600 Jahre Universität Rostock

Traditio et Innovatio

Master-Thesis on the subject of

Machine Learning on the DIN Rail - Detection of Anomalies on the Physical Layer in Fieldbus Networks

Degree course:	Electrical Engineering
Handed in by:	Nazmul Alam
Enrolment number:	218205174
Work period:	01. Aug 2022 – 19. Dec 2022
Supervisor:	Dr.-Ing. Thomas Mundt
Primary Reviewer:	Prof. Dr. rer. nat. Clemens H. Cap

Lehrstuhl für Informations- und Kommunikationsdienste
Institut für Informatik
Fakultät für Informatik und Elektrotechnik
Universität Rostock

44.5mm27.5mm

Printed on acid-free and elemental chlorine-free bleached paper.

As of:: December 19, 2022

Abstract

Building Automation System (BAS) defines the mechanism which is used to control and monitor intelligent buildings' operations such as lighting, climate control, and access control with the help of controllers, sensors, and actuators. Many devices are interconnected in a network through a common bus medium, known as fieldbus. There are different network protocols for field buses such as CAN, KNX, Modbus, etc. The increasing adoption of IP-connected IoT devices significantly increased the function of automated buildings. But unfortunately, the interconnection with the local area network (LAN) and with the Internet comes with great security risks that cannot be overlooked. Now attackers may have more opportunities to exploit the vulnerabilities of different protocols and gain information and knowledge about the building's infrastructure. The security of field buses is an important challenge that needs to be addressed. If a scenario of government facilities is considered, hospitals and public spaces are becoming more interconnected and the impacts of cyber attacks could cause large damage and harm to people. In this thesis, a method is proposed of an Intrusion Detection System (IDS) that could analyze the physical layer traffic of fieldbus and detect anomalies using machine learning. To train the machine learning model, the voltage over time or the time series data of different devices and for analysis of the feature space Convolutional Neural Network(CNN) was used.

Due to some limitations like the low data transfer rate of the building automation system (KNX data transfer rate 9600 bit/s) and we also may have to deal with difficult-to-reach locations in which case an arbitrary number of resources via the internet cannot be connected or powerful computers cannot be used locally. Furthermore, the complexity of the model and amount of the selection of the feature examined need more resources and time. Therefore, some available hardware accelerators for machine learning like Google Coral and NVIDIA Jetson nano will be used on DIN rail. This hardware will act as an intrusion detection system. In this thesis, it has been examined if it is feasible to deploy an intrusion detection system with the help of machine learning that can detect anomalies using physical layer data of a fieldbus network.

Contents

Chapter 1

Introduction

In this chapter

Technological evolution and the increasing need for energy efficiency led people to invent building automation systems. The primary application of a building automation system is to control and monitor different mechanisms inside the building namely heating, ventilation, and air conditioning. This mechanism is facilitated by field buses. Early building automation systems were introduced in the 1970s and they were designed to be isolated by nature and autonomous. Since they were isolated and using proprietary technology it was considered to be secured back then [1]. while newer system tackles these issues with help of cryptography. Since the 1990s the use of personal computers and the internet were becoming widely accessible and remote management become a reality. IP communication and Ethernet became prevalent and due to economical reasons and convenience, they were adopted in fieldbuses quickly. Nowadays a lot of modern buildings, and companies use automation for heating, ventilation, air conditioning, and lighting since they offer a good possibility to save energy and provide convenience.

BAS can help to reduce the operational cost which means lower electricity bills and since it can track how the system is performing, it can tell if something needs to be repaired and when which also means lower maintenance costs. Poor indoor air quality is common and underestimated threat to building occupants which we have realized more during the recent global pandemic. As BAS can automate indoor air quality, it improves the indoor air quality and therefore safety of the building occupants. BAS bus can also integrate with the security system to automate threat response. For example, if the motion sensor detects some activity but

no tenants or occupants have swiped their key card, BAS can determine it could be an intruder and send an alert to the building administrator.

Despite having many advantages a common factor among fieldbus protocols is lack of security. Regardless of the recent introduction of security mechanisms, it is still vulnerable to attack and relatively easy to interfere with the communication channel [1]. Most of the standard KNX installation does not offer either authentication or encryption. Someone with good knowledge and resources can connect to the bus and listen to the traffic and moreover inject malicious traffic [2]. Since the cables are often easily accessible due to the nature of the system, someone with an electromagnetic coil on the wall can capture traffic that is being transmitted [3]. Therefore, any personal information and privacy-sensitive information can be intercepted from the transmitted signal [4].

Since it is not easy to replace the existing fieldbus system as it is costly and time-consuming, we must think of some security measures to tackle the problem within their boundaries. There are almost no commonly available security tools that can be used to detect all potential anomalies, an intrusion detection system (IDS) needs to be established that can identify malicious activity on the network and increase security. With the help of machine learning, we can train our model on what is normal behavior, and thus we can detect when it is not normal, which could be a potential intrusion.

Earlier research showed optimistic results in the development of IDS [5]. But most processes operate on the upper layer of the OSI model. But there have been some researches that suggest [6] the electrical characteristics of a communication signal in field buses' physical layer could also be used in identifying devices. If the performance is stable and satisfactory then we could use it for designing a physical layer intrusion detection system. This system will detect if any new device is present in the network or whether the position of the device or cabling has changed or if any device is trying to spoof the identity and then send an alert according to the predefined security measurement. To detect this kind of anomaly IDS needs to be trained on sufficient physical layer characteristics of the maximum available devices in the network and try to learn the characteristics of a known set of devices. If any bus signal is significantly different than what it has learned previously it should raise an alarm.

To detect anomalies in our datasets Convolutional Neural Network (CNN) will be used. The operation of this neural network imitates the operation of the neuron in our brain, hence the name neural network. Generally, CNN works by getting an image as an input, assigning some weight based on different objects or features of the image, and then distinguishing them from each other as an output it predicts the probability of which class or category the input data may belong to. A convolution operation is applied to the input data to filter information and produce a feature map which is followed by a max pooling operation. The convolution and max pooling help to reduce the image dimension keeping all important features intact.

The objective of this thesis is to investigate the possibility of detecting anomalies using machine learning on a device's physical layer data.

1.1 Motivation

Low-cost sensors and wireless technologies are significantly changing the environment where we live and work, facilitating our safety, comfort, and efficiency as well. The Internet of Things (IoT) and cloud computing created an opportunity to control the building automation system remotely with the help of fieldbus. Fieldbus is simply a means of communication between input and output devices in a network of different interconnected devices with manageable wiring and easy extensibility. Prior to its introduction, computers used serial connections where only two devices could communicate with each other. On the other hand, fieldbus allows connecting several analog and digital points to connect at once. This minimizes the number of cables and length of cables as well. The such installation offers flexibility and convenience but also induces new vulnerabilities and a need for security. The systems currently used in building automation have several security issues and can be easily compromised as it lacks even basic cryptographic security [7].

Building automation systems consist of a lot of IoT components such as sensors and actuators which are often unattended and therefore can easily be attacked either physically or remotely. The more sensor or actuators are attached to a network the more complex the network and protocols become which can result in undesired interaction between components and can lead to security vulnerabilities [8]. If an attacker wants to inject malicious code into a network using fieldbus, they will observe and abuse the transmission protocol. Without security measures, a recipient can only assume the sender is legitimate and there is no way to recognize such an attack.

Some simple functionality like lighting control can become significant and attractive to an attacker. As an example, we can consider a scenario of an international conference where a lot of high-ranking government or non-government individuals are present. An attacker could turn off the light and maybe start messing with the heating, ventilation, or air conditioning system which is a major disturbance for such an international conference. In the worst-case scenario, this attacker may assist a real-world attack while the lights are down.
We can also think about a similar scenario for a hospital. While the doctors are performing some kind of time-sensitive and critical surgery any kind of lighting disturbance could cause major disruption and in the worst-case scenario, the patient could die. In a university building disturbance of lighting, heating or air conditioning could cause an interruption of lectures. There are numerous possibilities an attacker can carry out given the right context.

Thus, we would like to secure fieldbuses to tackle such security issues with help of IDS. The goal of this thesis is to set up an IDS that detect anomalies or outlier using physical

characteristics of fieldbuses with the help of machine learning. As fieldbus protocol, KNX was chosen as it is widely operated in Europe and a sufficient sample of telegrams was obtained from the computer science building of the University of Rostock.

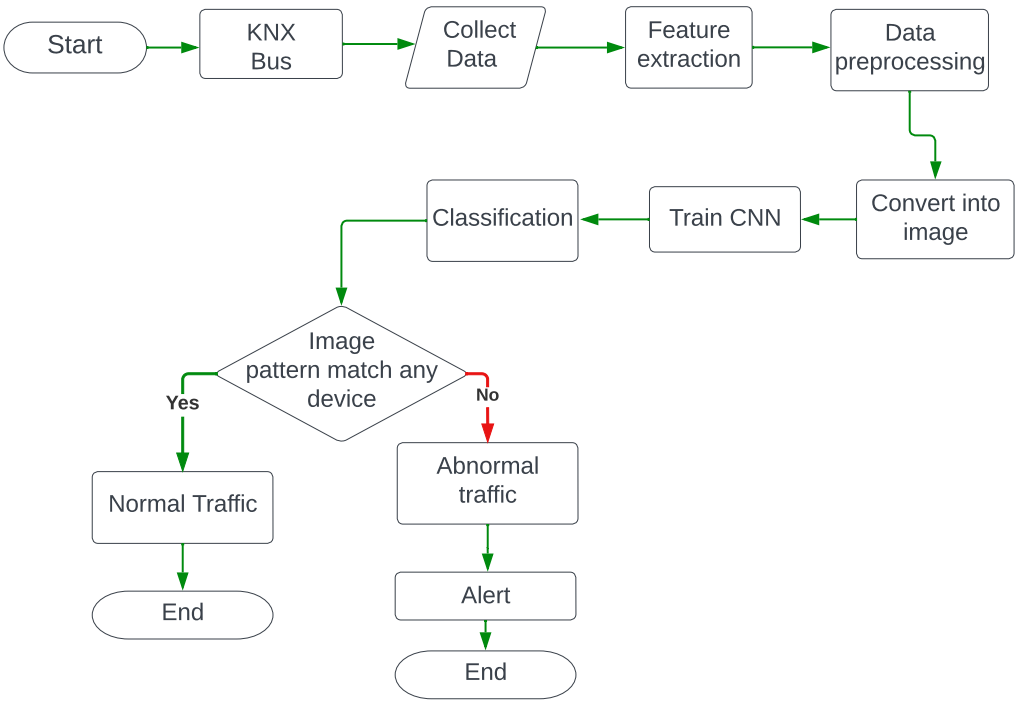


Figure 1.1: A flow chart of the entire working process

The complete workflow of this thesis is shown in the flowchart 1.1. The purpose is to train a CNN model with available data so that it can learn what is normal traffic and hence, can detect abnormal traffic. The data is collected from a KNX bus which is followed by some feature extraction and data preprocessing. As it is mentioned earlier, an accelerator will be used as IDS to detect the anomaly and those accelerators work with only CNN, accordingly, we need to convert the available time series data into an image so that it can be used to train a CNN model. After the model is trained, it will conduct some classification tasks to find out from which device the data is being transferred. If the prediction of incoming data matches any device which it has learned before then it is assumed normal traffic, however, if the traffic does not match with any device traffic pattern, then it could be an anomaly and will send an alert to the administrator.

This thesis has been divided into a few chapters. In the next chapter the fieldbus protocol KNX, a brief introduction to Intrusion Detection Systems and Machine learning has been discussed. The following chapter talks about the methodology, analysis of the data, and how the data was collected, preprocessed, and preprepared to be used for the machine learning algorithm. The next chapter comprises of implementation of different methods and checks if they can satisfy the requirement of this thesis. The evaluation chapter discusses the standard evaluation

process for machine learning, how the implementation result could be used for anomaly detection and whether the implemented methods can detect anomalies. The thesis concludes with a few suggestions to try on for further experiments.

Chapter 2

Background

In this chapter

In this chapter, some technical aspects and terminology will be discussed which is required to have a better understanding of the later parts. In the beginning, some brief insight about field buses will be discussed and as a fieldbus example, KNX has been chosen, and the details of the KNX system will be discussed. It will be followed by with concept of an Intrusion Detection System (IDS). This chapter concludes with an overview of machine learning and the algorithm CNN will be discussed as this specific algorithm has been used for the analysis of the feature space.

2.1 Fieldbus

Fieldbus refers to a family of industrial computer networks that allows communication with input devices such as sensors, ethernet switches, and output devices such as valves and drives without the complication of connecting all single devices back to the controller and facilitates real-time control and monitoring at the same time. Therefore, fieldbus can reduce costs and provide convenience. Fieldbus is not just a connection type but a group of protocols that are used in the industrial arena. The fieldbus has been standardized as IEC61158. In the past

industrial controller system were connected using RS232 serial communication and as it is known serial communication only allows two devices to communicate. In fieldbus, multiple field devices can be connected to a single connection point and that point would then connect to the controller. Fieldbus devices such as sensors, lamps, switches, and motors are connected to an I/O data block, which is connected to a field distribution device, which is again connected to a power supply and at last connected back to a Programmable Logic controller (PLC).

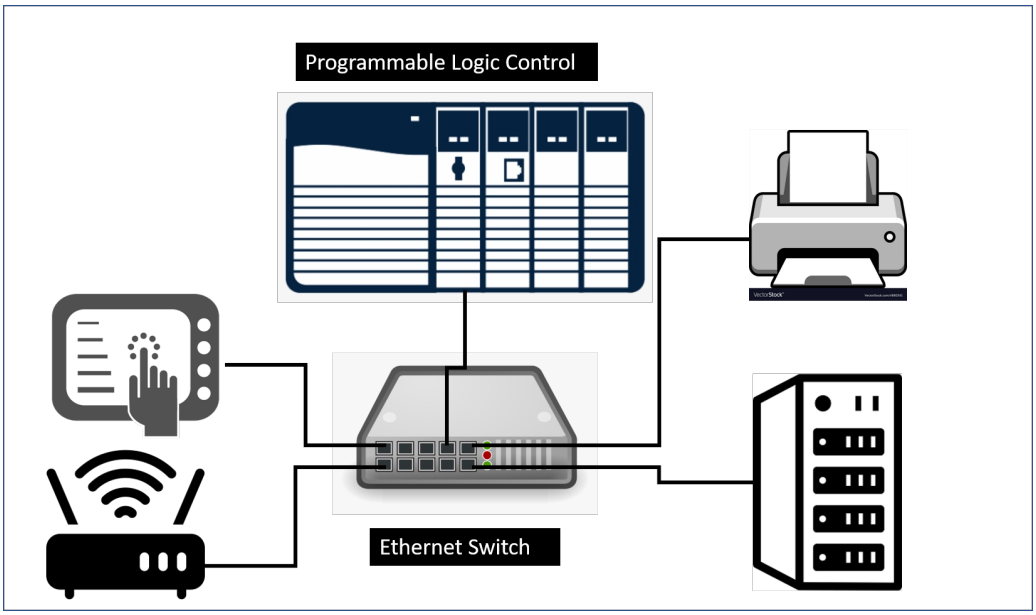


Figure 2.1: Field bus communication.

Depending on the application and industry the protocol of fieldbus can vary significantly. Some buses are better at specific operations than others. Therefore, it is imperative to consider the fieldbus protocol according to the need while designing a control system. It is also important to keep in mind that not all communications modules can communicate properly with every fieldbus protocol. Some most commonly used protocols are KNX, Modbus, CAN, Profibus, Profinet, HART, etc. Some benefits of fieldbus are:

- **Reduced cabling requirements:** It allows numerous devices to connect to a single point and controller. Decreases the number of required cables vastly.
- **Lower costs:** Reduced cabling comes with lower costs. It significantly reduces the cost of setting up and running a network.
- **Ease of installation:** In contrast to parallel wiring fieldbus needs to deal with lot fewer cables. Thus, planning, organizing, and installation require less time and labor.
- **Increased reliability:** Fieldbus systems are more reliable than parallel wiring due to the short signal pathway hence reliability is improved. It also provides improved protection against interference [9].

To represent different technological levels in an industrial automation plant, an automation pyramid is used, which is shown in figure 2.2, it allows different technologies to communicate with each other within each level and between different levels as well. The first level of the pyramid is known as the field level since it consists of sensors and actuators which are used to measure different process parameters like temperatures, pressure, flow, etc, and then it executes the command using electrical or electronic, hydraulic or mechanical devices. They are connected via field bus protocol like KNX or LON. Parallel wiring is typical at this level.

