Using artificial intelligence techniques in software engineering

200036034, 200028234, 200030397, 200009834, and 200030822

Group 4

Introduction

Artificial intelligence (AI) has been developed for nearly 70 years, and a variety of AI technologies such as fuzzy logic, intelligent knowledge systems and data mining have been gradually applied in the field of software engineering to solve many software engineering problems. AI's ultimate goal in the context of software engineering is automatic programming.

Since the creation of the modern computer, software engineering has become the most important part of computer science. Software engineering tasks involve activities throughout all phases of the Software Development Life Cycle (SDLC). Each stage of SDLC can produce a large amount of data which is of great significance in the areas of requirements analysis and specification, system and software design, implementation and unit testing, integration and system testing, and the operation and maintenance stages.

AI technology is changing rapidly and the solutions it provides often look very different from what software organisations and engineers are used to. The application of AI technology in software engineering brings a series of new and unique risks and opportunities to developers which need to be understood and analysed so appropriate strategies can be chosen. This essay will discuss the application of various AI technologies in the stages of software engineering, and analyse the current problems to provide help for software developers in the development process.

State of the art of the artificial intelligence techniques in the software engineering activities

Software specification

Software requirements specification (SRS) establishes the basis for an agreement between customers and suppliers on the functions of software products between customers and software development teams [1]. The purpose of SRS is to capture a complete description about how the system is expected to perform. Software requirements specification is a strict assessment of requirements before a more specific system design stage, and its purpose is to reduce the amount of redesign that could occur in the future. It should also provide a realistic basis for estimating product costs, risks and schedules.

First of all, the requirement analysis is the most important and basic stage in the software development life cycle. It is conducted by senior members of the team on the basis of the opinions of experts, from customers, sales departments, market surveys and industry fields. Then, the basic project method of information planning is used to study the feasibility of products in the fields of economy, operation and technology. The planning of quality assurance requirements and risk identification related to the project are also carried out in the planning stage [2].

Once the requirement analysis is done, the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an SRS document which consists of all the product requirements to be designed and developed during the project life cycle [2].

With the development of computer performance, human beings have ushered in the era of AI in the 21st century. In the past decade, machine learning (ML) has made a big advance as well as natural language processing (NLP) like Word2Vec and Doc2Vec. Many of the improved results have come from larger and more complex neural networks which are stacked many layers deep, but much of the progress can also be attributed to larger data sets and large-scale learning/training on graphics processing units [3].

At present, the fully automatic software requirements specification part has not been realised. However, with the development of annotation learning using Convolutional Neural Networks (CNN), the content of requirements can be learned. After learning, the system will select the similar requirements and provide them to customers and development teams, and provide the optimal solution according to the weight of each requirement in the project.

Software engineering will help to extract useful knowledge to identify and select potential reusable candidates, and the integration of AI technology will help to apply intelligence to the above process [4]. An example of this can be seen in an article by Tetsuo Tamai and Taichi Anzai called "Quality Requirements Analysis with Machine Learning" [5]. In this paper, 11538 cases of requirement sentences used in medical systems are collected from Japanese local governments and other public institutions. After tagging these requirement statements with aspects of speech and specific semantics, according to the eight normative characteristics of functional applicability, performance efficiency, compatibility, availability, reliability, security, maintainability and portability, the requirements statements are classified into four categories: requirements on external interface, user interface, system functions and finally, requirements on database. Finally, the training of the ML system is completed by a Convolution Neural Network. It shows the distribution of users' requirements in terms of structure, reliability, and their distribution in the whole document [5].

The case also proves the role of machine learning and data mining in software specification: [5]

1. Acquisition phase. When the acquisition requirements are not complete, we can complete the requirements according to the rough description of the

requirements, and can also deeply check the problems existing in the requirements.
2. Document preparation stage. Similarly, since incomplete requirements documents can be analysed with this tool, the results of data mining can be used to provide feedback when writing SRS.
3. Evaluation stage. In the evaluation stage, when the SRS is completed and submitted to the quality assurance team for review, it can be reviewed according to the analysis results.
4. Another approach. Artificial intelligence can help determine specifications. This can accurately and effectively express complex requirements. With the increase of ML learning samples, AI can easily help us to complete three different types of modeling needed in software engineering: (1) mathematical model (2) functional model (3) model to distinguish applications; that is, a linear programming model and the software model needed to be implemented [2].

