



PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education and Scientific Research
Mohamed Khider University – BISKRA
Faculty of Exact Sciences, Natural Sciences and Life
Department of Computer Science

Order n° : RTIC 13/M2/2022

Memory

Presented for the Academic Master's degree in

Computer Science

Option : Information and Communication Networks and Technologies (RTIC)

Vulnerability detection using deep learning

By :

BAKIR MOUNA

Defended on 26 / 06 / 2022, in front of the jury composed of :

Berghida Meryem	MCB	President
Boukhlof Djemaa	MCB	Supervisor
Benameur Sabrina	MCB	Examiner

Thanks

First and foremost, we thank Allah Almighty for wisdom and infinite knowledge. We would like to express our thanks to our framer Boukhlouf Djemaa , We would also like to express our gratitude to him for his patience and support that has been precious to us in order to make our work Harbor.

We send our sincere thanks to all the teachers who helped to improve our work and all the people who by their advice and their critics who guided our thinking and agreed to meet us and answer to our questions.

We would like to thank everyone who has supported us closely far, throughout this year, who will recognize themselves.

Finally, thank you in advance for those who would like to read us or us listen and above all help us to progress.

Dedication

I do offer my modest work to my parents, my brother and my sisters for their love, moral support, and outstanding patience during this long journey. And also to all my friends and classmates. Lastly, I offer my regards and blessing to all those who supported me in one way or another during the completion of this work.

Abstract

The detection of software vulnerabilities (or vulnerabilities forection. Deep learning is attractive for this purpose because it alleviates the requirement to manually define features. Despite the tremendous success of deep learning in other application domains, its applicability to vulnerability detection is not systematically understood. In order to fill this void, we propose framework for using deep learning to detect vulnerabilities in C/C++ programs with source code.

short) is an important problem that has yet to be tackled, as manifested by the many vulnerabilities reported on a daily basis. This calls for machine learning methods for vulnerability det In this work, methods to detect software gaps were mainly performed on function-level function codes C/C + +, We have conducted extensive experiments against a wide range of real-world weaknesses C/C + + obtained from many real-world projects and specifically propose a synthetic neural network for code analysis in which we propose a set of data imported from the Internet with the addition of data manually with the proposal of a model for the inclusion of a normal and unusual function.

keywords: Software Vulnerability Detection,security, Deep Learning, CNN

Résumé

La détection des vulnérabilités logicielles (ou des vulnérabilités en abrégé) est un problème important qui n'a pas encore été abordé, car manifestée par les nombreuses vulnérabilités rapportées quotidiennement. Cela nécessite des méthodes d'apprentissage automatique pour la détection des vulnérabilités. L'apprentissage profond est intéressant à cet effet, car il allège l'exigence de définir manuellement les fonctionnalités. Malgré l'énorme succès du deep learning dans d'autres domaines d'application, son applicabilité à la détection des vulnérabilités n'est pas systématiquement comprise. Afin de combler ce vide, nous proposons un cadre pour utiliser le deep learning pour détecter les vulnérabilités dans les programmes C/C++ avec du code source.

Dans ce travail, les méthodes de détection des lacunes logicielles ont été principalement réalisées sur les codes de fonction au niveau de la fonction C/C++, Nous avons mené des expériences approfondies contre un large éventail de faiblesses du monde réel projets mondiaux et spécifiquement proposer un réseau neuronal synthétique pour l'analyse de code dans lequel nous proposons un ensemble de données importées de l'Internet avec l'ajout de données manuellement avec la proposition d'un modèle pour l'inclusion d'une fonction normale et inhabituelle.

Mots-clés : Détection des vulnérabilités logicielles, sécurité, profondeur Apprentissage, CNN

Table of contents

General Introduction	1
1 Deep Learning	3
1.1 Introduction	3
1.2 What is artificial intelligence?	3
1.3 What is machine learning?	4
1.3.1 Support vector machines (SVMs)	5
1.4 What Is deep learning ?	5
1.5 Deep learning terminologies	6
1.5.1 Recurrent neuron	6
1.5.2 Vanishing Gradient Problem	7
1.5.3 Exploding gradient problem	7
1.5.4 max-pooling	7
1.5.5 Softmax	7
1.5.6 Neural network	8
1.5.7 Neuron	9
1.5.8 Activation function	10
1.5.9 Learning rate	10
1.5.10 Back propagation	11
1.6 Evolution of deep learning	11
1.7 Deep learning approaches	11
1.7.1 supervised learning	11
1.7.2 Unsupervised learning	12
1.7.3 Semi supervised learning(hybrid learning)	13
1.7.4 Deep reinforcement learning	14
1.7.5 Algorithms for Deep learning	14
1.8 Fundamental deep learning architectures Deep learning	15
1.8.1 Unsupervised Pre-trained Networks	15
1.8.2 Convolutional Neural Networks	17
1.8.3 Recurrent Neural Networks	17
1.9 Deep learning methods	18
1.9.1 Back propagation	18
1.9.2 Stochastic gradient descent	19
1.9.3 Learning Rate Decay	19

1.9.4	Dropout	19
1.9.5	Max-Pooling	19
1.9.6	Batch Normalization	19
1.9.7	Skip-gram	19
1.9.8	Transfer learning	20
1.10	frameworks deep learning	20
1.10.1	TensorFlow	20
1.10.2	Keras	21
1.10.3	Py Torch	21
1.10.4	Caffe	21
1.10.5	Deeplearning4j	21
1.11	The Applications of Deep Learning	21
1.11.1	Automatic Speech Recognition (ASR)	21
1.11.2	Game Playing	22
1.11.3	Autonomous Driving	22
1.11.4	Chatbots	22
1.11.5	Image captioning	23
1.11.6	News Aggregation and Fake News Detection	23
1.11.7	Text to Speech	23
1.11.8	Machine Translation	23
1.12	Conclusion	23
2	Software vulnerability detection	24
2.1	Introduction	24
2.2	Security objective	24
2.2.1	Confidentiality	24
2.2.2	Integrity	24
2.2.3	Authentication	24
2.2.4	Non-Repudiation	24
2.2.5	Availability	25
2.3	Terminology	25
2.3.1	Vulnerability	25
2.3.2	Attack(Exploit)	25
2.3.3	Threat	25
2.3.4	Software	25
2.3.5	Counter measure	26
2.3.6	Risk	26
2.4	The Open Web Application Security Project (OWASP) Top Ten	26
2.4.1	What is the OWASP?	26
2.4.2	Top Ten 2021 List :	26
2.5	National vulnerability database (NVD)	29
2.5.1	Common Vulnerabilities and Exposures (CVE)	29
2.6	SANS Top 25 Security Vulnerabilities In Software Applications	29
2.6.1	SANS	29
2.6.2	What the term CWE means?	30
2.6.3	The CWE Top 25	30
2.7	Vulnerability Detection methods	36

2.7.1	Fuzzing	36
2.7.2	Web Application Scanner	36
2.7.3	Static Analysis Techniques	36
2.7.4	Brick	36
2.7.5	CRED: C Range Error Detector	37
2.8	Related Work	37
2.8.1	VUDENC: Vulnerability Detection with Deep Learning on a Natural Code base for Python	37
2.8.2	VulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection	38
2.8.3	SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities	40
2.9	Conclusion	41
3	Design	42
3.1	Introduction	42
3.2	General system design	42
3.2.1	System objective	42
3.2.2	Architecture of the system	42
3.3	Detailed system design	43
3.3.1	Data collection	43
3.3.2	Data preparation	44
3.3.3	Classification and Training	45
3.3.4	Use of the Model	46
3.4	Design by UML	47
3.4.1	Sequence Diagram for "Inscription"	47
3.4.2	Sequence diagram "authenticate"	48
3.5	Conclusion	48
4	Implementation and results	49
4.1	Introduction	49
4.2	Environment and tools	49
4.2.1	Python	49
4.2.2	WAMP(Windows, Apache, MySQL, PHP)	50
4.2.3	TensorFlow	50
4.2.4	Keras	50
4.2.5	TfidfVectorizer	51
4.2.6	Tkinter	51
4.3	Data Structures	51
4.3.1	Data base	51
4.3.2	Dataset Used	52
4.3.3	Training and Testing	53
4.4	Our application interfaces	57
4.4.1	Interface "Login"	57
4.4.2	Interface "Registration"	57
4.4.3	Main interface	58
4.5	Conclusion	59

List of Figures

1.1	types of IA.[48]	4
1.2	Machine Learning.[34]	4
1.3	Types of Machine Learning.[36]	5
1.4	The relationship between artificial intelligence, ML and deep learning.[9]	6
1.5	machine or deep learning process.[13]	6
1.6	Recurrent Neuron.[40]	7
1.7	Pooling in Deep Learning.[59]	7
1.8	Softmax.[59]	8
1.9	Neural Network.[59]	8
1.10	Hidden layer in deep Learning.[33]	9
1.11	MLP (Multi-Layer perceptron) in Deep Learning.[70]	9
1.12	Neuron in Deep Learning.[33]	10
1.13	Activation Function.[33]	10
1.14	Learning Rate.[33]	11
1.15	Artificial Intelligence and Machine Learning and DL[2]	12
1.16	supervised learning .[17]	12
1.17	unsupervised learning.[19]	13
1.18	hybrid learning(SSL).[16]	13
1.19	hybrid learning(SSL).[16]	14
1.20	Autoencoders.[3]	15
1.21	Deep Belief Networks.[8]	16
1.22	Generative Adversarial Networks.[12]	16
1.23	Convolutional Neural Networks.[7]	17
1.24	Recurrent Neural Networks(RNN).[15]	18
1.25	DL in an audio recording [87]	22
1.26	Game Playing [11]	22
2.1	Types Of Software[6]	26
2.2	Changes in Top Ten Risks for 2021 .[65]	27
2.3	CVE[20]	29
2.4	2-CWE-125[75]	32
2.5	CVE-79[76]	32
2.6	CWE-20[75]	32
2.7	CWE-78[76]	33

2.8	CWE-416[75]	33
2.9	CWE-22[75]	34
2.10	CWE-190[75]	34
2.11	2-CWE-352[76]	35
2.12	1-CWE-434[76]	35
2.13	2-CWE-434[76]	35
2.14	Performance Comparison of Vulnerability Detection Methods.[23]	37
2.15	Overview of the Vudenc approach	38
2.16	Overview of VulDeePecker’s three modules (parser, vector representation extractor, and detector) used in the training and detection phases.	39
2.17	The μ VulDeePecker neural network architecture.	40
2.18	5. Experimental comparison between the effectiveness of μ VulDeePecker and VulDeePecker+ with respect to each of the 40 vulnerability types.	40
2.19	SySeVR: a framework for using deep learning to detect vulnerabilities	41
3.1	Architecture of the system.	43
3.2	Architecture of the CNN model.	43
3.3	Distribute our data with/without unbalanced data problem.	44
3.4	Example of TfidfVectorizer.	45
3.5	Train the Model.	45
3.6	Use of the Model.	46
3.7	Sequence Diagram for ”Inscription ”.	47
3.8	Sequence Diagram for ”authenticate ”.	48
4.1	Users list.	52
4.2	vulnerable data set	52
4.3	Non vulnerable data set	52
4.4	unbalanced data	53
4.5	balanced data.	53
4.6	building layers.	54
4.7	Building layers.	54
4.8	accuracy result(Model Accuracy,Model Loss)	55
4.9	Comparison accuracy result.	55
4.10	Matrix Of CNN Model.	56
4.11	Login Interface.	57
4.12	Registration Interface.	58
4.13	Testing phase.	59

List of Tables

1.1	Comparison of deep learning methods.[45]	20
1.2	Comparison of Deep Learning Frameworks.[45]	21
2.1	The CWE Top 25.[75]	31
3.1	Metadata of the Collected Data set.	44
4.1	CNN Empirical Results Analysis and LSTM for the same dataset	56

General introduction

Vulnerabilities in software are inevitable because writing them securely is very difficult and requires a lot of experience and professionalism. As humans are prone to error, even experienced and elite developers can make software errors lead to serious consequences for information security and are an underlying cause of the spread of attacks. Despite many developers seeking to improve software quality but this problem remains much bigger than everyone can imagine, and the evidence of this is that we record daily vulnerabilities .

Since vulnerabilities are inevitable, they must be discovered as soon as possible. Identifying potential vulnerabilities in programs is a decisive step in defence against cyberattacks, however, it can be difficult and it takes a long time. Since most systems are written in C, these vulnerabilities will cause the destruction of large systems.

Although previous studies have proposed various methods and tools that can be used in reducing or eliminating software vulnerabilities. Deep learning has received a lot of attention recently, so it has achieved results that have never been achievable in areas at a level of precision where the human being is outperformed by thousands of times, and in short when we want precision in use and achieve results it is an ideal solution.

In this study, through deep learning outcomes and more accurately Convolutional Neural Network (CNN), we want to apply to the field of vulnerabilities more specifically, we choose to recognize security gaps on C codes because many systems are written in them. For thus, we have collected a large number of C codes containing two natural and abnormal species then we train Convolutional Neural Network (CNN) to eventually get a trained model that can predict that the codes were safe first. After that, we have tested our model with different codes, and the obtained results gives a best accuracy.

We have divided the manuscript into four chapters:

- Chapter one: introduce basic concepts of deep learning and its approaches and methods.
- Chapter two: present top 10 important vulnerabilities according to the World Security Organization (OWASP), National Vulnerability Database NVD, and the different vulnerability detection methods. We present also the related work.
- Chapter three: Focus on general system design, detailed system components, model of test and

UML diagrams.

- Chapter four: intended for implementation details and obtained results.

We end our manuscript with a general conclusion.

Deep Learning

1.1 Introduction

In the last few years, the deep learning (DL) computing paradigm has been deemed the Gold Standard in the machine learning (ML) community. Moreover, it has gradually become the most widely used computational approach in the field of ML, thus achieving outstanding results on several complex cognitive tasks, matching or even beating those provided by human performance. One of the benefits of DL is the ability to learn massive amounts of data. The DL field has grown fast in the last few years and it has been extensively used to successfully address a wide range of traditional applications. More importantly, DL has outperformed well-known ML techniques in many domains, e.g., cybersecurity, natural language processing, bioinformatics, robotics and control, and medical information processing, among many others. so we can say In the field of ML, DL, due to its considerable success, is currently one of the most prominent research trends. In this chapter we present the basics of deep learning. [22]

1.2 What is artificial intelligence?

Artificial intelligence (AI) is defined as ‘a field of science and engineering concerned with the computational understanding of what is commonly called intelligent behaviour, and with the creation of artefacts that exhibit such behaviour’.[80] There are 4 types of IA here:

- Reactive Machines
- Limited Memory
- Theory of Mind
- Self Awareness

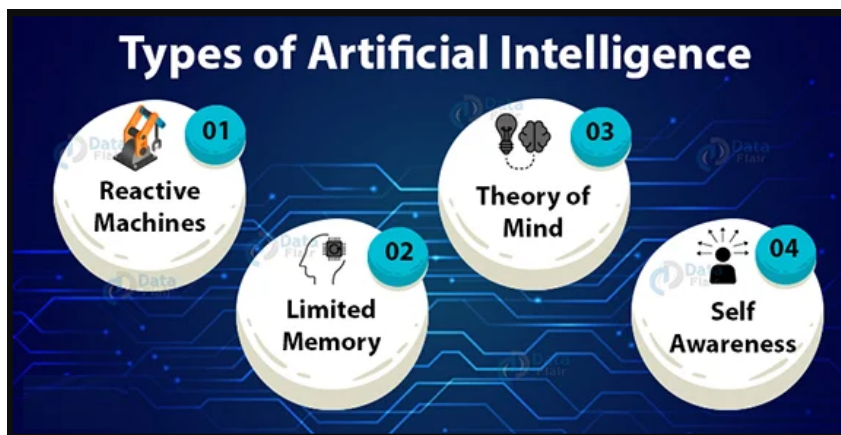


Figure 1.1: types of IA.[48]

1.3 What is machine learning?

Machine learning is a branch of computer science that broadly aims to enable computers to “learn” without being directly programmed . It has origins in the artificial intelligence movement of the 1950s and emphasizes practical objectives and applications, particularly prediction and optimization. Computers “learn” in machine learning by improving their performance at tasks through “experience”. In practice, “experience” usually means fitting to data; hence, there is not a clear boundary between machine learning and statistical approaches.Indeed, whether a given methodology is considered machine learning” or “statistical” often reflects its history as much as genuine differences, and many algorithms.[25]

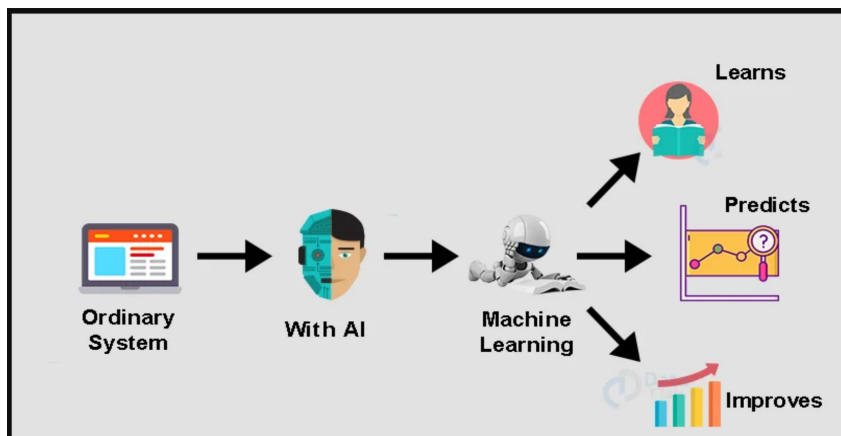


Figure 1.2: Machine Learning.[34]

Machine Learning Algorithms can be classified into 3 types as follows :

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

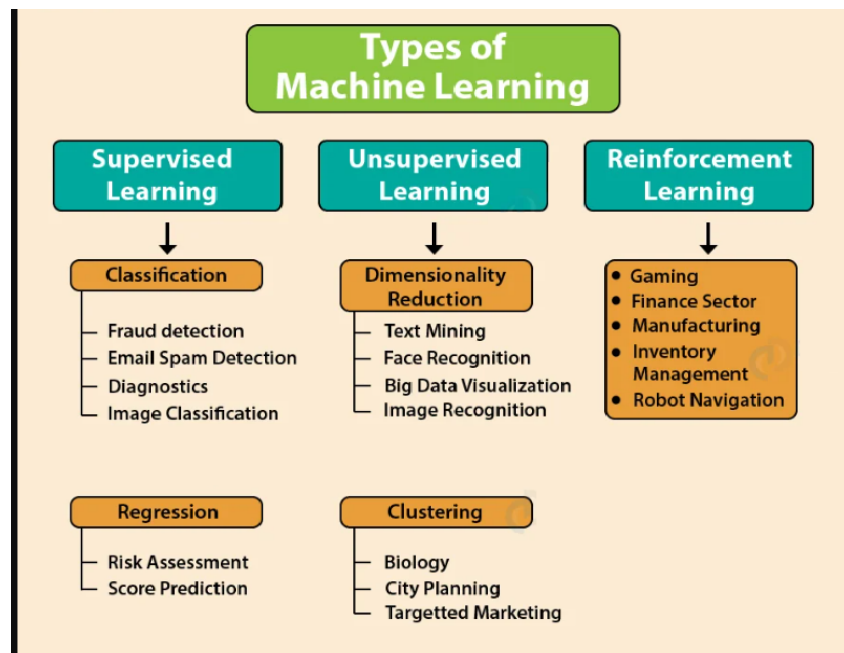


Figure 1.3: Types of Machine Learning. [36]

1.3.1 Support vector machines (SVMs)

Support vector machines (SVMs) [35] appeared in the early nineties as optimal margin classifiers in the context of Vapnik's statistical learning theory. Since then SVMs have been successfully applied to real-world data analysis problems, often providing improved results compared with other techniques. The SVMs operate within the framework of regularization theory by minimizing an empirical risk in a well-posed and consistent way. A clear advantage of the support vector approach is that sparse solutions to classification and regression problems are usually obtained: only a few samples are involved in the determination of the classification or regression functions. This fact facilitates the application of SVMs to problems that involve a large amount of data, such as text processing and bio-informatics tasks.

