Intrusion Detection System in IoT

Emanuel TRÎNC
Dept. Comunicatii
Politehnica Timisoara
Timişoara, România
emanuel.tranc@gmail.com

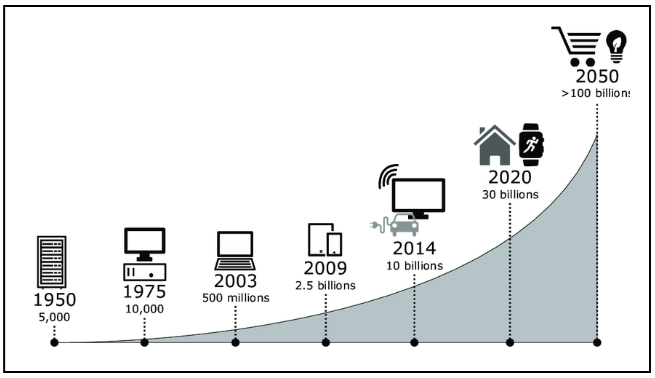
***Abstract*— With the rapid expansion of the Internet of Things (IoT), the security of interconnected devices has become a paramount concern. This paper introduces a real-world implementation of an Intrusion Detection System (IDS) designed specifically for IoT environments. The proposed IDS utilizes advanced anomaly detection techniques and Artificial Intelligence algorithms to identify and mitigate potential security threats. The system is tailored to accommodate the unique characteristics of IoT networks, including resource constraints and diverse communication protocols. We present a comprehensive evaluation of the IDS through experimental setups simulating real-world IoT scenarios, demonstrating its effectiveness in detecting various types of intrusions. The results contribute to the growing body of knowledge aimed at securing IoT deployments and provide insights for the development of robust intrusion detection strategies in the context of interconnected devices.**

***Keywords*— Intrusion Detection System (IDS), Artificial Intelligence (AI), Machine Learning (ML), Neural Network (NN), Deep Learning (DL), Internet of Things (IoT).**

I. Introduction

In the past couple of decades, the adoption of data and IoT devices has seen exponential growth. It is expected that by 2050 the number of IoT devices will exceed 100 billion [1], which is a huge difference to the current number of around 15-30 billion (depending on the information source).

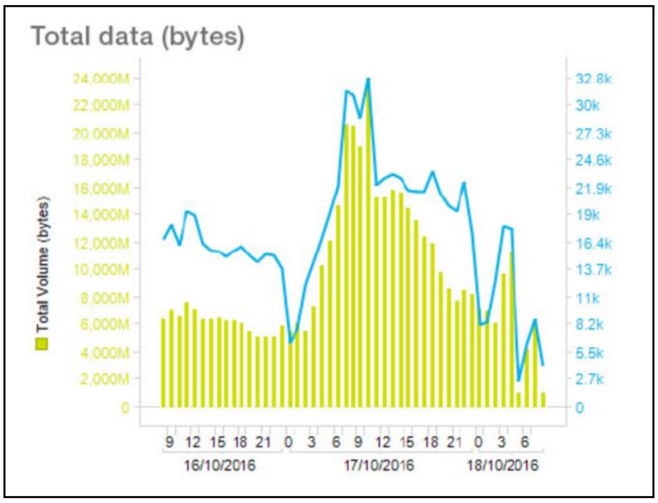
This is just an estimation, but judging by the way the technology was continuously beating all the estimations decade after decade in the past 3 or 4 decades, the number of IoT devices can be even much higher than 100 billion by 2050, if we take into account some other market statistics which envision that the number of IoT devices will hit 75 billion devices by 2025 already, than we could really expect much higher numbers by 2050 [2].



(Fig. 1.1) IoT Devices estimations by 2050

In terms of data usage/generation by the IoT devices, some estimations tell us that the average data volume per IoT device has peak values slightly over 30Kb/h, which translated in today's total number of IoT devices of around 26-30 billion means 10.8 Pb/12h. By 2050 that number could be 3 to 10 times as much, or even higher.

So, in the light of these metrics and based on the fact that one cannot just go there in an online store and buy an app or



(Fig. 1.2) Average data volume per IoT device

device to protect their IoT devices against all sorts of security threats, there is a high demand for some sort of IDS/IPS implementation so that all these billion of users with their trillion of IoT devices could feel safe connecting to the internet.

The purpose of this paper is to implement and IDS for IoT environment using Machine Learning and Neural Networks technologies taking into consideration the challenges of deploying such systems in the real world.

II. Intrusion Detection Systems Overview

Intrusion can be defined as any kind of unauthorized activity that causes damage to an information system. This means that any attack that could pose a possible threat to the information confidentiality, integrity or availability will be considered as intrusion.

A **Network Intrusion** refers to unauthorized access, monitoring, or manipulation of computer networks, systems, or data. It involves an individual or entity gaining unauthorized access to a network with the intent of exploiting vulnerabilities, stealing information, disrupting normal network operations, or causing malicious activities.

Preventing and mitigating network intrusions involve implementing robust cybersecurity measures, such as Firewalls, Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), secure network configurations, regular software updates, and employee training on security best practices. Additionally, monitoring network traffic and promptly responding to any suspicious activity is crucial for detecting and mitigating potential intrusions. The focus of this paper will be on Intrusion Detection Systems.

An Intrusion Detection System is a security technology designed to monitor and analyze network or system activities for signs of unauthorized access, malicious activities, or security policy violations. The primary purpose of an IDS is to identify and respond to potential security incidents, providing an additional layer of defense against cyber threats.

IDS systems can be classified as follows:

I. Based on Reaction and Response

1. Intrusion Detection System (IDS)
2. Intrusion Prevention System (IPS)

II. Based on Intrusion/Detection Methods (or Approach)

1. Signature-based IDS (SIDS)
2. Anomaly-based IDS (AIDS)
3. Heuristic-based IDS

III. Based on Deployment Location

1. Host-based IDS (HIDS)
2. Network-based IDS (NIDS)
3. Hybrid IDS (H-IDS)

IV. Based on Activities Monitored (Detection Model)

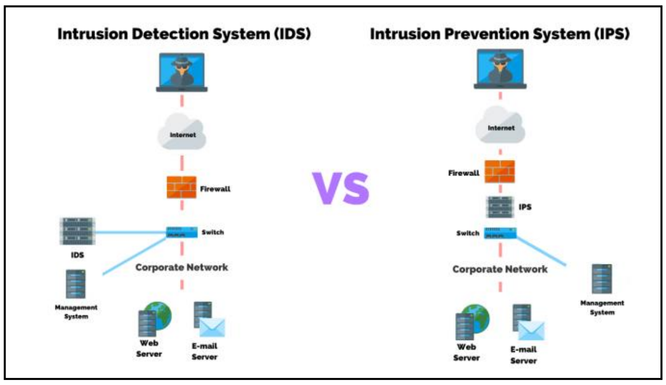
1. Misuse Detection System
2. Anomaly Detection System
3. Policy-based IDS

V. Based on Timing

1. Real-Time IDS
2. Batch Processing IDS

Intrusion Detection Systems for the Internet of Things (IoT) face unique challenges due to the characteristics of IoT environments, such as resource constraints, diverse device types, and the large-scale nature of IoT deployments.

It is important to make the distinction between Intrusion Detection System (IDS) and Intrusion Prevention System (IPS). Thus, the main difference between an IDS and IPS is that an IDS is used only to monitor the network, which then sends alerts when suspicious events on a system or a network are detected. An IPS, on the other hand, reacts to attacks in progress with the goal of preventing them from reaching targeted systems and networks.



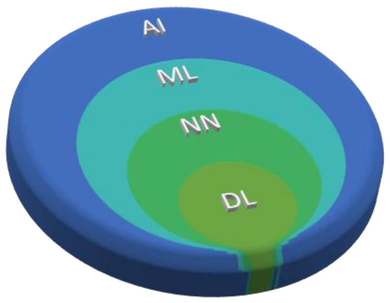
(Fig. 2.1) IDS vs IPS

While both IDS and IPS have the ability to detect attacks, the main difference is in their responses to an attack. However, its important to note that both IDS and IPS can implement the same monitoring and detection methods. So the two systems could work together or even complement each other, because in order for an IPS to block any traffic, it has to first detect and inspect the traffic, which in essence describes the main functionality of an IDS.

III. Related Work

There seem to be a couple of research papers published on the subject of AI Intrusion Detection Systems in the context of IoT or even in general. But when we try to search for real world implementations on the market right now, we seem to be in very short supply. Please see research paper [3] which was only recently published.

The aim of this research paper would be to investigate how it will be possible to architect and deploy such a system in the real world. Possibly even releasing the codebase with a public license, for the public to be able to install and make good use of such systems on their network or devices.



(Fig. 3.1) AI Algorithms family

We will first start to implement an IDS with various Machine Learning classification algorithms, and then go deeper into Deep Learning and Neural Networks. An analysis will be performed on obtained simulation results such that a decision could be made on which implementation will be best suited for each particular situation, based of course mainly on the accuracy obtained.

IV. Implementation Challenges

Starting from the challenges examined by research paper [3], we will try to consider how we can implement such a system that could be used in the real-world.

Designing a real-world Intrusion Detection System (IDS) for IoT devices involves addressing specific challenges associated with the IoT environment. Let us consider the following key design considerations for architecting an effective IDS for IoT devices:

1. Device Heterogeneity

Consideration: IoT devices vary widely in terms of hardware, operating systems, and communication protocols.