Software development

The design and implementation phase of a software development project consists of closely examining the specification and then turning it into a perceptible and usable system. The main activity that is carried out in this phase is programming, which can either be done by an individual or a large team of people working on different parts of the system [6]. AI has had a huge impact on many aspects of this process in both the areas of writing software and testing the code to ensure it is clean and well written [7].

One example of how AI can influence the software design and implementation phase is in the field of software intelligence. Software intelligence is the process of collecting data based on software systems to gain a better understanding of what makes maintainable and efficient code [8]. Over the past decade, the procedure of extracting software data and using it in a meaningful manner has been made autonomous and used for activities such as the architectural design of a system and development testing [9]. Research has been carried out into the field of programming assistants which utilise artificial neural networks to advise programmers in real-time, making suggestions about the code they are writing and making slight adjustments when necessary [10]. Another key area in which AI utilises software intelligence to help in the implementation phase of a project is automated software reuse. The different ML methods for this process are outlined in a 2018 conference paper [11]. The techniques outlined in the paper include clustering, classification and decision tree algorithms which can be used to identify suitable code and software and then recommend its optimal usage to the developers. In the context of a commercial project, this could help make the software design and implementation a lot more efficient as it would recommend code sources and libraries that the developers could use which would save a lot of time for the overall enterprise [12].

Over the course of writing a software application, a developer often needs to refactor the existing code. Refactoring is the action of changing source code in an attempt to improve its readability, effectiveness as well as it's maintainability [13]. This can be a long and difficult process depending on the size of the code base but in recent years, many propositions and attempts have been made to automate the process using AI. [14] One such example can be observed in [15], which discusses refactoring code at the basic level of method names. The proposed system utilised a Convolutional Neural Network which is a type of technique that is often used for language processing tasks and the classification of text [16]. Whilst the study produced positive conclusions, it exclusively focused on refactoring method names which consists of single identifiers and does not address how a whole code base made up of statements, terms and expressions could be automatically refactored.

The initial stage of refactoring code in any context is finding code smells, a term that is used to refer to code which does not meet suitable programming standards [17]. One proposed method of making the software implementation process faster and more efficient is real-time code smell detection. This might take the form of an application or a feature in an integrated development environment that would tell the programmer that they are writing sub-standard code based on a trained machine learning model. Whilst bad programming practices can be apparent to experienced software developers, there is still an underlying level of subjectivity involved in detecting code smells which has been a limitation in past research [18]. An article by Thirupathi Guggulothu and Salman Abdul Moiz called "Code smell detection using multi-label classification approach" [19] aimed to solve the issue of subjectivity in code practices by using a ML algorithm that classified code instances based on multiple identifiers. The research that was carried out was successful in finding a classifier that could accurately predict code smells and it also made progress in finding solutions to solving real-time code smell detection.

Software validation

AI started playing an important role in software validation due to the growing maturity of AI's algorithms and techniques as well as the top-notch technology that allowed computers to have incredible speed and memory. Software production and its demand is continuously growing, and organisations look for better ways to meet their challenges and to be ahead of the competitors, and so the time that is required to develop software can be shortened by using AI in software testing.

Source code, databases, inputs and outputs and any other component in software is made up of data. An advantage of using AI in software testing is that it can handle large amounts of data easily and effectively by using specific algorithms and techniques, which is something humans are not very good at. Different methods can be applied by AI for software testing purposes like classifications, regression, clustering and dimensionality reduction [20].

There are many ways in which different AI techniques can be implemented during the validation process of a software project. Researchers have used and combined many algorithms and techniques to target specific jobs in software testing and achieved competitive results. Some of these techniques include: Constraint-Based testing, AI planning and fuzzy logic.