1.4 What Is deep learning ?

We do not know of a crisp definition of what DL is,

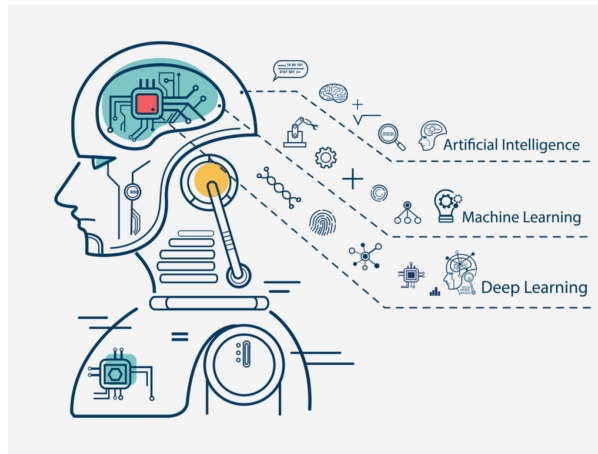


Figure 1.4: The relationship between artificial intelligence, ML and deep learning.[9]

but DL is a subset of machine learning. DL is based largely on Artificial Neural Networks (ANNs), a computing paradigm inspired by the functioning of the human brain. Like the human brain, it is composed of many computing cells or ‘neurons’ that each perform a simple operation and interact with each other to make a decision. Deep Learning is all about learning or ‘credit assignment’ across many layers of a neural network accurately, efficiently and without supervision. [66]

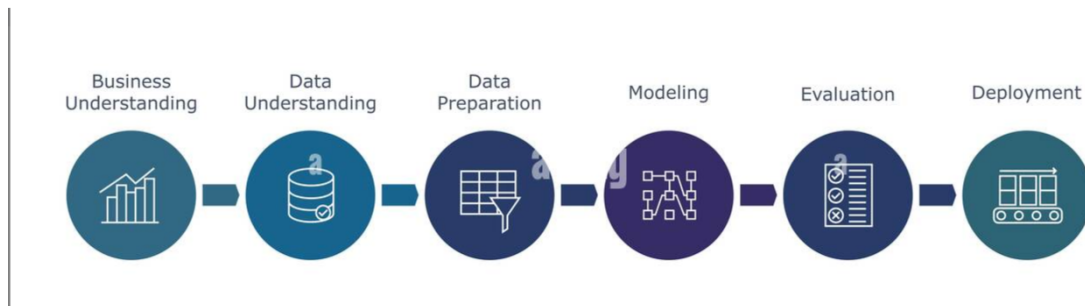


Figure 1.5: machine or deep learning process.[13]

1.5 Deep learning terminologies

1.5.1 Recurrent neuron

It’s one of the best from the Deep Learning Terminologies. Basically, in this output is sent back to the neuron for t timestamps. After looking at the diagram, we can say output is back as input t times. Also, we have to connect different together that will look like an unrolled neuron. Although, an important thing is that it provides us a more generalized output.[33]

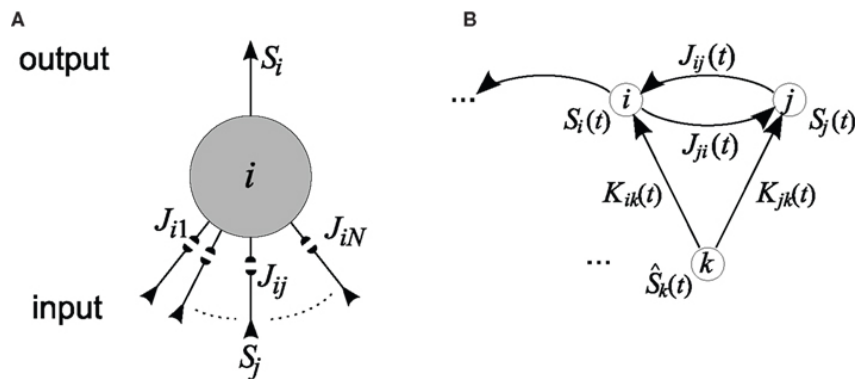


Figure 1.6: Recurrent Neuron.[40]

1.5.2 Vanishing Gradient Problem

The vanishing gradient problem since the derivative is usually small. While the gradients get smaller as they go through multiple layers (with activation functions) during the back propagation.[49]

1.5.3 Exploding gradient problem

The exploding gradient problem has been a major challenge for training very deep feedforward neural networks at least since the advent of gradient-based parameter learning. In a nutshell, it describes the phenomenon that as the gradient is backpropagated through the network, it may grow exponentially from layer to layer. [69]

1.5.4 max-pooling

max-pooling operation treats each dimension independently while for such representations the encoding dimensions are strongly correlated and should be treated jointly.[59]

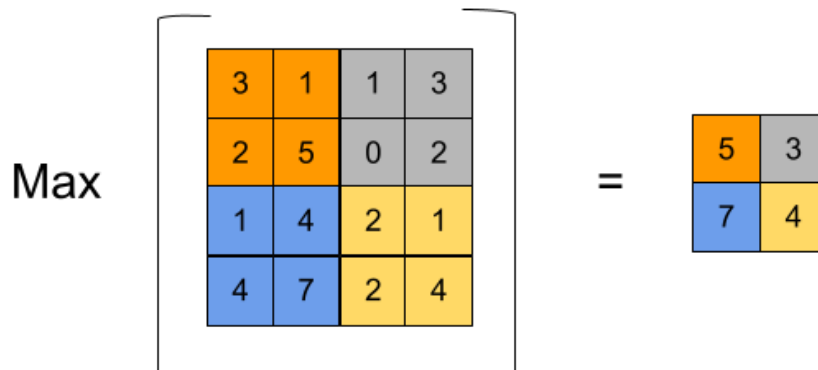


Figure 1.7: Pooling in Deep Learning.[59]

1.5.5 Softmax

The softmax link is used in many probabilistic model dealing with both discrete and continuous data. [27]

This constant defines an activation function that returns values using the following formula:

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

Figure 1.8: Softmax.[59]

1.5.6 Neural network

An artificial neural network (or simply neural network) consists of an input layer of neurons (or nodes, units), one or two (or even three) hidden layers of neurons, and a final layer of output neurons. [85]
Neural Networks Are A Set Of Algorithms, Which Is Based - Neural Network

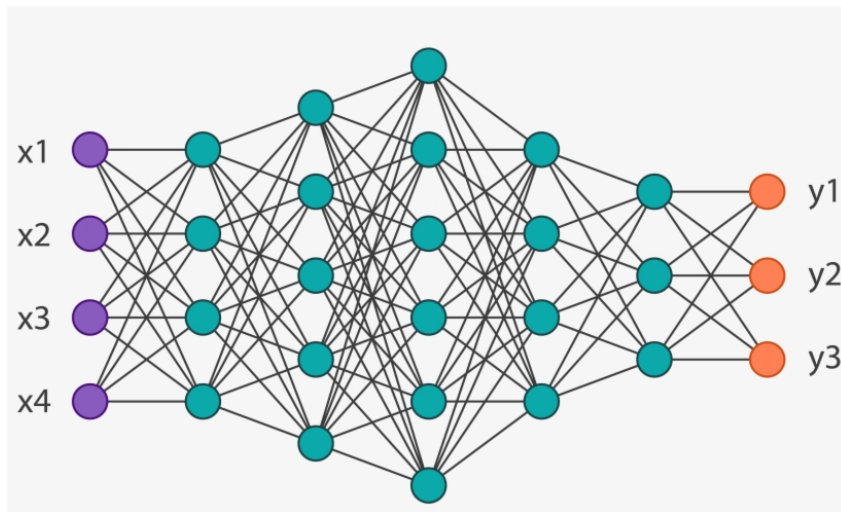


Figure 1.9: Neural Network.[59]

1.5.6.1 Input layer/ Output layer / Hidden layer

It's one of the best from the Deep Learning Terminologies. The input layer is the one which receives the input. Also, it's the first layer of the network. The output layer is the final layer of the network. These layers are the hidden layers of the network. We use these hidden layers to perform tasks on incoming data. Hence, pass generated output to the next layer. Although, both layers are visible but the intermediate layers are hidden.[33]

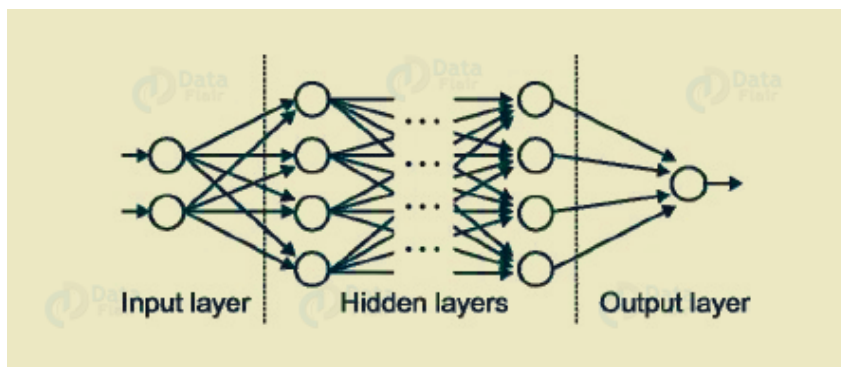


Figure 1.10: Hidden layer in deep Learning.[33]

1.5.6.1 MLP (Multi-Layer perceptron)

A multi layer perceptrons (MLP) is a finite acyclic graph. The nodes are neurons with logistic activation. nodes that are no target of any connection are called input neurons.

A MLP that should be applied to input patterns of dimension n must have n input neurons, one for each dimension. [70]

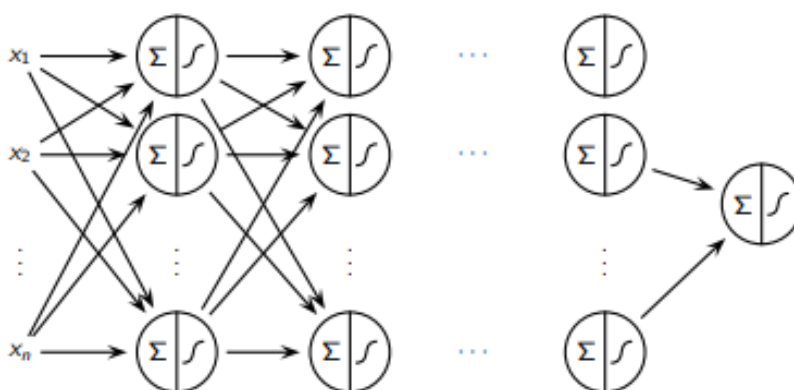


Figure 1.11: MLP (Multi-Layer perceptron) in Deep Learning.[70]

1.5.7 Neuron

As we can say that we use neuron to form the basic elements of a brain. Also, helps to form the basic structure of a neural network. As we get new information. We start to generate an output.[33]

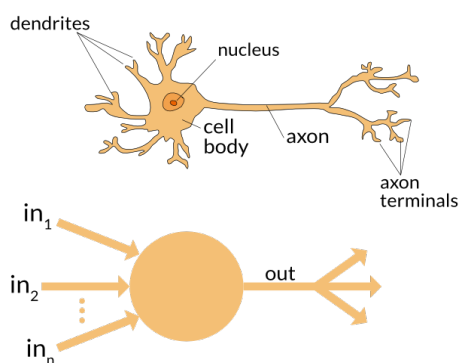


Figure 1.12: Neuron in Deep Learning.[33]

1.5.8 Activation function

As soon as we apply linear component to the input, a non-linear function is applied to it. As this is done by applying the activation function to the linear combination. Hence, this translates the input signals to output signals.[33]

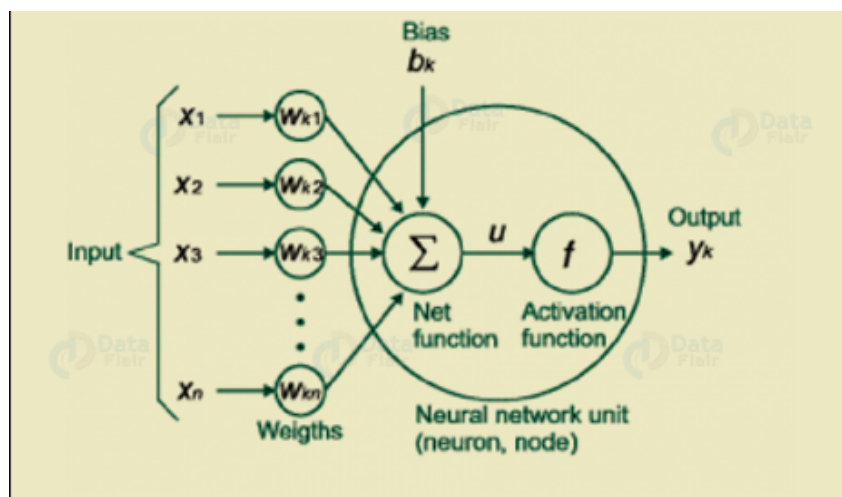


Figure 1.13: Activation Function.[33]

1.5.9 Learning rate

learning rate schedules have been proposed to automatically anneal the learning rate based on how many epochs through the data have been done. These approaches typically add additional hyperparameters to control how quickly the learning rate decays.[88]

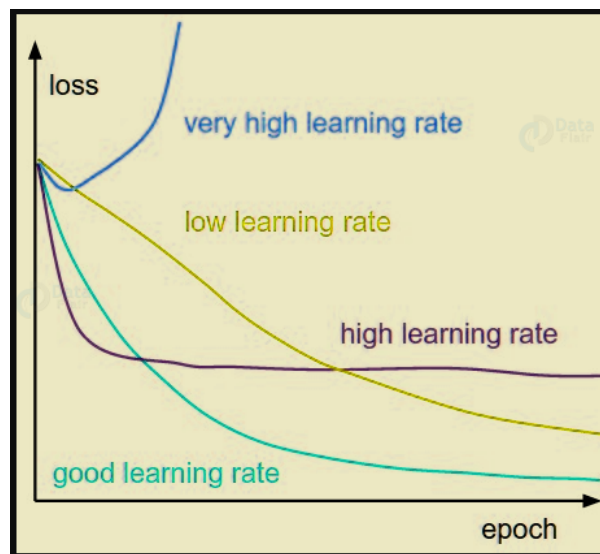


Figure 1.14: Learning Rate.[33]

1.5.10 Back propagation

Whenever we want to define a neural network, we assign random weights and bias values to our nodes. Also, as soon as we received the output for a single iteration. Thus, we can calculate the error of the network.

In back-propagation, the movement of the network is backward, the error along with the gradient flows back from the out layer through the hidden layers and updating of weights is done.[33]

1.6 Evolution of deep learning

It is easiest to understand deep learning with some historical context. Rather than providing a detailed history of deep learning, we identify a few key trends:

- Deep learning has had a long and rich history, but has gone by many names, reflecting different philosophical viewpoints, and has waxed and waned in popularity.
- Deep learning has become more useful as the amount of available training data has increased.
- Deep learning models have grown in size over time as computer infrastructure (both hardware and software) for deep learning has improved.
- Deep learning has solved increasingly complicated applications with increasing accuracy over time. [41]

1.7 Deep learning approaches

1.7.1 supervised learning

Supervised learning entails learning a mapping between a set of input variables X and an output variable Y and applying this mapping to predict the outputs for unseen data. Supervised learning is the most important methodology in machine learning and it also has a central importance in the processing of multimedia data.[31]

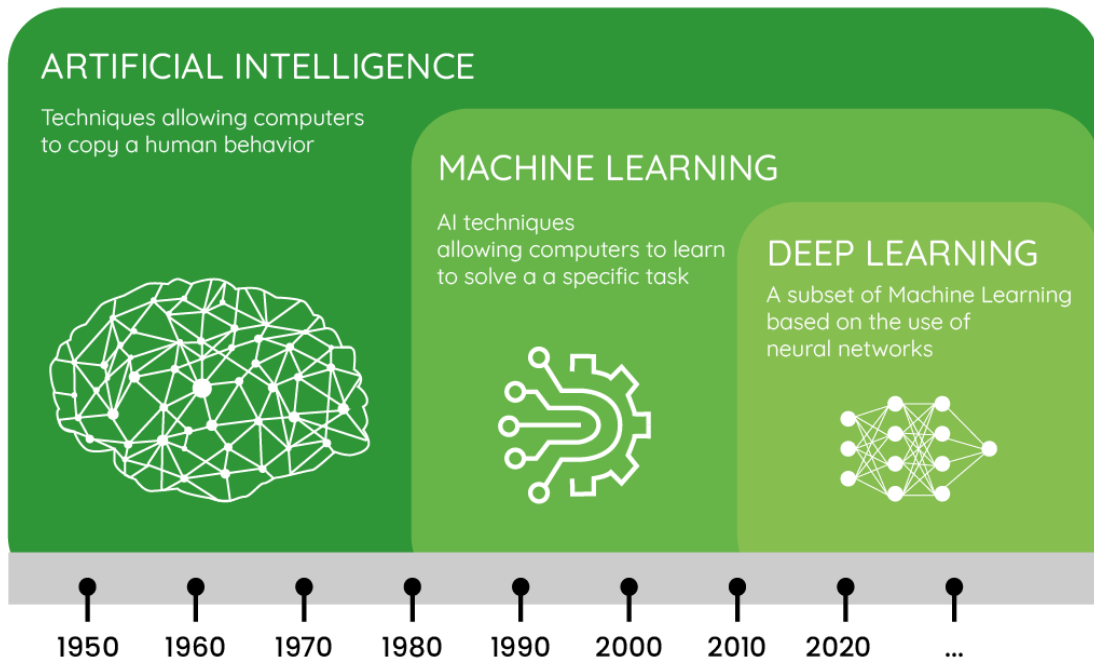


Figure 1.15: Artificial Intelligence and Machine Learning and DL[2]

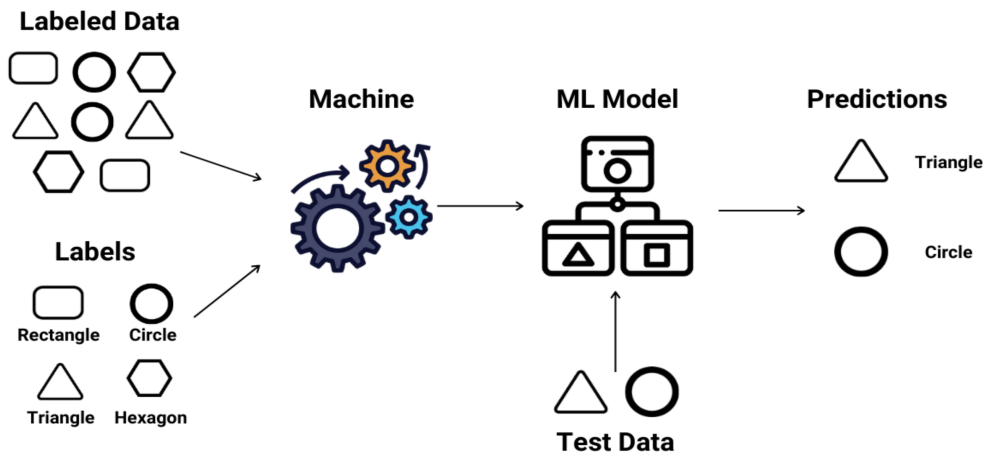


Figure 1.16: supervised learning .[17]

1.7.2 Unsupervised learning

unsupervised learning or “learning without a teacher.” is a type of algorithm that learns patterns from untagged data. The hope is that through mimicry, which is an important mode of learning in people, the machine is forced to build a compact internal representation of its world and then generate imaginative content from it. In contrast to supervised learning where data is tagged by an expert, e.g. as a “ball” or “fish”, unsupervised methods exhibit self-organization that captures patterns as probability densities or a combination of neural feature preferences. The other levels in the supervision spectrum are reinforcement learning where the machine is given only a numerical performance score as guidance, and semi-supervised

learning where a smaller portion of the data is tagged. Two broad methods in Unsupervised Learning are Neural Networks and Probabilistic Methods. [32]

