jia, Y., & Zhang, Y. (2015, April). Achievements, open problems and challenges for search based software testing. In 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pp. 1-12.

[42] Majumdar, R. (2010). Paul Ammann and Jeff Offutt Introduction to Software Testing. Cambridge University Press (2008). ISBN: 978-0-521-88038-1.£ 32.99. 322 pp. Hardcover. The Computer Journal, 53(5), 615-615.

[43] Maciel, C. P., Souza, E. F., Vijaykumar, N. L., Falbo, R. A., Meinerz, G. V., & Felizardo, K. R. (2018). An empirical study on the knowledge management practice in software testing. In Experimental Software Engineering Latin American Workshop (ESELAW'18). XXI Ibero-American Conference on Software Engineering (CIBSE).

[44] Xie, T., Tillmann, N., & Lakshman, P. (2016, May). Advances in unit testing: theory and practice. In Proceedings of the 38th International Conference on Software Engineering Companion, pp. 904-905.

[45] Böhme, M., & Paul, S. (2015). A probabilistic analysis of the efficiency of automated software testing. IEEE Transactions on Software Engineering, 42(4), 345-360.

[46] Clegg, B. S., Rojas, J. M., & Fraser, G. (2017, May). Teaching software testing concepts using a mutation testing game. In 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET), pp. 33-36.

[47] Angell, R., Johnson, B., Brun, Y., & Meliou, A. (2018, October). Themis: Automatically testing software for discrimination. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 871-875.