Constraint solving techniques are used to help improve the automation of software testing, and because of its adaptability, it has gotten much attention by researchers. AI planning was developed during the mid-1990's, and this area of research was used to generate test cases, consisting of "a sequence of commands by representing commands as operators, providing initial states and setting the goal as testing for correct system behavior" [21]. The same technique was also used for testing distributed systems and for the generation of test cases for graphical interfaces. In addition, another AI technique can be used in software testing using fuzzy logic to manage the uncertainty found during this software development phase.

Companies have invested largely in the use of AI in the software testing industry and the implementation of more AI techniques in this industry looks very promising. Within the next few years, more AI contributions in the area of software testing are expected to be implemented [20]:

- Software development life cycles will be shorter using AI techniques than with traditional testing methods, hence organisations will be more effective and produce more accurate results. Meeting deadlines is something that concerns all software developers, so they can benefit from these techniques, since AI will allow software validation time frames to be a lot faster.
- It is expected that specialised software will use AI deep learning and other AI algorithms to get more accurate results during testing within reasonable time frames.
- Software products will be more robust and reliable, thanks to the AI predictive analytics that will allow for the discovery of all possible test cases. This can even exceed customer expectations.

Software validation using AI techniques also faces multiple problems, mainly because the process of testing data generation is extremely difficult. There has been significant progress to achieve the challenging goal of having full automation in the validation software process. However, there are some techniques that are being developed that are often hampered by certain features of the program that is being tested. The lack of handling of the software's execution environment is the main problem with Search-Based software testing techniques and particularly, with Search-Based test data generation techniques. The latest methods for data generation testing "ignores or fails to handle interactions with the underlying operating system, the file system, network access and databases on which they may be dependent" [21]. In addition, programmers have to deal with thousands of lines of code, with dynamic constructions such as huge dynamic data structures, with non-linear numerical constraints extracted from complex statements that affect some AI techniques in software validation like scalability in the Constraint-Based testing technique.

In current times, more devices and ML applications are being connected to the internet and the number of mobile apps created are growing exponentially, and so is the cost for the validation of the software for these devices. It is now more important than ever to automate software verification in order to make software work in a safe predictable manner.

Software evolution

The last activity in software engineering is software evolution (maintenance). It is crucial for the continuity of the software given the changes based on the stakeholders and/or organisations decisions. Maintenance is a constant activity in software development and testing teams inside a company. In that sense, software evolution tackles the resolution of bugs and improvement in the code [22]. But this approach is time and cost consuming with detrimental results such as bug report duplication, lack of understanding in the severity of the bug reports and deficit in bug estimation. For this reason, AI [23] could be a great alternative to solve these issues in the software evolution activity.

Currently, when a bug is found in the software department, it is reported so it can be fixed. This procedure in some cases causes bug report duplication. For the reliability and maintainability of a software system, AI has been used to mitigate and detect this duplication. For instance, a method [24] was presented on software literature context method, where if given a certain word from a word-list, would improve the detection of bug report duplication. It applied different ML techniques with data sets from Android, Eclipse, Open Office and Mozilla. Another approach is proposed in [25], the data from the bug tracking system Jira were processed in Natural Language Processing (NLP), followed by word embedding (tokenised words to vectors), and finally this was applied as an input layer in CNN. One disadvantage is that the results cannot be generalised because it was just evaluated with four data sets.

The constraints in a company are crucial for selecting the severity of a bug to fix it, in some cases, the members of the team have no experience, and the outcome is a manual setup fix instead of an automatic fix. Thus, a group of authors [26] used different algorithms to predict the severity of bug reports. It extracted attributes such as bug ID, reported email ID, severity, product name, browser, etc from Bugzilla bug repositories. Two ML algorithms were used: bagging ensemble method and C4.5 algorithm. The former given, its design and structure, obtained a higher accuracy than C4.5 [26]. Yet, a more detailed explanation with a variety of ML techniques should be recommendable in contrast to the bagging ensemble method. In [27] a variety of ML algorithms are presented: Naive Bayes, RBF Networks, Functional Trees, Random Tress, Random Forests, and AdaBoost with the class being predicted as severe or non-severe. The Adabost algorithm has been excellent in improving the classifier performance.