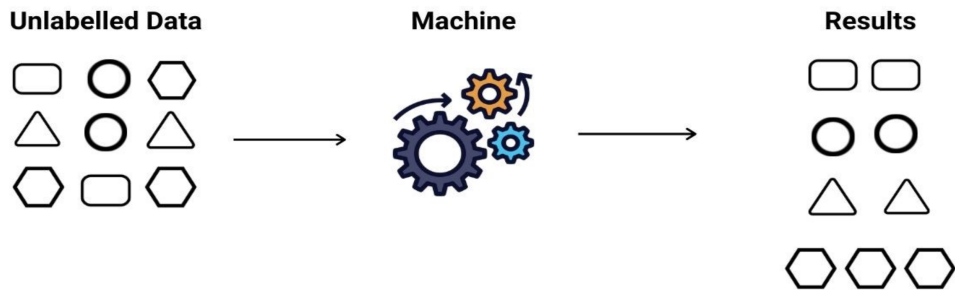


Figure 1.17: unsupervised learning.[19]

1.7.3 Semi supervised learning(hybrid learning)

Semi-supervised classification methods are suitable tools to tackle training sets with large amounts of unlabeled data and a small quantity of labeled data. This problem has been addressed by several approaches with different assumptions about the characteristics of the input data. Among them, self-labeled techniques follow an iterative procedure, aiming to obtain an enlarged labeled data set, in which they accept that their own predictions tend to be correct.[82]

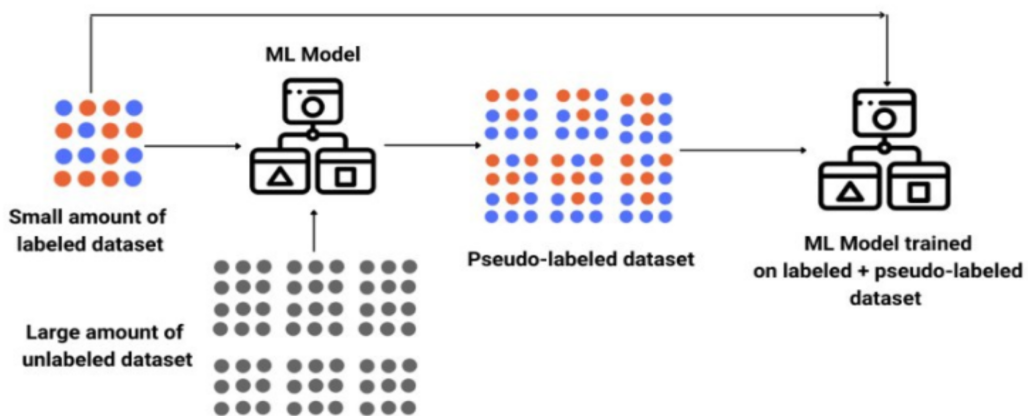


Figure 1.18: hybrid learning(SSL).[16]

1.7.4 Deep reinforcement learning

Deep reinforcement learning is the combination of reinforcement learning (RL) and deep learning. This field of research has been able to solve a wide range of complex decisionmaking tasks that were previously out of reach for a machine. Thus, deep RL opens up many new applications in domains such as healthcare, robotics, smart grids, finance, and many more. This manuscript provides an introduction to deep reinforcement learning models, algorithms and techniques.[39]

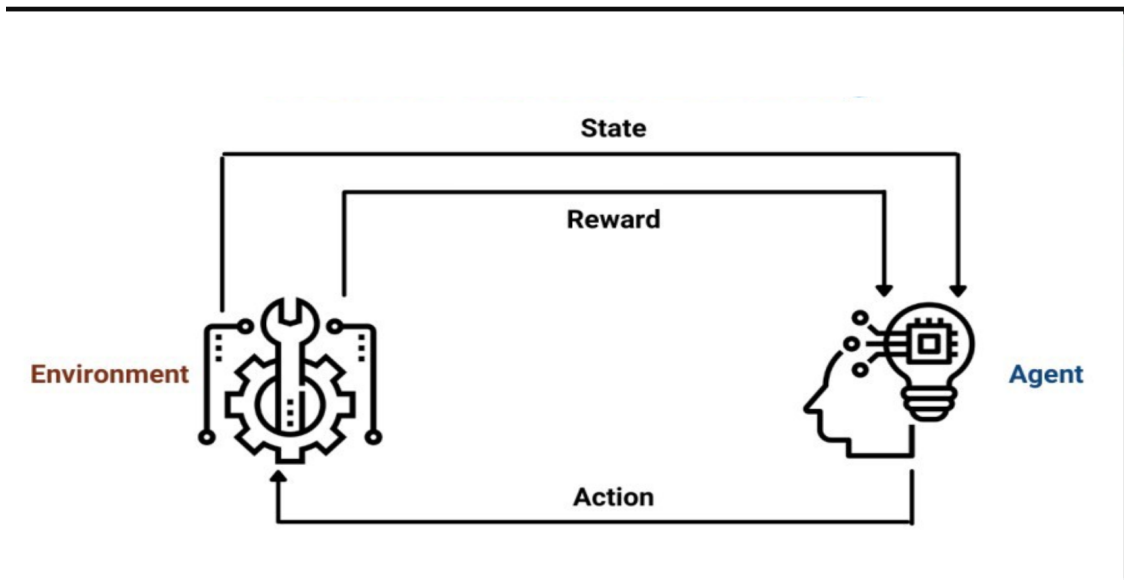


Figure 1.19: hybrid learning(SSL).[16]

1.7.5 Algorithms for Deep learning

The field of deep learning deals with many algorithms[38], each one designed to solve a specific task. Some of these algorithms are listed below :

- Feed Forward Neural Networks (Artificial Neuron).
- Radial Basis Function Neural Network.
- Multilayer Perceptron.
- Unsupervised pretrained network.
 - Autoencoders .
 - Deep Belief Networks .
 - Generative Adversarial Networks (GANs) .
- Convolutional Neural Networks.
- Recurrent Neural Network/LSTMs.
- Recursive Neural Networks.

1.8 Fundamental deep learning architectures Deep learning

1.8.1 Unsupervised Pre-trained Networks

In unsupervised pre-training, a model is trained unsupervised, and then the model used for prediction. Some unsupervised pre-training architectures are discussed below .[38]

1.7.1.1 Autoencoders

An autoencoder is a specific type of a neural network, which is mainly designed to encode the input into a compressed and meaningful representation, and then decode it back such that the reconstructed input is similar as possible to the original one.[24]

It is typically a feed forward neural network which aims to learn a compressed, distributed representation(encoding) of a dataset .[38]

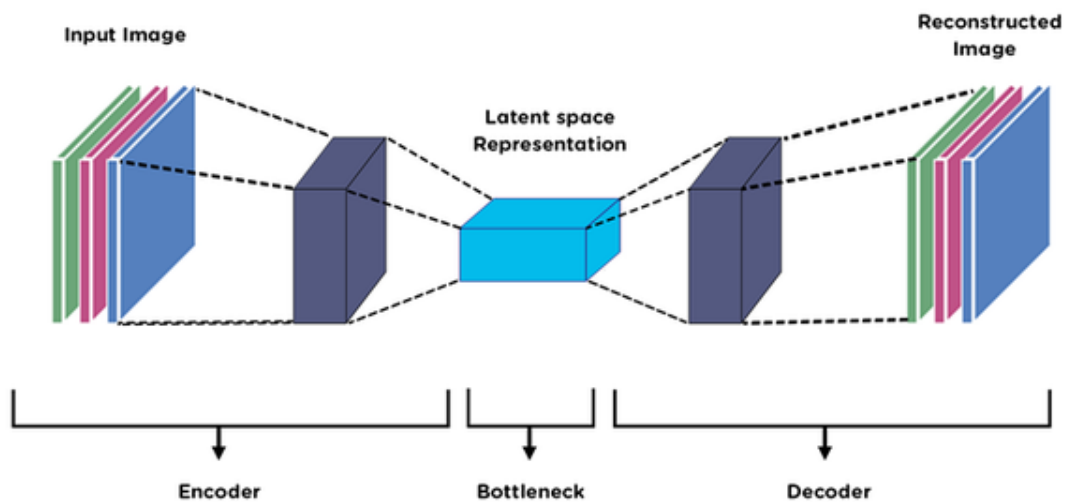


Figure 1.20: Autoencoders.[3]

1.7.1.2 Deep Belief Networks

are probabilistic generative models that are composed of multiple layers of stochastic, latent variables. The latent variables typically have binary values and are often called hidden units or feature detectors. The top two layers have undirected, symmetric connections between them and form an associative memory. The lower layers receive top-down, directed connections from the layer above. The states of the units in the lowest layer represent a data vector.[46]

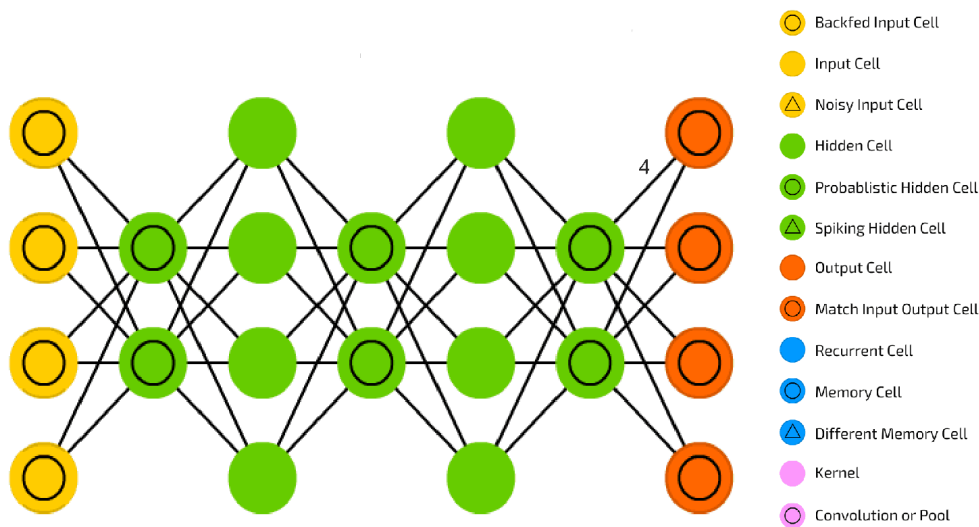


Figure 1.21: Deep Belief Networks.[8]

1.7.1.3 Generative Adversarial Networks

Generative adversarial networks (GANs) provide a way to learn deep representations without extensively annotated training data. They achieve this by deriving backpropagation signals through a competitive process involving a pair of networks. The representations that can be learned by GANs may be used in a variety of applications, including image synthesis, semantic image editing, style transfer, image super-resolution, and classification. [30]

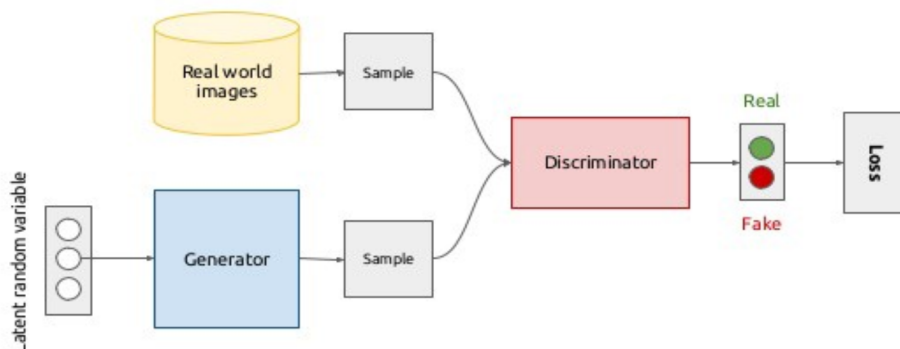


Figure 1.22: Generative Adversarial Networks.[12]

1.8.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are analogous to traditional ANNs in that they are comprised of neurons that self-optimize through learning. Each neuron will still receive an input and perform an operation (such as a scalar product followed by a non-linear function) - the basis of countless ANNs. From the input raw image vectors to the final output of the class score, the entire of the network will still express a single perceptive score function (the weight). The last layer will contain loss functions associated with the classes, and all of the regular tips and tricks developed for traditional ANNs still apply. [63]

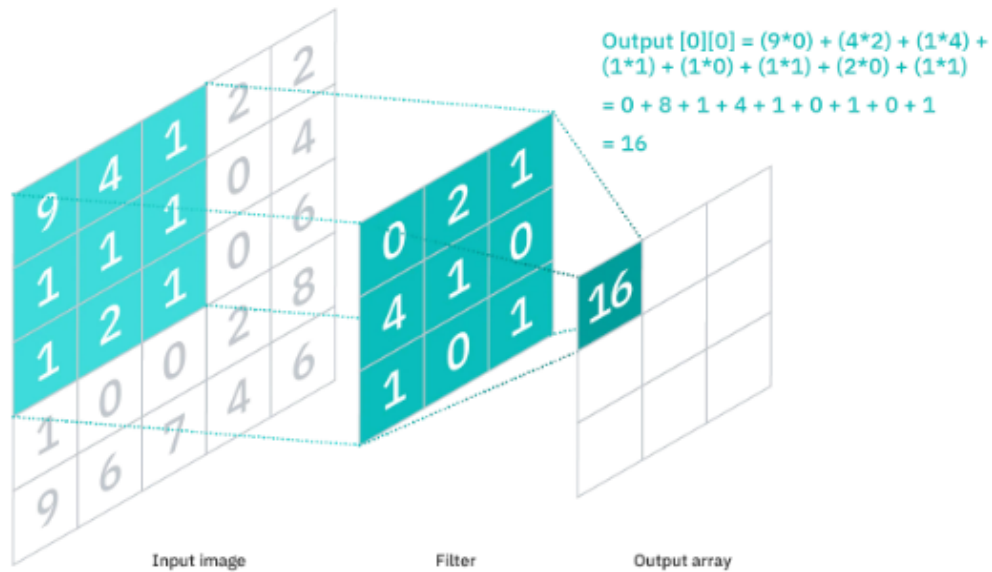


Figure 1.23: Convolutional Neural Networks. [7]

1.8.3 Recurrent Neural Networks

artificial neural networks (ANNs) are made from layers of connected units called artificial neurons. A “shallow network” refers to an ANN with one input layer, one output layer, and at most one hidden layer without a recurrent connection. As the number of layers increases, the complexity of network increases too. More number of layers or recurrent connections generally increases the depth of the network and empowers it to provide various levels of data representation and feature extraction, referred to as “deep learning”. [74]

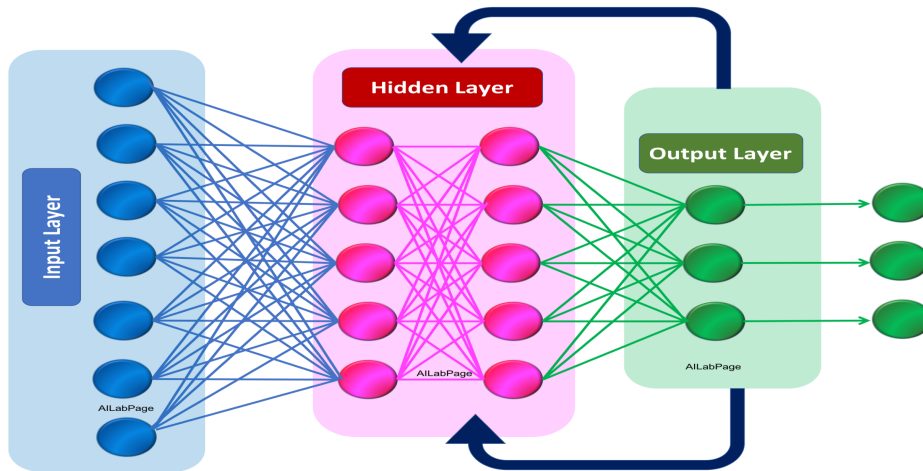


Figure 1.24: Recurrent Neural Networks(RNN).[15]

1.8.3.1 Long Short-Term Memory LSTM

Long short-term memory (LSTM) is a special type of recurrent neural network (RNN) architecture that was designed over simple RNNs for modeling temporal sequences and their long-range dependencies more accurately.[68]

1.8.3.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a classification and regression prediction tool that uses machine learning theory to maximize predictive accuracy while automatically avoiding over-fit to the data. Support Vector machines can be defined as systems which use hypothesis space of a linear functions in a high dimensional feature space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory.[50]

1.9 Deep learning methods

Some of the powerful techniques [45] that can be applied to deep learning algorithms to reduce the training time and to optimize the model are discussed in the following section. The merits and demerits of each method are comprised in the Table 1

1.9.1 Back propagation

While solving an optimization problem using a gradient-based method, back propagation can be used to calculate the gradient of the function for each iteration

1.9.2 Stochastic gradient descent

Using the convex function in gradient descent algorithms ensures finding an optimal minimum without getting trapped in a local minimum. Depending upon the values of the function and learning rate or step size, it may arrive at the optimum value in different paths and manners .

1.9.3 Learning Rate Decay

Adjusting the learning rate increases the performance and reduces the training time of stochastic gradient descent algorithms. The widely used technique is to reduce the learning rate gradually, in which we can make large changes at the beginning and then reduce the learning rate gradually in the training process. This allows fine-tuning the weights in the later stages.

1.9.4 Dropout

The overfitting problem in deep neural networks can be addressed using the drop out technique. This method is applied by randomly dropping units and their connections during training [9]. Dropout offers an effective regularization method to reduce overfitting and improve generalization error. Dropout gives an improved performance on supervised learning tasks in computer vision, computational biology, document classification, speech recognition .

1.9.5 Max-Pooling

In max-pooling a filter is predefined, and this filter is then applied across the nonoverlapping sub-regions of the input taking the max of the values contained in the window as the output. Dimensionality, as well as the computational cost to learn several parameters, can be reduced using maxpooling.

1.9.6 Batch Normalization

Batch normalization reduces covariate shift, thereby accelerating deep neural network. It normalizes the inputs to a layer, for each mini-batch, when the weights are updated during the training. Normalization stabilizes learning and reduces the training epochs. The stability of a neural network can be increased by normalizing the output from the previous activation layer .

1.9.7 Skip-gram

Word embedding algorithms can be modeled using Skip-gram. In the skip-gram model, two vocabulary terms share a similar context; then those terms are identical. For example, the sentences "cats are mammals" and "dogs are mammals" are meaningful sentences which shares the same meaning "are mammals." Skip-gram can be implemented by considering a context window containing n terms and train the neural network by skipping one of this term and then use the model to predict skipped term .

1.9.8 Transfer learning

In transfer learning, a model trained on a particular task is exploited on another related task. The knowledge obtained while solving a particular problem can be transferred to another network, which is to be trained on a related problem. This allows for rapid progress and enhanced performance while solving the second problem

Method	Description	Merits	Demerits
Backpropagation	is also used in optimization problems	calculatorfor gradient	sensitive to noisy data
stochastic gradient descent	To find optimal minimum in optimization problems	Avoids trapping in local minimum	Longer convergence time, computationally expensive
Learning rate decay	Reduce learning rate gradually	Increases performance and reduces training time	Computationally expensive
Dropout	Drops out units / connection during training	Avoids overfitting	Increases number of iterations required to converge
Max pooling	Applies a max filter	Reduces dimension and computational cost	Considers only the maximum element which may lead to unacceptable result in some cases
batch normalization	Batch-wise normalization of input to a layer	Reduces covariant shift, increases stability of the network, network trains faster, allows higher learning rates	Computational overhead ring training
skip gram	Used in word embedding algorithms	Can work on any raw text and requires less memory	Softmax function is computationally expensive, and training time is high
transfer learning	Knowledge of first model is transferred to second problem	Enhances performance, rapid progress in training of second problem	Works with similar problems only

Table 1.1: Comparison of deep learning methods.[45]

1.10 frameworks deep learning

A deep learning framework [45] helps in modeling a network more rapidly without going into details of underlying algorithms. Some deep learning frameworks are discussed below and are summarized in Table 2.