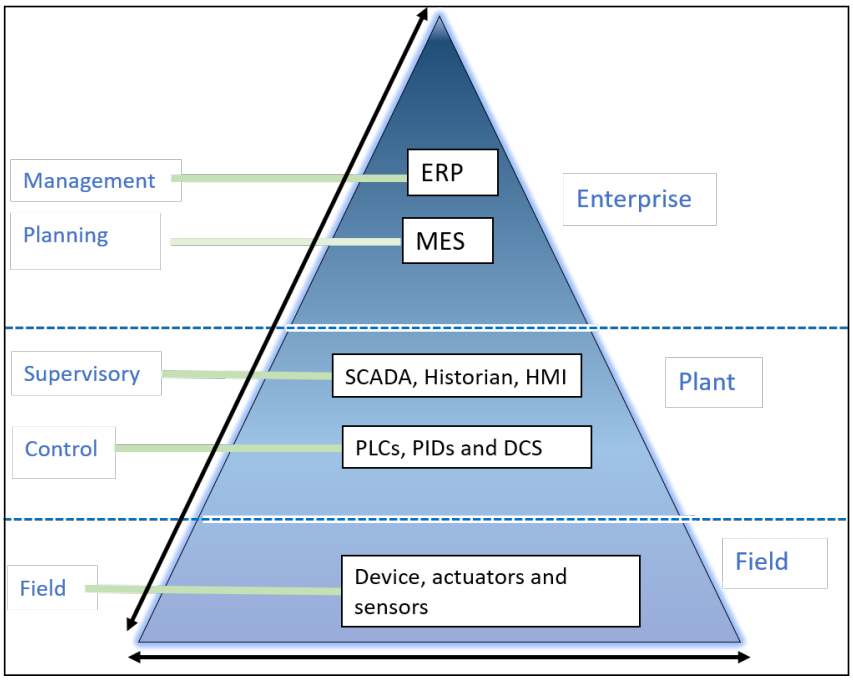


Figure 2.2: The automation pyramid of a typical industrial plant.

The control level is responsible for controlling logical devices like PLC, distributed control system (DCS), or proportional-integral-derivative (PID) controllers. This level performs the actual physical work using control and logical devices to regulate or control the field-level devices. They receive measurements from field-level devices and determine what needs to be done by the field-level actuators to accomplish predefined criteria. For example: If the motion sensor picks up any motion in the hallway, turn on the light.
The third level is the supervisory control and data acquisition (SCADA) level or supervisory level which accesses all data and controls multiple systems. The SCADA collects all information from all subsystems or subprocesses of the automation plant, performs essential analysis, and displays all information in an organized manner. This level also includes a Human-machine interface (HMI) and workstation. It also often uses a database or process historian to access past process data.
The fourth level is the manufacturing execution system (MES). It monitors the complete production systems from raw materials to the completed goods in the industrial automation sys-

tem. This level takes care of production scheduling, production equipment management, performance analysis, and quality control.
The top level is the management level. Enterprise resource planning (ERP) systems are used for material management features and plant scheduling. ERP is an integrated software that helps to monitor everyday business activities to sales, procurement, accounting, and many more [10].
In the subsequent section details about KNX fieldbus, its technical characteristics, and important terminology will be discussed. In this thesis, only KNX fieldbus has been observed since it is commonly used in Europe and all the data has been collected from KNX fieldbus.

2.1.1 KNX

KNX is an open standard protocol for industrial and home building automation systems. It emerged from the past three standards; European Installation Bus (EIB), European Home Systems Protocol (EHS), and BatiBus Club International (BCI) back in May 1999. Since then, it has been operating under the name KNX. They are also responsible for the distribution of Engineering Tool Software (ETS) which is used to plan and configure KNX installation. It can connect and control different field devices such as sensors, actuators, controllers, etc. There are several other bus technologies in the market and different technologies are appropriate for different applications but KNX is supported by many different manufacturers. It almost supports all communication media. All devices in a KNX system use the same bus network for the transmission and exchange of data. It is a decentralized structure and does not need a central control unit. If one device goes down it does not affect the other functions except for the application of the failed device. In the KNX system devices are generally divided into different categories. They are system components (power supply, USB interface, programming interface), sensors, actuators, logical components, and control panels.

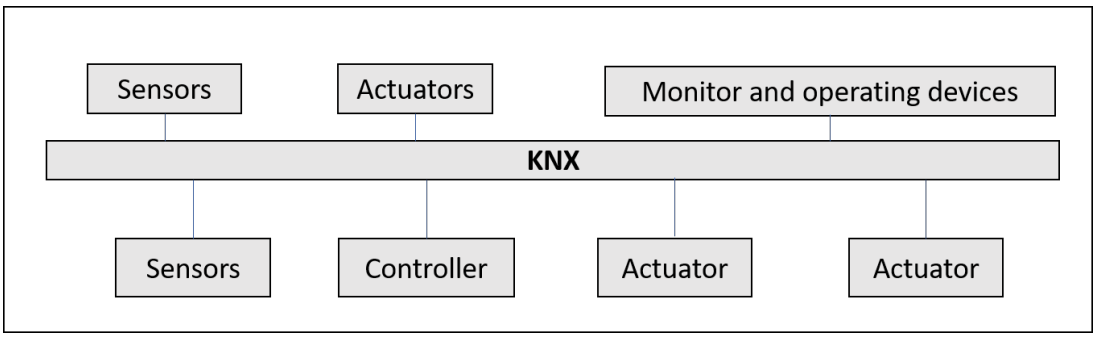


Figure 2.3: Field devices connected over KNX.)

Due to the decentralized structure, KNX can be adjusted according to the need. Theoretically, a KNX system can connect more than 50,000 devices. [11]

2.1.1.1 Communication Media

KNX uses a different kind of communication medium for transmission and data exchange. They are:

- **Twisted pair (KNX TP)** Via twisted pair cable.
- **Powerline (KNX PL)** Uses existing 230V mains
- **Radio Frequency (KNX RF)** Over radio signal
- **KNX IP** Ethernet communication.

Twisted pair (KNX TP)

It is the most common communication medium for the KNX system. It uses a two-core pair cable. All devices are connected through the same bus cable. It is cost-effective and easy to install. In this installation bus cable supplies both data and power. The rated voltage for the power supply is 30V while for bus data is 24V. The DC supply voltage created by a capacitor gives power to the device while the AC supply voltage is used for data transmission. DC voltage is separated by the device while the transformer separates the AC voltage.
The data transfer rate of TP is 9600 bit/s, and it travels one byte at a time through asynchronous data transfer. In normal data transfer, KNX TP needs 20 ms to send a telegram while it goes up to 40 ms when the device is being programmed therefore, it can process 50 telegrams per second. During data transmission when a logical zero is transmitted the voltage drop for a moment due to the inductor effect on the choke and then, within 104 µs it goes up again at the original voltage. When a logical one is transmitted the bus remains idle[11]. It is illustrated in the following figure(2.4)

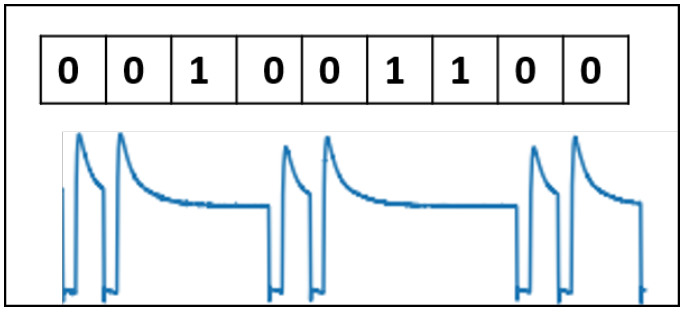


Figure 2.4: Signal shape of TP

Powerline (KNX PL)

KNX PL uses existing electricity cables of a building as communication media therefore, it is relatively easy and convenient to install a KNX system in a building. One of three phases and one neutral of the electricity cable acts as a data transfer medium hence no dedicated bus cable is required. Data signals are superimposed on the main voltage. No additional power is required since it is getting the necessary power from the 230 V mains. To make sure data can transfer via all three phases, a phase coupler is used while and band-stop filter is used to stop the data signal transmission out of the building and toward the main grid.

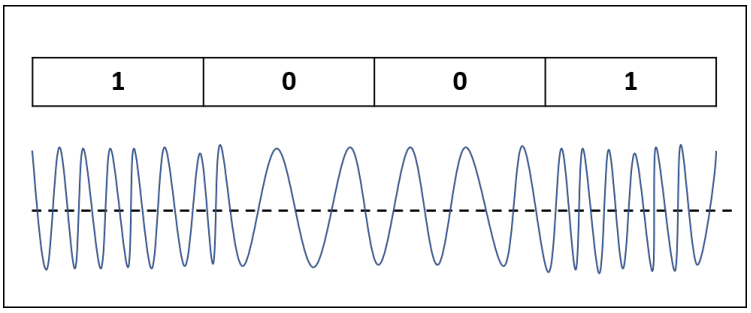


Figure 2.5: Signal shape in KNX PL

The data transfer rate of KNX PL is 1200 bit/s. Data is transmitted via spread frequency shift keying(S-FSK). A logical zero is represented by a signal frequency of 105.6 kHz, while a logical one corresponds to a frequency of 115.2 kHz (see figure 2.5). Signals are superimposed onto the main voltage. The center frequency between the two waves is 110 kHz, that's why it is also known as PL110. To evaluate the signal, digital processing methods are used where the receptive sensitivity of the device and transmission power are constantly adapted[11].

KNX Radio Frequency (RF)

A Radio-frequency medium is the best choice when some part of the building may be inaccessible or it is not possible to lay new cables. It is appropriate to use this medium as an extension of the already installed KNX installation. In theory, it is possible to control and manage all the devices and the entire operations wirelessly. This setup is suitable when the RF sensors don't need power from power mains rather they are battery-operated devices. Therefore, those devices cannot be always in ready-to-receive mode. A unidirectional device model has been defined in KNX which only sends telegrams when necessary and there is no receiver. Actuators always have to be ready-to-listen states therefore, they take power from 230 V main and are bidirectional.

Radio technology passes information using modulation. There are different modulation techniques are used like amplitude modulation, phase modulation, frequency modulation, or

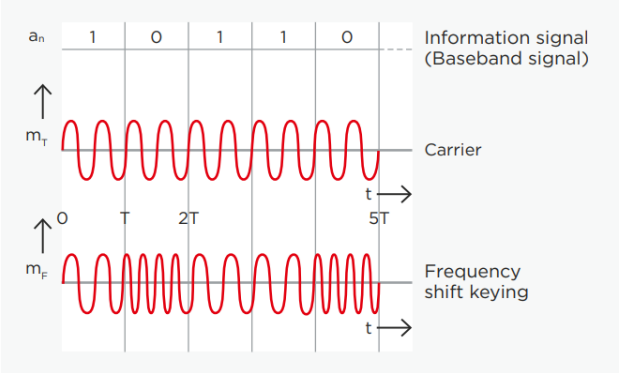


Figure 2.6: Frequency modulation in KNX RF. [11]

maybe a combination of all. KNX RF uses frequency modulation. The logical zero and one are transmitted by a slight modification of the frequency of the carrier wave (Fig. 2.6). There are two versions of KNX RF; KNX RF Ready and KNX RF Multi. In KNX RF Ready the center frequency is 868.3 MHz but only has one communication channel which is vulnerable to interference. To overcome this interference KNX Multi allows the device to switch to a different free channel from the occupied channel. There are two fast channel F1 and F2 which are intended to use for human-operated application like lights, blinds, etc., and two slow channels S1 and S2 which does not need to be in receive mode all the time[11].

KNX IP

TCP/IP and UDP are two widely used protocols used for data transmission. They are used in the transport layer of the OSI model. The base protocol of TCP and UDP is IP protocol. TCP is a connection-oriented protocol that establishes a permanent connection and ensures data transmission reliability while UDP is a connection-less protocol that does not ensure transmission reliability but it is much faster than TCP. The UDP protocol is often used in building automation. KNX can also be linked with Ethernet. The building can be monitored and controlled via Ethernet from anywhere. Existing KNX installation in the building can be used for KNX main and backbone lines[11].

Two Ethernet communication methods are used in the KNX system; tunneling and routing. Both of these use the UDP protocol. Tunneling is used from a local area network or internet to access the bus for KNX installation or configuration while routing is used to exchange telegram over an Ethernet network. They are known as KNXnet/ IP tunneling and KNXnet/ IP routing. IP communication in KNX is shown in figure 2.7. Communication starts from the application layer then continue to the transport layer (UDP), to the network layer (IP), and ends up in Ethernet or physical layer.

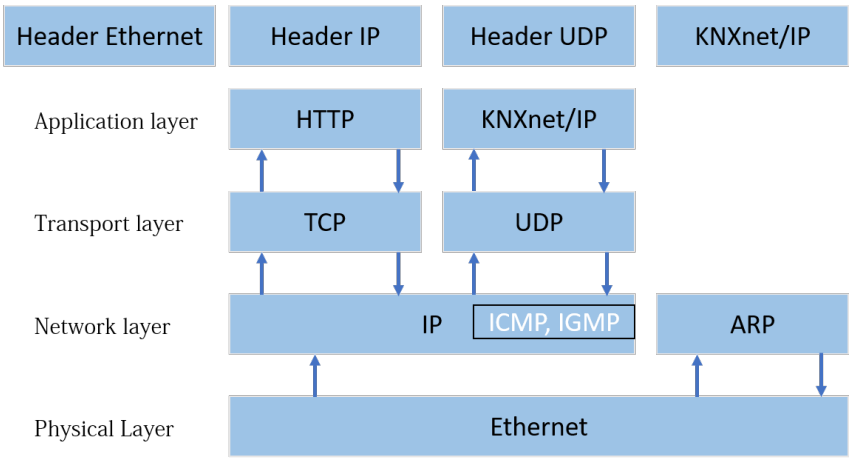


Figure 2.7: KNX IP in the OSI reference model

2.1.1.2 Telegram Structure & bus access methods

Information between different KNX devices is exchanged via telegram which typically consists of a sequence of character where each character are 8 bits or 1 byte long. Since all devices use the same bus therefore only one telegram can be transmitted at a time. To avoid collision among telegrams of different devices different bus access methods are used. The telegram structure and bus access method of different KNX installations will be discussed here.

KNX TP

KNX TP has four fields (Figure 2.8). The Control field determines the priority of the telegram and also mentions whether it is a repeated telegram. The address field defines the individual address of the sender and the destination address of one receiver or group address in case multiple receivers belong to a specific group. The Data field contains the payload of the telegram which can be up to 16 bytes long. The checksum field performs the parity check.

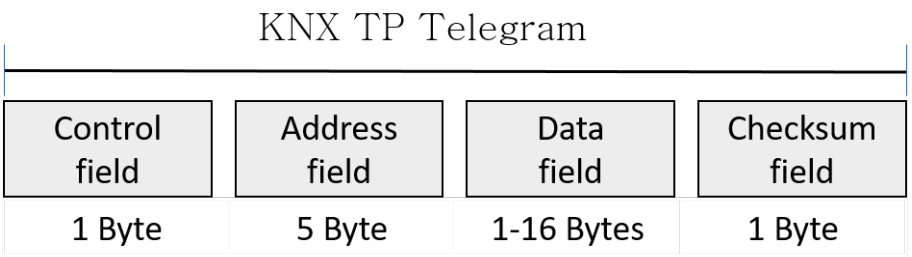


Figure 2.8: Telegram structure in KNX TP

Bus access method

A collision can happen when multiple devices want to transmit data at the same time. Only

one telegram can be transmitted at one time. To prevent collision while transmitting telegrams Carrier Sense Multiple Access/ Collision Avoidance (CSMA/CA) method is used. All transmitting devices listen for incoming data transmission along the bus, when there is no transmission one can send the telegram. If in the meantime another device wants to transmit data it senses the transmission of other incoming data and stops its own transmission until the previous transmission is finished, and then it resumes transmitting data. The priority level of a telegram can be specified in the control field. If two telegram has the same priority, then priority is determined by the physical address where 0 has priority over 1 [11].

KNX PL

KNX PL telegrams also have four fields and it is basically extended KNX TP telegrams (Figure 2.9). The training field synchronizes and sets the senders and receiver levels. The preamble field announces the start of the transmission and inhibits telegrams from colliding while accessing the bus. The third field contains a complete telegram. The system ID field contains a system ID that is the same for one system and it helps to prevent communication among different KNX PL devices which does not have the same system ID.