On the other hand, a large number of releases of the software are going to have bugs. These are detrimental for new updates. The goal to predict the required time to fix the bugs has a tremendous benefit to a company and its planning in the software evolution activity. Thus, these researchers [28] decided to contrast different ML techniques to predict the time in fixing the bug and the name of the developer. The data extracted for this research is from Kaggle with the attributes: bug status, bug's description, bug initial and end date, among others. The algorithms tested were J48 (Decision Tree Algorithm), Random Forest, and SVM (Support Vector Machine) - linear, polynomial, radial, and sigmoid [28]. The accuracy was more than sixty percent; however, it can be improved with the selection of new attributes, and with a specific case study of different industries. An innovative technique is stated in [29], the researchers develop an NLP model known as BERT (Bidirectional Encoder Representations from Transformers) to perform the prediction of bug-fixing time. This model uses a deep learning approach with the use of transfer learning (a neural network model) commonly used in computer vision. BERT took data from bug reports from the Bugzilla open source platform related to the description of the issue and developer's comments, and the bug-fixing time was classified into fast and slow classes [29]. A point to enhance is the validation of the data to find the time to fix the problem, because the researchers make an assumption [29] by calculating a uniform distribution of developers work, but it is not generalising the real contribution of the developers.

Challenges, and future trends of artificial intelligence techniques in software engineering

With the continuous development of artificial intelligence, mankind faces not only technical difficulties but also many future uncertainties.

The main idea in the 90s was to create an expert system to assist software engineers during software development [30]. In [31], this proposal is called the programmer's apprentice project. The purpose of this project is to achieve human-computer interaction, which is similar to the relationship between programmers and assistants in reality. This would help to achieve an increase in productivity. At first, this expert system could only implement the simplest directional programming. With training, the expert system could implement more complex programming, but the key logic processing still needs to be processed by human programmers. The programmer could provide a logical idea, allowing the expert system to focus on generating functions, data structures, and even the entire program [32]. At this stage, the problems that AI can handle in software engineering are [31]:

1. Disambiguating natural language requirements.
2. Developing knowledge-based systems and ontologies to manage the requirements and model problem domains.
3. The use of computational intelligence to solve the problems of incompleteness and prioritisation of requirements.

One area of ML that is being used to solve these problems is deep learning (deep structured learning), which includes supervised, semi-supervised, and unsupervised learning [33]. The biggest challenges of deep learning at present are [34]:

1. A large amount of labeled data is required for deep learning to make it as efficient as possible, which makes researchers carry out studies in fields with rich data resources, instead of doing research in areas which may be deemed more important. Although there are some methods that can reduce the dependence on data, such as transfer learning, few sample learning, unsupervised learning, and weakly supervised learning. But so far, their performance has not been comparable to supervised learning.
2. Overfitting the benchmark data. Deep neural networks perform well on benchmark data sets, but on real-world images outside the data sets, the effects are not satisfactory.
3. Excessively sensitive to image changes. Deep neural networks are very sensitive to standard adversarial attacks. These attacks can cause changes in images that are imperceptible to humans but may change the neural network's perception of an object. Moreover, the neural network is too sensitive to changes in the scene.

For deep neural networks to handle all the problems, it seems that an infinite data set is needed, which brings huge challenges to training and testing data sets.

What impact do these have on software engineering? It can be considered from the software life cycle. AI can automate requirements analysis, code analysis, coding, software testing, and fault diagnosis [35]. In AI, uncertainty appears on the human side. Specifically, the uncertainty of knowledge and of software development. The uncertainty of knowledge is mainly reflected in the uncertainty of common-sense knowledge and language. Among them, common sense knowledge can be expressed in natural language, and the corresponding concepts show obvious uncertainties such as vagueness and randomness. There is also uncertainty about software quality. The mainstream of computer software engineering is object-oriented technology and methods, which mainly include object design, object analysis, and object realisation. However, with the continuous improvement of software complexity and the continuous increase of software scale, the quality of software products has gradually become more difficult to control and grasp. The implementation process of each sub-project can be regarded as a process of human-computer interaction. In human-computer interaction, the human factor is a particularly important influencing factor. The reliability of the actual software is mainly affected by the operator. Therefore, until now, AI has not been able to play its maximum role in software engineering.