1.10.1 TensorFlow

TensorFlow, developed by Google Brain, supports languages such as Python, C++, and R. It enables us to deploy our deep learning models in CPUs as well as GPUs .

1.10.2 Keras

Keras is an API, written in Python and run on top of TensorFlow. It enables fast experimentation. It supports both CNNs and RNNs and runs on CPUs and GPUs .

1.10.3 Py Torch

Py Torch can be used for building deep neural networks as well as executing tensor computations. PyTorch is a Python-based package that provides tensor computations. PyTorch delivers a framework to create computational graphs.

1.10.4 Caffe

Caffe Yangqing Jia developed Caffe, and it is open source as well. Caffe stands out from other frameworks in its speed of processing as well as learning from images. Caffe Model Zoo framework facilitates us to access pre-trained models, which enable us to solve various problems effortlessly .

1.10.5 Deeplearning4j

Deeplearning4j is implemented in Java, and hence, it is more efficient when compared to Python. The ND4J tensor library used by Deeplearning4j

Deep Learning Framework	Release Year	Language written in	CUDA supported	Pre-trained models
TensorFlow	2015	C++	Yes	Yes
Keras	2015	Python	Yes	Yes
PyTorch	2016	Python, C	Yes	Yes
Caffe	2013	C++	Yes	Yes
Deeplearning4j	2014	C++, Java	Yes	Yes

Table 1.2: Comparison of Deep Learning Frameworks.[45]

1.11 The Applications of Deep Learning

We have many areas of application in deep learning, including the following:

1.11.1 Automatic Speech Recognition (ASR)

Automatic Speech Recognition (ASR) is a task used to convert speech waves or signals to its mapping sequence of words or units using a determined algorithm . These sequences are represented like the human transcription. The observations of speech vectors are used for representing input of speech audios .[71]

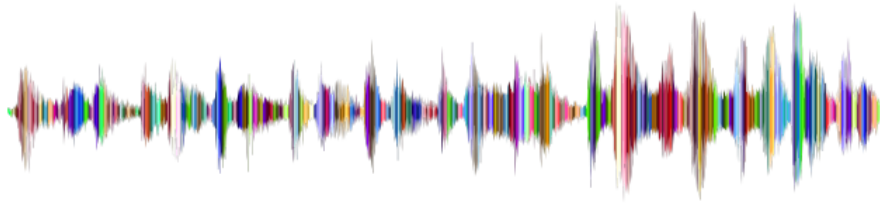


Figure 1.25: DL in an audio recording [87]

1.11.2 Game Playing

Deep learning has come a long way since its inception. As the capacity and capability of the concept is larger now than ever before, the benefits of deep learning have found their way into games as well. This can be seen in both single player games, whose outcomes are primarily used for research purposes to determine how well a system can adapt to a situation when a set of constraints is present, as well as in multiplayer games, where a player is able to play against the computer without the need for another human to be present.[4]



Figure 1.26: Game Playing [11]

1.11.3 Autonomous Driving

self-driving cars are autonomous decision-making systems that process streams of observations coming from different on-board sources, such as cameras, radars, LiDARs. where sensory information is directly mapped to control outputs. The components of the modular pipeline can be designed either based on AI and deep learning methodologies, or using classical non-learning approaches.[43]

1.11.4 Chatbots

Chatbots are applied in many different domains. As far as Education and Research go. Deep Learning application have been used in these fields. The choice seems justified by the fact that chatbots created for educational purposes are often aimed at providing specific information (such as class schedules) or educational material.[28]

1.11.5 Image captioning

Image captioning is a process of automatically describing an image with one or more natural language sentences. In recent years, image captioning has witnessed rapid progress, from initial template-based models to the current ones, based on deep learning. [47]

1.11.6 News Aggregation and Fake News Detection

Spreading of misinformation on the web nowadays represents a serious issue, as their influence on peoples opinions may be significant. Fake news represents a specific type of misinformation. While its detection was mostly being performed manually in the past, automated methods using machine learning and related fields became more critical.[51]

1.11.7 Text to Speech

This is the inverse of automatic speech recognition. In other words, the input is text and the output is an audio file. In this case, the output is much longer than the input. While it is easy for humans to recognize a bad audio file, this is not quite so trivial for computers.[89]

1.11.8 Machine Translation

Unlike the case of speech recognition, where corresponding inputs and outputs occur in the same order (after alignment), in machine translation, order inversion can be vital. In other words, while we are still converting one sequence into another, neither the number of inputs and outputs nor the order of corresponding data examples are assumed to be the same. Consider the following illustrative example of the peculiar tendency of Germans to place the verbs at the end of sentences.[89]

```
German:      Haben Sie sich schon dieses grossartige Lehrwerk angeschaut?  
English:     Did you already check out this excellent tutorial?  
Wrong alignment: Did you yourself already this excellent tutorial looked-at?
```

1.12 Conclusion

Deep learning is continuously evolving faster; still, there are a number of problems to deal with and can be solved using deep learning. Even though a full understanding of the working of deep learning is still a mystery, we can make machines smarter using Deep learning, sometimes even smarter than human.[45]

Software vulnerability detection

2.1 Introduction

Software vulnerability remains a serious problem among industry players in the world today because of the numerous security related challenges it possess to end-users and stakeholders; there has been an increased reportage on several security vulnerabilities with high devastating effects on customers; this has brought to the public domain the need to focus on software vulnerability detection tools and methods. Although previous studies have proposed various methods and tools that can be used in reducing or eliminating software vulnerability.[23]

2.2 Security objective

2.2.1 Confidentiality

Confidentiality is the prevention of the disclosure of secret or sensitive information to unauthorized users or entities.[54]

2.2.2 Integrity

Integrity is the prevention of unauthorized modification of protected information without detection.[54]

2.2.3 Authentication

Authentication is process of validating the user's identity. Users are identified using different authentication mechanisms.[53]

2.2.4 Non-Repudiation

non-repudiation is the ability to protect against denial by one of the entities involved in an action of having participated in all or part the action.[78]

2.2.5 Availability

Availability is the provision of services and systems to legitimate users when requested or needed. Availability is also a goal in providing reliable, dependable or fault-tolerant systems except that availability.[54]

2.3 Terminology

2.3.1 Vulnerability

A flaw in a software, firmware, hardware, or service component resulting from a weakness that can be exploited, causing a negative impact to the confidentiality, integrity, or availability of an impacted component or components.[20] or is a property of system security requirements, design, implementation, or operation that could be accidentally triggered or intentionally exploited and result in a security failure .[62]

National Institute of Standards and Technology (NIST): Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.

2.3.2 Attack(Exploit)

An exploit is a technique that takes advantage of a vulnerability to cause a failure. An attack is a specific application of an exploit [5]. In other words, an attack is an action (or sequence of actions) that takes advantage of vulnerability [62]; or An attack is a specific instance of a threat. It involves detailed descriptions of the system, the vulnerabilities exploited, the attack path and the assets attacked.[54]

2.3.3 Threat

A threat is any action that can damage an asset, or cause a security breach. For example, there are disclosure threats, modification threats and denial of service threats, which threaten the main security goals of protecting confidentiality, integrity and availability, respectively.[54] Anything or anyone that has the ability or desire to exploit a vulnerability.[79]

2.3.4 Software

is a set of programs [5] that instructs the computer about the tasks to be performed. Software tells the computer how the tasks are to be performed or hardware carries out these tasks. Different sets of software can be loaded on the same hardware to perform different kinds of tasks.

Software can be broadly classified into three categories

1. System Software.
2. Application Software.
3. Programming Software[6]

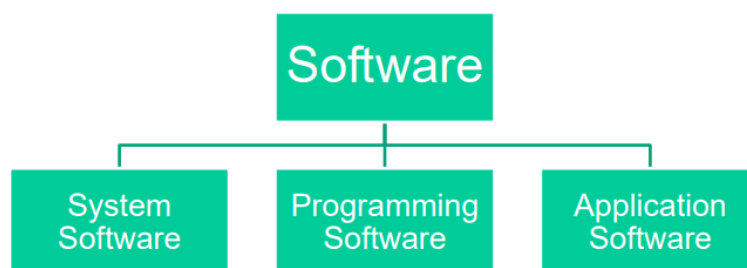


Figure 2.1: Types Of Software[6]

2.3.5 Counter measure

A countermeasure is a defense against an attack. The term indicates the reactive stance of security measures in the past: after an attack occurs.[54]

2.3.6 Risk

The intersection between assets, vulnerabilities, and threats. Risk is the amount an asset's owner would suffer if a threat exploited an asset's vulnerability.[79]

risk = threat * vulnerability/countermeasure

2.4 The Open Web Application Security Project (OWASP) Top Ten

2.4.1 What is the OWASP?

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted.[14]



2.4.2 Top Ten 2021 List :

The OWASP Top 10 [65] is a list of the 10 most important security risks affecting web applications. It is revised every few years to reflect industry and risk changes. The list has descriptions of each category of application security risks and methods to remediate them. The latest version of the OWASP Top 10 list is published(2021) are the following:

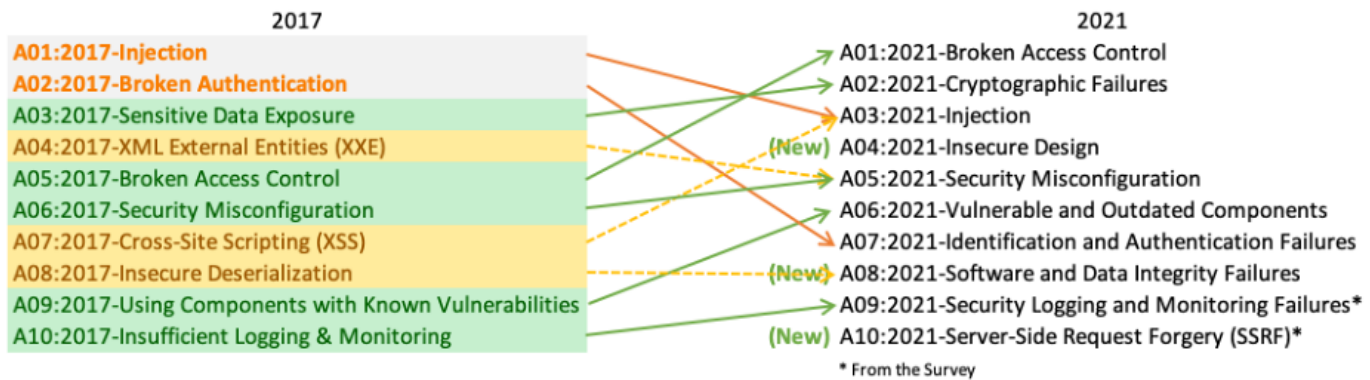


Figure 2.2: Changes in Top Ten Risks for 2021 .[65]

2.4.2.1 A01:2021-Broken Access Control

At its core, Broken Access Control is simply a scenario in which attackers can access, modify, delete or perform actions outside an application or systems' intended permissions. Many vulnerabilities can be classified as a form of Broken Access Control, such as when normal users are able to access admin-only features



by changing parameters in a URL, viewing or modifying another user's data or privilege escalation.[81]

2.4.2.2 A02:2021-Cryptographic Failures

As per the OWASP cryptographic failure definition (2021), it's a symptom instead of a cause. This failure is responsible for the exposure/leaking of data of critical and sensitive nature to ill-intended resources/people. Missing out on safeguarding such data leads to theft, public listing, breaches, and other problems.[84]



2.4.2.3 A03:2021-Injection

Injections are amongst the oldest and most dangerous attacks aimed at web applications and can lead to data theft, data loss, loss of data integrity, denial of service, as well as full system compromise. The primary reason for injection vulnerabilities is usually insufficient user input validation.[56]



2.4.2.4 A04:2021-Insecure Design

is a new category for 2021, with a focus on risks related to design flaws. If we genuinely want to “move left” as an industry, it calls for more use of threat modeling, secure design patterns and principles, and reference architectures.[65]



2.4.2.5 A05:2021–Security Misconfiguration

One of the widely encountered vulnerabilities of web applications is security misconfiguration, which can happen at any level of an application stack, including the underlying platform, web server, database server, framework, and business logic code. According to the report by Open Web Application Security Project (OWASP)



, security misconfiguration, This vulnerability is listed as one of the top ten risks (ranked number 5 in 2021) which could potentially lead to risks ranging from unauthorized access to some system data or functionality to a complete system compromise.[29]

2.4.2.6 A06:2021–Vulnerable and Outdated Components

A software component is part of a system or application that extends the functionality of the application, such as a module, software package, or API. Component-based vulnerabilities occur when a software component is unsupported, out of date, or vulnerable to a known exploit. You may inadvertently use vulnerable software components in production environments, posing a threat to the web application.[1]



2.4.2.7 A07:2021–Identification and Authentication Failures

Previously known as Broken Authentication, this category slid down from the second position and now includes Common Weakness Enumerations (CWEs) related to identification failures. Notable CWEs included are CWE-297: Improper Validation of Certificate with Host Mismatch, CWE-287: Improper Authentication, and CWE-384: Session Fixation.[81]



2.4.2.8 A08:2021–Software and Data Integrity Failures

A new category for 2021 focuses on making assumptions related to software updates, critical data, and CI/CD pipelines without verifying integrity. One of the highest weighted impacts from Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS) data. Notable Common Weakness Enumerations (CWEs) include CWE-829: Inclusion of Functionality from Untrusted Control Sphere, CWE-494: Download of Code Without Integrity Check, and CWE-502: Deserialization of Untrusted Data.[81]



2.4.2.9 A09:2021–Security Logging and Monitoring Failures

Returning to the OWASP Top 10 2021, this category is to help detect, escalate, and respond to active breaches. Without logging and monitoring,



breaches cannot be detected. Insufficient logging, detection, monitoring, and active response occurs any time.[81]

2.4.2.10 A10:2021–Server-Side Request Forgery (SSRF)

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).[81]



2.5 National vulnerability database (NVD)

The NVD is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol (SCAP). This data enables automation of vulnerability management, security measurement, and compliance. The NVD includes databases of security checklist references, security-related software flaws, misconfigurations, product names, and impact metrics.[61]

2.5.1 Common Vulnerabilities and Exposures (CVE)

A unique, alphanumeric identifier assigned by the CVE Program. Each identifier references a specific vulnerability. A CVE ID enables automation and multiple parties to discuss, share, and correlate information about a specific vulnerability, knowing they are referring to the same thing.[20]

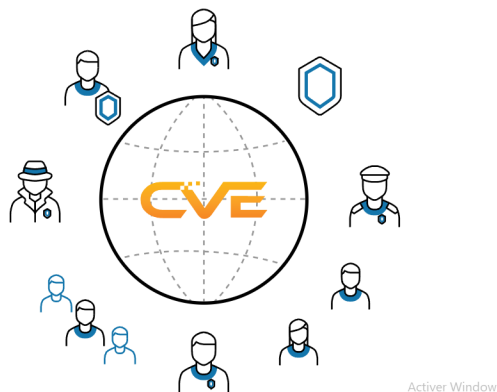


Figure 2.3: CVE[20]

2.6 SANS Top 25 Security Vulnerabilities In Software Applications

2.6.1 SANS

The word SANS [76] is not just an ordinary dictionary word rather it stands for SysAdmin, Audit, Network, and Security. In this tutorial, we will learn about the SANS top 20 security weaknesses we can

find in software programs and what we can do to mitigate it.

2.6.2 What the term CWE means?

The Common Weakness Enumeration (CWE) is a community accepted list of software and hardware vulnerabilities with identification code assigned for each weakness. The goal is to identify various flaws in software and hardware to be able to fix and mitigate all those flaws.

2.6.3 The CWE Top 25

Below is a brief listing of the weaknesses in the 2021 CWE Top 25 :

Rank	ID	Name	Score	2020 Rank Change
[01]	CWE-787	Out-of-bounds Write	65.93	+1
[02]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1
[03]	CWE-125	Out-of-bounds Read	24.9	+1
[04]	CWE-20	Improper Input Validation	20.47	-1
[05]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
[06]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
[07]	CWE-416	Use After Free	16.83	+1
[08]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
[09]	CWE-352	Cross-Site Request Forgery (CSRF)	14.46	0
[10]	CWE-434	Unrestricted Upload of File with Dangerous Type	8.45	5
[11]	CWE-306	Missing Authentication for Critical Function	7.93	+13
[12]	CWE-190	Integer Overflow or Wraparound	7.12	-1
[13]	CWE-502	Deserialization of Untrusted Data	6.71	+8
[14]	CWE-287	Improper Authentication	6.58	0
[15]	CWE-476	NULL Pointer Dereference	6.54	-2
[16]	CWE-798	Use of Hard-coded Credentials	6.27	+4
[17]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	5.84	-12
[18]	CWE-862	Missing Authorization	5.47	+7
[19]	CWE-276	Incorrect Default Permissions	5.09	+22
[20]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	4.74	-13
[21]	CWE-522	Insufficiently Protected Credentials	4.21	-3
[22]	CWE-732	Incorrect Permission Assignment for Critical Resource	4.2	-6
[23]	CWE-611	Improper Restriction of XML External Entity Reference	4.02	-4
[24]	CWE-918	Server-Side Request Forgery (SSRF)	3.78	+3
[25]	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	3.58	+6

Table 2.1: The CWE Top 25.[75]

1.CWE-787: Out-of-bounds Write Error [75]

The software writes data past the end, or before the beginning, of the intended buffer.

The following is an example of code that may result in a buffer underwrite, if find() returns a negative value to indicate that ch is not found in srcBuf:

```

Example Language: C
int main() {
  ...
  strncpy(destBuf, &srcBuf[find(srcBuf, ch)], 1024);
  ...
}

```

Figure 2.4: 2-CWE-125[75]

If the index to `srcBuf` is somehow under user control, this is an arbitrary write-what-where condition.

2. CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')[75]

The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

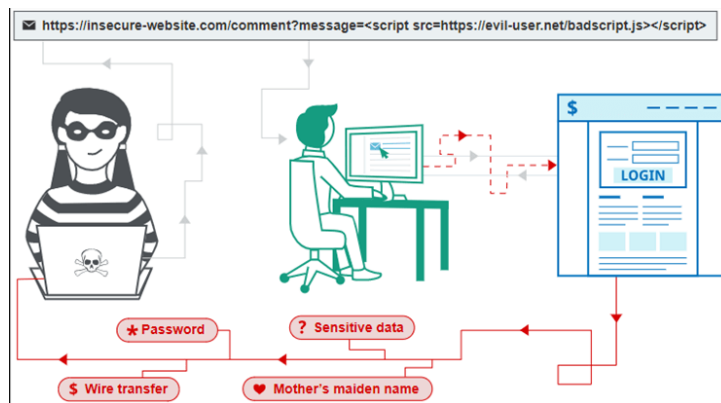


Figure 2.5: CVE-79[76]

3. CWE-125: Out-of-bounds Read (Out-of-bounds Read Error)

The software reads data past the end, or before the beginning, of the intended buffer.[75]

4. CWE-20: Improper Input Validation (Unvalidated Input Error) [75]

The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly. The below images show that a good application should not accept script or command as an input. If such inputs are not properly sanitized, the application will process it thinking it's a valid request.

This example demonstrates a shopping interaction in which the user is free to specify the quantity of items to be purchased and a total is calculated.

```

Example Language: Java
...
public static final double price = 20.00;
int quantity = currentUser.getAttribute("quantity");
double total = price * quantity;
chargeUser(total);
...

```

Figure 2.6: CWE-20[75]

The user has no control over the price variable, however the code does not prevent a negative value from

being specified for quantity. If an attacker were to provide a negative value, then the user would have their account credited instead of debited.

5.CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.[75]

The example below reads the name of a shell script to execute from the system properties. It is subject to the second variant of OS command injection.

Example Language: Java

```
String script = System.getProperty("SCRIPTNAME");
if (script != null)
    System.exec(script);
```

Figure 2.7: CWE-78[76]

If an attacker has control over this property, then they could modify the property to point to a dangerous program.[75]

6.CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')[75]

The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.