Solution: Design a flexible IDS that can accommodate diverse devices. Use protocol agnostic detection methods and consider lightweight agents for resource-constraint devices.

2. Network Diversity

Consideration: IoT networks can be heterogeneous, including wired and wireless connections.

Solution: Implement IDS components that understand and analyze different network protocols. Account for the challenges of wireless communication, such as packet loss and signal interference.

3. **Resource Constraints**

Consideration: Many IoT devices have limited processing power, memory, and energy resources.

Solution: Develop lightweight detection algorithms optimized for resource-constrained devices. Use edge-computing to distribute detection tasks and reduce the burden on individual devices.

4. **Encryption and Privacy**

Consideration: IoT devices often handle sensitive data, and traffic may be encrypted.

Solution: Implement techniques for analyzing encrypted traffic without compromising privacy. Use methods such as traffic correlation and metadata analysis.

5. **Scalability Concerns**

Consideration: The number of IoT devices will be massive even in the near future, leading to scalability issues while deploying such systems in the field.

Solution: Design a scalable IDS architecture with the ability to handle a large number of devices. Consider distributed and cloud-based solutions for scalability.

6. **Real-time Detection**

Consideration: Some IoT applications require real-time response to security threats.

Solution: Implement real-time detection mechanisms, leveraging edge computing and fast processing algorithms (this could conflict with resource constraints, so we have to find a good trade-off). Prioritize critical alerts to ensure timely response.

7. **Adaptability to Network Changes**

Consideration: IoT networks may experience dynamic changes due to device mobility or network reconfigurations.

Solution: Design the IDS to dynamically adapt to network changes. Use Machine Learning (ML) models that continuously learn and update based on evolving network conditions.

8. **Behavior Profiling**

Consideration: Understanding normal behavior is crucial for anomaly detection.

Solution: Develop stronger behavior profiling mechanisms to establish baselines for normal device behavior. Regularly update these profiles to adapt to changes in device behavior over time.

9. **Compliance and Regulations**

Consideration: Different industries and regions may have specific regulations and compliance requirements.

Solution: Ensure that the IDS complies with relevant regulations. Allow for customization to meet industry-specific standards.

10. **Incident Response and Reporting**

Consideration: A clear incident response plan is essential for handling detected threats.

Solution: Integrate the IDS with an incident response system. Provide reporting and visualization tools to help security teams understand and respond to detected incidents effectively.

11. **Continuous Monitoring and Updates**

Consideration: Security threats and attack vectors evolve over time.

Solution: Implement continuous monitoring and update mechanisms. Regularly update detection signatures, algorithms, and threat intelligence feeds.

12. **Cross-layer Analysis**

Consideration: Threats may manifest across different layers of the IoT stack.

Solution: Design the IDS to perform cross-layer analysis, considering both network and application-layer threats. Collaborate with application and network security mechanisms.

13. **User Education and Awareness**

Consideration: End-users may inadvertently contribute to security vulnerabilities.

Solution: Include user education and awareness components. Provide clear communication and alerts to users about potential security risks and best practices.

14. **Legal and Ethical Considerations**

Consideration: Privacy laws and ethical considerations must be respected.

Solution: Ensure that the IDS complies with legal requirements. Implement privacy-preserving measures and be transparent about the data collection and analysis process.

V. Simulation Results and Discussion

A. Simulation using prepared Dataset

The proposed laboratory simulation of IDS is by using KDD CUP 1999 dataset and applying the following Machine Learning classification algorithms against it:

- Gaussian Naïve Bayes
- Decision Tree
- Random Forest
- Support Vector
- Logistic Regression
- Gradient Descent

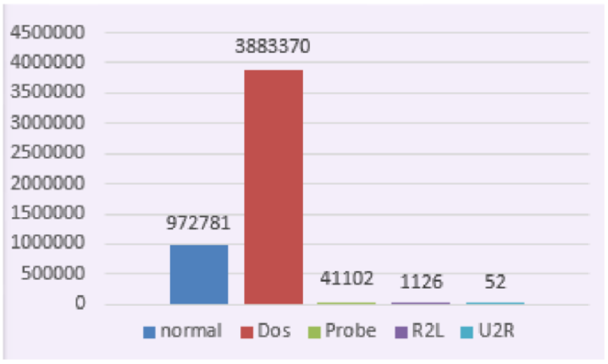
KDD CUP 1999 is an extremely large dataset that is used with Intrusion Detection Systems experiments, the complete dataset having approximately 4.5 million records. This dataset has 41 features that can be classified into three categories [9]:

- TCP Connection Features
- Content Features
- Traffic Features

More details can be seen in the table in Appendix A.

This dataset also contains 22 types of attacks, which can be grouped into 4 major categories [4]:

1. Denial of Service Attack (DoS)
2. User-to-Root Attack (U2R)
3. Remote-to-Local Attack (R2L)
4. Probing: occurs when the attacker tries to gain information about the network to find some vulnerability.



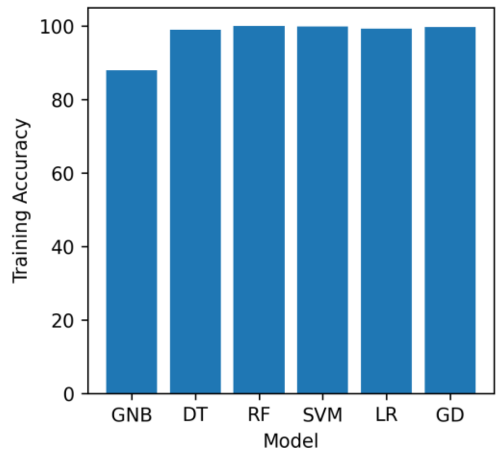
(Fig. 5.1) Attack types classification for KDD CUP 99

Appendix B presents a more detailed table with the distribution of attack types of KDD CUP 99 Dataset.

The 1st step in implementing an IDS with Machine Learning is to train the ML models with KDD CUP 99 Dataset and then train the models to see the accuracy. Below is presented the comparison of the training and testing accuracy of each model.

Classifier	Train score	Test score
Gaussian Naive Bayes	87.951%	87.904%
Decision Tree	99.058%	99.052%
Random Forest	99.997%	99.967%
Support Vector Machines	99.875%	99.879%
Logistic Regression	99.353%	99.353%
Gradient Descent	99.793%	99.772%

(Fig. 5.2) Machine Learning Classifiers scores



(Fig. 5.3) Training Accuracy plot for ML Classifiers

Classifier/Model	Train time [s]	Test time [s]
Gaussian Naive Bayes	0.62	1.10
Decision Tree	1.41	0.17
Random Forest	11.10	1.33
Support Vector Machines	346.27	235.57
Logistic Regression	24.95	0.08
Gradient Descent	416.61	2.38

(Fig. 5.4) Machine Learning Classifiers Train and Test times

From the above metrics we could observe that the best performing Machine Learning algorithm for IDS is Random forest, which takes 11 seconds to train. Support Vector Machines and Gradient Descent are not bad performers, but they take quite a long time to train, 346 seconds for SVM and 416 seconds for GD.

B. IDS Implementation with Neural Networks

A more interesting and advanced approach would be to try to train the IDS with some Neural Networks Models. 3 models were chosen:

1. Shallow Neural Network Model
2. Deep Neural Network Model
3. Convolutional Neural Network Moel

NN Model	Train accuracy [%]	Train loss [%]
Shallow NN	99.89%	0.42%
Deep NN	99.93%	0.31%
Convolutional NN	99.91%	0.37%

(Fig. 5.5) NN Models Accuracy and Loss

NN Model	Train time [s]	Test time [s]
Shallow NN	213.62	6.00
Deep NN	1440.82	13.54
Convolutional NN	321.78	7.50

(Fig. 5.6) NN Models Train and Test time

After these models were trained and tested with KDD CUP 99 dataset, the metrics from the above figures were obtained. It seems that all models have pretty good accuracy, but the training time is significantly higher and much more resource intensive compared to the Machine Learning counterparts. The Shallow NN takes 213 seconds to train, Deep NN Model takes 1440 seconds to train, and Convolutional NN Model takes 321 seconds to train.

The codebase for the Python implementation can be found publicly on Github, see [5] for more details.

The machine specifications on which these simulations were executed are the following: Windows 11 Home, 64-bit, 11th Gen Intel Core i7-1166G7 @ 2.80GHz (8 CPUs) ~2.8GHz, 16GB RAM. If the trained model needs to be deployed in a mobile IoT device, then it is best to do the training on a more powerful computer and then deploy the pre-trained model into the IoT device.

It seems that the best train score (accuracy) on this dataset is obtained by Random Forest Classifier. Regarding testing the classifiers, the dataset is split into 67% train data and 33% test data. See Appendix C for more details about the data sets. Hence we can observe that the testing score could be in some cases slightly less or greater than the train score, so there could be a small variation, but still not far away from the training score.

C. *Simulation using Real Dataset*

In order to test the IDS on a real environment, a network sniffer program has been implemented for Mobile or PC, to hookup on the network interface of the respective device and listen for incoming network packages.

The program is flexible enough to be able to inspect network interface traffic data on various network levels (Ethernet, IP, TCP).