Conclusion

In conclusion, there are a number of AI techniques that can be utilised in the software development process which consists of the specification, development, validation and evolution of a project. Different ML algorithms such as Neural Networks and Decision Trees have been implemented in a variety of applications in software engineering from SRS processing to bug estimation time prediction during the evolution phase. A large amount of data has been processed by researchers around the world to solve the challenges in each one of the activities, with the goal of providing better solutions in the software development cycle. As a result, AI will benefit the customers, organisations and businesses to create optimal and automatic processes. On the other hand, AI still faces many challenges. For example, labeling data is expensive, deep learning requires a huge amount of data which might not be available in small scale projects and large calculations require expensive material and potentially large time costs. Whilst these issues are yet to be resolved, AI is making an impact on the software engineering landscape and will continue to do so for the following decades.

Word count: 3839

References

[1] Pierre Bourque, Richard E Fairley, et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0.* IEEE Computer Society Press, 2014.

[2] VIPAN Kumari and SANDEEP Kulkarni. "Use of Artificial Intelligence in Software Development Life Cycle: Requirements and Its Model". In: *Int. Res. J. Eng. Technol.(IRJET)* 5.8 (2018), pp. 398–403.

[3] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural networks* 61 (2015), pp. 85–117.

[4] "IEE Colloquium on 'Software Engineering and AI (Artificial Intelligence)' (Digest No.087)". In: *IEE Colloquium on Software Engineering and AI (Artificial Intelligence) (Digest No.087).* 1992, 0_1–.

[5] Tetsuo Tamai and Taichi Anzai. "Quality Requirements Analysis with Machine Learning." In: *ENASE.* 2018, pp. 241–248.

[6] Nabil Mohammed Ali Munassar and A Govardhan. "A comparison between five models of software engineering". In: *International Journal of Computer Science Issues (IJCSI)* 7.5 (2010), p. 94.

[7] Mohammad Shehab, Laith Abualigah, Muath Ibrahim Jarrah, et al. "Artificial intelligence in software engineering and inverse". In: *International Journal of Computer Integrated Manufacturing* (2020), pp. 1–16.

[8] Ahmed E Hassan and Tao Xie. "Software intelligence: the future of mining software engineering data". In: *Proceedings of the FSE/SDP workshop on Future of software engineering research.* 2010, pp. 161–166.

[9] Vaibhav Narayan. "The Role of AI in Software Engineering and Testing". In: *International Journal of Technical Research and Applications* (2018).

[10] Thomas Dean, Maurice Chiang, Marcus Gomez, et al. "Amanuensis: The Programmer's Apprentice". In: *arXiv preprint arXiv:1807.00082* (2018).

[11] Divanshi Priyadarshni Wangoo. "Artificial intelligence techniques in software engineering for automated software reuse and design". In: *2018 4th International Conference on Computing Communication and Automation (ICCCA).* IEEE. 2018, pp. 1–4.

[12] Majdi Rawashdeh, Awny Alnusair, Muder Almiani, et al. “JMentor: An Ontology-Based Framework for Software Understanding and Reuse”. In: *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*. IEEE. 2019, pp. 1–8.

[13] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[14] Marvin Wyrich and Justus Bogner. “Towards an autonomous bot for automatic source code refactoring”. In: *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE. 2019, pp. 24–28.

[15] Kui Liu, Dongsun Kim, Tegawendé F Bissyandé, et al. “Learning to spot and refactor inconsistent method names”. In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE. 2019, pp. 1–12.

[16] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. “Understanding of a convolutional neural network”. In: *2017 International Conference on Engineering and Technology (ICET)*. IEEE. 2017, pp. 1–6.

[17] Eva Van Emden and Leon Moonen. “Java quality assurance by detecting code smells”. In: *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.* IEEE. 2002, pp. 97–106.

[18] Dario Di Nucci, Fabio Palomba, Damian A Tamburri, et al. “Detecting code smells using machine learning techniques: are we there yet?” In: *2018 ieee 25th international conference on software analysis, evolution and reengineering (saner)*. IEEE. 2018, pp. 612–621.