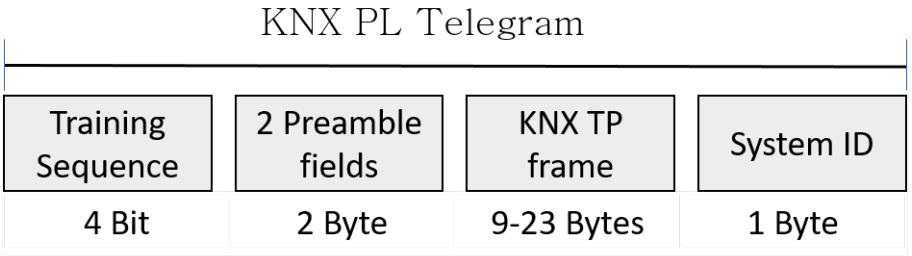


Figure 2.9: Telegram structure in KNX PL

Bus access method

KNX PL also uses the bus access method to stop the collision of telegrams. In this method, all devices are in receiving mode by default and it can only change to sending mode if certain conditions are met. If a device detects the bit string of a preamble it assumes the bus is occupied and it delays the transmission until a later point in time. It diminishes the possibility of a collision.

KNX RF

IN KNX RF data are sent using multicast telegrams. That means several bus devices can receive one telegram at the same time. For example, several lights can be turned off at once. KNX

RF telegram has several data fields which are partitioned by checksum fields (Figure 2.10). The data field contains bus-related information and actual payload as well. The first data block has three fields. The control field contains different information like the length of the telegram, transmission quality, if the RF device is battery operated then the status of the battery and whether the device is unidirectional or bidirectional. The second field contains a serial number or domain address. The serial number is set by the manufacturer and it is not changeable. Checksum field checks for any data transmission error. The second data block contains the individual or source address, the destination address, and the actual payload. It can contain more data blocks depending on the length of the payload.

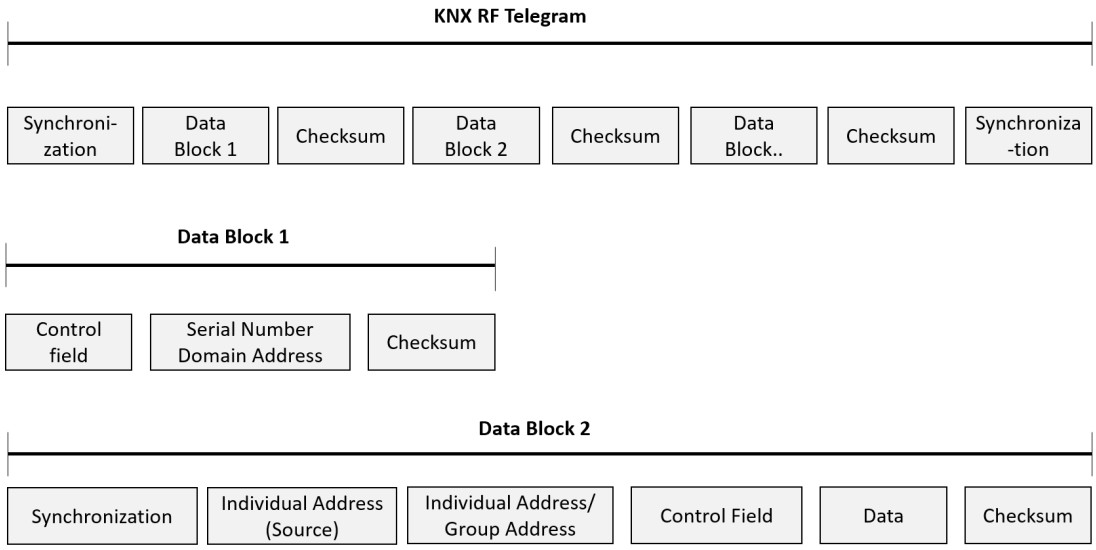


Figure 2.10: Telegram structure in KNX RF

Bus access method

Unidirectional devices send a telegram when necessary and the rest of the time they stay idle. The duty cycle of unidirectional devices is very low, around 1% therefore the chance of telegram collision is very thin [11]. Bidirectional devices check the radio channel occupancy before sending a telegram. If occupied, then the device waits until it is free again.

KNXnet/IP

In addition to the KNX TP telegram, KNX IP contains some more information. In the first field, the header length is always the same. The function of the header is to indicate the start of the telegram. The protocol version indicates the KNXnet/IP protocol. The service type identifier indicates what action needs to be performed. The total length of the telegram is mentioned in the service type identifier. The last field carries the payload (Figure 2.11).

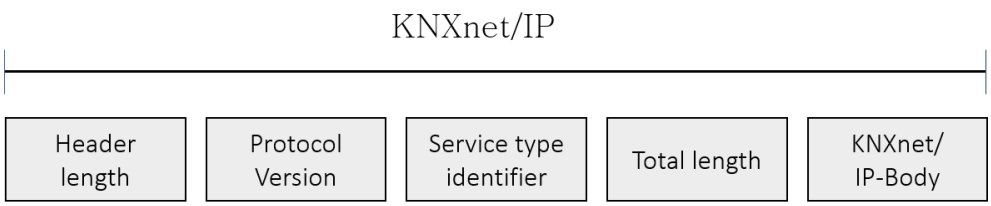


Figure 2.11: Telegram structure in KNXnet/IP

2.1.1.3 Topology

KNX TP

The logical tree topology of the KNX system is divided into devices, lines, and areas. An exemplary tree topology is shown in figure 2.12. The basic unit of this installation is a line. A line consists of a KNX power supply including a choke and a maximum of 64 devices. The devices get power from the power supply and the bus cable is used for data exchange among devices. The bus cable can be laid as desired which offers great flexibility. If more than 64 devices need to be connected then a line repeater is used. This addition of a line repeater is known as line segmentation. It includes a line repeater, a power supply including a choke, and 64 more devices where the line repeater is also counted as one bus device in the line. A maximum of three repeaters can be connected with a line which means the highest number of bus devices is 255 [11].

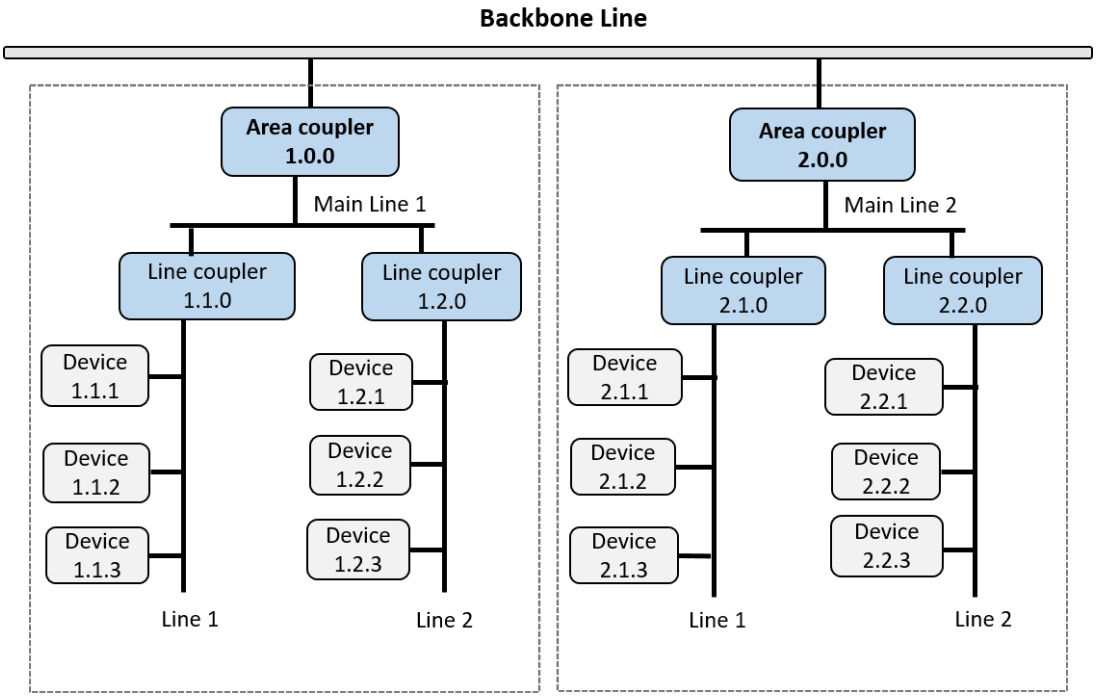


Figure 2.12: Exemplary tree topology of KNX TP.

To expand the installation new lines are added via a line coupler since a line repeater cannot be used to expand the installation and take the advantage of KNX system's full potential. Line repeater and line coupler are most often similar hardware therefore it is more convenient to use a line coupler since it can also filter telegram, which means less telegram traveling along each line. Line coupler will not allow a telegram to travel to another line where it does not belong. Each line comprises a power supply and 64 more devices likewise where the line coupler is counted as a device. The main line can accommodate up to 64 devices and it needs power as well. The highest number of line couplers that can be connected and operated with the main line is 15. 15 line connected with the main line forms an Area. Up to 15 areas can be connected using an area coupler and that makes a complete system. The area line can accommodate 64 more devices just like the main line. This area line is also known as the backbone line and it needs a power supply of its own [12].

Cable Length

To ensure optimum performance and uninterrupted data exchange the distance of devices from the power supply or between devices also needs to be considered. The maximum distance from the power supply to a device should be 350m. The maximum distance between arbitrary two devices in a line should be 700m and the length of a line segment is a maximum of 1,000m. [11]

Addressing

Physical address

Each device in the KNX system is assigned a unique number called the individual address or physical address. The number is separated by dots. The physical address indicates where this device sits in network topology and also identifies the device. The first digit denotes the area, the second digit denotes the number of lines and the third digit indicates the position of the device in a line. The address is 16 bits in size and it is divided into follows, the first 4 bit is for the area, the second 4 bit is for the line, and the last 8 bit is for the device number (Figure 2.13) [13]. When assigning physical address area or line coupler must always be given the number 0 to define they are area/line coupler. In the example physical address, 2.3.20 means bus device 20 is in the second area and third line.

Group address

KNX also offers multicast support in the network layer which means one packet can be received by many devices. If there are some common devices belonging to a group for example

Byte 0		Byte 1
4 bit	4 bit	8 bit
Area	Line	Device

Address e.g. 2 . 3 . 20

Figure 2.13: Structure of physical address

lights or blinds, then one network packet can turn on or off all the lights in a room or open or close the window blind. Therefore, the same type of devices can be assigned to the same group and it will be provided with a group address or logical address. A group address also consists of 16 bits but has a difference in structure. There are two kinds of group addresses.

- Group address with a main group and subgroup (two-level addressing)
- Group address with the main group, middle group, and sub-group (three-level addressing)

In a two-level subgroup, the first bit is always 0, then later 4 bits belong to the main group and the last 11 bit belongs to the subgroup (Figure 2.14). That means the maximum number of the main group is 16 and the subgroup is 2048.

1 bit	4 bit	11 bit
0	Main group	Sub group

Figure 2.14: Structure of 2-level group address

In the three-level subgroup, the address is divided into 3 parts. It takes 3 bits from the subgroup and creates a middle group while the rest is the same as two-level addressing (Figure 2.15). [13]

1 bit	4 bit	3 bit	8 bit
0	Main group	Middle group	Sub group

Figure 2.15: Structure of 3-level group address

The addressing scheme of the physical address and group address is differentiated by '.' and '/'. The physical address is denoted by x.x.x while the group address is denoted by x/x/x.

KNX PL

The topology of KNX PL is mostly the same as KNX TP. It consists of areas and lines. The basic unit of an installation is a line and it can contain up to 255 devices. In an area, there can be a maximum of 15 KNX PL lines and the highest number of areas in PL is eight. A system coupler is used instead of a line coupler. A band stop filter is used to separate one PL line from another. This system coupler also has a filter function which helps to minimize the number of telegrams along the line. To prevent telegram congestion KNX PL is a good choice since the number of telegrams in KNX PL is lower than in KNX TP.

Individual Address

The system coupler is assigned with the number 0 and all other device gets an individual address based on their location in the topology. For example, individual address 2.3.20 means a PL bus device with the number 0 in the second area and third line.

KNX RF

In this installation, the device can be installed virtually anywhere since they communicate via radio media therefore there is no constraint of lines, and does not need to be arranged hierarchically. But it needs to be ensured the devices are within range and they can communicate with each other, any sensor can communicate with any actuator. A device from a nearby KNX RF installation can receive a transmission from another KNX RF installation, thus, one device from a different installation must not cause any interference. It has been discussed earlier that (Figure 2.10) the RF telegram contains a serial number or domain address, this number is a unique identifier of the device and only the receiver paired with the transmitter can process the telegram. A KNX system can be made of a complete radio network or a combination of different communication mediums like KNX TP.

KNX IP

KNX IP can be used in conjunction with the KNX TP connection. The backbone of the KNX system can be based on Ethernet allowing more flexibility and higher speed. KNXnet/IP routing is a multicast-based telegram. A multicast is a group-oriented connection method where all devices listen to a multicast address. This allows all the telegrams to be found by all IP devices which are listening to this address. The KNX Association has preserved the address 224.0.23.12 as a multicast address but any address can be used if it is the same on all devices [14].

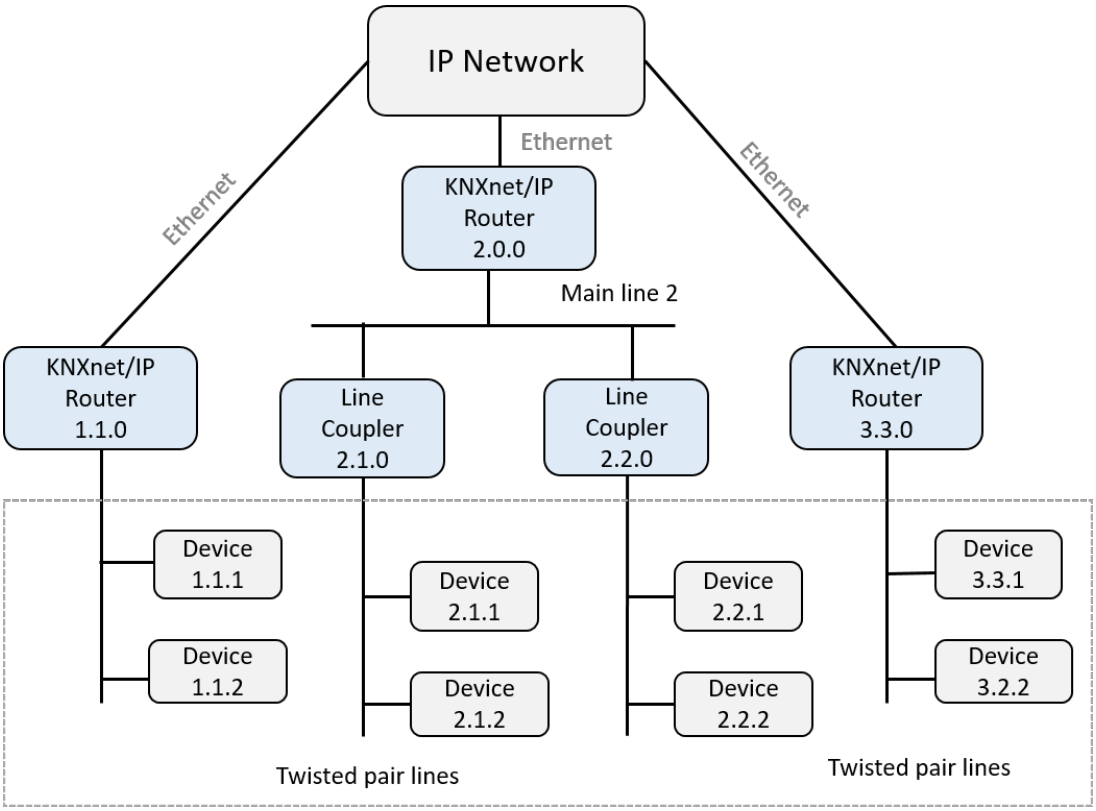


Figure 2.16: A model tree structure of KNXnet/IP

In the place of the main and area line, KNX IP can be used. KNXnet/IP routers are used for this purpose. Ethernet is to connect all the KNXnet/IP routers and KNX TP is used with the router. Using the routing method router exchanges telegram with another router. As a line and area coupler, KNXnet/IP router can also be used and similar to other couplers this router is also capable of filtering telegram. Some router has the functionality of filtering telegram with an individual address. Router use routing method to communicate with one another and other KNX devices as well via Ethernet. ETS can use the router as an IP programming interface using tunneling. A completely different KNX/IP installation can be connected with an existing installation via Ethernet if both buildings are installed with KNX TP and are combined into a single system [11]. It is more flexible since both buildings can be monitored or managed from one place. A model tree structure of KNX IP and KNX TP combined is shown in figure 2.16.