The information extracted from the packet listener program could then be fed to the pre-trained ML/NN algorithm to detect intrusions in real-time. So, one strategy could be to start with a pretrained model using predefined datasets like KDD CUP 99, and then improve the algorithm further by continuously learning from the production environment where it will be deployed.

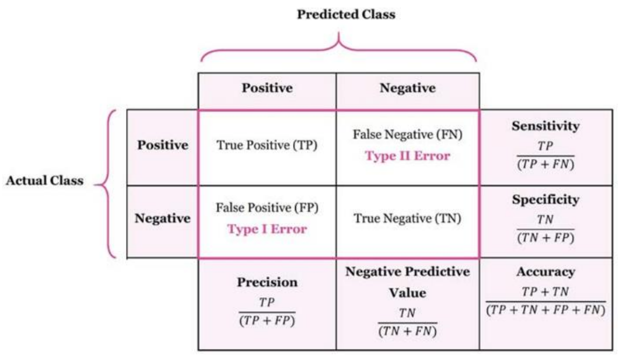
From the Attack Types Distribution in *Appendix C*, we can see that most of the data is related to DoS attacks, so Machine Learning should be strongly trained in that area. Using a network sniffer program, we will try to grab in real-time some of the most important parameters for the DoS data row and fill-in with default data the ones that are hard to reach in real-time.

For the purposes of this simulation, a test server was setup with NPM, which listens for HTTP connections to localhost:80, and a Python program will try to simulate a DoS attack on that API endpoint while the trained model is fetching and analyzing data of incoming packets and let us analyze how the trained Gaussian Naïve Bayes algorithm will perform on a scenario one step closer to the real-world (real-world simulation).

VI. Performance Metrics

Classification metrics and confusion matrices go hand in hand. Confusion matrices are the result of classification problems. There are four possible values that make up the result:

- **True Positive**: classification model created correctly predicted positive
- **False Negative**: classification model predicted false, but are actually positive. These are also called Type II errors.
- **False Positive**: classification model predicted positive, but are actually negative. These are considered Type I errors.
- **True Negative**: classification model correctly predicted negative.



		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$

(Fig. 6.1) Confusion Matrix

The chart above is an example of Confusion Matrix. The four resulting values mentioned earlier are labeled and displayed in the white boxes in the chart. The pink boxes with formulas on the periphery are Classification Metrics. To understand the metrics, one must understand the results first.

Once the results of the classification problem have been received, classification metrics can be calculated. These metrics include, but are not limited to the following: *Sensitivity*, *Accuracy*, *Negative Predicted Value*, and *Precision*.

Sensitivity. Also known as *True Positive Rate (TPR)*, or *Recall*. It represents the outcomes that are correctly predicted as positive. It is calculated as the number of correctly predicted attacks and the total number of attacks.

$$Recall = \frac{True\ Positives}{All\ Positives} = \frac{TP}{TP+FN} = \frac{TP}{P}$$

False

Specificity. Also known as *True Negative Rate*. It represents the outcomes that are correctly predicted as negative.

$$Specificity = \frac{True\ Negatives}{All\ Negatives} = \frac{TN}{TN+FP} = \frac{TN}{N}$$

Accuracy. It is also known as *Classification Rate (CR)*. It represents the outcomes that are correctly labeled as true. The CR measures how accurate the IDS is in detecting normal or anomalous traffic behavior. It is described as the percentage

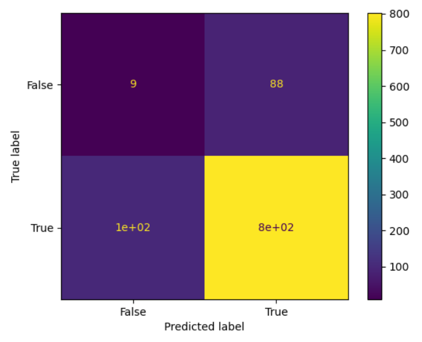
$$Accuracy = \frac{All\ Correct}{All\ Predictions} = \frac{TP+TN}{TP+FP+TN+FN}$$

Precision. It represents the outcomes that are correctly predicted positive.

$$Accuracy = \frac{True\ Positives}{Predicted\ Positives} = \frac{TP}{TP+FP}$$

Sensitivity and Specificity mathematically describe the accuracy of a test which reports the presence or absence of a condition, in comparison with a Gold Standard or definition. In a diagnostic test, sensitivity is a measure of how well a test can identify true positives and specificity is a measure of how well a test can identify true negatives [6].

Evaluating the performance of an IDS involves considering various metrics that assess its effectiveness, efficiency and the ability to mitigate security threats. For example, in the figure below is represented the Confusion Matrix for the Machine Learning with Support Vector Classifier training of the algorithm to recognize the hand written digits [7].



(Fig. 6.2) Confusion Matrix generated from scikit-learn hand written digits ML algorithm example

Appendix E presents a more thorough explanation of Confusion Matrices.

VII. Deoployemt in IoT Environment

This section will explore how a trained IDS model could be deployed in the IoT environment. As we have seen in the metrics presented in section V, IDS AI models could take a significant amount of computing resources, even on a PC/Laptop device. Training especially the Neural Networks models in an IoT environment will be close to impossible. Maybe on a mobile device it will work somehow, but it will potentially drain the battery while training, but in another IoT device, like a kitchen appliance, that will not be possible.

So, we are left with just a handful of choices here. On mobile devices there is a possibility to deploy a pre-trained IDS with NN and just monitor the traffic, because the model can be trained on a more powerful computer and save the trained model to a file, which can be deployed in a mobile device, or a more capable IoT device. See /src/out directory in [5] for files with extensions .h5, .tflite, or .joblib. But, in the case of IoT devices with really reduced hardware and software capabilities, even that option will not be possible, so an alternative approach is to install the IDS in or somewhere after the router that the IoT device will be connected to.

VIII. Conclussions

Designing an effective Intrusion Detection System for IoT devices requires a holistic approach that considers the unique characteristics of the IoT environment. This includes accommodating device heterogeneity, addressing resource constraints, implementing adaptive detection mechanisms, and ensuring compliance with legal and ethical standards. Regular updates, continuous monitoring, and user education contribute to a comprehensive security strategy for IoT ecosystems.

IX. Acknowledgments

I would like to express my deepest appreciation to Conf. Dr. Ing. Corina BOTOCA and Lect. Dr. Ing. Cristina STOLOJESCU, for all their help, mentoring and support, without which the completion of this research paper would not have been realizable.

References

[1] https://financesonline.com/iot-device-statistics/

[2] Intrusion Detection: Cisco IDS Overview, Dec. 28 2001, https://www.ciscopress.com/articles/article.asp?p=24696

[3] Sowmya T. Mary Anita E. A., "A comprehensive review of AI-based intrusion detection system", Measurement Sensors, Volume 28, August 2023.

[4] The UCI KDD Archive, Information and Computer Science, University of California, Irvine, https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[5] T. Emanuel, "Intrusion Detection Implementation", Github page, https://github.com/trincema/master-communication-networks/tree/main/dissertation_thesis

[6] True Positive Success Rate, https://www.gabormelli.com/RKB/True_Positive_Success_Rate

[7] Python Scikit-learn, Recognizing hand-written digits, https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html

Identify applicable funding agency here. If none, delete this text box.

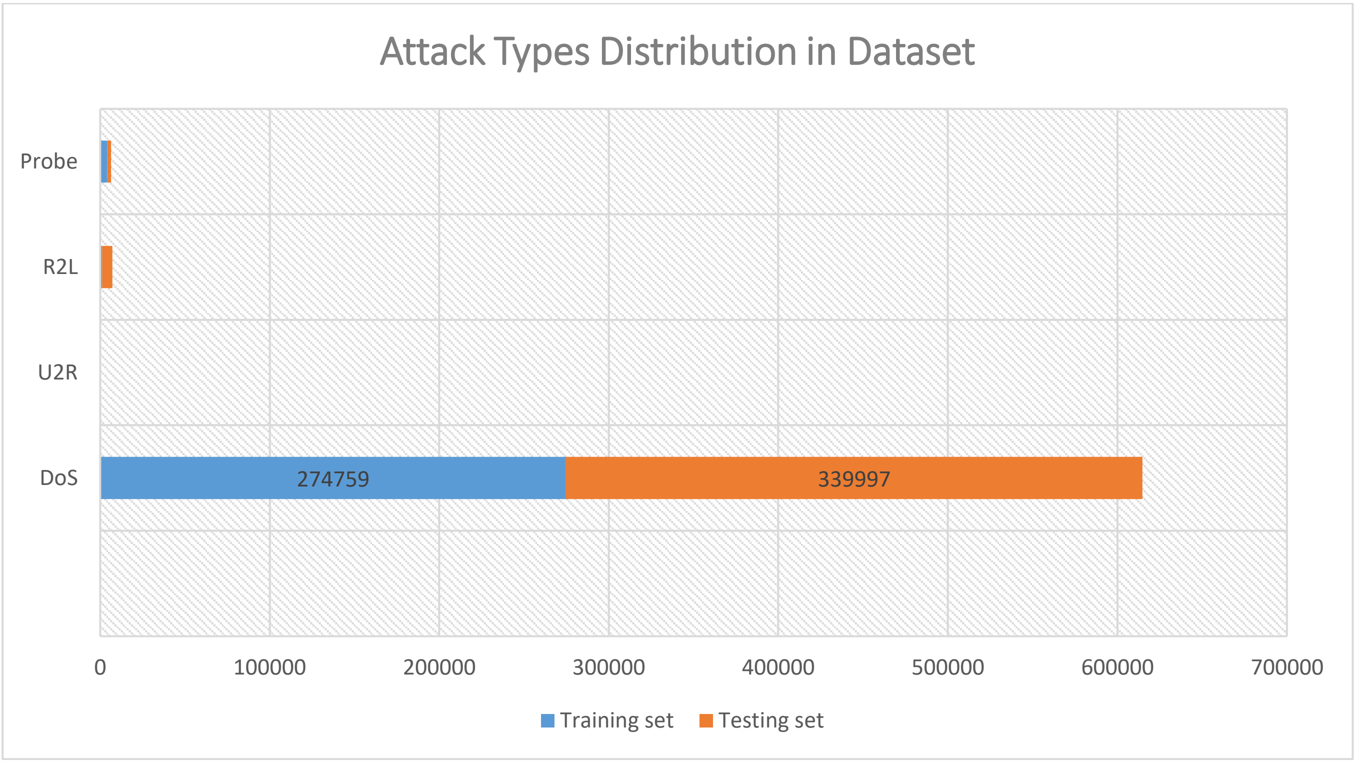
Appendix A: *Features of KDD 199 Dataset*