Example :

In 2008, a large number of web servers were compromised using the same SQL injection attack string. This single string worked against many different programs. The SQL injection was then used to modify the web sites to serve malicious code.

7.CWE-416:Use After Free(Free Memory Error)[75]

Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.

The following code illustrates a use after free error:

Example Language: C

```
char* ptr = (char*)malloc (SIZE);
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

Figure 2.8: CWE-416[75]

When an error occurs, the pointer is immediately freed. However, this pointer is later incorrectly used in the logError function.

8.CWE-22:Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

or Directory Traversal

The software uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the software does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.[75] Example: In the example below, the path to a dictionary file is read from a system property and used to initialize a File object.

```
Example Language: Java
String filename = System.getProperty("com.domain.application.dictionaryFile");
File dictionaryFile = new File(filename);
```

Figure 2.9: CWE-22[75]

However, the path is not validated or modified to prevent it from containing relative or absolute path sequences before creating the File object. This allows anyone who can control the system property to determine what file is used. Ideally, the path should be resolved relative to some kind of application or user home directory.[75]

8.CWE-190:Integer Overflow or Wraparound(Integer Overflow Error)[75]

The software performs a calculation that can produce an integer overflow or wraparound, when the logic assumes that the resulting value will always be larger than the original value. This can introduce other weaknesses when the calculation is used for resource management or execution control. Example : Integer overflows can be complicated and difficult to detect. The following example is an attempt to show how an integer overflow may lead to undefined looping behavior:

```
Example Language: C
short int bytesRec = 0;
char buf[SOMEBIGNUM];

while(bytesRec < MAXGET) {
    bytesRec += getFromInput(buf+bytesRec);
}
```

Figure 2.10: CWE-190[75]

In the above case, it is entirely possible that bytesRec may overflow, continuously creating a lower number than MAXGET and also overwriting the first MAXGET-1 bytes of buf.

9.CWE-352:Cross-Site Request Forgery (CSRF)

The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.[75]

The below image shows an attacker inducing a user to perform actions that they do not intend to perform.[76]

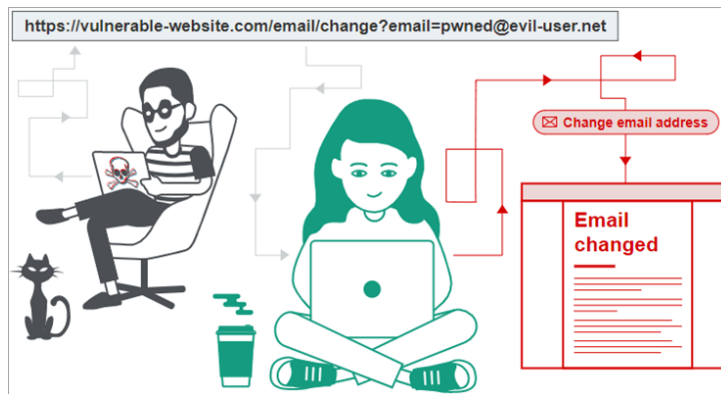


Figure 2.11: 2-CWE-352[76]

10.CWE-434:Unrestricted Upload of File with Dangerous Type

The software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product’s environment.[75]

The following program shows an upload of a PHP file. The file type was not verified and validated before uploading within the webroot directory. As a result of this weakness, an attacker may upload an arbitrary PHP file and execute it by directly accessing the uploaded file.[76]

```
HTML form
<form action="upload_image.php" method="post" enctype="multipart/form-data">
File:
<input type="file" name="filename"/>
<br/>
<input type="submit" name="submit" value="Upload"/>
</form>
```

Figure 2.12: 1-CWE-434[76]

```
upload_image.php
// Unrestricted Upload of File with Dangerous Type [CWE-434] vulnerable code example
// (c) HTB Research
$image = "images/" . basename($_FILES["uploadedfile"]["name"]);