2.2 Intrusion detection system

An intrusion is an attempt to compromise Confidentiality, Integrity, and Availability (CIA) or to neglect the security structure of a network or computer. An Intrusion Detection System (IDS) is the mechanism of monitoring the events occurring in a computer or network and analyzing them to detect any sign of suspicious activity and issue an alert when such type of activity is found [15]. An IDS is basically a software application that inspects the network

for any kind of policy violation or harmful activity. The administrator normally gets an alert when any kind of breach occurs or is collected centrally using a security information and event management (SIEM) system. A SIEM system collects output from multiple sources and uses an alarm filtering method to characterize malicious activity from false alarms [16]. The intrusion detection system is the eye of the network. It keeps an eye on the network traffic and alerts the administrator when some kind of anomaly is found.

Intrusion detection systems monitor the network to detect any kind of anomaly but they could also produce false alarms. Therefore, organizations need to fine-tune the IDS when configuring it for the first time. IDS learns what normal traffic looks like on the network and then raises an alarm when something occurs out of normal observation. An IDS uses a different algorithm to detect anomalies and there is no chance an IDS can give entirely accurate detection [17]. There will be either false positives or false negatives. In the case of false positive IDS defines an activity as an intrusion but the activity is acceptable behavior while the false negative is identifying some activity or process as acceptable behavior when the activity is a real attack. For example, if the motion sensor detects some movement in the building at 4:00 in the morning, it could send an alert but in reality, it could be some cleaning crew which is acceptable behavior, this is a false positive. But if someone comes in the early morning intending to break into the server room for example and the IDS accepts it as acceptable behavior then it is a false negative. The consequence of a false negative is much worse than a false positive hence IDS are usually configured to generate more false positives rather than false negatives [18] [17].

The function of an IDS are as follows as [19]:

- Monitoring user's activity.
- Observe system activity.
- Data files assessment.
- Audition of system configuration.
- Recognizing abnormal activity and known attacks.
- Auditing system configuration and correcting system configuration errors.
- Save information about attackers.

In addition to IDS, there is also the Intrusion Prevention System (IPS) which scans the network for any potential security breach or policy violation as IDS does but has the primary

objective of preventing threats once detected. It differs from IDS since IDS only detects the threat and sends an alert but does not carry out any preventive measures. An IDS also differs from a traditional network firewall which uses a set of rules to allow or deny network packets. Basically, a firewall restricts access between network to inhibit intrusion but do not signal an alarm inside the network. In contrast, an IPS manages access control like an application layer firewall [20]. The goal of this thesis is to use an intrusion detection system possibly on every line of the fieldbus system to detect any anomaly. A tiny accelerator device that is capable of running neural networks in parallel will be installed on the line of the KNX system which will act as an intrusion detection system.

2.2.1 Classification of Intrusion Detection System

Different types of IDS can be installed to aid administrators to secure the network. The mostly used IDS are network-based intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS). Some other types of IDS are also used, however, that include file integrity and log file checker, and use of honeypots which is essentially a decoy device. Finally, some IDS combines different functionalities known as hybrid IDS [21]. Different kind of IDS and their functionality will be discussed here.

1. **Network Intrusion Detection System (NIDS):**
   Network-based IDS are devices that are intelligently allocated within the network to examine traffic from all devices traveling on the network. It carries out passive inspection of passing traffic on the entire network and checks any matches with the known attacks. Once the threat is identified or suspicious activity is discovered it raises an alarm. NIDS can be software-based or hardware-based systems and depending on the network requirement and manufacturer it can be connected via different network interfaces. NIDS are either anomaly-based or signature-based systems, both are techniques to distinguish normal traffic from malicious traffic. Potential issues with NIDS are encryption, tuning difficulties, signature development lag time, and high-speed network data overload [21]. An example of NIDS installation is, installing it on the network where firewalls are located to detect if an attacker is trying to break the firewall.

2. **Host Intrusion Detection System (HIDS):**
   Unlike NIDS, HIDS sits at the service end-point rather than in the network transit point. It runs on an independent host or devices to monitor incoming and outgoing traffic and when any suspicious activity is discovered it sends an alert to the administrator. It takes snapshots of the file system from time to time and compares them with earlier snapshots to find out any discrepancies like whether any analytical systems or files have been edited or deleted. It is mostly installed on servers or machine-critical

machines and is more focused on exploring the host operating system and host application functionality. HIDS can detect a variety of potential attacks targeting an organization's server such as web servers, mail, and DNS [21].

3. **Protocol-based Intrusion Detection System (PIDS):**
   PIDS monitors the state of the protocol and dynamic behavior of the web server. It sits at the front end of a server, interpreting and controlling the protocol between the user and the server. It checks the HTTPS protocol stream constantly to secure the web server by filtering the IP address or port number [19].

4. **Application Protocol-based Intrusion Detection System (APIDS):**
   APIDS is IDS that focuses on a specific application protocol or protocols in use by the computing system. It usually resides within a group of servers and detects intrusion by examining and interpreting the communication on the application-specific protocol. It can be installed between a web server and the database management system to monitor the SQL protocol as it interacts with the database [19].

5. **Hybrid Intrusion Detection System:**
   The hybrid intrusion detection system is generally a combination of two or more IDS. It combines both Host IDS and Network IDS functionality on the same platform to develop a complete view of the network system. It can check the file system's integrity like Host IDS but the Network IDS functionality is only useful when some traffic is destined for this specific device or host. It is mostly installed on an organization's most critical machine [21]. It is much more effective compared to other IDS.

2.2.2 Detection Methods of IDS

The means by which, IDS functions can also be different. In the following subsection, some detection methods of IDS will be discussed.

1. **Signature-based Method:**
   Signature-based intrusion detection is the most prevalent type of IDS. This system monitors the network or server and tries to find any match against some predefined attack lists or signatures. The IDS looks for a series of bytes or packets sequence known to be harmful, for instance, a sequence of bytes in the data stream or a sequence of requests in HTTP traffic. If some particular network conversation matches a signature predefined on the IDS, the system administrator gets an alert. It is quite effective in

security monitoring yet has some drawbacks. To detect potential intrusion, the signature database must be large. It is difficult to keep pace with the network traffic as the speed of the network increases [21]. It can only detect a known attack that has been inspected already. New variants of known attacks cannot be detected and it requires enough effort to keep the database up to date [17].

2. **Anomaly-based Method:**
   Anomaly-based intrusion detection follows the principle that intrusion can be detected by distinguishing between expected or normal behavior and unexpected behavior. Deviation from observed normal behavior beyond a threshold level raises an alarm. In the beginning, the system is trained to learn what is normal and anything that does not correspond to previously learned behavior is assumed to be an intrusion. Features to be considered are throughput, events per time, and any other measurable data. If the attack profile changes quickly then the anomaly detection performs weakly. While rebuilding the attacker's behavior profile, the system becomes unavailable [17]. This method can detect unknown attacks unlike the signature-based method but in reality, it also generates a lot of false positive alarms [21].

3. **Stateful protocol analysis:**
   This method compares pre-defined profiles of the normally accepted definition of protocol state with observed events and recognizes any deviation of protocol state. It knows which protocol steps are permitted during all the states of communication and can differentiate unexpected sequences of instructions. This method needs more resources than other IDS types for protocol state tracking and inspection [15] [17].

2.3 Machine learning

Machine Learning (ML) is a subset of artificial intelligence that uses the study of algorithms and statistical models to build a system that has the ability to learn itself automatically from previous data or experience and can improve the outcome from experience without being specifically programmed. It imitates the way human learns and solve problems and gradually improves accuracy. AI pioneer Arthur Samuel has defined machine learning as the field of study that gives computers the ability to learn without explicitly being programmed [22, p. 21]. ML teaches machines how to use data more effectively and efficiently. Nowadays machine learning is being used in many places, starting from a simple Google search to self-driving cars. Traditional programming requires detailed instructions for a computer to follow, for example, recognizing different people from a photo while humans can do this task easily. Machine learning follows this approach and lets the computer learn and program itself based on previous data or experience. It starts with data, photos, numbers, or text for

example flower images, time series data from sensors, and bank transaction records. This data is then prepared to be trained, it is then called training data. The more data the better the program performs. Afterward, programmers choose the machine learning algorithm, provide the data to the model, and the model train itself to find any pattern and make predictions.

There are different kinds of algorithms available because a specific algorithm works best on specific types of data and specific types of problems one wishes to solve. There is no single algorithm that is suitable to solve any kind of problem, for example, a Convolutional Neural Network or CNN works best with image data, and Natural Language Processing (NLP) works best with text and speech data. In the specific context of this thesis, machine learning will be used in IDS to detect anomalies by finding abnormal and possibly malicious data. The model will learn what the traffic from normal operations looks like and raise an alarm if any traffic is found that does not correlate with normal traffic. It is assumed the abnormal traffic lies beyond the cluster of normal observation and appears as an outlier. The machine learning algorithm can be divided as follows [23]:

- Supervised
- Unsupervised
- Semi-supervised

Supervised

In supervised machine learning model is trained with labeled data sets where the labeling is done by a human, which allows the models to learn and improve accuracy over time. There are Input variables (x) and an output variable (y), and an algorithm is used to learn the mapping from input to output. Then a learning algorithm trains the model to predict output in response to new datasets. For example, classifying spam messages to the spam folder from the inbox folder. It is best suited for static data and stable systems. Since the labeling process needs to be repeated every time the baseline behavior changes this method is less suited for a dynamic system but it gives good results due to the clear decision surface or normal and abnormal data [2].

Unsupervised

It uses a machine learning algorithm to analyze and group unlabeled data sets. Unlike supervised, there is no labeled data and no human intervention hence the name unsupervised.

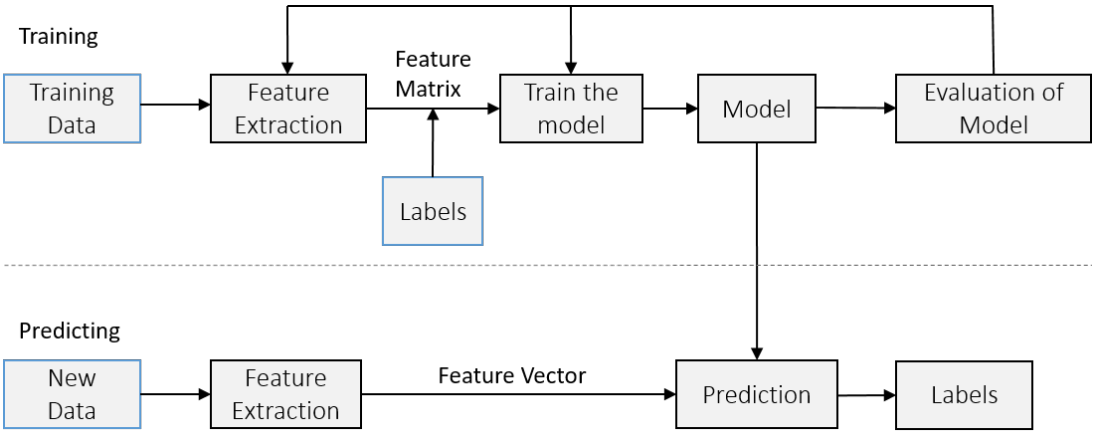


Figure 2.17: Supervised learning workflow

These algorithms find underlying hidden patterns and structures of data. An unsupervised model should be able to find some features to group similar inputs together from the information of all possible inputs. Two inputs are considered to be similar when they produce the same output label[22, p. 24]. The ability to discover similarities and differences in data sets makes it an ideal suit for data analysis and image and pattern recognition. It is performed mostly as a part of exploratory data analysis. It uses a process of dimensionality reduction to reduce the number of features in a model and is also used to find clusters of data. Unsupervised learning is usually cheaper to collect since a human does not need to label the input to the corresponding output.

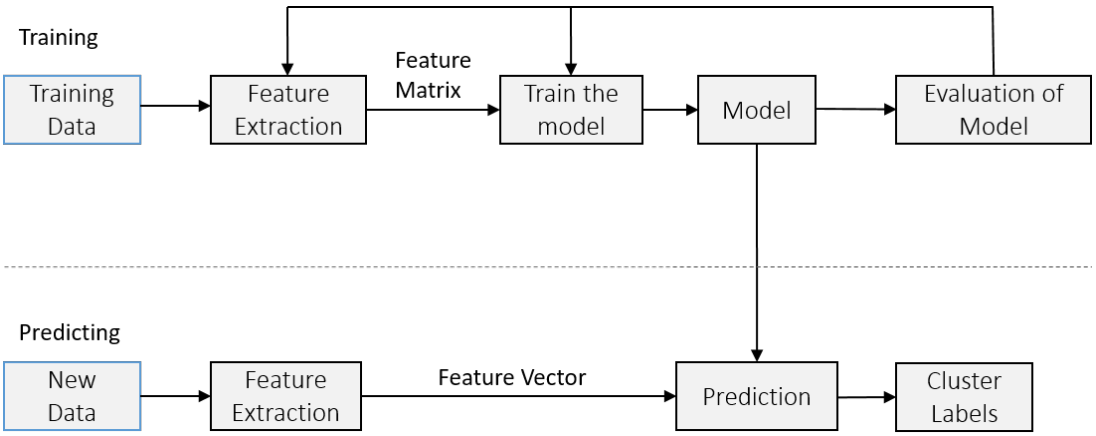


Figure 2.18: Unsupervised learning workflow

Semi-supervised

Semi-supervised is a happy medium between supervised and unsupervised learning. We can put together a small amount of labeled data to guide classification and feature extraction with a large amount of unlabeled data during training. The training data must be free of outliers.

A decision boundary is derived from the training session which clearly outlines the training data. When the model is trained every new incoming data is checked against the decision boundary derived earlier. If it lies within the decision surface, it is classified as normal observation otherwise classified as abnormal or outlier. It is useful when enough labeled data is not available for supervised learning algorithms since labeling a large data set is costly, therefore, a readily available small amount of labeled data is used. It is a matter of concern when the training data is not completely clean and does not include the full range of normal data as all unknown data will be marked as outlier[2].

2.3.1 Convolutional Neural Network (CNN)

Convolutional Neural Network or CNN is a deep learning model that works with grid pattern data such as images. It is mostly used to analyze visual images. CNN was inspired by the organizations of the animal visual cortex and designed to learn spatial hierarchies of features automatically and adaptively [24]. A biological neuron corresponds to an artificial neuron. The CNN kernel acts as a receptor that can respond to different features. Activation function mimics the biological function of the neuron that if only neural electric signal exceeding a certain threshold can be transmitted to the next neuron [25]. CNN is composed of three types of layers: convolution, pooling, and fully connected layer. Convolution and pooling layers extract the feature and then feed it to the fully connected layer which maps the extracted features to the final output, for instance, classification. In a digital image, pixel values are stored in a 2D grid or array of numbers, then there is another small grid of numbers known as a kernel which is basically a feature extractor, and is applied to every position of the image. This process is called convolution which makes CNN greatly efficient. In the training period, the algorithm attempts to optimize parameters like kernels by minimizing the difference between output and ground truth labels through some learning algorithms called backpropagation and gradient descent. A simple architecture of CNN is shown in figure 2.19. The objective of this thesis is to set up an IDS with the help of machine learning on some accelerators like NVidia Jetson Nano or Google coral ai, whereas, those devices are only capable of running neural networks. Therefore, CNN has been chosen as the machine learning algorithm to carry out the experiment.
In the following subsection, the architecture of CNN and how CNN works will be discussed.

Working principle of CNN

The building blocks of CNN consist of convolutional layers, pooling layers, and fully connected layers. A CNN architecture is formed when these layers are stacked. Typically, it comprises iterations of various convolutional layers and a pooling layer, followed by a fully connected layer.