ID	Feature name	Description	Type
TCP Connection Features			
1	**duration**	Connection length (seconds)	continuous
2	**protocol_type**	Protocol Type (TCP, UDP, etc)	discrete
3	**service**	Network service on the destination (http, telnet, etc)	discrete
4	**src_bytes**	Number of data bytes from source to destination	continuous
5	**dst_bytes**	Number of data bytes from destination to source	continuous
6	**flag**	Connection status (normal/error)	discrete
7	**land**	1 if connection is from/to the same host/port; 0 otherwise	discrete
8	**wrong_fragment**	Number of 'wrong' fragments	continuous
9	**urgent**	Number of urgent packets	continuous
Content Features			
10	**hot**	Number of hot indicators	continuous
11	**num_failed_logins**	Number of failed login attempts	continuous
12	**logged_in**	1 if successfully logged in; 0 otherwise	discrete
13	**num_compromised**	Number of 'compromised' conditions	continuous
14	**root_shell**	1 if root shell is obtained; 0 otherwise	discrete
15	**su_attempted**	1 if 'su root' command attempted; 0 otherwise	discrete
16	**num_root**	Number of 'root' accesses	continuous
17	**num_file_creations**	Number of file creation operations	continuous
18	**num_shells**	Number of shell prompts	continuous
19	**num_access_files**	Number of operations on access control files	continuous
20	**num_outbound_cmds**	Number of outbound commands in FTP session	continuous
21	**is_hot_login**	1 if the login belongs to 'hot list'; 0 otherwise	discrete
22	**is_guest_login**	1 if login is a 'guest' login; 0 otherwise	discrete
Traffic Features			
23	**count**	Number of connections to the same host as current connection in the past 2 seconds	continuous
24	**dst_host_count**	Number of connections having the same destination host	continuous
25	**serror_rate**	% of connections that have 'SYN' errors	continuous
26	**rerror_rate**	% of connections that have 'REJ' errors	continuous
27	**same_srv_rate**	% of connections to the same service	continuous
28	**diff_srv_rate**	% of connections to different services	continuous
29	**srv_count**	Number of connections to the same service as the current connection in the past 2 seconds	continuous
30	**srv_serror_rate**	% of connections that have 'SYN' errors	continuous
31	**srv_rerror_rate**	% of connections that have 'REJ' errors	continuous
32	**srv_diff_host_rate**	% of connections to different hosts	continuous
33	**dst_host_srv_count**	Number of connections that have the same destination host using the same service.	continuous

34	**dst_host_same_srv_rate**	% of connections that have the same destination port and using the same service	continuous
35	**dst_host_diff_srv_rate**	% of different services and current host	continuous
36	**dst_host_same_src_port_rate**	% of connections to the current host having the same source port	continuous
37	**dst_host_srv_diff_host_rate**	% connections to the same service coming from different hosts	continuous
38	**dst_host_serror_rate**	% connections to the current host that have an S0 error	continuous
39	**dst_host_srv_serror_rate**	% connections to the current host and specified service that have an S0 error	continuous
40	**dst_host_rerror_rate**	% connections to the current host that have an RST error	continuous
41	**dst_host_srv_rerror_rate**	% connections to the current host and specified service that have an RST error	continuous

Appendix B: *The distribution of Attack Types of KDD CUP 1999 Dataset*

ID		Attack Type	KDD CUP 1999	
			Training set kddcup.data10percent	**Testing set** corrected.gz
1	**Denial of Service (DoS)**	Smurf	164091	280790
2		Neptune	107201	58001
3		Back	2203	1098
4		Teardrop	979	12
5		Pod	264	87
6		Land	21	9
7	**User to Root (U2R)**	Buffer-overflow	30	22
8		Rootkit	10	13
9		Load-module	9	2
10		Pearl	3	2
11		Spy	2	
12	**Remote to Local (R2L)**	Warezclient	1020	-
13		Guess-password	53	4367
14		Warezmaster	20	1602
15		Imap	12	1
16		Ftp-write	8	3
17		Multihop	7	18
18		Phf	4	2
19	**Probe**	Satan	1589	1633
20		IP-sweep	1247	306
21		Port-sweep	1040	354
22		Nmap	231	84



Appendix C: *KDD CUP 1999 dataset detailed data representation in table format*

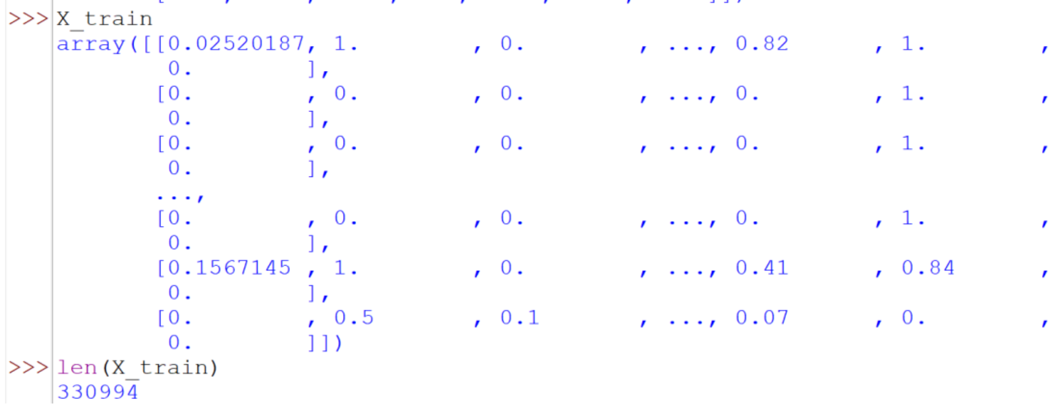
Here is represented in more detail the data used to train the Machine Learning models. We can see that there are only 32 features in the table (some of them were ignored for the analysis), in the form of columns, and the 33rd column represents the Attack type mapped to each set of data from the 32 columns.

1st 11 columns from the dataset												
ID	duration	protocol_type	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	…
0	0	1	0	181	5450	0	0	0	0	0	1	…
1	0	1	0	239	486	0	0	0	0	0	1	…
2	0	1	0	235	1337	0	0	0	0	0	1	…
3	0	1	0	219	1337	0	0	0	0	0	1	…
4	0	1	0	217	2032	0	0	0	0	0	1	…
…	…	…	…	…	…	…	…	…	…	…	…	…
494016	0	1	0	310	1881	0	0	0	0	0	1	…
494017	0	1	0	282	2286	0	0	0	0	0	1	…
494018	0	1	0	203	1200	0	0	0	0	0	1	…
494019	0	1	0	291	1200	0	0	0	0	0	1	…
494020	0	1	0	219	1234	0	0	0	0	0	1	…

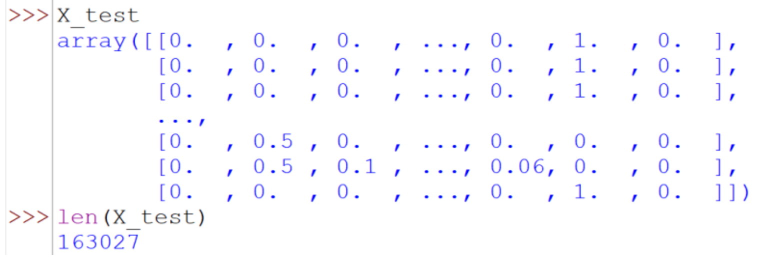
2nd set of 11 columns from the dataset												
ID	num_compromised	root_shell	su_attempted	num_file_creations	num_shells	num_access_files	num_outbound_cmds	is_host_login	is_guest_login	count	srv_count	…
0	0	0	0	0	0	0	0	0	0	8	8	…
1	0	0	0	0	0	0	0	0	0	8	8	…
2	0	0	0	0	0	0	0	0	0	8	8	…
3	0	0	0	0	0	0	0	0	0	6	6	…
4	0	0	0	0	0	0	0	0	0	6	6	…
…	…	…	…	…	…	…	…	…	…	…	…	…
494016	0	0	0	0	0	0	0	0	0	4	5	…
494017	0	0	0	0	0	0	0	0	0	6	6	…
494018	0	0	0	0	0	0	0	0	0	6	18	…
494019	0	0	0	0	0	0	0	0	0	6	12	…
494020	0	0	0	0	0	0	0	0	0	6	35	…