[19] Thirupathi Guggulothu and Salman Abdul Moiz. “Code smell detection using multi-label classification approach”. In: *Software Quality Journal* (2020), pp. 1–24.

[20] H. Hourani, A. Hammad, and M. Lafi. “The Impact of Artificial Intelligence on Software Testing”. In: *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*. 2019, pp. 565–570.

[21] Hany Ammar, Walid Abdelmoez, and Mohamed Hamdi. “Software Engineering Using Artificial Intelligence Techniques: Current State and Open Problems”. In: (Mar. 2012).

[22] Meir M. Lehman and Juan F. Ramil. “Software Evolution and Software Evolution Processes”. en. In: *Annals of Software Engineering* 14.1 (Dec. 2002), pp. 275–309. ISSN: 1573-7489. DOI: 10.1023/A:1020557525901. URL: https://doi.org/10.1023/A:1020557525901 (visited on 10/09/2020).

[23] Lucia, David Lo, Giuseppe Scanniello, et al. “Leveraging machine learning and information retrieval techniques in software evolution tasks: summary of the first MALIR-SE workshop, at ASE 2013”. In: *ACM SIGSOFT Software Engineering Notes* 39.1 (Feb. 2014), pp. 1–2. ISSN: 0163-5948. DOI: 10.1145/2557833.2560584. URL: https://doi.org/10.1145/2557833.2560584 (visited on 10/09/2020).

[24] Karan Aggarwal, Tanner Rutgers, Finbarr Timbers, et al. "Detecting duplicate bug reports with software engineering domain knowledge". In: ISSN: 1534-5351. Mar. 2015, pp. 211–220. DOI: 10.1109/SANER.2015.7081831.

[25] Qi Xie, Zhiyuan Wen, Jieming Zhu, et al. "Detecting Duplicate Bug Reports with Convolutional Neural Networks". In: *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. ISSN: 2640-0715. Dec. 2018, pp. 416–425. DOI: 10.1109/APSEC.2018.00056.

[26] M N Pushpalatha and M Mrunalini. "Predicting the severity of bug reports using classification algorithms". In: *2016 International Conference on Circuits, Controls, Communications and Computing (I4C)*. Oct. 2016, pp. 1–4. DOI: 10.1109/CIMCA.2016.8053276.

[27] Ahmed Fawzi Otoom, Doaa Al-Shdaifat, Maen Hammad, et al. "Severity prediction of software bugs". In: Apr. 2016, pp. 92–95. DOI: 10.1109/IACS.2016.7476092.

[28] Rucha Sawarkar, Naresh Kumar Nagwani, and Sanjay Kumar. "Predicting Bug Estimation Time for Newly Reported Bug Using Machine Learning Algorithms". In: *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*. Mar. 2019, pp. 1–4. DOI: 10.1109/I2CT45611.2019.9033749.

[29] Pasquale Ardimento and Costantino Mele. "Using BERT to Predict Bug-Fixing Time". In: *2020 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*. ISSN: 2473-4691. May 2020, pp. 1–7. DOI: 10.1109/EAIS48028.2020.9122781.

[30] Phil for Humanity. *The Use of Artificial Intelligence for Program Development*. Jan. 2010. URL: https://www.philforhumanity.com/The_Use_of_Artificial_Intelligence_for_Program_Development.html.

[31] Derek Partridge. *Artificial Intelligence in Software Engineering*. Jan. 2002. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471028959.sof013.

[32] Divyanshi Kothari. "How Artificial Intelligence Accelerates Software Development". In: *International Research Journal of Engineering and Technology* 06.08 (Aug. 2019), pp. 1392–1394.

[33] Y. Bengio, A. Courville, and P. Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013), pp. 1798–1828. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2013.50.

[34] Alan L. Yuille and Chenxi Liu. *Deep Nets: What have they ever done for Vision?* Jan. 2019. URL: https://arxiv.org/abs/1805.04025.

[35] Farid Meziane and Sunil Vadera. *Artificial intelligence applications for improved software engineering development: new prospects*. Information Science Reference, 2010.