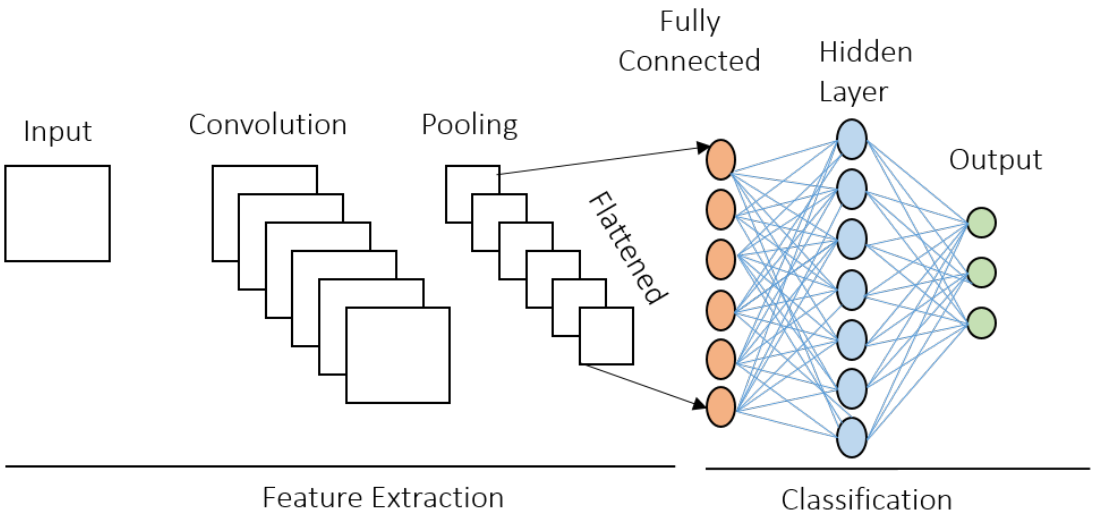


Figure 2.19: A simple Convolutional Neural Network (CNN) architecture

In this section, 2D-CNN will be discussed for simplicity purposes but a similar operation can also be executed for 3D-CNN. Different parts of the whole operation of CNN will be discussed in the following.

2.3.1.1 Convolution layer

The convolution layer performs the feature extraction which typically consists of linear and nonlinear operations such as convolution operation and activation function. The objective of the convolution layer is to extract high-level features such as edges from the image. In convolution operation, small arrays of numbers known as kernels or filters are applied across the input which is basically a big array of numbers known as a tensor. An element-wise product between each element of the input at each location and kernel is calculated and summed to obtain the output value. This operation is often referred to as a scaler product. The kernel slides through the whole input and calculates the scalar product of the whole input. The output value sits in the corresponding position in the output tensor, also known as a feature map. Input tensor, kernel, and feature map and how they were calculated are shown in figure 2.20. In this example, the first value of the feature map is calculated as follows:

$1*1 + 2*0 + 1*1 + 2*0 + 0*1 + 0*0 + 1*1 + 0*0 + 2*1 = 5$

The rest of the values are calculated following the same process. Different kernel extracts different features of the input image therefore, multiple kernels is applied. The number of feature maps will be the same as the number of kernels applied. Depending on the problem number and size of the kernel need to be defined. The typical size of kernels is 3 x 3 but it could also be 5 x 5 and 7 x 7.

The example shown in 2.20 has no padding with the input tensor therefore the width and

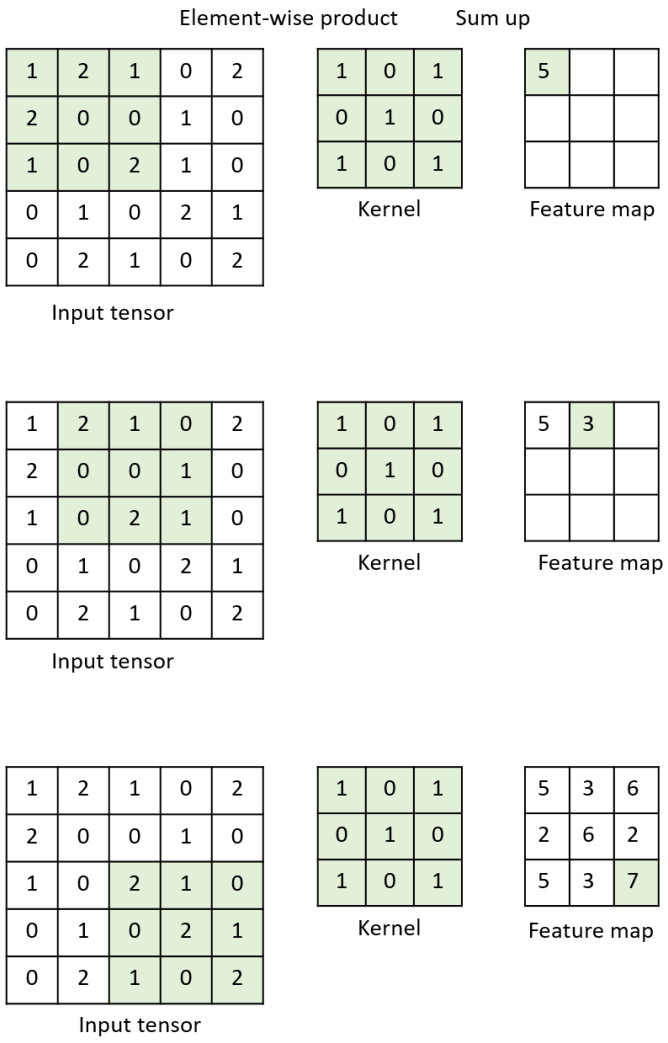


Figure 2.20: An example of convolutional operation with kernel size of 3x3, no padding and stride of 1

height of the image will reduce after every convolution. To prevent that, zero padding is used where a row and a column of zeros are added to the outer frame of the input tensor which allows more space for the kernel to cover the image and keep the same in-plane dimension throughout the convolutional operation. Convolution with zero padding is shown in figure 2.21

When the kernel slide through the input tensor, this slide is called stride. Users need to define

the stride in convolutional operation. Usually, the stride is 1 but if the feature maps need to be downsampled, a stride of more than 1 is used. The alternative technique to downsample the image is a pooling function which will be discussed later on. Based on the task and given training dataset, the CNN model discovers the best kernel during the training process. Kernels are the only parameters that are learned automatically during training, except that, the size and number of kernel, padding, and stride which are known as hyperparameters need to be specified before the training process begins.

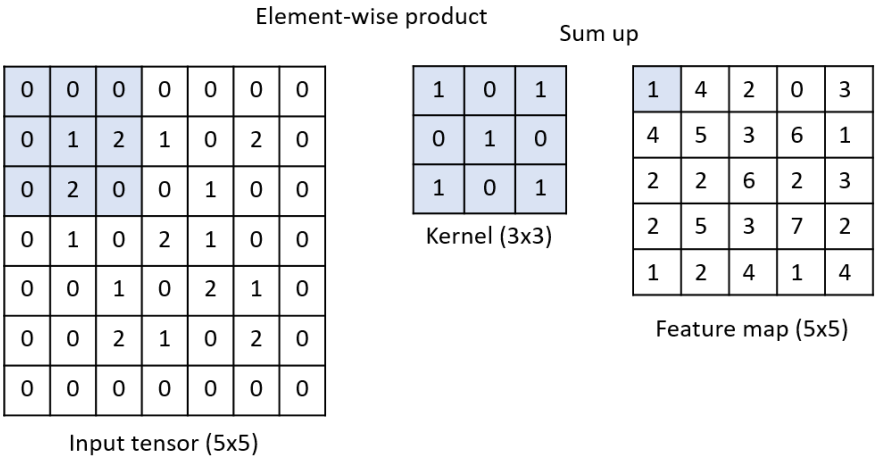


Figure 2.21: An example of convolutional operation with kernel size of 3x3, with zero padding and stride of 1

Activation function

The output of the convolutional layer is passed through a nonlinear function called the activation function which simply activates a neuron if it exceeds a certain threshold value. Different types of linear functions are available such as sigmoid, hyperbolic tangent (tanh) function, and rectified linear unit (Relu). Due to similarities with biological neuron behavior the first two functions were used earlier whereas nowadays most commonly used activation function in a neural network is ReLU [24]. It simply computes the function: f(x)= max(0,x) which means it rounds up any value less than zero to zero and keeps the other values the same. The general structure of the activation function is shown in figure 2.22.

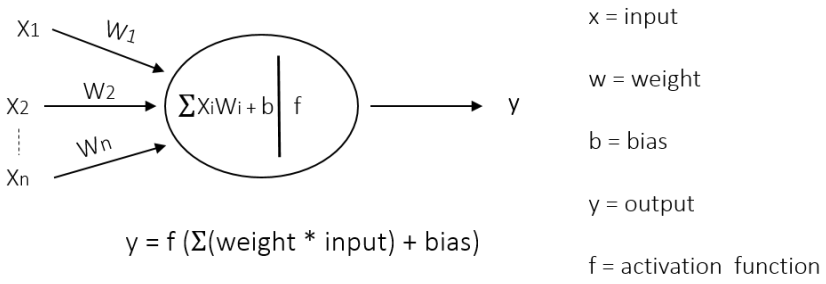


Figure 2.22: General structure of activation function

Here X_i represents the input features and W_i represents the weight assigned to the connection between input and neuron and b is the bias value. y is the output of the neuron and f is the activation function which could be ReLU, sigmoid, or tanh. The activation function introduces non-linearity in the equation. Without an activation function, the input of each layer will be a linear function of the previous output layer hence it does not matter how many layers the neural network has, the output will always be a linear combination of the input which

means the hidden layer will be of no use [25]. That is why the activation function is used in the neural network to bring in non-linearity.

2.3.1.2 Pooling layer

A pooling operation is performed to downsample the feature map to reduce the computation process and also to introduce a translation invariance to small shifts and distortion. The most popular pooling operation is max pooling whereas average pooling is also used sometimes. It simply extracts the maximum value in each patch from the feature map and discards other values. The common practice of max pooling is with a filter of size 2 x 2 with a stride of 2. It reduces the dimension of feature maps by a factor of 2. The depth dimension of feature maps remains the same. A max pooling operation is shown in figure 2.23.

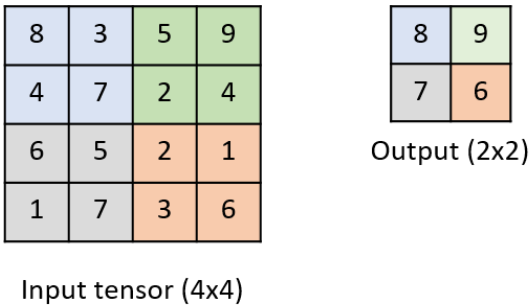


Figure 2.23: An example of max pooling operation with a filter size of 2x2, no padding, and stride of 2

2.3.1.3 Fully connected layer

After performing convolution and max pooling operation the output feature map is flattened, i.e., transformed into a 1D array of numbers. It is then connected to one or more fully connected layers also known as the dense layer where every input is connected to every output by a learnable weight. If there are more layers between the flattened and output layer they are called the hidden layer. The extracted features are then passed through the fully connected layer and they are mapped to the final output of the network. If it is a classification task the final output layer has the same number of nodes as the number of classes and it outputs the probabilities for each class. Each fully connected layer is followed by an activation function such as ReLU. The last layer activation function is different from other activation functions and it varies depending on the task. For instance, for the binary classification task sigmoid is used and for the multiclass classification task, a softmax function is used. Some commonly used activation function in the last layer is shown in the table 2.1:

Task	Last layer activation function
Binary classification	Sigmoid
Multiclass single-class classification	Softmax
Multiclass multiclass classification	Sigmoid
Regression to continuous values	Identity

Table 2.1: Commonly used last layer activation functions for different tasks [24]

Training network

In the training process, appropriate kernels for the convolutional network are identified and weights of fully connected layers are adjusted by minimizing the difference between prediction and ground truth. This is known as backpropagation. The model performance under specific kernels and weights is measured by loss function through forward propagation. Kernels and weights are then continuously updated through backpropagation to minimize the loss value. A complete operation of CNN including the training process can be apprehended from the figure 2.24

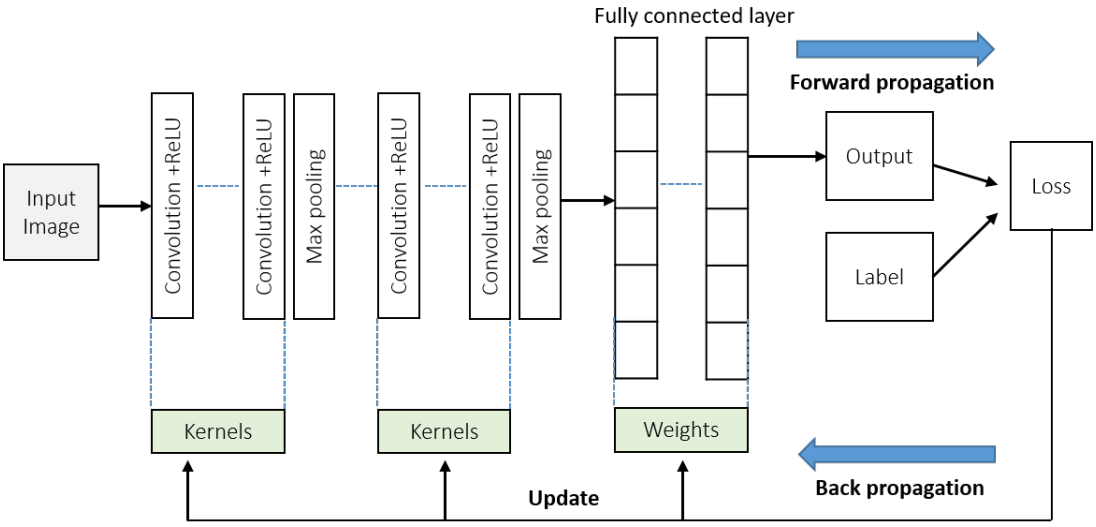


Figure 2.24: An overview of CNN architecture with the training process. based on [24]

Since the CNN algorithm will be used for the implementation, therefore, it is important to comprehend the complete working principle of CNN, what is the purpose of the activation function and how the accuracy and loss are measured, and how the algorithm tries to minimize the loss. This will help in better understanding the implementation part.

Chapter 3

Analysis

In this chapter

This chapter is divided into three parts. In the first section, attacker motivation, different attack scenarios, the security of fieldbus, their vulnerability, and defense mechanism will be reviewed. The second part will describe how the data from KNX fieldbus devices have been collected, followed by the third section, the collected data will be analyzed and how the data can be used for machine learning algorithm will be discussed.

3.1 Fieldbus security

The building automation system is already an important part of our life. It has been widely used all over the world since it offers great flexibility and convenience but this flexibility and comfort come with some risks. If the system gets hacked then the building could become useless, moreover, it can put people in more difficulty than comfort. The author Johannes Goltz in [26] has mentioned some incidents where the failure of a building automation system can be critical. For instance, a hotel building got hacked and the locking system stopped working, in Finland, the heating system was out of service due to distributed denial of service (DDoS) attack and it can get more critical if the malfunction occurs in the prison. It is important to

improve the security of new installations and existing installation as well. Some protocols offer encrypted and authenticated solutions such as KNX secure [27] but it requires a massive upgrade and the connected device needs to be exchanged. It is a good solution for new installation but existing installation cannot be updated that easily as it takes a lot of time moreover it is costly. For that reason, it is vital to find another way to increase the security of existing installations. At first, it is important to know the attacker's motivation, what kind of security is already offered by the existing fieldbus protocol, and why it is useful to use an IDS as a security mechanism. The attack can be executed by different people with different backgrounds and motives. It includes:

Internal attackers Internal attackers could be the employees or resident of the building who has prior knowledge about the system and may have authorized access to the system and devices. They can exploit the system's vulnerability and execute unauthorized actions. The motivation includes financial gain, revenge for any internal reason, blackmail, or maybe working for a criminal organization.

External attackers People who do not have direct access to the facility and act from outside. They can use social engineering to get access to the system and exploit vulnerabilities. They could be hackers, competitors, or criminals with the motivation of economic gain or social reputation. Their action could have from low to very high impact.

3.1.1 Attack scenarios

To have a better understanding of the consequences of an attack a brief discussion of possible attack scenarios will be given here. All the scenarios have strong feasibility of happening and they can cause a significant impact on people, business operations, and resources. It could be destructive and lead to potential damage.

Data center damage

Almost all businesses and organizations use data centers to store and process large amounts of data. The purpose is to distribute the data and keep backups for an emergency. Since this kind of facility host a lot of devices and equipment, the temperature is always high and a proper cooling system needs to be there as exposure to high temperature for some time could do potential damage to these devices and equipment. Nowadays. HVAC systems are integrated with the BAS system therefore it is possible to monitor and adjust the temperature according to the need. This HVAC system could be an attractive target for the attacker. If some attacker gains access to the system he can manipulate the system and even stop the air conditioning which could cause overheating, this can lead to the shutting down of the data center, and in the worst-case scenario the data center could be damaged due to overheating and data will be lost. Big organizations usually have several data centers and backups of data but still, it will take time to restore the affected system which can cause business interruption.