3rd set of 11 columns from the dataset											
ID	serror_rate	rerror_rate	same_srv_rate	diff_srv_rate	srv_diff_host_rate	dst_host_count	dst_host_srv_count	dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	Attack Type
0	0.00	0.0	1.0	0.0	0.00	9	9	0.0	0.11	0.00	normal
1	0.00	0.0	1.0	0.0	0.00	19	19	0.0	0.05	0.00	normal
2	0.00	0.0	1.0	0.0	0.00	29	29	0.0	0.03	0.00	normal
3	0.00	0.0	1.0	0.0	0.00	39	39	0.0	0.03	0.00	normal
4	0.00	0.0	1.0	0.0	0.00	49	49	0.0	0.02	0.00	normal
…	…	…	…	…	…	…	…	…	…	…	…
494016	0.00	0.0	1.0	0.0	0.40	86	255	0.0	0.01	0.05	normal
494017	0.00	0.0	1.0	0.0	0.00	6	255	0.0	0.17	0.05	normal
494018	0.17	0.0	1.0	0.0	0.17	16	255	0.0	0.06	0.05	normal
494019	0.00	0.0	1.0	0.0	0.17	26	255	0.0	0.04	0.05	normal
494020	0.00	0.0	1.0	0.0	0.14	6	255	0.0	0.17	0.05	normal

As we can see, we have 494,020 rows of test data in this data set, which is split into 77% training data, that is to drive the learning process of the ML classifiers, and the remaining 33% of the dataset is left for the testing phase. Below you have 2 pictures from the Python console to see how Python has split the data into the train and test set. It allocated 330,994 rows for the training part, and 163,027 data rows for the testing part.

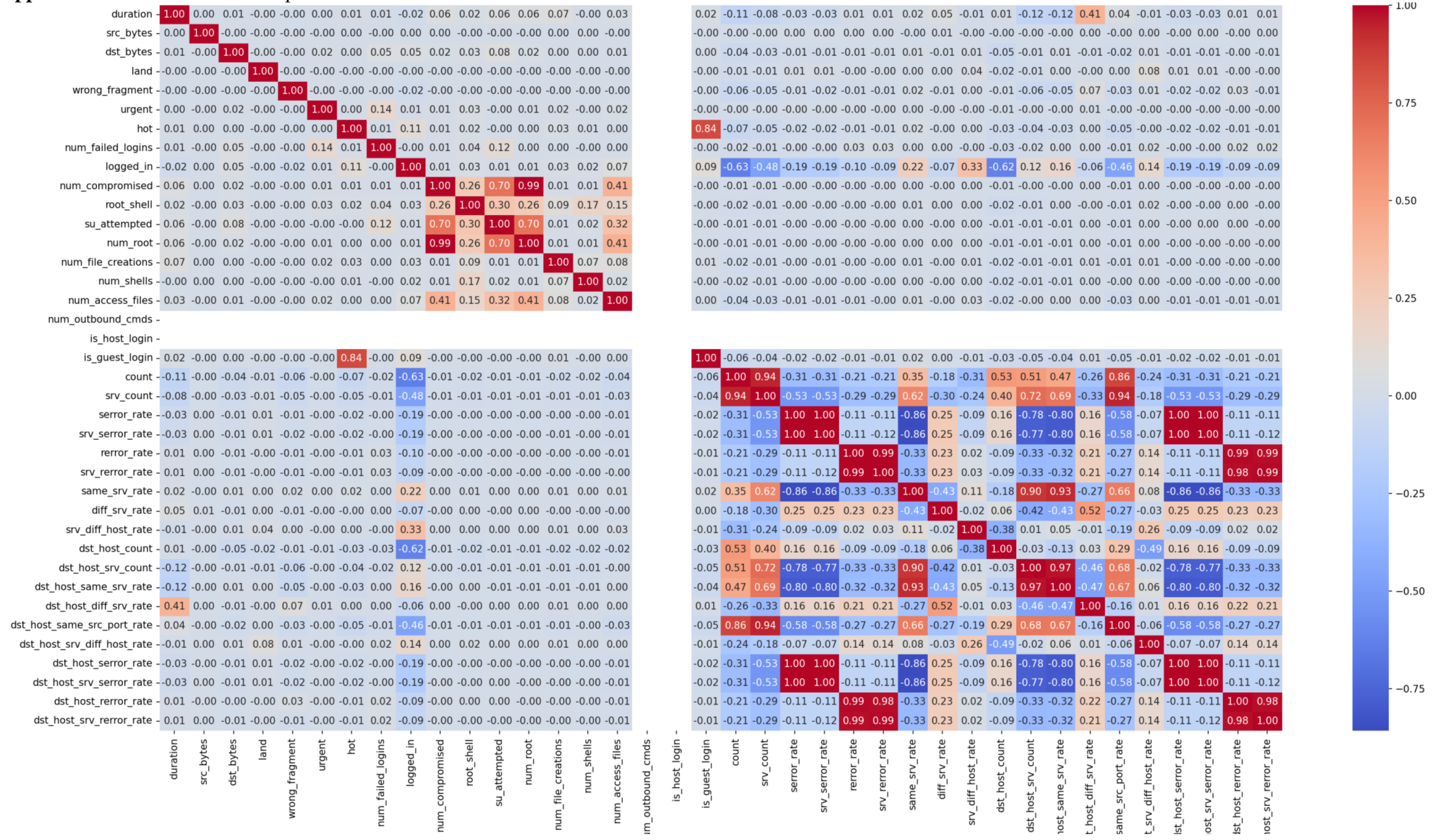


```
>>> X_train
array([[0.02520187, 1.        , 0.        , ..., 0.82      , 1.        ,
         0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 1.        ,
         0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 1.        ,
         0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 1.        ,
         0.        ],
       [0.1567145 , 1.        , 0.        , ..., 0.41      , 0.84      ,
         0.        ],
       [0.        , 0.5       , 0.1       , ..., 0.07      , 0.        ,
         0.        ]])
>>> len(X_train)
330994
```



```
>>> X_test
array([[0.  , 0.  , 0.  , ..., 0.  , 1.  , 0.  ],
       [0.  , 0.  , 0.  , ..., 0.  , 1.  , 0.  ],
       [0.  , 0.  , 0.  , ..., 0.  , 1.  , 0.  ],
       ...,
       [0.  , 0.5 , 0.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 0.5 , 0.1 , ..., 0.06, 0.  , 0.  ],
       [0.  , 0.  , 0.  , ..., 0.  , 1.  , 0.  ]])
>>> len(X_test)
163027
```

Appendix D: *Test Data Heatmap*

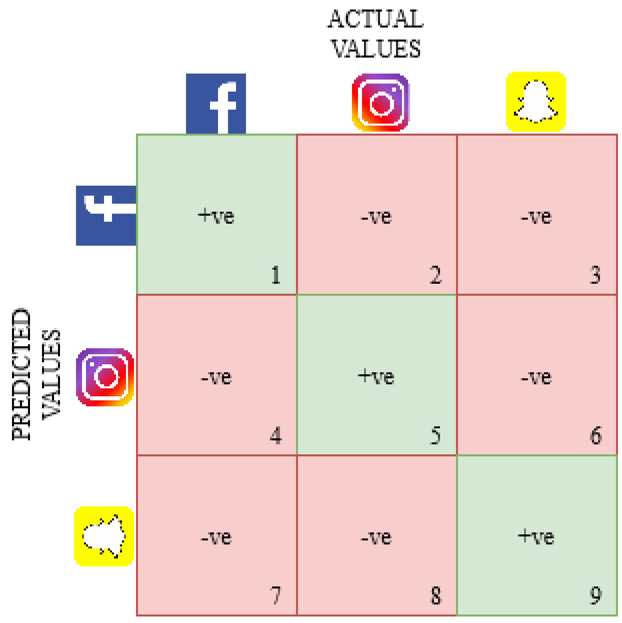


Appendix E: *Classifiers Confusion Matrices*

A confusion matrix is a performance evaluation tool in Machine Learning, representing the accuracy of a classification model. It displays the number of true positives, true negatives, false positives and false negatives. The matrix aids in analyzing model performance, identifying miss-classifications, and improving predictive accuracy.