if(move_uploaded_file($_FILES["uploadedfile"]["tmp_name"], $image))
{
    echo "You have successfully uploaded you picture.";
}
else
{
    echo "Error!";
}
```

Figure 2.13: 2-CWE-434[76]

2.7 Vulnerability Detection methods

2.7.1 Fuzzing

Fuzzing [26] is a method to discover software bugs and vulnerabilities by automatic test input generation which has found tremendous recent interest in both academia and industry. Fuzzing comes in the form of several techniques.

- symbolic execution, which enables a particularly effective approach to fuzzing by systematically enumerating the paths of a program.
- random input generation, which generates large amounts of inputs per second with none or minimal program analysis overhead.

2.7.2 Web Application Scanner

A web application scanner is an automated program that examines web applications for security vulnerabilities. In addition to searching for web application specific vulnerabilities, the tools also look for evidence of software coding errors, such as unchecked input strings and buffer overflows.

Some commercial web application scanners are AppScan, WebInspect, Hailstorm, Acunetix WVS, WebKing.

commercial web application scanners can also be used in detecting software vulnerability, they are; AppScan WebInspect NTOspider Some open source web application [62]

2.7.3 Static Analysis Techniques

Automated static analysis vocabulary includes the following terms: false positives, true positives and false negatives. A false positive occurs when a tool alerts to the presence of a nonexistent fault. A false negative occurs when a fault exists, but it is not detected due to the fact that static analysis tools are not perfectly accurate and may not detect all errors. Examples of automated static analysis tools are FindBugs, PMD and CheckStyle. The disadvantage of these tools is that they produce a high rate of false positives [37]. Many static analysis approaches have been introduced in various research aims at detecting BOF vulnerabilities.

These approaches can be classified to these six main areas: inference technique analysis sensitivity Analysis granularity soundness completeness language. [23]

2.7.4 Brick

Binary Run-time Integer Based Vulnerability Checker which detects integer based vulnerability at run-time. It is a very effective approach which results in low false positive and negative. BRICK process involves three stages: it converts the binary code to intermediate representation VEX on Valgrind (dynamic binary instrumentation framework Valgrind intercepts integer related statements at run-time, and records the necessary information to detect and locate vulnerability with a set checking scheme. [23]

2.7.5 CRED: C Range Error Detector

One of the vulnerability detecting approach that is not widely use is the Dynamic Buffer Overrun Detector. This is because its lack the power to protect against all buffer overrun attacks, break existing code and also produce too high overhead. CRED: C Range Error Detector approach corrects the above in- competencies and finds all buffer overruns attacks. CRED proved effective in detecting buffer overrun attacks on programs with known vulnerabilities, and is the only tool found to guard against tested 20 different buffer overflow attacks.[23]

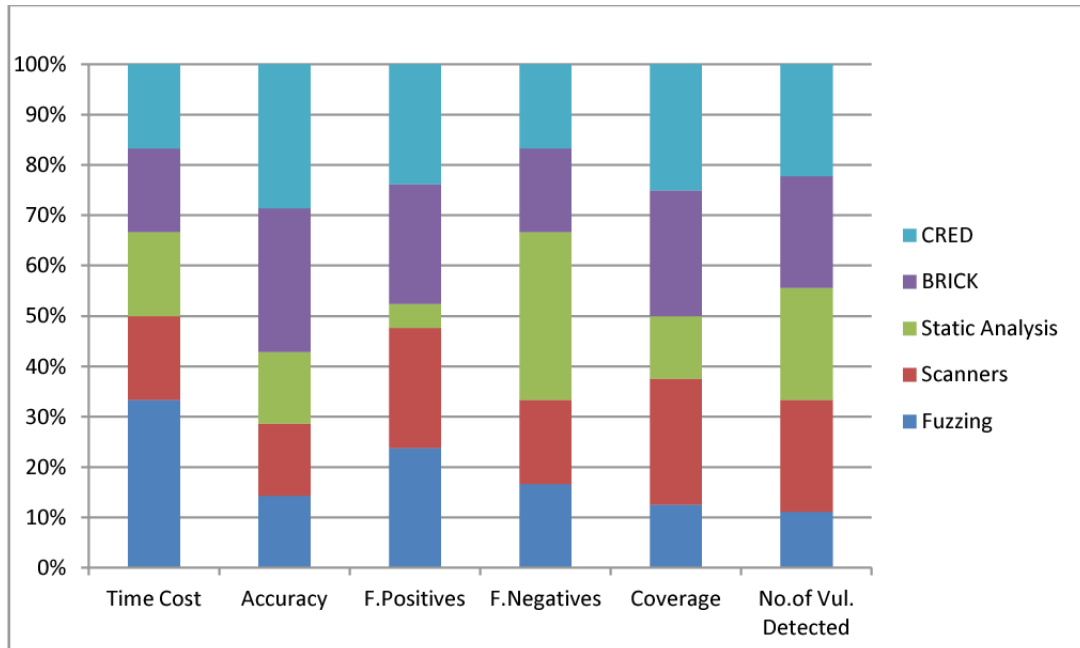


Figure 2.14: Performance Comparison of Vulnerability Detection Methods.[23]

2.8 Related Work

We discuss related work on vulnerability detection using deep learning techniques :

2.8.1 VUDENC: Vulnerability Detection with Deep Learning on a Natural Code base for Python

2.5.1.1 Principle

The main idea of this work [86] is to present a deep learning-based vulnerability detection tool that automatically learns vulnerable code features from the large real-world Python codebase. In this article, we present Vudenc (Vulnerability Detection with Deep Learning on a Natural Codebase), a deep learning-based vulnerability detection tool that automatically learns features of vulnerable code from a large and real-world Python codebase. Vudenc applies a word2vec model to identify semantically similar code tokens and to provide a vector representation. A network of long-short-term memory cells (LSTM) is then used to classify vulnerable code token sequences at a fine-grained level, highlight the specific areas in the source code that are likely to contain vulnerabilities, and provide confidence levels for its predictions.

2.5.1.2 Method

In this work: (Vulnerability Detection with Deep Learning on a Natural Codebase), a deep learning-based vulnerability detection tool that automatically learns features of vulnerable code from a large and real-world Python codebase. Vudenc applies a word2vec model to identify semantically similar code tokens and to provide a vector representation. A network of long-short-term memory cells (LSTM) is then used to classify vulnerable code token sequences at a fine-grained level, highlight the specific areas in the source code that are likely to contain vulnerabilities, and provide confidence levels for its predictions.

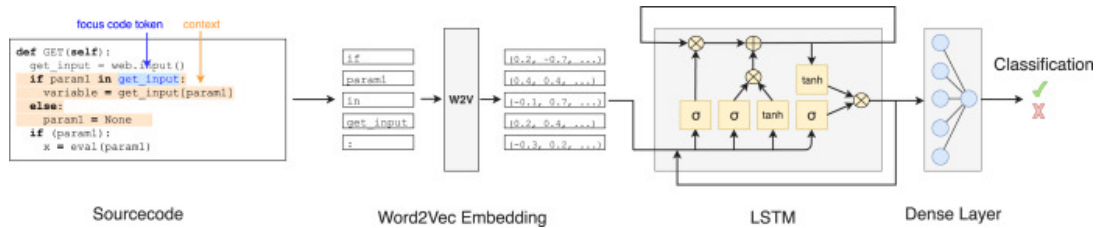


Figure 2.15: Overview of the Vudenc approach

2.5.1.3 Results

To evaluate Vudenc, he used 1,009 vulnerability-fixing commits from different GitHub repositories that contain seven different types of vulnerabilities (SQL injection, XSS, Command injection, XSRF, Remote code execution, Path disclosure, Open redirect) for training. In the experimental evaluation, Vudenc achieves a recall of 78%-87%, a precision of 82%-96%, and an F1 score of 80%-90%. Vudenc's code, the datasets for the vulnerabilities, and the Python corpus for the word2vec model are available for reproduction.

2.8.2 VulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection

2.5.2.1 Principle

The main idea of this work [90] is they propose the deep learning-based system for multi class vulnerability detection, ubbed VulDeePecker. The key insight underlying VulDeePecker is the concept of code attention, which can capture information that can help pinpoint types of vulnerabilities, even when the samples are small.[90]

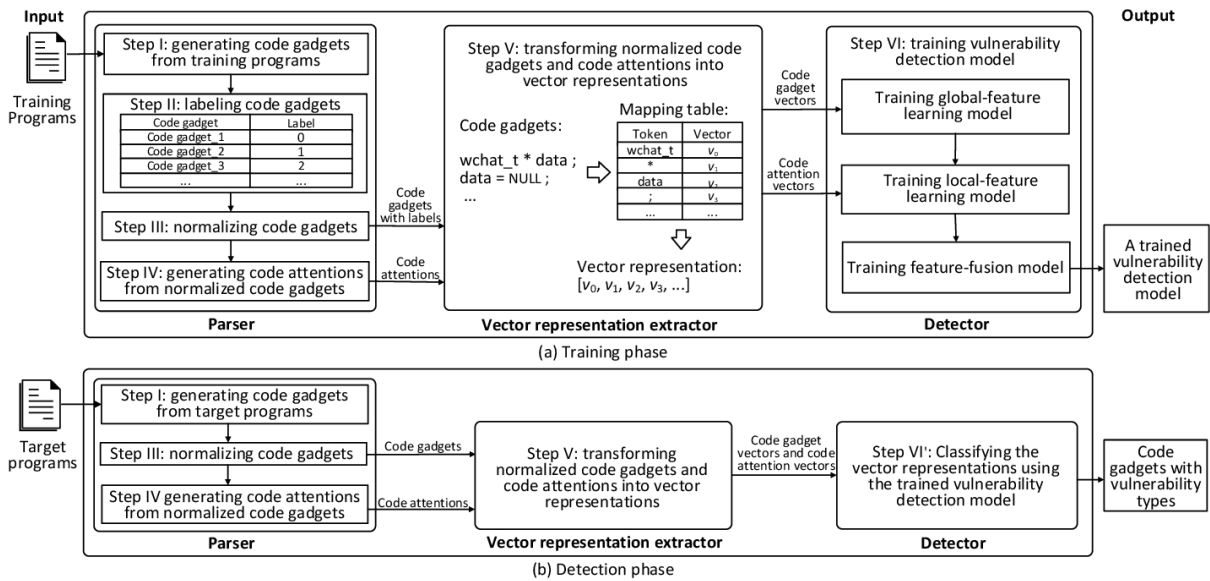


Figure 2.16: Overview of VulDeePecker’s three modules (parser, vector representation extractor, and detector) used in the training and detection phases.

2.5.2.2 Method

1)Dataset Preparation

- sources of vulnerabilities are Software Assurance Reference Dataset (SARD) and National Vulnerability Database (NVD) .
- SARD provides the statements containing the vulnerability and the vulnerability type in CWE-ID (Common Weakness Enumeration Identifier).
- NVD contains a large number of vulnerabilities in production software and provides the software products affected, the vulnerable versions, the CWE-IDs, and the patch files.

2)Training Phase

Step I: for generating code gadgets from training programs, we use the open source C/C++ code analysis tool.

Step II: they Collect all data from (SARD and NVD) easily available from the dataset.

Step III: and Step IV: for each code gadget, they wrote an automated program based on lexical analysis to analyze each token in the data.

Step V :Step VI: they use the BLSTM as a buildingblock for constructing μ VulDeePecker neural network architecture.

- implementation of the model uses Keras,They use lexical analysis to generate a set of program tokens.

3) Detection Phase they perform 5 steps for target programs, of which the first 4 steps are the same as Step I, III, IV,V in the training phase.

In Step VI', they input the code gadget vectors and their corresponding code attention vectors to be tested into the trained μ VulDeePecker model.

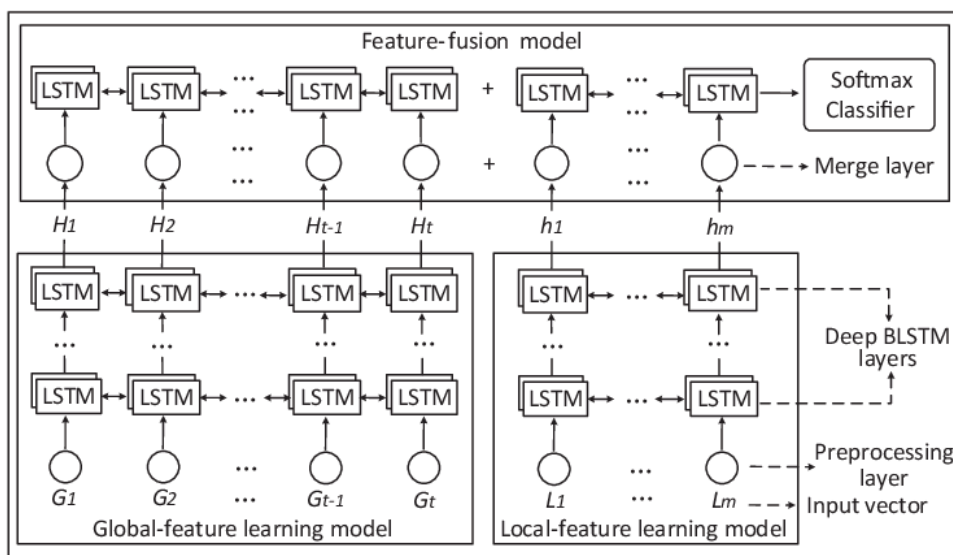


Figure 2.17: The μ VulDeePecker neural network architecture.

2.5.2.3 Results

Research Question (RQ): • RQ1: how effective is μ VulDeePecker for multiclass vulnerability detection?

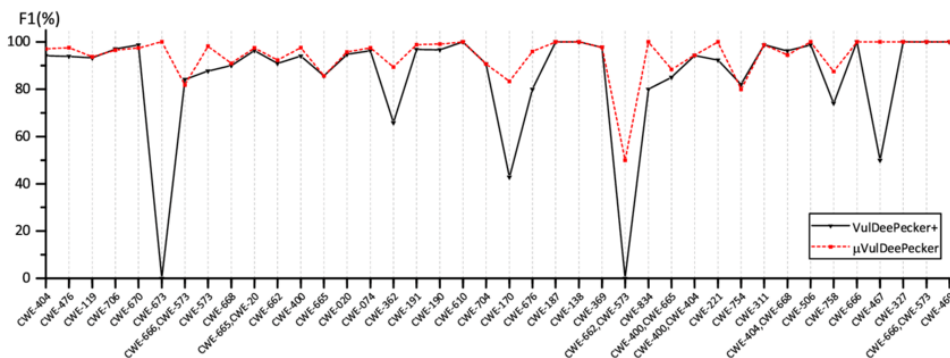


Figure 2.18: 5. Experimental comparison between the effectiveness of μ VulDeePecker and VulDeePecker+ with respect to each of the 40 vulnerability types.

2.8.3 SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities

2.5.3.1 Principle

The main idea of this work [55] is that Deep learning is successful in image processing and other applications. In particular, inspired by the area proposal concept, in image processing inspires us to adapt it to the context of vulnerability detection. However, the problem vulnerability detection is very different from the problem image processing because the latter has natural structural representations.

2.5.3.2 Method

- they propose using vulnerability syntax characteristics to identify pieces of code as initial candidates for vulnerability detection. - Vulnerability dataset. vulnerability dataset from two sources: NVD and SARD. SARD contains production, synthetic and academic programs (also known as test cases), which are categorized as “good” (having no vulnerabilities), “bad” (having vulnerabilities), and “mixed” (having vulnerabilities whose patched versions are also available). - For NVD, he focus on 19 popular C/C++ open source

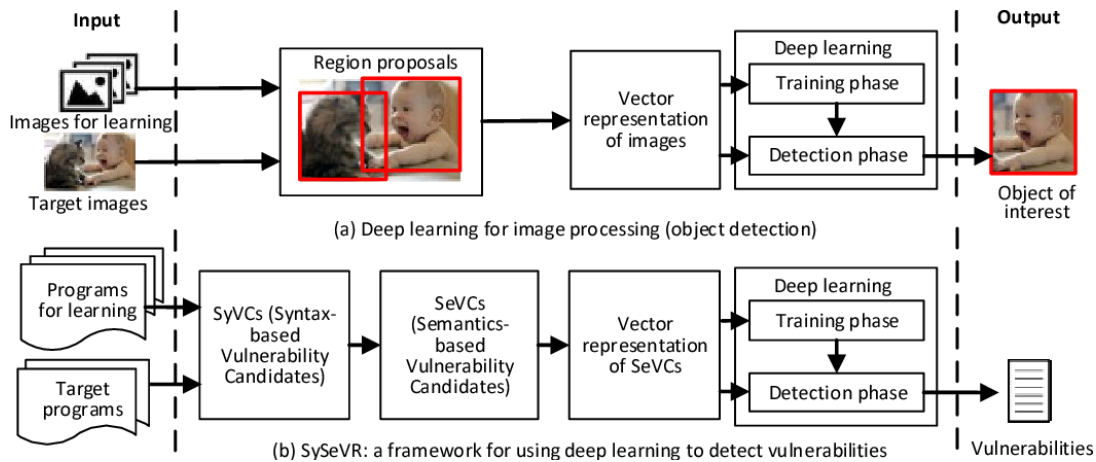


Figure 2.19: SySeVR: a framework for using deep learning to detect vulnerabilities

2.5.3.3 Results

they presented the SySeVR framework for using deep learning to detect vulnerabilities. Based on a large dataset of vulnerability they collected, they drew a number of insights, including an explanation on the effectiveness of deep learning in vulnerability detection. Moreover, they detected 15 vulnerabilities that were not reported in the NVD. Among these 15 vulnerabilities, 7 are unknown and have been reported to the vendors, and the other 8 have been “silently” patched by the vendors when releasing newer versions.

2.9 Conclusion

Vulnerabilities of the program We have attempted to provide a list of vulnerabilities, starting with the first Top Ten :2021 List and the National Vulnerability Database (NVD), We have explained old methods for detection vulnerabilities, Highlight lessons learned in related work regarding vulnerability detection. For this work, we aim to apply the deep learning to detection vulnerabilities.

Chapter 3

Design

3.1 Introduction

In the previous chapters, we presented a theoretical study on vulnerability detection and deep learning. In this chapter, we will put this concept into practice and propose a vulnerability detection system based on the source code of C/C++ functions and deep learning based on CNN and LSTM. It is organized as follows: In the first part, we describe the outline of our proposed model. In the second part, we describe in detail the design of the proposed model giving the details of each module of the design. We then define the learning method used.

3.2 General system design

In this next title, we will broadly describe our system and give the shape of its structure, such as its components and its purpose.

3.2.1 System objective

The objective of this application is the discovery of vulnerable source code of C/C++ functions from code analysis enter and to extract the characteristic points from the latter compared with the real code of the vulnerabilities published in national vulnerability NVD database, interpret and recognize the code to detect a vulnerable state or not. For this, we will use a learning system using a code base.

3.2.2 Architecture of the system

In general, we can represent the structure of the system as follows:

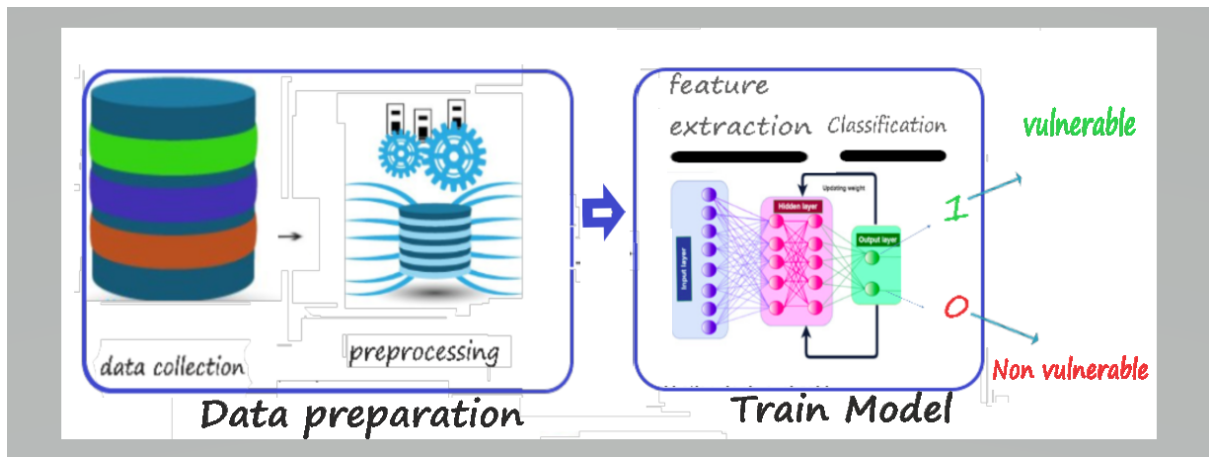


Figure 3.1: Architecture of the system.

3.3 Detailed system design

In the following, we will detail each of the steps described. The general architecture of the CNN model is as follows:

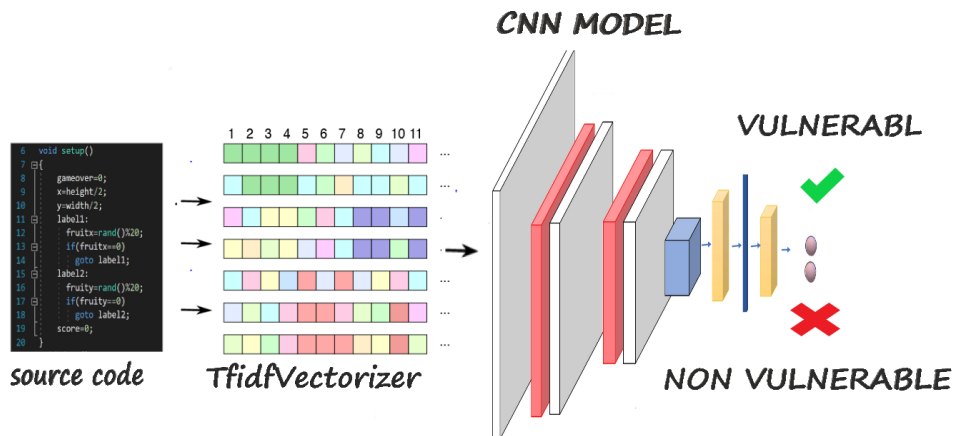


Figure 3.2: Architecture of the CNN model.

3.3.1 Data collection