Physical access control

The access control system can allow or restrict people from access to certain areas. It consists of a card, card reader, controller, and databases with the user's credentials. When someone requests access it checks the user's credentials in the database whether he has access authorization and then allows or denies access. Since the access control system can be integrated with BAS and if the attacker has access to BAS he can easily mess up with the system and allows himself access to some forbidden areas for an instance data center to carry out a physical attack. The attacker can also lock out the building doors, cause panic, create insufferable conditions by manipulating the temperature, and put the residents' health in danger.

Damaging healthcare facility

Healthcare facility contains a lot of equipment which are vulnerable to cyber-physical attack. An attacker can raise the temperature of the hospital refrigerators which can spoil the blood, drugs even organs. The medical device includes PLC to control the machine that communicates with the network. If an attacker manages to take control of the PLC over the network they can perform any kind of action since this device lack basic security mechanism like authentication and authorization [28].

The attack scenarios described above, some of those are hypothetical but some attacks also did happen. For example, In November 2016 hackers got access to two buildings in Finland and were able to turn off the heating system remotely. In 2017 in Austria, a hotel got hacked the attacker locked the occupant inside and asked for ransom to unlock the door [29]. To access the fieldbus network attacker can access it physically or remotely. In the first case, an attacker can connect a device to the wire that is controlled by the attacker and in the second case, an attacker can access a network using an IP gateway from a compromised IP network. It has been found that a critical facility can be compromised by malicious code transferred through a USB stick with the help of a dishonest employee with direct network access [30]. Attack on field devices and network components becomes possible once the system is breached. Gaining access to the network in such a way requires knowledge of the protocols and network topology. When inside the system, the attacker can temper the behavior of the devices which can threaten the safety of occupants in case of device malfunction. As a consequence, it is important to have a security mechanism to prevent such occurrences.
In the following, some defense mechanism will be discussed that is important for a fieldbus to accommodate secure communication [31][32].

- **Authentication**
  Authentication is the process of recognizing someone's or something's identity. It provides access control for systems by matching credentials stored in the database of the authorized user. Users are typically identified by different knowledge factors such as

user ID, password, biometric signature, thumbprint, facial scan, or maybe smart chip card.

- **Confidentiality**
  Confidentiality ensures the data is protected from a third party who is not the sender and intended recipient. Anyone without proper authorization cannot see the message content.

- **Integrity**
  Integrity assures the data has not been modified by an unauthorized entity while it was traveling from sender to receiver. It assures the purity of the data.

- **Availability**
  Information and resources should be accessible to the authorized user whenever they need them. It guarantees the system, application and data are available to the user in a timely and uninterrupted manner.

- **Data freshness**
  Data freshness ensures the information received is recent and not an old packet or replay of previously transmitted data.

- **Update mechanism**
  There should be some mechanism to improve the security by eliminating the flaws or patching some vulnerabilities that haven't been found in the early stage of the development and implementing security updates from time to time.

3.1.2 Possible attacks on fieldbuses

It is important to know the types of different attacks that are possible to execute on the fieldbuses. The authors in [7] have exhibited an attack tree 3.1 that shows various attacks which is possible to execute on fieldbuses. They divided the attacks into network attacks, where the adversary gains access to the network medium such as radio frequency or powerline, and device attacks where the attacker uses the network interface of another compromised device such as sensors, actuators, and controllers. They also have divided device attacks into three parts. In software attacks, an attacker can exploit the device's software flaw through the regular communication channel. Physical intrusion or manipulation to interface with a device is a Physical attack. Side channel attacks are when an attacker observes some external parameters of a device that could be measurable to collect information about internal operation [7].

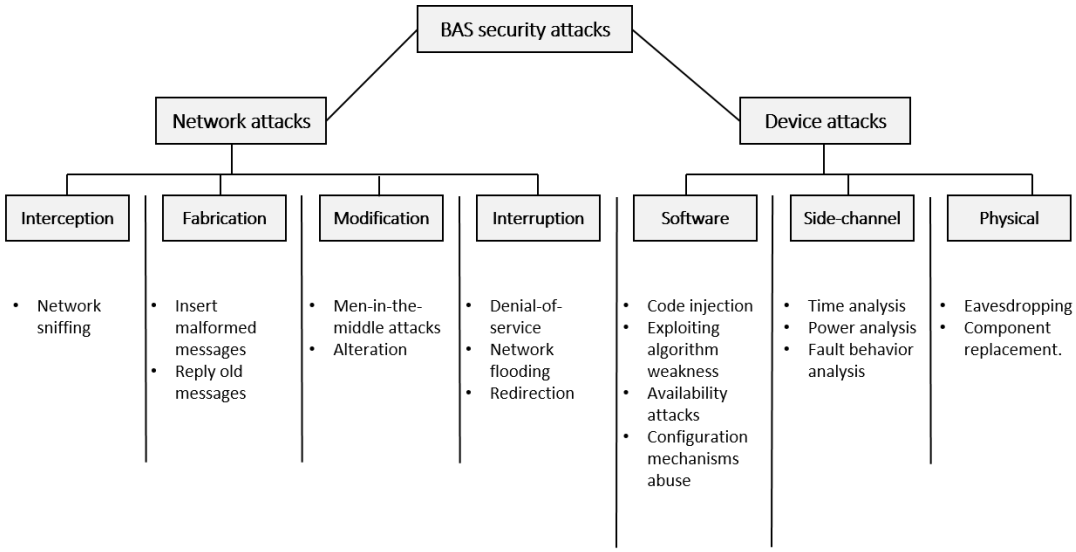


Figure 3.1: Possible attacks in BAS. Based on [7]

3.1.3 Security of KNX

Extensive research has been conducted to have a better understanding of the security status of the BAS. It shows that before the KNX security extension was published, there was almost no security existed in KNX installation [33]. KNX doesn't guarantee data confidentiality, data integrity, data freshness, and dedicated authentication service [32] [7]. It only provides a basic control scheme based on the cleartext password. A maximum of 255 different access levels can be defined and each of them has a different set of privileges. Access level 0 has the most privilege and access level 255 has the least privilege. Only a 4-byte password can be specified for each access level. But this scheme is only available for management communication. In addition, KNX has more security flaws [7], for instance:

- The keys are transmitted in cleartext.
- A single management tool named ETS is used to configure and maintain the KNX network but it does not offer a full access protection mechanism since there is only one key for the whole installation.
- KNX does not manage, generate or distribute keys securely. Hence, keys have to be uploaded manually to the device. It is up to the system administrator to ensure a secure environment to upload the key.
- The access protection mechanism cannot be applied to process data communication.

The authors in [34] have shown an attack on the authentication of KNX. The authentication mechanism only uses 4 bytes length password which is necessary to reprogram KNX. The necessary condition for this attack is to be able to install malware on any machine via any means,

which are connected with the same local area network (LAN) to which KNX is connected to. The malware will search for a KNXnet/IP gateway in the LAN to use and when it finds one it will check if the installation has been secured with a password. If any password has been used, then the malware will attempt to turn on and off the light switch repeatedly which will force the system administrator to reprogram the setup. While the reprogramming process begins it sends the password in cleartext and the malware can intercept the password. Once the attacker has the password, he can reprogram the installation with a new password, and eventually, the system administrator is locked out in his own system.

KNX secure

KNX secure was first introduced in 2015 by the KNX association to ensure data integrity, freshness, confidentiality, and authentication [35]. Depending on the different mediums used there are two variants, namely KNX Data secure and KNX IP secure. They ensure:

Data integrity: an authentication code is appended to every message to prevent an attacker from injecting manipulated frame.
Freshness: In KNX data freshness is ensured by a sequence number and in KNX IP it is secured by a sequence identifier. It helps prevent replay attacks.
Confidentiality: It encrypts the network traffic so that attackers cannot see the transmitted data in plain text. KNX devices ensure encryption according to the AES-128 CCM algorithm with a symmetrical key the message authentication code used is HMAC SHA256.

For KNXnet/IP medium the entire telegram is completely encrypted whereas in KNX Data secure telegrams are partly encrypted. It only encrypts the payload. KNX Data-secured devices can coexist with classic devices. Encrypted communication can take place among compatible devices while the other devices can perform in a standard way. Using a pre-shared key in KNX data secure key distribution is provided. However, that is not the case for KNX IP-secure. The complete installation needs to be upgraded at once to make it work. KNX IP secure uses an Elliptic-curve Diffie-Hellman key exchange over NIST curve K-283 to set up secure communication. Thus it acquires perfect forward secrecy. To authenticate the communication partner it uses pre-shared secret keys. For multicast communication is called group keys and for unicast communication, it is called device key [27].

The fieldbus network is a very simple network that consists of different bus segments and a tree-shaped structure. From the above discussion, it is unambiguous that the existing installation of KNX provides no security at all. KNX secure offers some security but to utilize it, the existing installation and devices have to be upgraded completely which is expensive and

requires a lot of time. Most building automation systems' architecture remains unchanged over their lifetime without greater changes. Such extensive installation and rare changes in bus system over the decades makes it complicated to upgrade something just for security reason. With the help of the security extension, new installation could be made secure but in the meantime, there has to be some way to secure the existing installation. If that can be achieved by adding some devices which can discover attacks that would increase the security of the existing installation. IDS is such a way where devices monitor the network traffic to look for any unusual operation and raise an alert. The data rate of the building automation system is typically very low which makes it very difficult to transmit all telegrams to a central controller for inspection. Therefore, it is important to only distribute necessary information based on what, an alert should be raised [26].

3.2 Data collection from KNX

The time series data was collected by Willi Brekenfelder, a student at the University of Rostock, as a part of his bachelor thesis [36]. The objective was, whether it is possible to find any attack from the physical layer data. The test setup used is shown in 3.2. An oscilloscope (PicoScope 2206A) was used to measure the data. The oscilloscope was connected to the KNX network to read and process the data on the bus. It reads continuously from the bus. After a predefined trigger has been set, the data was processed. The data was stored as a comma-delimited or CSV file for further experiments. In this thesis, those files containing the device's voltage readings will be used to find anomalies with the help of machine learning.

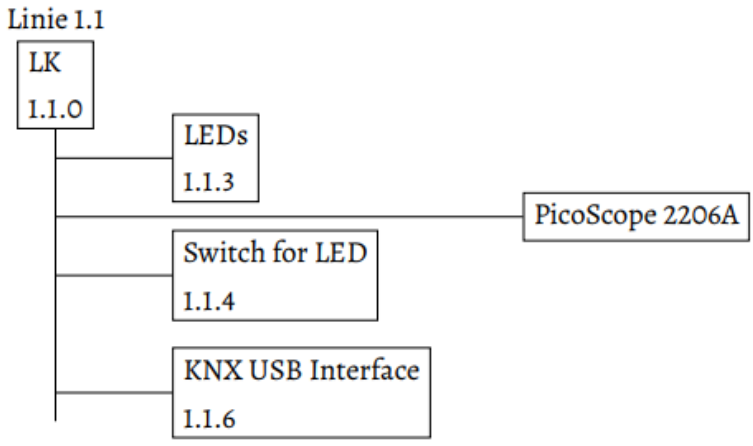


Figure 3.2: Test setup of KNX data collection [36]

3.3 Methodology, data analysis, and feature selection

The objective of this thesis is to create a CNN model and train it with the available time series data from different devices in order to classify the devices. The time series data only contains time and voltage output at that specific time. The data available are not directly usable to use by the CNN algorithm since CNN requires images, therefore, the data has to be processed before supplying to the model to train it. Hence, the first requirement is to convert our time series data into images. There are different ways to convert time series data into images, in this thesis at least two different ways will be executed to transform the data into images.

First technique
In the first procedure, the time series data will be shifted and split by a threshold value. It will then form a numbered array where the voltage value will be considered as the grayscale value of images. This is how the image will be produced from the time series data to train the model.

Second technique
In the second procedure, a graph of time vs voltage change will be plotted from the time series data using the `python matplotlib` library. Later on, this image of the graph will be used to train the model.

The details of that first technique will be discussed in this chapter while the second technique will be reviewed in the following chapter. In this section, the collected data will be analyzed and the detailed operation of the first method will be studied.

3.3.1 Devices and data

As mentioned before the collected data from the KNX bus was stored in CSV format and a typical name looks like `3_8_1_3_8_11`, where `3_8_1_3_8_11` is a device identification number and -1 is the measurement number which increases with each measurement. Therefore, the number `3_8_1_3_8_1500` stands for, device `3_8_1_3_8_1` and measurement number `500`. I have split the data into different folders according to the device name. There was a total of `3000` files that belongs to `6` different devices and each device has `500` measurements. For simplicity purposes, I have named the folder `Device_1:` , `Device_2`,..... `Device_6`. In the following the folder name which contains the specific device measurements is given:

`Device_1:` `3_8_1_3_8_1`
`Device_2:` `3_8_2_3_8_2`

```
Device_3:   3_8_5_3_8_5
Device_4:   3_8_6_3_8_6
Device_5:   3_8_7_3_8_7
Device_6:   3_8_8_3_8_8
```

3.3.2 Tools, programming language, and libraries

To analyze and visualize the data Jupyter notebook has been used originally. Python 3 was used as the programming language and the important libraries used to carry out the tests are Pandas, Matplotlib, NumPy, Scikit-learn, OpenCV, and TensorFlow.

3.3.3 Exploring the data

Each CSV file contains two columns, which are Time μs and Voltage (V), and has a total of 7168 rows. If I read the file with pandas `read_csv` function, it looks like the figure 3.3
.

The time is regularly spaced by 0.5 μs. If I plot a figure with the given time and voltage data, then it will look like the figure 3.4. On the X axis, it shows time in μs, and on the Y axis, it has the voltage in V.

3.3.4 Time series to image generation process

The objective is to make an image out of the time series data which can be used to train a CNN model that can distinguish different images and outputs the probability from which device the data came. The idea is to use a trigger or threshold value so that all the waveform starts at the same time. A way has to be found to split all the waveform and stack it vertically so that all waveform starts at the same position. Each waveform will be called a bit. Each bit is made of many voltage values. When the trigger value split the waveform and outputs voltage values that belong to a bit, it will form an array of numbers that will correspond to a grayscale value of an image. Then this image will be used to train a CNN model. Figure 3.5 illustrates a possible way of how the individual waveform can be split in a such way that the associated voltage values can be arranged in a 2D matrix format. The voltage values shown in the figure are arbitrary.

```
df = pd.read_csv('3_8_1_3_8_1-1.csv')
df.columns = ['time', 'voltage']
df
```

	time	voltage
0	0.0	32.965541
1	0.5	32.914965
2	1.0	32.904850
3	1.5	32.864389
4	2.0	32.854273
...	...	...
7163	3581.5	21.171181
7164	3582.0	21.161065
7165	3582.5	21.171181
7166	3583.0	21.201526
7167	3583.5	21.161065

7168 rows × 2 columns

Figure 3.3: A typical time series data

At this point, we will learn briefly what is an image and how a computer sees an image. An image is just a collection of pixels which are basically some tiny rectangular box and the smallest unit of an image. In an image, all pixels are arranged in a grid format. Each pixel represents the amount of gray intensity to be displayed for a specific location of the image. For images, pixel values are integers that have 256 possible gray intensity values, starting from 0 being black to 255 being white. Figure 3.6 shows what an image actually looks like and an image-to-pixel representation.