A confusion matrix is an NxN matrix used for evaluating the performance of a classification model, where N is the total number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

In the context of multi-class classification, things are a bit different. Lets consider an example where we have to predict that a person loves Facebook, Instagram, or Snapchat. The confusion matrix will be a 3x3 matrix like in the picture below:



The true positive, true negative, false positive, and false negative for each class will be calculated by adding the cell values as follows:

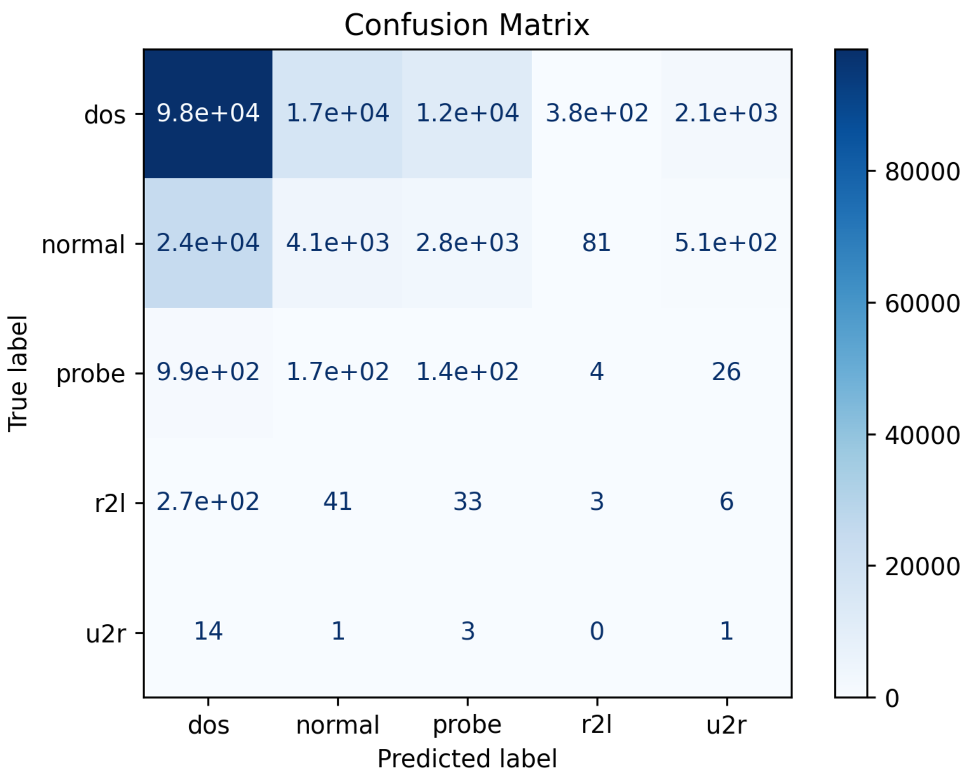
Facebook	**Instagram**	**Snapchat**
$\boldsymbol{TP} = Cell_1$	$\boldsymbol{TP} = Cell_5$	$\boldsymbol{TP} = Cell_9$
$\boldsymbol{FP} = Cell_2 + Cell_3$	$\boldsymbol{FP} = Cell_4 + Cell_6$	$\boldsymbol{FP} = Cell_7 + Cell_8$
$\boldsymbol{TN} = Cell_5 + Cell_6 + Cell_8 + Cell_9$	$\boldsymbol{TN} = Cell_1 + Cell_3 + Cell_7 + Cell_9$	$\boldsymbol{TN} = Cell_1 + Cell_2 + Cell_4 + Cell_5$
$\boldsymbol{FN} = Cell_4 + Cell_7$	$\boldsymbol{FN} = Cell_2 + Cell_8$	$\boldsymbol{FN} = Cell_3 + Cell_6$

Now, lets try to calculate the Sensitivity, Accuracy for Gaussian Naïve Bayes classifier.

$$\boldsymbol{Accuracy_{dos}} = \frac{All\ Correct}{All\ Predictions} = \frac{TP + TN}{TP + FP + TN + FN}$$

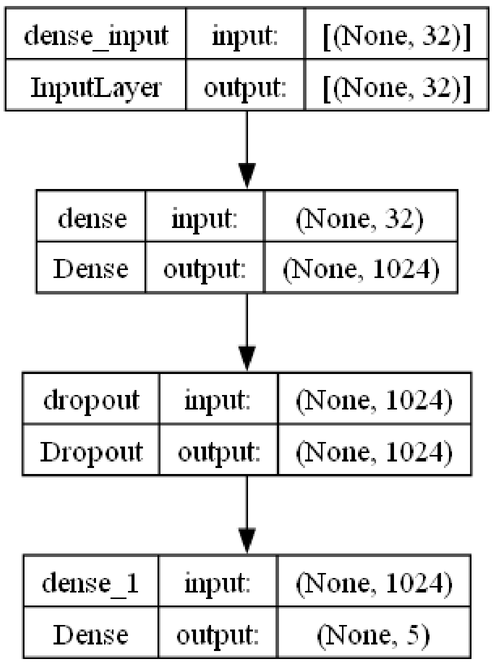
$$= \frac{9.8 \cdot 10^4 + 4.1 \cdot 10^3 + 2.8 \cdot 10^3 + 81 + 5.1 \cdot 10^2 + 1.7 \cdot 10^2 + 1.4 \cdot 10^2 + 4 + 26 + 41 + 33 + 3 + 6 + 1 + 3 + 1}{9.8 \cdot 10^4 + 4.1 \cdot 10^3 + 2.8 \cdot 10^3 + 5.1 \cdot 10^2 + 1.7 \cdot 10^2 + 1.4 \cdot 10^2 + 199 + FP + FN}$$

$$Recall_{dos} = \frac{TP}{TP + FN} = \frac{9.8 \cdot 10^4}{9.8 \cdot 10^4 + 3.8 \cdot 10^2}$$



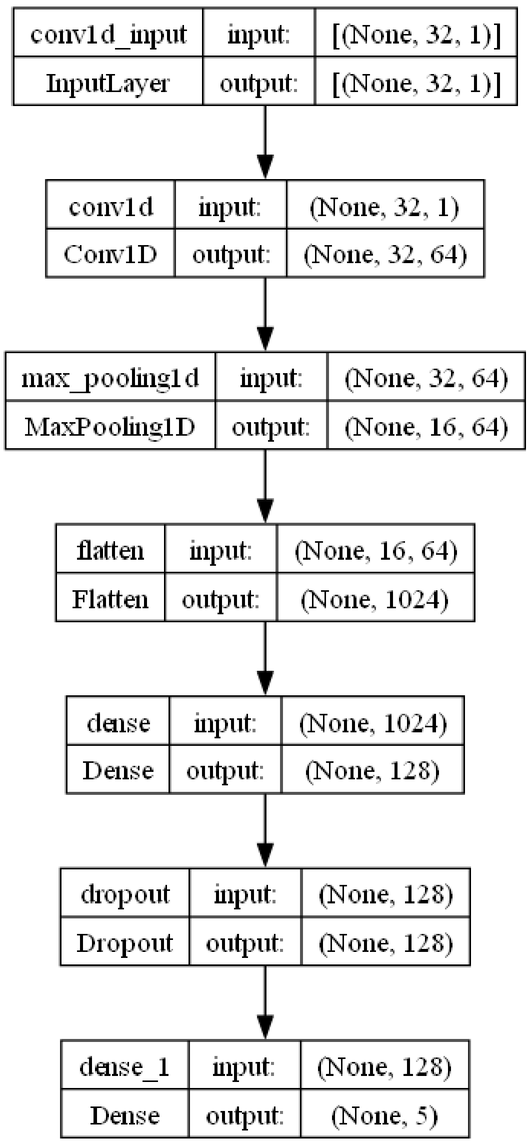
(Fig. E.2) Gaussian Naïve Bayes Classifier confusion Matrix

Appendix F: *Neural Networks Model Diagrams*



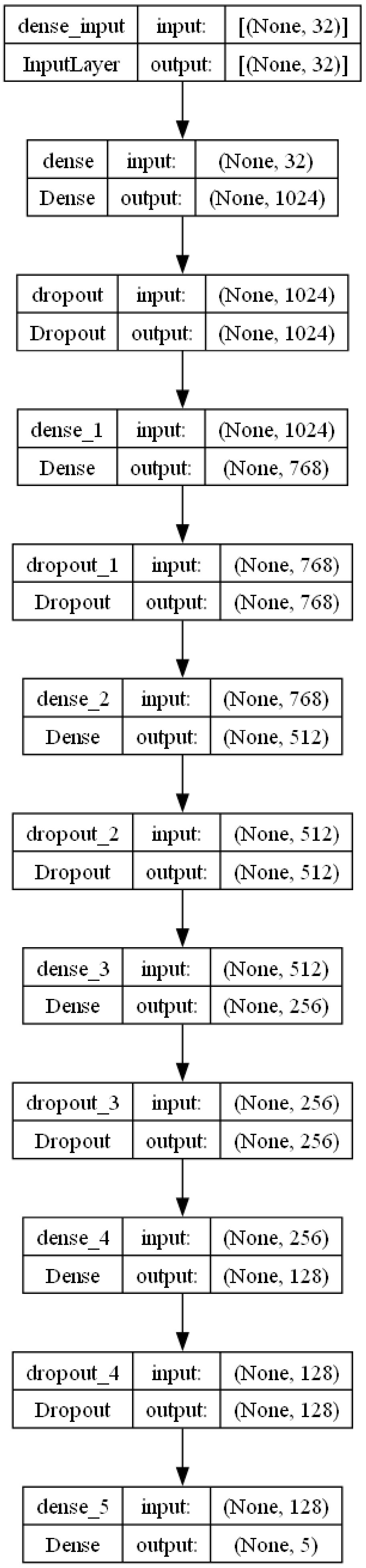
dense_input | input: | [(None, 32)]
InputLayer | output: | [(None, 32)]
dense | input: | (None, 32)
Dense | output: | (None, 1024)
dropout | input: | (None, 1024)
Dropout | output: | (None, 1024)
dense_1 | input: | (None, 1024)
Dense | output: | (None, 5)