data collection is the first stage of our system. It allows us to acquire data from the real world, and to design a database of examples (learning examples), which will be used in the following steps. we will collect as much data as possible, which is a collection of vulnerable c/c++ function code leading to exploitation by attackers and not vulnerable code. The process of collecting this dataset was difficult as most datasets were not free and did not have public access, but we managed to find one on the GitHub(github.com) [10]platform, which didn't contain much data, just a satisfying amount. We encountered a problem of unbalanced data and it was addressed by collecting data from the real data in GitHub (github.com) [18] to make the data balanced as shown in the following diagram :

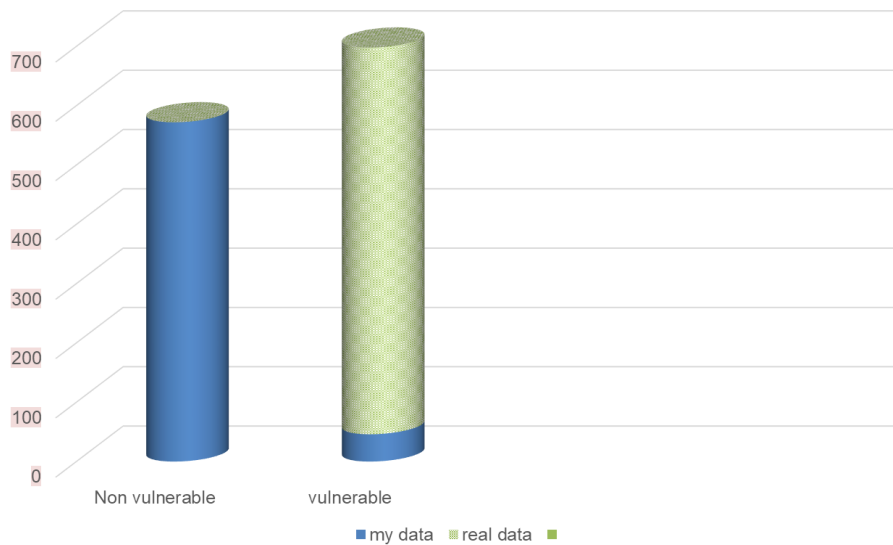


Figure 3.3: Distribute our data with/without unbalanced data problem.

Dataset	Dataset size	Total entries	Vulnerable code entries	Non-vulnerable code entries
unbalanced data	1,56 Mo +3,63 Ko	618 code	46 code	572 code
balanced data	5,83 Mo +7,52 Ko	1283 code	698 code	585 code

Table 3.1: Metadata of the Collected Data set.

The dataset contains two columns, a sentence and a Label column. The Label column value 1 means the `c/c++` code is vulnerable and 0 means the data is not vulnerable.

3.3.2 Data preparation

In this component we will try to train our system, which in turn learns in the field of classification and division of applications through specific data, where the algorithm Tfidf Vectorizer codes vulnerable and non-vulnerable `c/c++` into a matrix to easily identify and manipulate, then preserve a model and if it is high resolution, use it or not reshape it using other more accurate and efficient parameters.

TFIDFVectorization

In our system we have chosen to use TFIDFVectorization as a data vectorization technique, in which we strive to extract features from our textual dataset.

TfidfVectorizer: a vectorizer that uses the “TF-IDF” tune scale of frequencies by counting words in each text to focus on meaningful spam messages.[\[57\]](#)

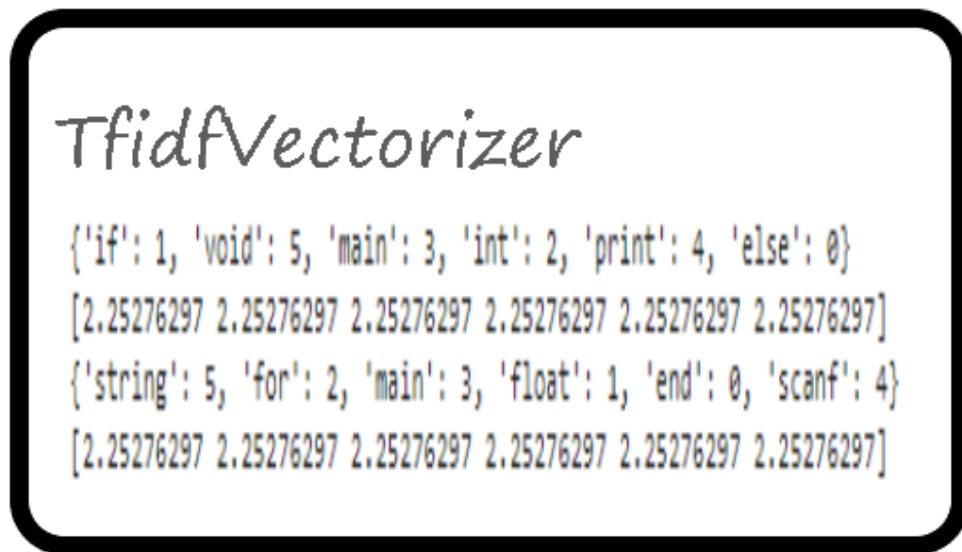


Figure 3.4: Example of TfidfVectorizer.

3.3.3 Classification and Training

In this part we will try to train our system, which in turn learns in the field of order classification and division from specific data, then saves a model and if it has high accuracy, we exploit it or not reform it with other parameters. We chose to use CNN (convolutional neural network) to train our model, which provided us with better accuracy compared to other machine learning algorithms such as LSTM and others, the following figure shows the steps of training.

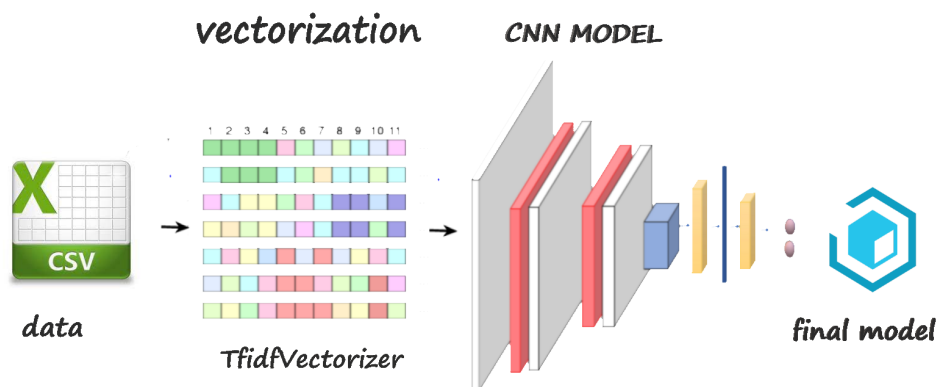


Figure 3.5: Train the Model.

We will test our simple system, by testing our model and we calculate the accuracy. In the condition-tested group our model did not see it and the result was more than 75%. Thus, we can say that our model is efficient, and we either run it or retreat another way by using other good metrics and training again.

3.3.4 Use of the Model

We can use our saved model, as follows:

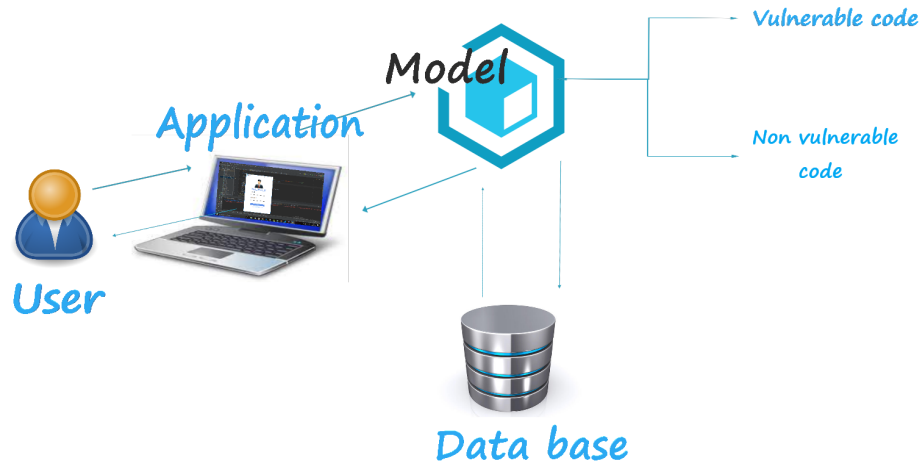


Figure 3.6: Use of the Model.

We take a text (source code in c/c++....) after processing it and give it to the model we previously trained which it considers as inputs and the form will do it and classify it categorize vulnerable code or not vulnerable.

3.4 Design by UML

3.4.1 Sequence Diagram for "Inscription "

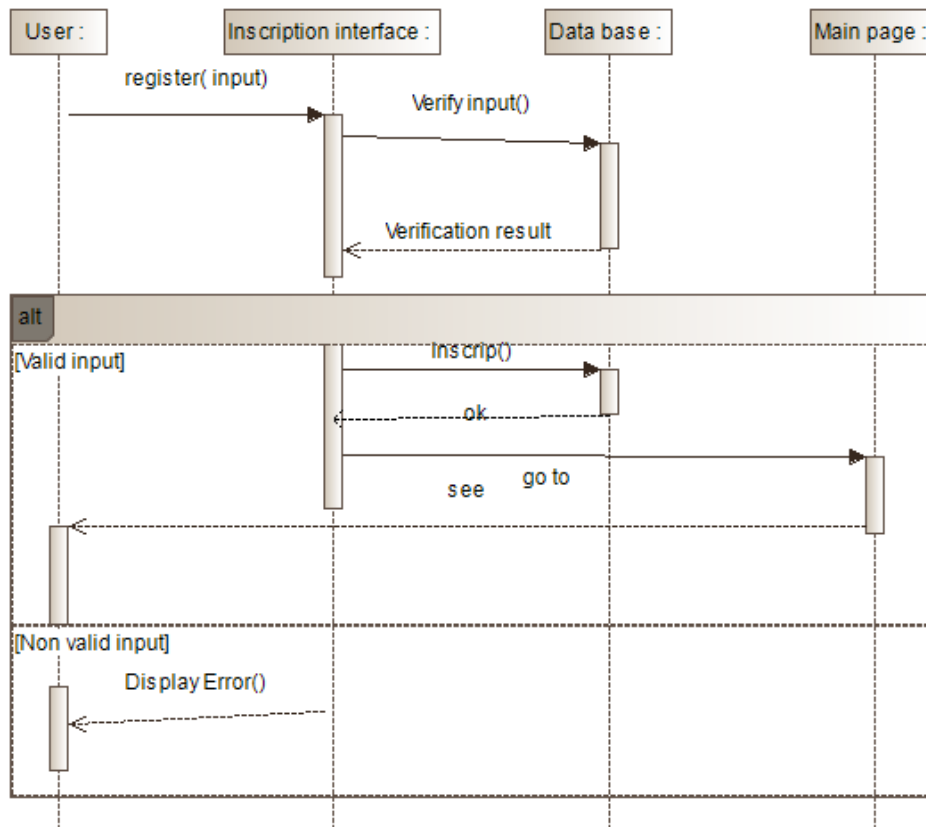


Figure 3.7: Sequence Diagram for "Inscription ".

3.4.2 Sequence diagram "authenticate"

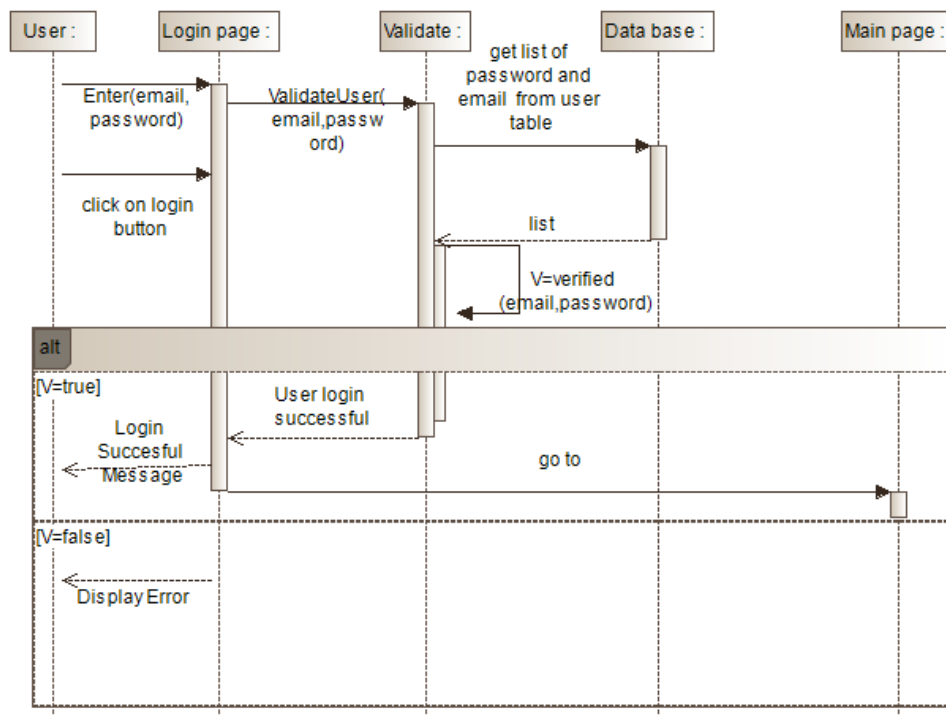


Figure 3.8: Sequence Diagram for "authenticate".

3.5 Conclusion

In this chapter, we presented the design of our system, where we presented the global design, and also detailed the measures we have taken to access our system, and we detailed the components used and the steps of base (data collection, data preparation and training).

Chapter 4

Implementation and results

4.1 Introduction

In this chapter, we will present the working environment, the programming language, and the tools we used to build the system. Thereafter we will explain all the experiments that we have applied to the proposed method and the results obtained.

4.2 Environment and tools

The following section presents the working environment, programming languages and software tools used in development.

4.2.1 Python

Is an object-oriented, inter-preted, mid-level programming language that is easy to learn and use, and is now considered among the top programming languages to learn. It is free to use, has cross-platform compatibility (Mac, Windows, Linux, Ubuntu) and has low system requirements, giving any individual the opportunity to code in Python. [72]



Google Colab

Google Colab in short. Google Colab is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. With Google Colab, it is possible to write and execute code, save and share our analyses, and access powerful computing resources, all for free from the browser.

[44]



why use google colab ?

Google has made Google Colaboratory which is the online framework where one can write, execute Deep Learning and Machine Learning codes. Here, different versions of python and different runtime environments are available. Also, it can download bigger datasets directly from the servers to google drive at very high speed. You can mount your drive with Google Colab and it can fetch the required file after your authentication. How long the amount of memory and computation provided by Google is available for user is also discussed. It provides very high-speed computations and then saves the Learning model.[60]

4.2.2 WAMP(Windows, Apache, MySQL, PHP)

WAMP is an acronym for(Windows, Apache,MySQL, PHP) is a web development package in-tended for Windows OS.35 . [67]

This stack provides developer an operating system, Web server, database and Web scripting software. The AMP stacks are also available for Linux and Macintosh operating systems, known as LAMP and MAMP respectively.[21]



4.2.3 TensorFlow

TensorFlow is also a library (open-source) for various computer programming with a range of variety of tasks. TensorFlow is the free representative mathematical library, which was used for ML functions mainly neural networks . [58]



Why Use TensorFlow ?

The TensorFlow platform was released in November 2015, in a climate of stiff competition from well-established deep learning and ML packages such as Torch and Theano. [The performance comparisons between the packages goes beyond the scope of this piece). The platform has been undergoing constant improvements since its release, thanks in part to a nascent community of contributors. TensorFlow packs quite a punch owing to its significant positive characteristics. Firstly, the package's open source nature makes it transparent, customizable, and extensible by the end user. Secondly, the documentation, support is rich and elaborate, which is a critical factor in accelerating user adoption. Lastly, the fact that the fact that the package has been backed by Google. [77]

4.2.4 Keras

Keras is a neural network library (open-source) and coding in Python programming language that able to run on most of the high levels of Ten-sorFlow, Theano, or Microsoft Cognitive Toolkit.Usually, Keras developed to fa-



Facilitate rapid experimentation using deep neural networks with user approachable, modular, and extensible. Keras library provides a higher-level approach, an extra impulsive set of concepts that create it simple to develop DL based models irrespective use of computational backend. [58]

Why Use keras ?

Of the available deep learning libraries, Keras is the most popular among practitioners, which provides users the ability to implement quickly, train, and test networks. [64]

Keras abstracts many of the complicated aspects of TensorFlow while still providing customizability and ease of use. This combination makes Keras the first choice of many for deep learning applications. As a result of its popularity and ease of use, Keras is the clear choice on which to build one end of the two-way bridge. [64]

4.2.5 TfidfVectorizer

TfidfVectorizer: Each Fasta sequences of RdRP was transformed with TfidfVectorizer method with in sklearn module. TfidfVectorizer, Convert a collection of raw documents to a matrix of TF-IDF features. [52]



4.2.6 Tkinter

Tkinter provides Python applications with an easy-to-program user interface. Tkinter supports a collection of Tk widgets that support most application needs. Tkinter is the Python interface to Tk, developed by John Ousterhout, who was originally at University of California at Berkeley and later at Sun Microsystems. Currently, Tcl/Tk is developed and supported by the Scriptics Corporation. [42]



Why Use Tkinter ?

- * Python with Tkinter makes it simple to design graphical user interfaces.
- * It gives the Tk GUI toolkit a rich object-oriented interface.
- * It supports many controls in a GUI programme, such as buttons, labels, and text boxes. [83]

4.3 Data Structures

4.3.1 Data base

The following database was used. This database contains the users of the system.

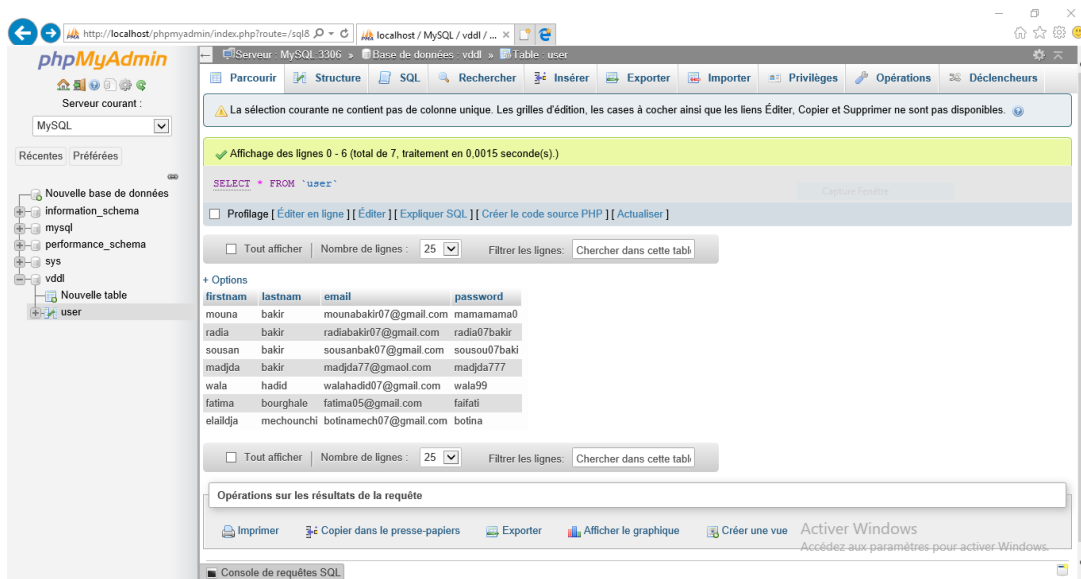


Figure 4.1: Users list.

4.3.2 Dataset Used

Dataset for CNN training

To train the CNN model, we used a GITHUB(github.com) database which was downloaded from the website which is size(1.56MB+3.63KB) but we notice that this data is unbalanced data because number of non vulnerable data is 572 code but vulnerable data is 46 code and we also added some addition for balanced data through real code in another site and we split them into two types where the first type contains 698 items which are 1 positive and 572 items which are 0 all negative and we enter them into the CNN form for training we will display an image which is shown below.

1263	static ssize_t CVE_2013_4512_PATCHED_exitcode_proc_write(struct file *file,	1
1264	static int CVE_2013_4511_VULN_au1200fb_fb_mmap(struct fb_info *info, struct vm_area_struct *vma)	1
1265	int CVE_2013_4511_VULN_au1100fb_fb_mmap(struct fb_info *fbi, struct vm_area_struct *vma)	1
1266	static int CVE_2013_4511_PATCHED_au1200fb_fb_mmap(struct fb_info *info, struct vm_area_struct *vma)	1
1267	int CVE_2013_4511_PATCHED_au1100fb_fb_mmap(struct fb_info *fbi, struct vm_area_struct *vma)	1
1268	static inline int CVE_2013_4470_VULN_ip6_ufo_append_data(struct sock *sk,	1
1269	static inline int CVE_2013_4470_VULN_ip_ufo_append_data(struct sock *sk,	1
1270	static inline int CVE_2013_4470_PATCHED_ip6_ufo_append_data(struct sock *sk,	1

Figure 4.2: vulnerable data set .

1271	static void writepng_error_handler(png_structp png_ptr, png_const_charp msg);	0
1272	/* unexpected pnmtyp; note that outfile might be stdout */	0
1273	int writepng_encode_image(mainprog_info *mainprog_ptr)	0
1274	static int wpng_isvalid_latin1(uch *p, int len)	0
1275	static void wpng_cleanup(void)	0
1276	int main(int argc, char **argv)	0
1277	int main(int argc, char **argv)	0
1278	#endif	0
1279	}	0
1280		0
1281	}	0
1282	}	0
1283	#endif	0

Figure 4.3: Non vulnerable data set .

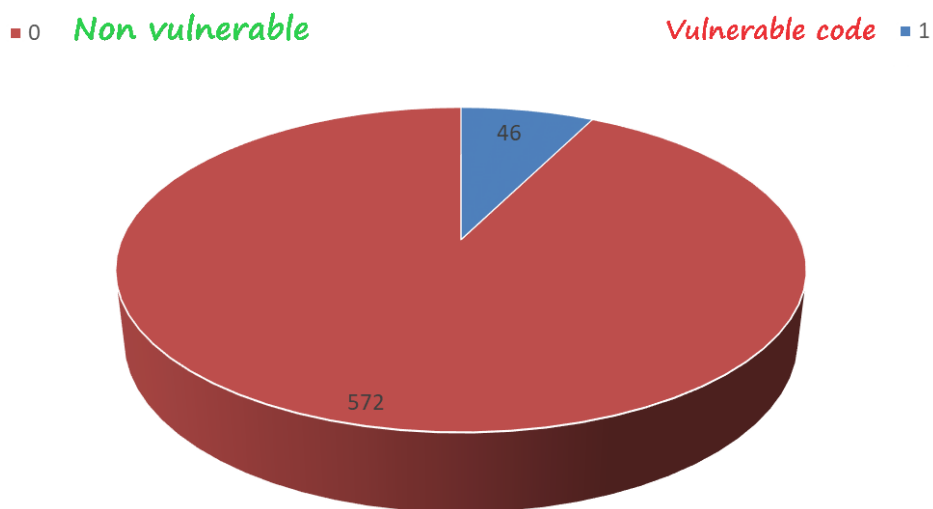


Figure 4.4: unbalanced data .

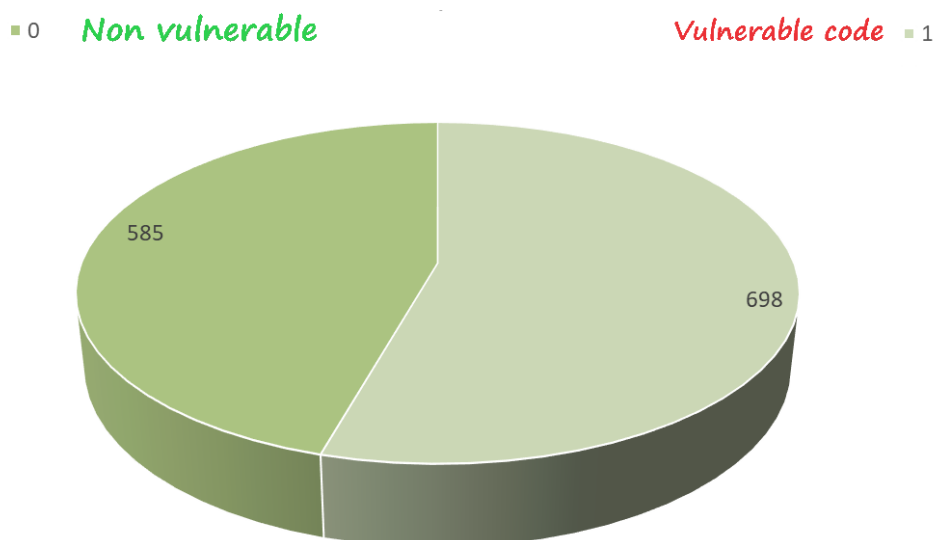


Figure 4.5: balanced data.

4.3.3 Training and Testing

Why Use CNN ?

Convolutional neural networks (CNNs) have interesting and widely varied architectures catering to the requirement of learning features from structured data volumes. However, there are certain design considerations common to all CNN architectures that enable us to craft correct implementations of the architectures suited to our objectives. It is more than just mathematical calculations-it's an art in itself! Thus, in the present study, we'll put our knowledge of CNNs and their distinct layers in practice by

making use of Google’s open-source machine learning platform.

We built the layers of the sequential model as follows :

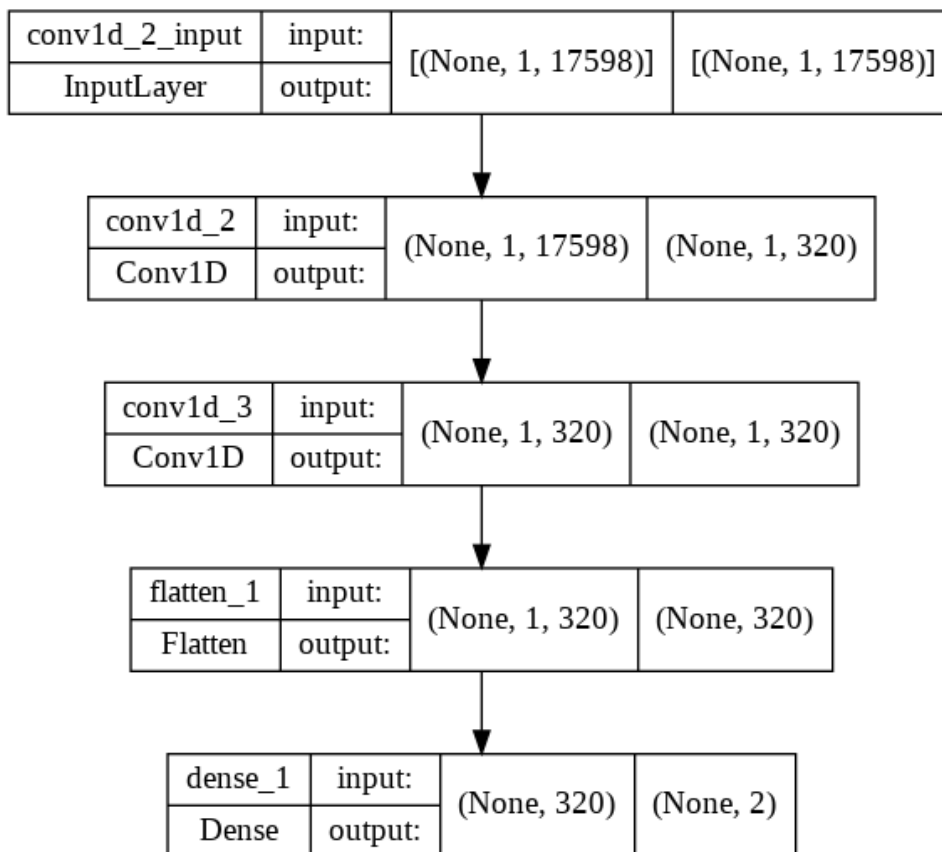


Figure 4.6: building layers.

We created the model and tested it with 20% of the dataset as follows :