Now if we take a look at figure 3.5, the voltage values corresponding to a bit are placed in a row while the bits are in a column. The time of each bit starts at the same position. Now if we look at the values only as shown in the figure 3.7, it is apparent it is an array of numbers or a 2D matrix where we can assume each block of numbers are pixels which represents the gray intensity of an image. This is as same as the pixel representation of figure 3.6. The resulting image would be mostly black to a little grayish image since most of the values are near 0. Following this method, it is feasible to make an image out of the time series data. In this thesis, this process has been implemented to generate the image out of the time series data stored in

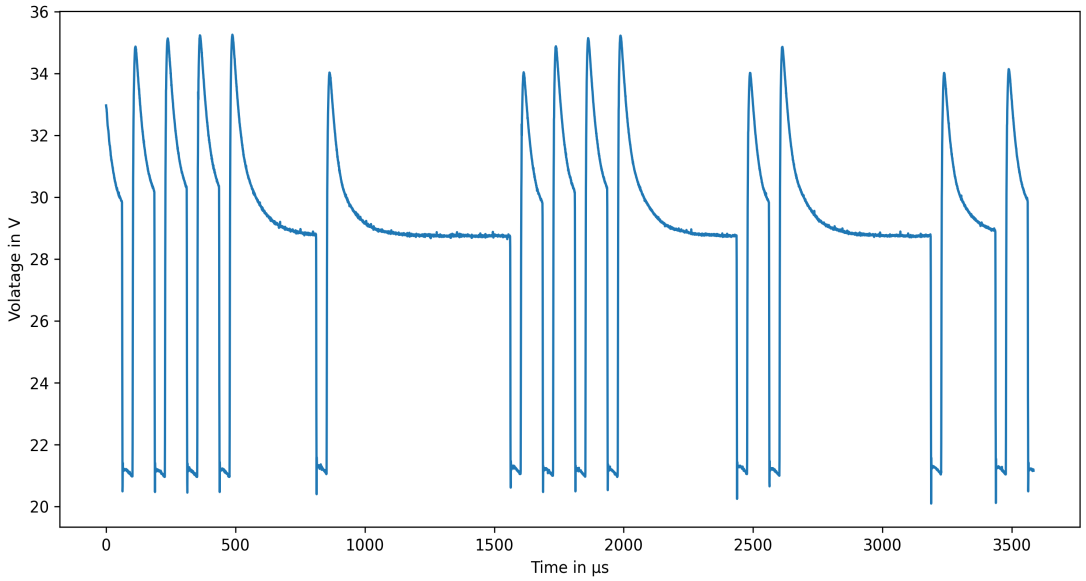


Figure 3.4: Time vs voltage plot

CSV files.

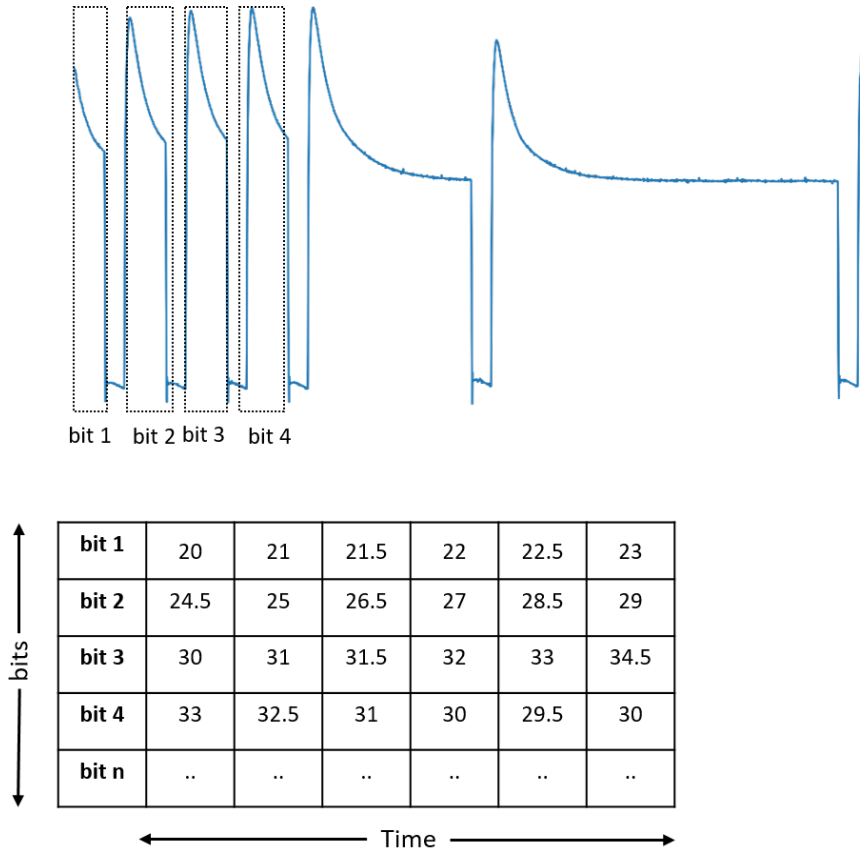
3.3.5 Shifting the waveform/Triggering

As has been discussed already, to generate an image the first step is to split the waveform and shift them to the left so that all waveform starts at the same time. The goal is to use a trigger value or threshold value. When the waveform goes above the threshold value, it will end there and will be counted as a bit. The next waveform will start from the initial position of the time and keep on going as long as it does not hit the threshold value and then it continues until the end. I have written a function in python to perform this process. At first, I calculated the row-to-row voltage difference using the pandas `diff()` function and then plotted a figure with voltage and the calculated time difference which gives me the figure 3.8

Now to identify the peak start I have tested with the threshold value of 2. That means, if the row-to-row voltage difference is more than 2 then it will be considered a new waveform. The function written to shift the waveform is shown in 3.1

```
import pandas as pd
df = pd.read_csv('3_8_1_3_8_1-1.csv')
df.columns = ['time', 'voltage']
threshold = df['voltage'].diff().gt(2)
group = (threshold&~threshold.shift(fill_value=False)).cumsum().add(1)
time = lambda i: i['time'].groupby(group).apply(lambda j: j - j.iloc[0])
df_2 = (df.assign(bit=group,time=time))
df_2
```

Listing 3.1: A python function to split and shift the waveform



bit 1	20	21	21.5	22	22.5	23
bit 2	24.5	25	26.5	27	28.5	29
bit 3	30	31	31.5	32	33	34.5
bit 4	33	32.5	31	30	29.5	30
bit n	..	..	..	..	..	..

Figure 3.5: A possible way to convert the waveform to an array of numbers

In the function, at first, it checks if the row-to-row voltage difference is more than 2, which yields `False` or `True`, then do a cumulative sum where `False` is zero `True` is 1. To put it differently, the cumulative sum of (True, False, False, True, False, True, False) would be (1,1,1,2,2,3,3) which indicates, that all the False following by a True will be in the same group. this value will be placed in a new row corresponding to the voltage value.
To calculate the time, I have used pandas `gourpby()` and used the previously calculated 'group' to categorize according to the 'group' value. Each group is a new waveform and has a starting time. Now in each group, If I subtract the continuous time from the group start time then every time a new waveform starts, the calculated time will start from zero which will put the beginning of all the waveforms at the same place. The executed function gives the output as shown in figure 3.9a

If I do a pivot of `df_2` as shown in 3.2

```
1 df_2 = (df.assign(bit=group,time=time).pivot(index='bit', columns='time',
      values='voltage'))
2 df_2
```

Listing 3.2: Pivot of df_2

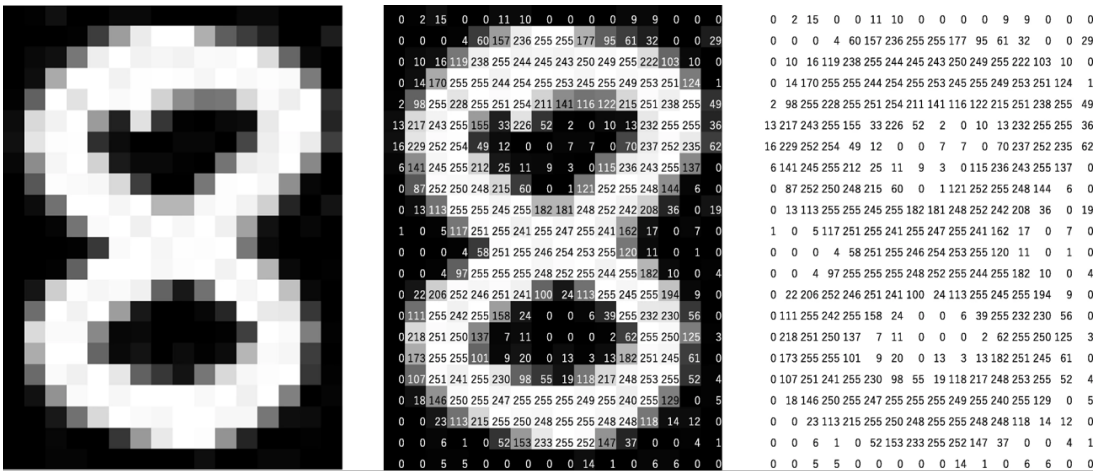


Figure 3.6: An example of an image to pixel representation [24]

.

20	21	21.5	22	22.5	23
24.5	25	26.5	27	28.5	29
30	31	31.5	32	33	34.5
33	32.5	31	30	29.5	30
..	..	..	..	..	..

Figure 3.7: An instance of 2D array of voltage values representing a gray image

then it gives the following output shown in figure 3.9b. We finally have a split waveform where all waveform starts at the same position. This array of numbers is the 2d matrix which can represent a gray image. To inspect whether the waveforms are starting from the same position, if I do a transpose of `df_2` and then if I plot a figure with the given data the resulting plot would be seen in the figure 3.10

It is evident from the figure that all the waveforms are starting from time 0.
From figure 3.9b it can be seen the threshold value has identified a total of 14 bits and at least one bit has a maximum count of voltage values of 1499. To visualize a gray image out of this real data I saved the values to a CSV file and used Microsoft excel Conditional Formatting, and Color Scales. The outcome can be seen in figure 3.11

If the different device outputs different voltage readings that will produce distinguishable images and a CNN model should be able to differentiate among the images. If some image which is representing the time series data, does not belong to any devices, then it could be an anomaly.

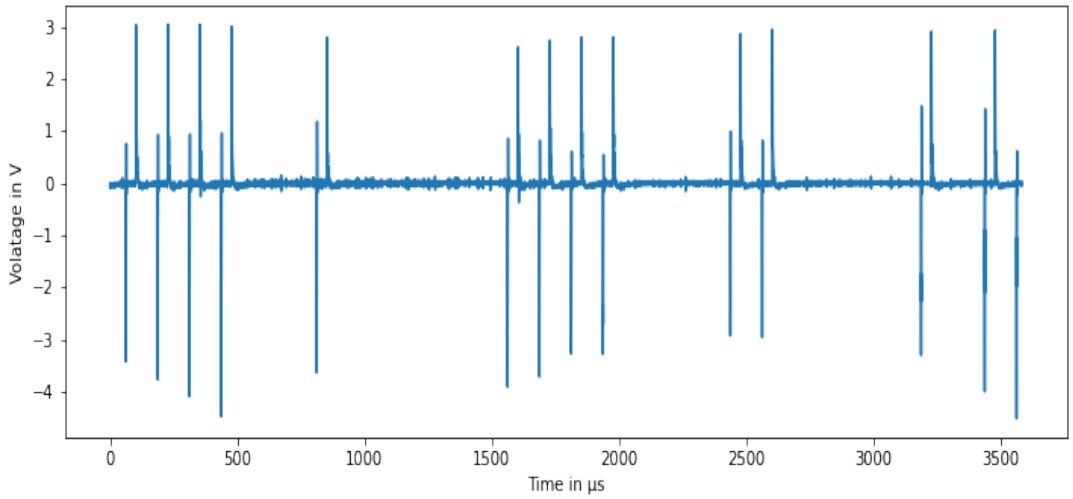


Figure 3.8: A plot with time and row-to-row voltage difference

	time	voltage	bit
0	0.0	32.965541	1
1	0.5	32.914965	1
2	1.0	32.904850	1
3	1.5	32.864389	1
4	2.0	32.854273	1
...	...	...	...
7163	104.0	21.171181	14
7164	104.5	21.161065	14
7165	105.0	21.171181	14
7166	105.5	21.201526	14
7167	106.0	21.161065	14

7168 rows × 3 columns

(a) A new dataframe df_2 with time and bit value

time	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5
bit										
1	32.965541	32.914965	32.904850	32.864389	32.854273	32.763236	32.743006	32.651969	32.581162	32.510356
2	25.045314	27.543777	29.182444	30.588462	31.114454	31.984364	32.510356	32.793582	33.299344	33.511764
3	25.166697	27.746081	29.415095	30.719960	31.326873	32.125977	32.530586	33.440957	33.511764	33.865797
4	25.277965	27.877579	29.536477	30.912149	31.367334	32.206899	33.410611	33.157730	33.774760	33.926488
5	25.379117	27.978732	29.667975	30.780651	31.670791	32.338397	32.844158	33.329689	33.744414	34.027640
6	25.631998	27.634814	28.959909	30.173737	30.659268	31.053762	31.630330	32.125977	32.368742	32.692430
7	23.528030	26.137759	27.948386	29.253251	30.244544	30.649153	31.428026	32.368742	32.004594	32.550817
8	23.639297	26.380525	28.464263	29.971432	30.902034	31.458371	32.520471	32.581162	33.117270	33.400496
9	23.740449	26.542369	28.707028	30.295120	30.881803	31.862981	32.368742	32.884619	33.420727	33.734299
10	23.871948	26.673867	28.889103	30.305235	31.185260	31.873096	32.803697	32.965541	33.562340	33.845566
11	24.387824	26.694097	28.342880	29.678091	30.315350	31.134684	31.417911	32.095631	32.156323	32.500240
12	24.499092	27.078476	28.909333	30.295120	31.003186	31.701137	32.247360	32.803697	33.177961	33.531994
13	24.964393	27.149283	28.626107	29.981548	30.365926	31.114454	31.721368	31.984364	32.378858	32.571047
14	25.197043	27.371818	28.828411	30.092815	30.659268	31.286413	31.761828	32.318166	32.378858	32.793582

14 rows × 1499 columns

(b) Dataframe df_2 after pivot

Figure 3.9: A conversion of a waveform to an image bitmap using real values

3.3.6 Automating the process using a loop

In the earlier section, a waveform of one time-series data has been split and shifted. As it has been mentioned before there is a total of 6 devices and each device has 500 measurements. Consequently, all the measurement from all devices has to go through the same process so that the waveform starts at the same place and can be used to train a CNN model. To run this test, I selected the first 100 measurements instead of all 500 from each device and placed them in a different folder with their respective name. I have written a function which is basically a `for` loop which will find out all the files, inside all the folders and run the function to split and shift the waveform. The function is shown in the listing 3.3.

```
1 import pathlib
```

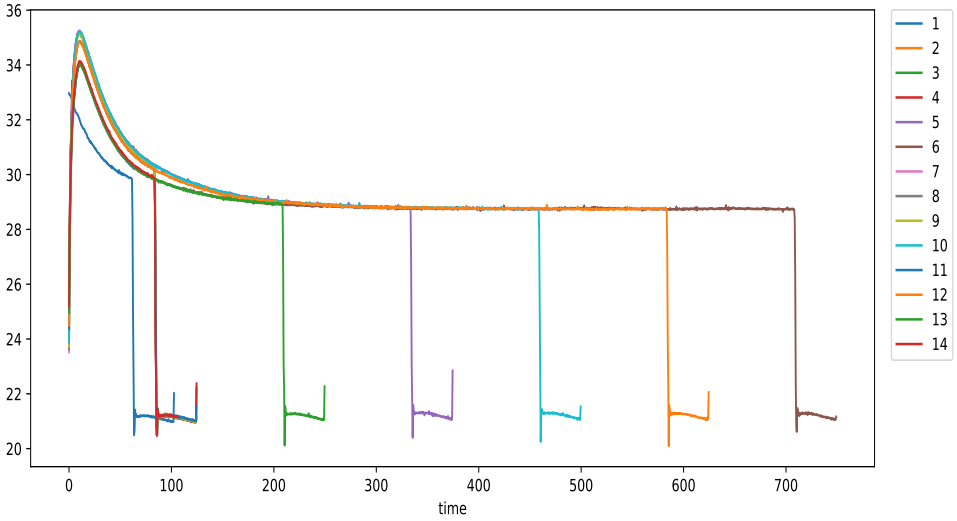


Figure 3.10: Plot after split and shift of the waveform. All split waveform starts from time 0