(Fig. F.1) Shallow Neural Network Model



conv1d_input | input: | [(None, 32, 1)]
InputLayer | output: | [(None, 32, 1)]
conv1d | input: | (None, 32, 1)
Conv1D | output: | (None, 32, 64)
max_pooling1d | input: | (None, 32, 64)
MaxPooling1D | output: | (None, 16, 64)
flatten | input: | (None, 16, 64)
Flatten | output: | (None, 1024)
dense | input: | (None, 1024)
Dense | output: | (None, 128)
dropout | input: | (None, 128)
Dropout | output: | (None, 128)
dense_1 | input: | (None, 128)
Dense | output: | (None, 5)

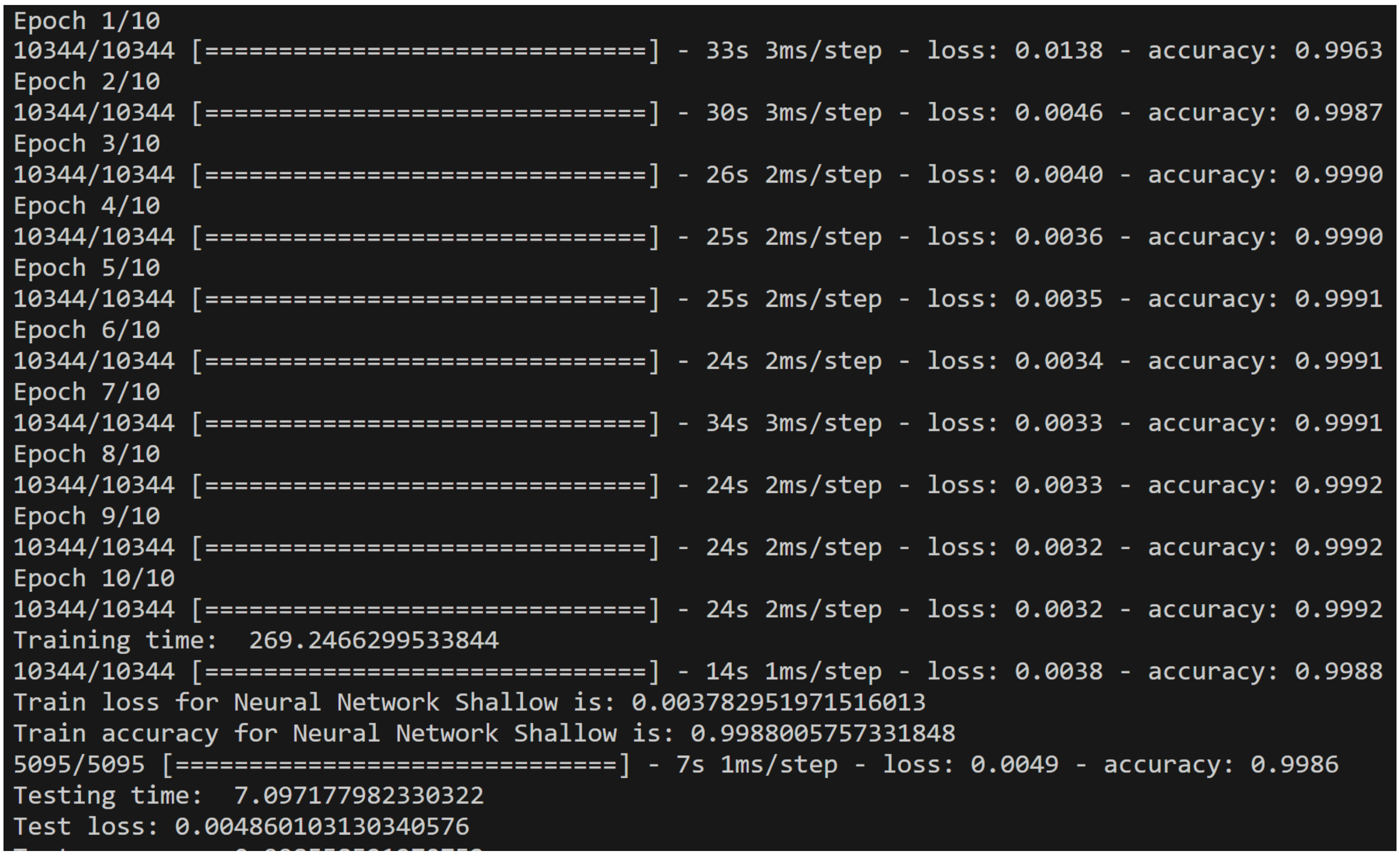
(Fig. F.2) Convolutional Neural Model Diagram



dense_input | input: | [(None, 32)]
InputLayer | output: | [(None, 32)]
dense | input: | (None, 32)
Dense | output: | (None, 1024)
dropout | input: | (None, 1024)
Dropout | output: | (None, 1024)
dense_1 | input: | (None, 1024)
Dense | output: | (None, 768)
dropout_1 | input: | (None, 768)
Dropout | output: | (None, 768)
dense_2 | input: | (None, 768)
Dense | output: | (None, 512)
dropout_2 | input: | (None, 512)
Dropout | output: | (None, 512)
dense_3 | input: | (None, 512)
Dense | output: | (None, 256)
dropout_3 | input: | (None, 256)
Dropout | output: | (None, 256)
dense_4 | input: | (None, 256)
Dense | output: | (None, 128)
dropout_4 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_5 | input: | (None, 128)
Dense | output: | (None, 5)