```

number of learning
sessions
1 model.compile(optimizer='adam', loss=tf.keras.losses.BinaryCrossentropy(), metrics=['accuracy'])
2
3 his = model.fit(X_train, y_train, epochs=1000, validation_split=0.2)
test
epoch 48/1000
23/23 [=====] - 4s 169ms/step - loss: 8.6777e-06 - accuracy: 1.0000 - val_loss: 0.0621 - val_accuracy: 0.9667
epoch 49/1000
23/23 [=====] - 4s 170ms/step - loss: 8.3107e-06 - accuracy: 1.0000 - val_loss: 0.0622 - val_accuracy: 0.9667
epoch 50/1000
23/23 [=====] - 4s 173ms/step - loss: 7.9500e-06 - accuracy: 1.0000 - val_loss: 0.0624 - val_accuracy: 0.9667
epoch 51/1000
23/23 [=====] - 4s 170ms/step - loss: 7.6153e-06 - accuracy: 1.0000 - val_loss: 0.0625 - val_accuracy: 0.9667
epoch 52/1000
23/23 [=====] - 4s 172ms/step - loss: 7.3055e-06 - accuracy: 1.0000 - val_loss: 0.0626 - val_accuracy: 0.9667
epoch 53/1000
23/23 [=====] - 4s 173ms/step - loss: 6.9987e-06 - accuracy: 1.0000 - val_loss: 0.0627 - val_accuracy: 0.9667
epoch 54/1000
23/23 [=====] - 4s 172ms/step - loss: 6.7189e-06 - accuracy: 1.0000 - val_loss: 0.0628 - val_accuracy: 0.9667

```

Figure 4.7: Building layers.

Accuracy results : The method for calculating the accuracy is as follows : [73]

$$accuracy = \frac{(Truepositive + truenegative) * 100}{Totalknees}$$

The method for calculating the precision is as follows : [73]

$$Precision = \frac{Truepositive}{Truepositive + FalsePositives}$$

The method for calculating the The recall is as follows : [73]

$$Recall = \frac{Truepositive}{Truepositive + Falsenegative}$$

	precision	recall	f1-score	support
0	0.98	0.99	0.99	132
1	0.99	0.98	0.99	125
accuracy			0.99	257
macro avg	0.99	0.99	0.99	257
weighted avg	0.99	0.99	0.99	257

Figure 4.8: accuracy result(Model Accuracy,Model Loss)

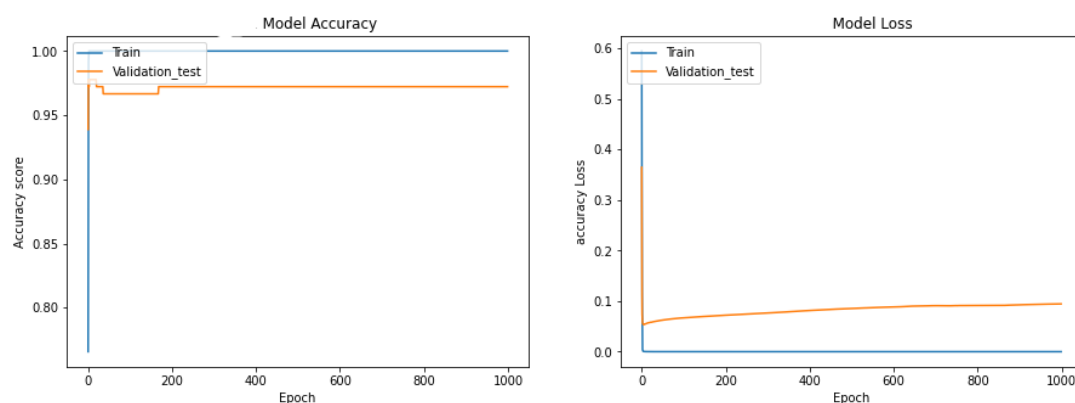


Figure 4.9: Comparison accuracy result.

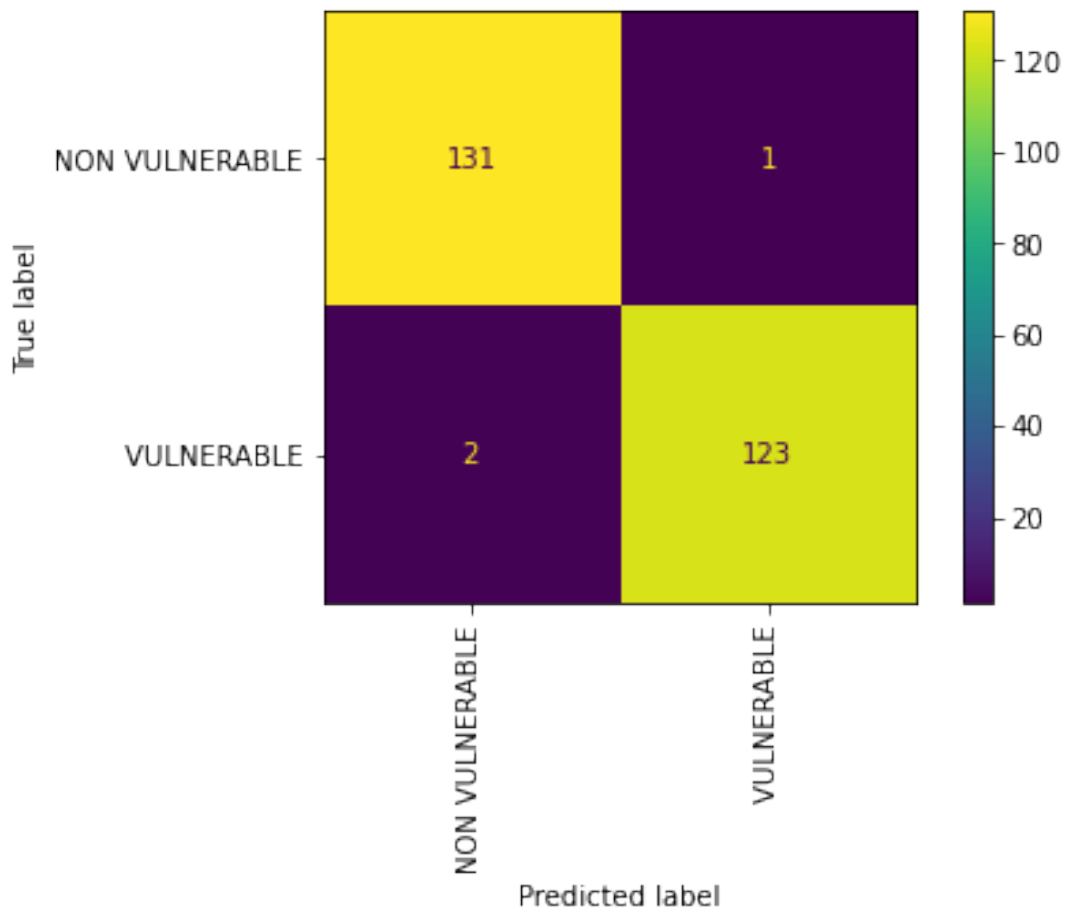


Figure 4.10: Matrix Of CNN Model.

Matrix Description :

- case 0 (Non vulnerable) : Discover on 131 codes And fail to discover on 1 code
- case 1 (vulnerable) : Discover on 123 codes And fail to discover on 2 codes To access a good model, we tested our dataset on LSTM and got the results below, which is not enough compared to our model’s achievement of accuracy. **Comparing results**

Name of classifier	CNN	LSTM
Evaluation Accuracy	99%	97%

Table 4.1: CNN Empirical Results Analysis and LSTM for the same dataset

So the best model is CNN because it’s bigger than them in terms of accuracy, it’s convenient and effective in detecting and preventing vulnerabilities.

4.4 Our application interfaces

4.4.1 Interface "Login"


This interface allows a pre-registered user to enter once they have written their email and password,As shown in the following figure :



Figure 4.11: Login Interface.

4.4.2 Interface "Registration"

This interface allows the server to register for the first time ,It allows you to enter the username and requires email and password... As shown in the following figure :



The image shows a web browser window titled "Inscription". The main heading is "Vul_Dete_DL" in blue. Below the heading is an illustration of a man in a suit with a green arrow pointing up. There are four input fields labeled "Firstname", "Lastname", "Email", and "Password". At the bottom, there are two blue buttons: "Sign up" and "Reinitialise".

Figure 4.12: Registration Interface.

4.4.3 Main interface

Main interface that allows you to choose the file that will be processed, As shown in the following figure

:

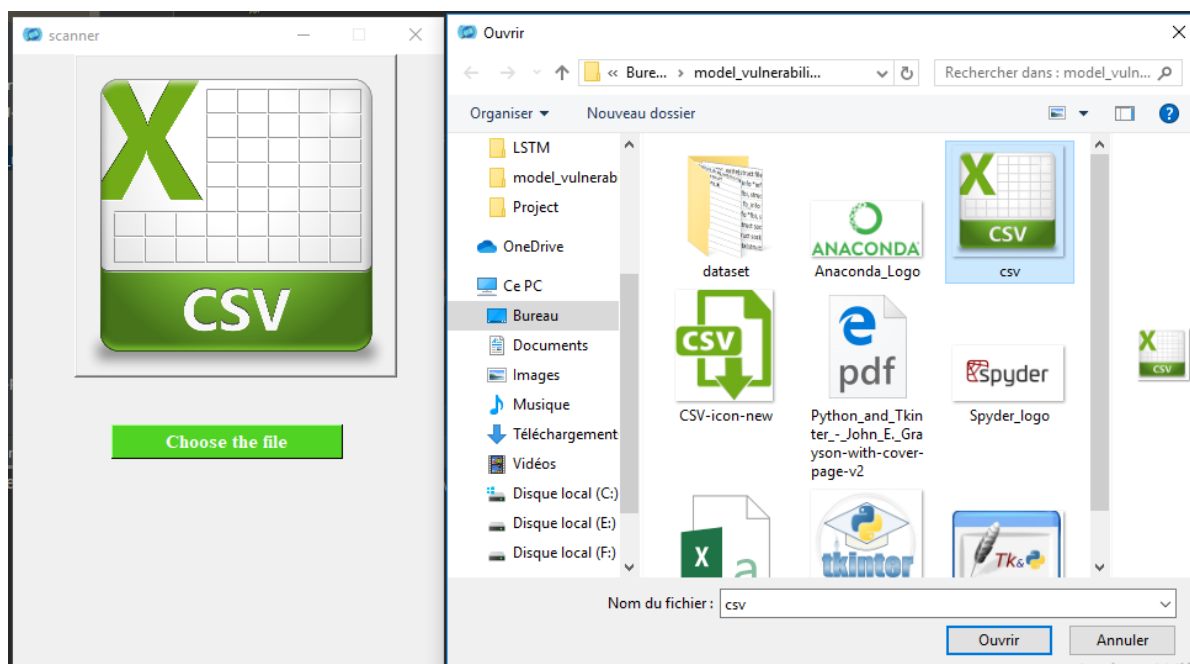


Figure 4.13: Testing phase.

4.5 Conclusion

In this chapter, we presented the implementation stages of our system, describing the environment and development tools we used plus a database. The trained model was tested on the database, like we also introduced graphical interfaces to our system. Finally, we have explained the experiences and results obtained.

General conclusion

Preventing vulnerabilities in software is not an easy task as hackers and attackers find new ways to break protections what makes these techniques obsolete, researchers are considering applying machine learning in the field of cyber security to provide more dynamic and robust protection for new types of attacks that we have never seen before.

Machine learning is still a developing area in the field. of cyber security, and there is a lack of libraries, frameworks and open source tools to use for issues related to threats and attacks. In this work, we have proposed a method to prevent common and dangerous vulnerabilities that occur in C code . The results were effective. While we struggled to collect data, it was not easy. To continue work on this project, we could increase the size of the dataset by collecting more or even using the data already built to increase efficiency.

Bibliography

- [1] A06:2021,sited as 23/04/2022. URL: <https://support.f5.com/csp/article/K17045144>.
- [2] Artificial intelligence. sited as 23/04/2022. URL: <https://www.dilepix.com/hs-fs/hubfs/IA-machine-learning-deep-learning.png?width=600&name=IA-machine-learning-deep-learning.png>.
- [3] Autoencoders , sited as 23/04/2022. URL: https://miro.medium.com/max/1200/1*nqzWupxC60iAH2dYrFT78Q.png.
- [4] A compendium of deep learning frameworks , sited as 23/04/2022. URL: https://www.ripublication.com/ijaer19/ijaerv14n10_24.pdf.
- [5] Computer software , sited as 23/04/2022. URL: <https://sasakawaeducation.org/wp-content/uploads/2020/10/Computer-Software.pdf>.
- [6] Computing basics, sited as 23/04/2022. URL: https://ftms.edu.my/v2/wp-content/uploads/2019/02/csca0101_ch07.pdf.
- [7] Convolutional neural networks ,sited as 23/04/2022. URL: https://1.cms.s81c.com/sites/default/files/2021-01-06/ICLH_Diagram_Batch_02_17A-ConvolutionalNeuralNetworks-WHITEBG.png.
- [8] Deep belief networks , sited as 23/04/2022. URL: <https://imgur.com/CodLXb0>.
- [9] deep-learning-et-machine-learning , sited as 23/04/2022. URL: <https://www.robot-magazine.fr/wp-content/uploads/2021/06/deep-learning-et-machine-learning.jpg>.
- [10] Function-level-vulnerability-detection , sited as 26/04/2022. URL: <https://github.com/DanielLin1986/Function-level-Vulnerability-Detection/projects?type=beta>.
- [11] game , sited as 23/04/2022. URL: <https://towardsdatascience.com/building-a-deep-neural-network-to-play-fifa-18-dce54d45e675>.
- [12] Generative adversarial networks , sited as 23/04/2022. URL: https://miro.medium.com/max/1276/0*npFxMmmyRVQnNVHq.jpg.
- [13] machine or deep learning process,sited as 12/05/2022. URL: <https://www.alamy.com/data-science-machine-or-deep-learning-process-image271805675.html>.
- [14] owasp, sited as 23/04/2022. URL: <https://owasp.org/Top10/A00-about-owasp/>.

- [15] Recurrent neural networks(rnn), sited as 23/04/2022. URL: https://miro.medium.com/max/3309/1*5bjD7kmtaJI-n3qztBC2Ig.png.
- [16] semi supervised learning(hybrid learning) , sited as 23/04/2022. URL: <https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning>.
- [17] supervised learning ,sited as 23/04/2022. URL: https://cdn-images-1.medium.com/max/1600/1*Iz7bCLrPTImnBD00EyE3LA.png.
- [18] Sysevr , sited as 26/04/2022. URL: <https://github.com/SySeVR/SySeVR>.
- [19] unsupervised learning,sited as 23/04/2022. URL: https://cdn-images-1.medium.com/max/1440/1*YU1_BcqFPgX49sSb5yrk3A.jpeg.
- [20] Vulnerability , sited as 23/04/2022. URL: <https://www.cve.org/ResourcesSupport/Glossary?activeTerm=glossaryCVEID#>.
- [21] Sonam Agrawal and Rajan Dev Gupta. Development and comparison of open source based web gis frameworks on wamp and apache tomcat web servers. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(4):1, 2014.
- [22] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8(1):1–74, 2021.
- [23] Richard Amankwah, Patrick Kwaku Kudjo, and Samuel Yeboah Antwi. Evaluation of software vulnerability detection methods and tools: a review. *International Journal of Computer Applications*, 169(8):22–27, 2017.
- [24] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.
- [25] Qifang Bi, Katherine E Goodman, Joshua Kaminsky, and Justin Lessler. What is machine learning? a primer for the epidemiologist. *American journal of epidemiology*, 188(12):2222–2239, 2019.
- [26] Marcel Boehme, Cristian Cadar, and Abhik Roychoudhury. Fuzzing: Challenges and reflections. *IEEE Softw.*, 38(3):79–86, 2021.
- [27] Guillaume Bouchard. Efficient bounds for the softmax function, applications to inference in hybrid models, 2007.
- [28] G Caldarini, S Jaf, and K McGarry. A literature survey of recent advances in chatbots. *information* 2022, 13, 41, 2022.
- [29] Gaurav Chadha. Web application security,sited as 23/04/2022. URL: <https://cs-coe.iisc.ac.in/wp-content/uploads/2020/07/Web-Application-Security.pdf>.
- [30] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018. doi:10.1109/MSP.2017.2765202.
- [31] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

- [32] Bob Dadae. Unsupervised learning. 1999.
- [33] data flair.training. Deep learning terminologies,sited as 23/04/2022. URL: https://data-flair.training/blogs/deep-learning-terminologies/#google_vignette.
- [34] data flair.training. ML,sited as 23/04/2022. URL: <https://data-flair.training/blogs/wp-content/uploads/sites/2/2017/07/what-is-machine-learning.jpg>.
- [35] data flair.training. Svm,sited as 23/04/2022. URL: <https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/introduction-to-SVM.png>.
- [36] data flair.training. types of ml,sited as 23/04/2022. URL: <https://data-flair.training/blogs/wp-content/uploads/sites/2/2017/07/Types-of-Machine-Learning-1.jpg>.
- [37] Vinicius Rafael Lobo de Mendonca, Cassio Leonardo Rodrigues, Fabrízio Alphonsus A de MN Soares, and Auri Marcelo Rizzo Vincenzi. Static analysis techniques and tools: A systematic mapping study. *ICSEA*, 2013.
- [38] Harshavardhan CA Deekshith Shetty, M Jayanth Varma, Shrishail Navi, and Mohammed Riyaz Ahmed. Diving deep into deep learning: History, evolution, types and applications.
- [39] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *CoRR*, abs/1811.12560, 2018. URL: <http://arxiv.org/abs/1811.12560>, [arXiv:1811.12560](https://arxiv.org/abs/1811.12560).
- [40] Matthieu Gilson, Anthony Burkitt, and J Leo van Hemmen. Stdp in recurrent neuronal networks. *Frontiers in computational neuroscience*, 4:23, 2010.
- [41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [42] John E Grayson. *Python and Tkinter programming*. Manning Publications Co. Greenwich, 2000.
- [43] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.
- [44] Teddy Surya Gunawan, Arselan Ashraf, Bob Subhan Riza, Edy Victor Haryanto, Rika Rosnelly, Mira Kartiwi, and Zuriati Janin. Development of video-based emotion recognition using deep learning with google colab. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 18(5):2463–2471, 2020.
- [45] Aboul Ella Hassanien, M Tolba, and Ahmad Taher Azar. *Advanced machine learning technologies and applications*. Springer, 2021.
- [46] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.
- [47] Ingrid Hrga and M Ivašić-Kos. Deep image captioning: An overview. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 995–1000. IEEE, 2019.
- [48] <https://data-flair.training/>. Types of artificial intelligence,sited as 23/04/2022. URL: <https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/10/Types-of-AI.jpg>.
- [49] Zheng Hu, Jiaojiao Zhang, and Yun Ge. Handling vanishing gradient problem using artificial derivative. *IEEE Access*, 9:22371–22377, 2021. [doi:10.1109/ACCESS.2021.3054915](https://doi.org/10.1109/ACCESS.2021.3054915).

- [50] Vikramaditya Jakkula. Tutorial on support vector machine (svm). *School of EECS, Washington State University*, 37(2.5):3, 2006.
- [51] Viera Maslej Krešňáková, Martin Sarnovský, and Peter Butka. Deep learning methods for fake news detection. In *2019 IEEE 19th International Symposium on Computational Intelligence and Informatics and 7th IEEE International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics (CINTI-MACRo)*, pages 000143–000148. IEEE, 2019.
- [52] Gudipati Pavan Kumar. Hierarchical clustering on rna dependent rna polymerase using machine learning. *bioRxiv*, 2021.
- [53] Nilesh A Lal, Salendra Prasad, and Mohammed Farik. A review of authentication methods. *vol*, 5:246–249, 2016.
- [54] Ruby B Lee. Security basics for computer architects. *Synthesis Lectures on Computer Architecture*, 8(4):1–111, 2013.
- [55] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, and Zhaoxuan Chen. Sysevr: A framework for using deep learning to detect software vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, 2021.
- [56] lifars. injection,sited as 23/04/2022. URL: <https://lifars.com/2020/04/injection-attacks-explained/>.
- [57] Moohong Min, Jemin J Lee, and Kyungho Lee. Detecting illegal online gambling (iog) services in the mobile environment. *Security and Communication Networks*, 2022, 2022.
- [58] Wazir Muhammad, Irfan Ullah, and Mohammad Ashfaq. An introduction to deep convolutional neural networks with keras. In *Machine learning and deep learning in real-time applications*, pages 231–272. IGI Global, 2020.
- [59] Naila Murray and Florent Perronnin. Generalized max pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.
- [60] Mark J Nelson and Amy K Hoover. Notes on using google colaboratory in ai education. In *Proceedings of the 2020 ACM conference on innovation and Technology in Computer Science Education*, pages 533–534, 2020.
- [61] NVD. National vulnerability database ,sited as 23/04/2022. URL: <https://nvd.nist.gov/>.
- [62] Elizabeth Fong Romain Gaucher Vadim Okun, Paul E Black, and Eric Dalci. Building a test suite for web application scanners.
- [63] Keiron O’Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.
- [64] Jordan Ott, Mike Pritchard, Natalie Best, Erik Linstead, Milan Curcic, and Pierre Baldi. A fortran-keras deep learning bridge for scientific computing. *Scientific Programming*, 2020, 2020.
- [65] owasp. Top10,sited as 23/04/2022. URL: <https://owasp.org/Top10/>.
- [66] Niall O’Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In *Science and information conference*, pages 128–144. Springer, 2019.

- [67] P Padmavathi, K Chandrashekar, Anagha S Setlur, and Vidya Niranjana. Mutaxome: A novel database for identified somatic variations of in silico analyzed cancer exome datasets. *Cancer Informatics*, 21:11769351221097593, 2022.
- [68] Subarno Pal, Soumadip Ghosh, and Amitava Nag. Sentiment analysis in the light of lstm recurrent neural networks. *International Journal of Synthetic Emotions (IJSE)*, 9(1):33–39, 2018.
- [69] George Philipp, Dawn Song, and Jaime G Carbonell. Gradients explode-deep networks are shallow-resnet explained. 2018.
- [70] Martin Riedmiller and AM Lernen. Multi layer perceptron. *Machine Learning Lab Special Lecture, University of Freiburg*, pages 7–24, 2014.
- [71] Cecko Robert, Jamrózy Jerzy, Ješko Waldemar, Kuśmierk Ewa, Lange Marek, and Owsiany Mariusz. Automatic speech recognition and its application to media monitoring. *CMST*, 27(2):41–55, 2021.
- [72] Damien Rolon-Mérette, Matt Ross, Thaddé Rolon-Mérette, and Kinsey Church. Introduction to anaconda and python: Installation and setup. *Python for research in psychology*, 16(5):S5–S11, 2016.
- [73] REC Rose. The accuracy of joint line tenderness in the diagnosis of meniscal tears. *West Indian Med J*, 55(5):323–326, 2006.
- [74] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.
- [75] SANS. Cwe,sited as 23/04/2022. URL: <https://www.sans.org/top25-software-errors/>.
- [76] SANS. Cwe,sited as 23/04/2022. URL: <https://www.softwaretestinghelp.com/sans-top-20-security-vulnerabilities/>.
- [77] Abhineet Saxena. Convolutional neural networks: An illustration in tensorflow. *XRDS*, 22(4):56–58, jun 2016. doi:10.1145/2951024.
- [78] Enrico Schiavone. Providing continuous authentication and non-repudiation security services. In *Student Forum of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016.
- [79] Simha Sethumadhavan, Adam K Hastings, and Mohammed Ibn Zaid Tarek. Introduction to security for computer architecture students. 2019.
- [80] Stuart C Shapiro. *Encyclopedia of artificial intelligence second edition*. John, 1992.
- [81] synack. Broken access control,sited as 23/04/2022. URL: <https://www.synack.com/blog/preventing-broken-access-control-the-no-1-vulnerability-in-the-owasp-top-10-2021/>.
- [82] Isaac Triguero, Salvador García, and Francisco Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information systems*, 42(2):245–284, 2015.
- [83] Kuldeep Vayadande, Samruddhi Pate, Naman Agarwal, Dnyaneshwari Navale, Akhilesh Nawale, and Piyush Parakh. Modulo calculator using tkinter library. *EasyChair Preprint*, (7578), 2022.

- [84] wallarm. A02:2021,sited as 23/04/2022. URL: <https://www.wallarm.com/what/a02-2021-cryptographic-failures>.
- [85] Sun-Chong Wang. Artificial neural network. In *Interdisciplinary computing in java programming*, pages 81–100. Springer, 2003.
- [86] Laura Wartschinski, Yannic Noller, Thomas Vogel, Timo Kehrer, and Lars Grunske. Vudenc: Vulnerability detection with deep learning on a natural codebase for python. *Information and Software Technology*, page 106809, 2022.
- [87] Tao Xu, Han Zhang, Cheng Xin, Edward Kim, L Rodney Long, Zhiyun Xue, Sameer Antani, and Xiaolei Huang. Multi-feature based benchmark for cervical dysplasia classification evaluation. *Pattern recognition*, 63:468–475, 2017.
- [88] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL: <http://arxiv.org/abs/1212.5701>, [arXiv:1212.5701](https://arxiv.org/abs/1212.5701).
- [89] Aston Zhang, ZC Lipton, Mu Li, and AJ Smola. Dive into deep learning release 0.7. 1, 2020.
- [90] Deqing Zou, Sujuan Wang, Shouhuai Xu, Zhen Li, and Hai Jin. uvuldeepecker: A deep learning-based system for multiclass vulnerability detection. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2224–2236, 2019.