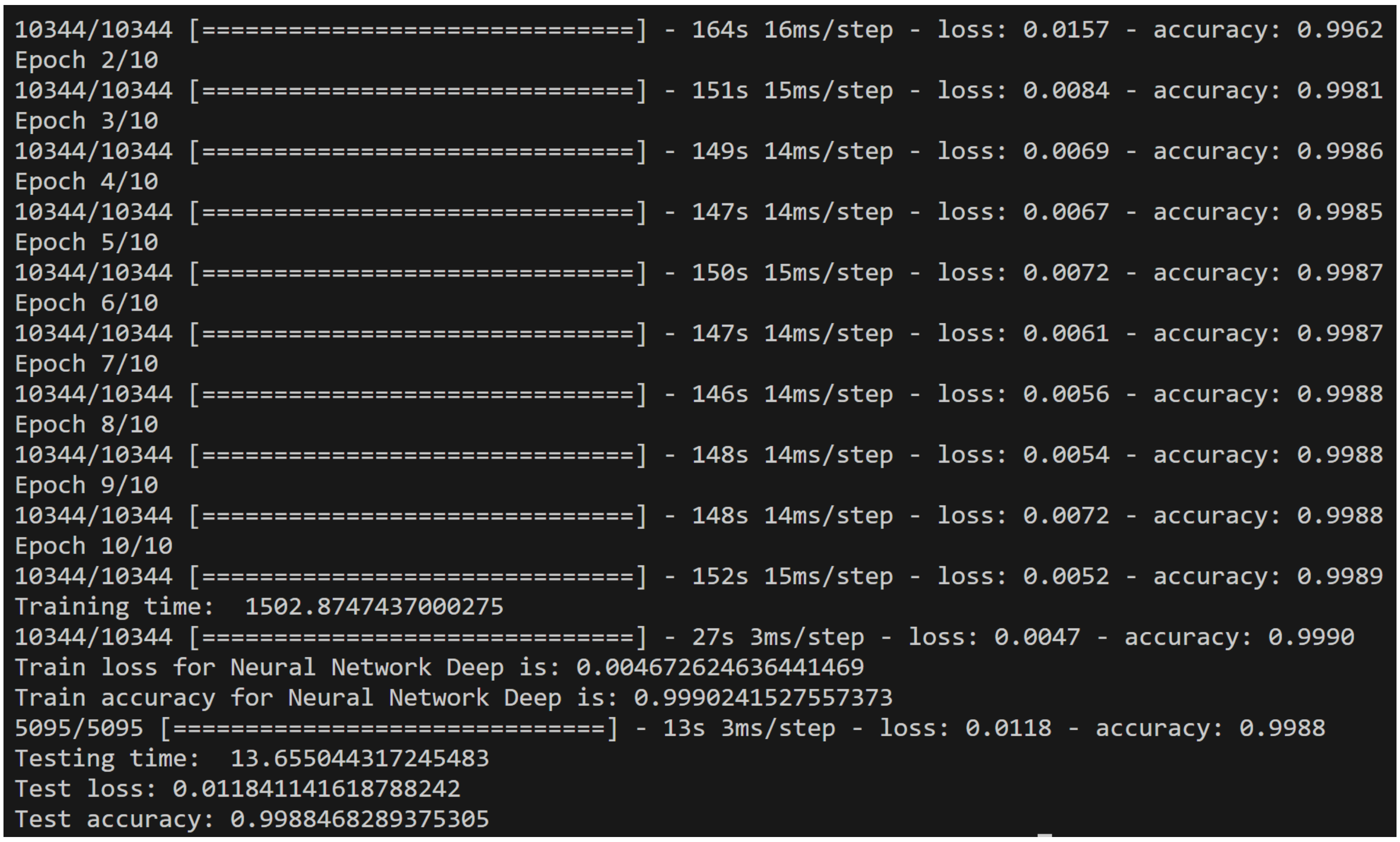
(Fig. F.3) Deep Neural Network Model

Appendix G: *Neural Network Training and Testing*



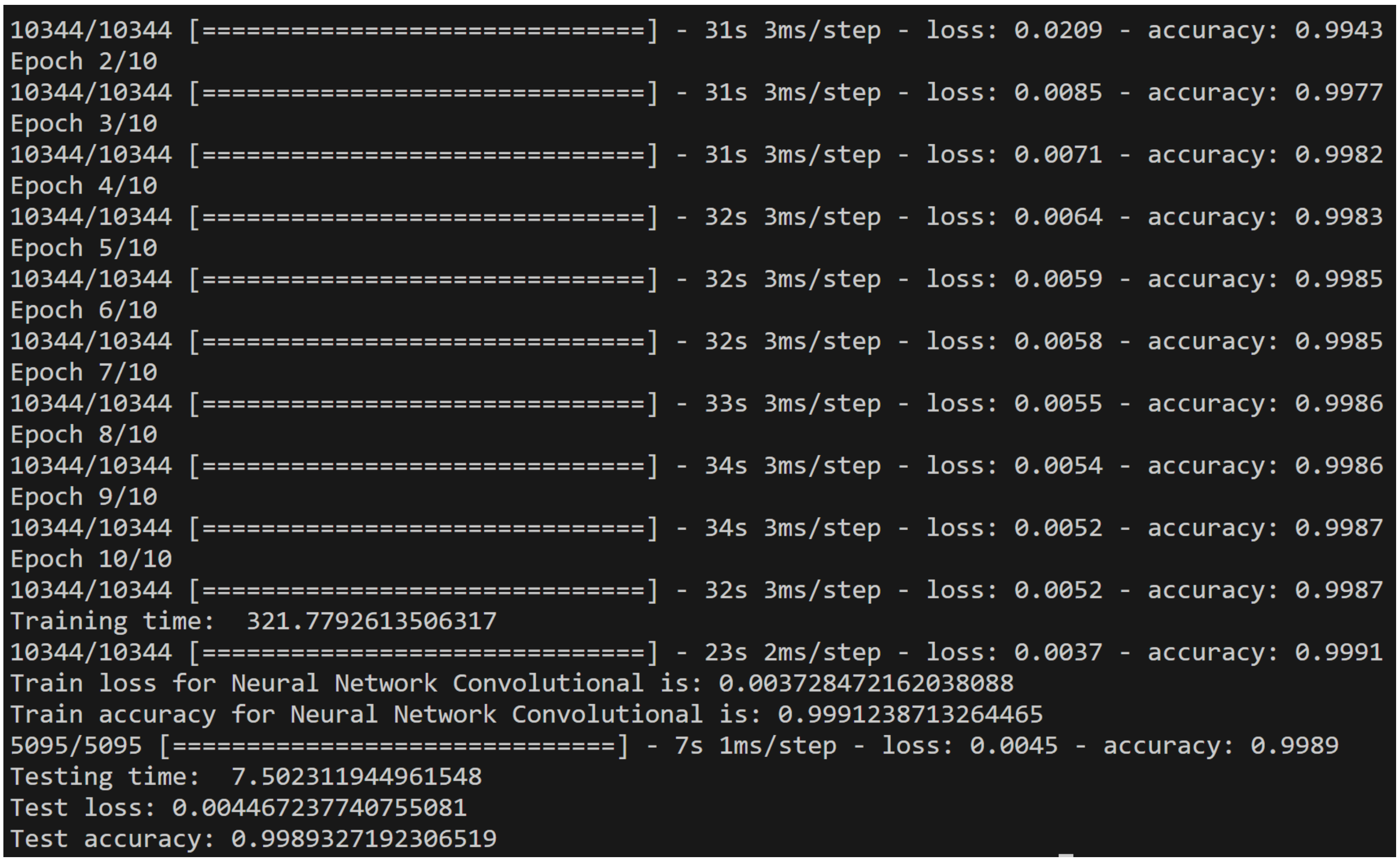
```
Epoch 1/10
10344/10344 [==============================] - 33s 3ms/step - loss: 0.0138 - accuracy: 0.9963
Epoch 2/10
10344/10344 [==============================] - 30s 3ms/step - loss: 0.0046 - accuracy: 0.9987
Epoch 3/10
10344/10344 [==============================] - 26s 2ms/step - loss: 0.0040 - accuracy: 0.9990
Epoch 4/10
10344/10344 [==============================] - 25s 2ms/step - loss: 0.0036 - accuracy: 0.9990
Epoch 5/10
10344/10344 [==============================] - 25s 2ms/step - loss: 0.0035 - accuracy: 0.9991
Epoch 6/10
10344/10344 [==============================] - 24s 2ms/step - loss: 0.0034 - accuracy: 0.9991
Epoch 7/10
10344/10344 [==============================] - 34s 3ms/step - loss: 0.0033 - accuracy: 0.9991
Epoch 8/10
10344/10344 [==============================] - 24s 2ms/step - loss: 0.0033 - accuracy: 0.9992
Epoch 9/10
10344/10344 [==============================] - 24s 2ms/step - loss: 0.0032 - accuracy: 0.9992
Epoch 10/10
10344/10344 [==============================] - 24s 2ms/step - loss: 0.0032 - accuracy: 0.9992
Training time:  269.2466299533844
10344/10344 [==============================] - 14s 1ms/step - loss: 0.0038 - accuracy: 0.9988
Train loss for Neural Network Shallow is: 0.003782951971516013
Train accuracy for Neural Network Shallow is: 0.9988005757331848
5095/5095 [==============================] - 7s 1ms/step - loss: 0.0049 - accuracy: 0.9986
Testing time:  7.097177982330322
Test loss: 0.004860103130340576
```

(Fig. G.1) Shallow Neural Network Training and Testing



```
10344/10344 [==============================] - 164s 16ms/step - loss: 0.0157 - accuracy: 0.9962
Epoch 2/10
10344/10344 [==============================] - 151s 15ms/step - loss: 0.0084 - accuracy: 0.9981
Epoch 3/10
10344/10344 [==============================] - 149s 14ms/step - loss: 0.0069 - accuracy: 0.9986
Epoch 4/10
10344/10344 [==============================] - 147s 14ms/step - loss: 0.0067 - accuracy: 0.9985
Epoch 5/10
10344/10344 [==============================] - 150s 15ms/step - loss: 0.0072 - accuracy: 0.9987
Epoch 6/10
10344/10344 [==============================] - 147s 14ms/step - loss: 0.0061 - accuracy: 0.9987
Epoch 7/10
10344/10344 [==============================] - 146s 14ms/step - loss: 0.0056 - accuracy: 0.9988
Epoch 8/10
10344/10344 [==============================] - 148s 14ms/step - loss: 0.0054 - accuracy: 0.9988
Epoch 9/10
10344/10344 [==============================] - 148s 14ms/step - loss: 0.0072 - accuracy: 0.9988
Epoch 10/10
10344/10344 [==============================] - 152s 15ms/step - loss: 0.0052 - accuracy: 0.9989
Training time:  1502.8747437000275
10344/10344 [==============================] - 27s 3ms/step - loss: 0.0047 - accuracy: 0.9990
Train loss for Neural Network Deep is: 0.004672624636441469
Train accuracy for Neural Network Deep is: 0.9990241527557373
5095/5095 [==============================] - 13s 3ms/step - loss: 0.0118 - accuracy: 0.9988
Testing time:  13.655044317245483
Test loss: 0.011841141618788242
Test accuracy: 0.9988468289375305
```

(Fig. G.2) Deep Neural Network Training and Testing



```
10344/10344 [==============================] - 31s 3ms/step - loss: 0.0209 - accuracy: 0.9943
Epoch 2/10
10344/10344 [==============================] - 31s 3ms/step - loss: 0.0085 - accuracy: 0.9977
Epoch 3/10
10344/10344 [==============================] - 31s 3ms/step - loss: 0.0071 - accuracy: 0.9982
Epoch 4/10
10344/10344 [==============================] - 32s 3ms/step - loss: 0.0064 - accuracy: 0.9983
Epoch 5/10
10344/10344 [==============================] - 32s 3ms/step - loss: 0.0059 - accuracy: 0.9985
Epoch 6/10
10344/10344 [==============================] - 32s 3ms/step - loss: 0.0058 - accuracy: 0.9985
Epoch 7/10
10344/10344 [==============================] - 33s 3ms/step - loss: 0.0055 - accuracy: 0.9986
Epoch 8/10
10344/10344 [==============================] - 34s 3ms/step - loss: 0.0054 - accuracy: 0.9986
Epoch 9/10
10344/10344 [==============================] - 34s 3ms/step - loss: 0.0052 - accuracy: 0.9987
Epoch 10/10
10344/10344 [==============================] - 32s 3ms/step - loss: 0.0052 - accuracy: 0.9987
Training time:  321.7792613506317
10344/10344 [==============================] - 23s 2ms/step - loss: 0.0037 - accuracy: 0.9991
Train loss for Neural Network Convolutional is: 0.003728472162038088
Train accuracy for Neural Network Convolutional is: 0.9991238713264465
5095/5095 [==============================] - 7s 1ms/step - loss: 0.0045 - accuracy: 0.9989
Testing time:  7.502311944961548
Test loss: 0.004467237740755081
Test accuracy: 0.9989327192306519
```

(Fig. G.3) Convolutional Neural Network Training and Testing