32.9655	32.915	32.9048	32.8644	32.8543	32.7632	32.743	32.652	32.5812	32.5104	32.4294	32.389	32.3182	32.2878
25.0453	27.5438	29.1824	30.5885	31.1145	31.9844	32.5104	32.7936	33.2993	33.5118	33.7545	33.9467	34.149	34.321
25.1667	27.7461	29.4151	30.72	31.3269	32.126	32.5306	33.441	33.5118	33.8658	34.0984	34.2805	34.4929	34.6649
25.278	27.8776	29.5365	30.9121	31.3673	32.2069	33.4106	33.1577	33.7748	33.9265	34.2299	34.4221	34.6244	34.7559
25.3791	27.9787	29.668	30.7807	31.6708	32.3384	32.8442	33.3297	33.7444	34.0276	34.2401	34.4525	34.6548	34.7863
25.632	27.6348	28.9599	30.1737	30.6593	31.0538	31.6303	32.126	32.3687	32.6924	32.8644	33.0768	33.2589	33.4106
	26.1378	27.9484	29.2533	30.2445	30.6492	31.428	32.3687	32.0046	32.5508	32.7025	32.915	33.097	33.3095
	26.3805	28.4643	29.9714	30.902	31.4584	32.5205	32.5812	33.1173	33.4005	33.6736	33.8253	34.0883	34.2603
	26.5424	28.707	30.2951	30.8818	31.863	32.3687	32.8846	33.4207	33.7343	33.9771	34.149	34.3918	34.5941
	26.6739	28.8891	30.3052	31.1853	31.8731	32.8037	32.9655	33.5623	33.8456	34.0883	34.321	34.5031	34.7155
24.3878	26.6941	28.3429	29.6781	30.3154	31.1347	31.4179	32.0956	32.1563	32.5002	32.7835	32.9655	33.1476	33.3297
24.4991	27.0785	28.9093	30.2951	31.0032	31.7011	32.2474	32.8037	33.178	33.532	33.6938	33.9366	34.0984	34.3008
24.9644	27.1493	28.6261	29.9815	30.3659	31.1145	31.7214	31.9844	32.3789	32.571	32.8037	33.006	33.178	33.3398
25.197	27.3718	28.8284	30.0928	30.6593	31.2864	31.7618	32.3182	32.3789	32.7936	32.9858	33.1577	33.3398	33.532

Figure 3.11: A gray image generated from a real-time series data

```
data_dir = (r'E:\Rostock-Masters\Thesis\Test')
data_dir = pathlib.Path(data_dir)

file_name_dict = {
    'Device_1' : list(data_dir.glob('Device_1/*.csv')),
    'Device_2' : list(data_dir.glob('Device_2/*.csv')),
    'Device_3' : list(data_dir.glob('Device_3/*.csv')),
    'Device_4' : list(data_dir.glob('Device_4/*.csv')),
    'Device_5' : list(data_dir.glob('Device_5/*.csv')),
    'Device_6' : list(data_dir.glob('Device_6/*.csv'))
}
df_lst = []
df_compressed = []

for file in file_name_dict['Device_1']:
    file_df = pd.read_csv(str(file))
```

```
18    file_df.columns = ['time', 'voltage']
19
20        threshold = file_df['voltage'].diff().gt(2)
21        group = (threshold&~threshold.shift(fill_value=False)).cumsum().add(1)
22        time= lambda i: i['time'].groupby(group).apply(lambda j: j- j.iloc[0])
23        df_2 = (file_df.assign(bit=group,time=time).pivot(index='bit',
              columns='time', values='voltage'))
24
25    df_3 = df_2.copy()
26    df_3.reset_index(drop=True, inplace=True)
27    df_3 = df_3.rename_axis(None, axis=1)
28    df_lst.append(df_3)
29
30    df_4=df_3.iloc[:14,:100]
31    df_compressed.append(df_4)
```

Listing 3.3: A python function to automate the split and shift of waveform of all measurement

Before running this function, I explored a few measurements and noticed a few things. As it is shown in figure 3.9b the split and shift function recognized a total of 14 bits and at least one bit has a maximum count of 1499 voltage values. I noticed the threshold value used in the function identifies a different number of bits and the number of voltage value counts belonging to a specific bit are different. In another word, different measurement outputs a different number of bits and voltage count belonging to a bit. Hence, there are a lot of `Null` values. For that reason, I have decided to use the first 14 bits and the first 100 voltage counts for each bit, in other words, the first 14 rows and the first 100 respective columns that will make a 14 x 100-pixel image.

In the function, in addition to other libraries used previously, I have used the `Pathlib` library and `glob` function. The Pathlib library works with operating system paths and `glob` returns all the file paths that match a specific file pattern. It can search specific file types in a given path. `file_name_dict` is a python dictionary where each device name and their corresponding files are listed using `glob` . It checks each folder in the given path and lists all the files which have `.csv` extension. I decided to save each measurement after executing the function to a CSV file for later use. Therefore, I created two empty lists `df_lst = []`, `df_compressed = []` where the first one store the measurement after the split and shift function and the latter one stores measurements after minimizing them to 14 x 100 I.e. 14 rows and 100 columns.

The loop will go through all the devices stored in `file_name_dict`, however, since I want to save each measurement to a CSV file, in the function shown in 3.3, I selected the specific device

`Device_1`. Later, I removed the index row and header column from the resulting dataframe since they will be of no use to making an image. Figure 3.12 shows how the data of one measurement looks after running the function. This is the final data or array of numbers, which will be used in CNN for training.

```
1 df_compressed[99]
```

	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	...	45.0	45.5
0	30.649153	30.689614	30.679499	30.659268	30.659268	30.639038	30.628922	30.568231	30.578346	30.507540	...	21.201526	21.181296
1	23.386416	26.157990	28.352995	29.748897	30.851458	31.347104	32.186668	32.844158	32.985772	33.400496	...	31.610100	31.620215
2	23.487569	26.309718	28.575530	30.193967	31.033532	32.227129	32.368742	33.117270	33.410611	33.734299	...	31.842750	31.792174
3	23.639297	26.441216	28.727259	30.123161	31.073993	31.984364	32.712660	32.955426	33.471303	33.896142	...	31.893326	31.852866
4	23.730334	26.572714	28.787950	30.365926	31.134684	32.105746	32.368742	33.086924	33.501648	33.896142	...	31.903442	31.842750
5	24.195635	26.562599	28.241728	29.506132	30.386157	30.730075	31.428026	32.156322	32.004594	32.500240	...	31.073993	31.033532
6	24.762088	27.017785	28.535069	29.880395	30.436733	30.952610	31.731483	32.065285	32.146207	32.631738	...	31.104338	31.043647
7	24.903701	27.422394	29.101522	30.497424	30.942495	31.832635	32.065285	32.965541	33.208307	33.511764	...	31.670791	31.529178
8	25.035199	27.604468	29.324058	30.679499	31.245952	32.166438	32.844158	33.097039	33.592685	33.855681	...	31.812405	31.741598
9	25.146467	27.746081	29.455556	30.871688	31.549409	32.045055	32.995887	33.289228	33.683723	33.926488	...	31.893326	31.822520
10	25.470154	27.543777	28.909333	30.163622	30.568231	30.922264	31.822520	31.974248	32.338397	32.692430	...	31.053762	31.043647
11	25.672459	28.009077	29.485901	30.760420	31.215606	31.903442	32.611508	32.834043	33.420726	33.562340	...	31.610100	31.599985
12	23.416762	25.955685	27.877579	29.202675	30.264774	30.790766	31.144799	32.358627	31.954018	32.469895	...	31.084108	31.003186
13	23.639297	26.228797	28.069769	29.404979	30.477194	30.730075	31.468487	32.247360	32.146207	32.662084	...	31.165030	31.134684

14 rows × 100 columns

Figure 3.12: A figure of the time-series data after going through the loop

Chapter 4

Implementations

In this chapter

The implementation of the prototype is discussed in this chapter. It includes some preprocessing of the data, labeling the data, training of the CNN model, different test run, and their results. All tests have been run initially on a local computer on Jupyter notebook.

4.1 Imbalanced dataset

In the last chapter, it has been discussed that all the measurements will have the first 14 x 100 rows and columns respectively which will make a 14x100 pixel image. The purpose was to maintain the same number of bits and voltage counts per bit to produce the same size image for all measurements. However, after the loop was executed on all the data, I uncovered some issues with the resultant shape of the data. I have discovered a lot of measurement from

different devices does not have at least 100 counts of voltage values for each bit. I have written a function to have an overview of all the files from different devices. This function applies the split and shift function on each file which actually represents the time series data and then checks each file/measurement for the minimum and maximum voltage count for each bit and shows the output for each file. If there is any file that has at least one bit among the first 14 bits which does not have a minimum 100 voltage value count it shows the index number of that bit. I have run the function individually for each device's measurements and found out all the measurement of Device_1 and Device_2 has at least 14 rows and 100 columns each with voltage values while the other device has many measurements which do not have at least 14 rows and 100 columns each with voltage values. Figure 4.1 illustrates the issues that have been encountered. In the figure, we can see, that the index number 2,5,9,11, and 13 only has 2 voltage count each. This is only one measurement's output while there are a lot of other files which has more or less similar issues. Therefore, primarily, I have chosen to train and test the classification task with Device_1 and Device_2 data only.

	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5
0	30.548	30.5177	30.5379	30.548	30.5278	30.5075	30.5075	30.4671	30.4671	30.3558	30.3558
1	27.5741	28.2316	30.113	30.7807	31.0538	32.1057	32.652	32.8947	33.2285	33.4308	33.6331
2	27.6955	28.3833									
3	30.4064	30.9121	31.7719	32.8543	32.7127	33.2387	33.4511	33.7242	33.9063	34.0984	34.2704
4	27.7663	28.444	30.3862	30.9728	31.9945	32.2878	32.9858	33.2083	33.623	33.795	34.0681
5	27.7865	28.3631									
6	30.4974	30.9526	31.7416	32.5205	32.652	33.4207	33.5623	33.8658	33.9973	34.2502	34.3918
7	27.1493	27.7157	29.3949	30.2041	30.4873	30.983	31.4887	31.9136	32.2575	32.4395	32.6722
8	27.0582	27.5438	29.405	30.0524	31.428	31.1853	31.3977	31.9034	32.2575	32.48	32.6419
9	27.4426	27.9686									
10	29.9815	30.639	31.3168	31.9742	32.1664	32.9554	33.097	33.3803	33.5623	33.7646	33.9164
11	27.5741	28.1709									
12	30.2445	30.7807	31.5292	31.8832	32.7734	33.1577	33.5118	33.6736	33.8658	34.0984	34.2198
13	27.5741	28.1203									

Figure 4.1: An example of the imbalanced dataset

Implementations

4.2 Classification of 2 devices

4.2.1 Labelling and preprocessing

A CNN model will be trained with Device_1 and Device_2 data and it will be tested to check the accuracy of the model. The complete coding with processing and labeling of the data is shown in the listing 4.1. The first part of the code is the same that has already been shown in the listing 3.3 except for `file_label_dict_2` which is the label of the device. At the end of the classification task CNN model will predict the probability of incoming data belonging to

which class/device. Where 0 is `Device_1` and 1 is `Device_2`. As it is shown in the coding at the end of the loop, the device label has been appended with the respective device name.
The data and the label were converted to a `numpy` array afterward. From `sklearn` library `model_selection` package has been used to split the training and testing set. I used 30% of the data for testing. The `random_state=0` denotes the training set and the testing set of the data remains the same across all execution. In the model `keras Conv2D` has been used, which is shown in listing 4.2 and `Conv2D` expects to have 3 dimensions which are width, height, and color channel. Since I am working with the grayscale image I have added one more dimension to each image. If it was a color image, then a 3 dimension(RGB) needs to be added. Using `numpy reshape` function that 1 dimension has been added. The function `numpy.reshape` gives a new shape to an array without changing its data. Finally, the data was scaled down from 0 to 1. Scaling is an important preprocessing step. The entire data is scaled down to a fixed range to normalize the data so that it is easier for the model to learn and the model is not biased. Now all the data are preprocessed and ready to be used for training.

```
import pandas as pd
import numpy as np
import pathlib
from sklearn.model_selection import train_test_split

data_dir = (r'E:\Rostock-Masters\Thesis\Test')
data_dir = pathlib.Path(data_dir)

file_name_dict_2 = {
    'Device_1' : list(data_dir.glob('Device_1/*.csv')),
    'Device_2' : list(data_dir.glob('Device_2/*.csv'))
}
file_label_dict_2 = {
    'Device_1' : 0,
    'Device_2' : 1
}
device_list, device_label = [],[]
for device_name, folder in file_name_dict_2.items():
    for file in folder:
        file_df = pd.read_csv(str(file))
        file_df.columns = ['time', 'voltage']

      #Applying split and shift function
        threshold = file_df['voltage'].diff().gt(2)
        group = (threshold&~threshold.shift(fill_value=False)).cumsum().add(1)
        time= lambda i: i['time'].groupby(group).apply(lambda j: j- j.iloc[0])
        df_2 = (file_df.assign(bit=group,time=time).pivot(index='bit',
            columns='time', values='voltage'))
        df_3 = df_2.copy()
```

```
        df_3.reset_index(drop=True, inplace=True)
        df_3 = df_3.rename_axis(None, axis=1)

        #Compressing to 14 rows 100 columns
        df_4=df_3.iloc[:14, :100]
        device_list.append(df_4)
        device_label.append(file_label_dict_2[device_name])

#Converting to numpy array
X = np.array(device_list)
y = np.array(device_label)

#Train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
    0.30,random_state=0)

#Adding 1 dimension
X_train = X_train.reshape(X_train.shape[0],14,100,1)
X_test = X_test.reshape(X_test.shape[0],14,100,1)

#scaling data from 0 to 1
X_train_scaled = X_train/36 #Max voltage value 36
X_test_scaled = X_test/36
```

Listing 4.1: A python code to process the data for training

4.2.2 Training and testing

To train the model, `keras` from `tensorflow` has been used.TensorFlow is a library developed by google for deep learning applications. The details working procedure of CNN, filter, kernel size, stride, padding, activation layer, pooling, and dense layer have been explained in section Convolutional Neural Network (CNN). The model is shown in the listing 4.2. Initially, I used 8 filters with 3x3 size, strides of 1: defines the step size of the filter when sliding over the image, padding is kept as same: represents 1 layer of zero will be added all around the image so the output size does not shrink but stays the same as the input, max pooling operation with pool size 2x2 and activation layer is relu which is widely used activation layer during convolution.

The selection of the number of filters is arbitrary. In every layer, I doubled the number of filters. Since every convolutional layer is followed by a max pooling layer, after the third max pooling layer the output shape was reduced to 1x 12 x32. Therefore, it is not possible to add

more max pooling layers. Later, the layer was flattened and the dense layer started with activation function relu. Since we are classifying between two devices, the last layer will provide two outputs with the activation layer softmax, another widely used activation function in the final layer.

At which position which activation layer should be used has been shown in the table 2.1. The summary of the model is displayed in 4.2. It shows the shape of the output after each layer is executed.

```
1 model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 14, 100, 3)        30
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 50, 3)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 50, 6)          168
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 25, 6)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 25, 12)         660
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 12, 12)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 144)               0
_________________________________________________________________
dense_3 (Dense)              (None, 72)                10440
_________________________________________________________________
dense_4 (Dense)              (None, 36)                2628
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 74
=================================================================
Total params: 14,000
Trainable params: 14,000
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.2: A summary of the CNN model

```
#Buiding a model
import tensorflow as tf
model_2 = tf.keras.Sequential([

    tf.keras.layers.Conv2D(filters=8, kernel_size=3, strides=1,
        padding="same",
        activation="relu",input_shape=(X_train_scaled[0].shape)),
    tf.keras.layers.MaxPool2D(pool_size=2),
    tf.keras.layers.Conv2D(16,3, padding="same", activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2),
    tf.keras.layers.Conv2D(32,3, padding="same", activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2),

    tf.keras.layers.Flatten(),
```

```
    tf.keras.layers.Dense(72, activation='relu'),
    tf.keras.layers.Dense(36, activation='relu'),
    tf.keras.layers.Dense(2, activation= 'softmax') #Output layer
])
#Training the model
from tensorflow import keras
opt = keras.optimizers.Adam(lr=0.001)
model_2.compile(optimizer=opt,
             loss='sparse_categorical_crossentropy',
               metrics= ['accuracy'])

history= model_2.fit(X_train_scaled, y_train, epochs=250,
    batch_size=32,validation_split=0.20, verbose=1, shuffle= True)
    #validation_split=0.33
history

#Testing the model
loss, accuracy= model_2.evaluate(X_test_scaled, y_test)
print(f'Loss: {loss}, Accuracy: {accuracy}')
```

Listing 4.2: Training and testing the CNN model

Now the model needs to be compiled for training. I have used `loss='sparse_categorical_crossentropy'` since our data is not one hot-encoded rather the output is in integer format. For training, I have chosen `epochs=200`, which is arbitrary and basically refer to the number of iteration. The model is now trained and ready to be tested. 50% of the data was used for training, 20% was used for validation, and the rest 30% was kept for testing. To evaluate the model, those test data were provided to the model to make predictions and for this specific test run, I got **88.33%** accuracy. Figure 4.3 Shows the training and validation loss and accuracy of this specific run.

4.3 Classification of 6 devices: Method-1

For the classification task of the 6 devices, I have implemented a total of four methods. This is the first method. As it has been mentioned earlier due to some imbalanced dataset in Device_2 - Device_6, it has been decided to run two device classifications first. In this subsection, the first method of classification of the 6 devices will be observed.

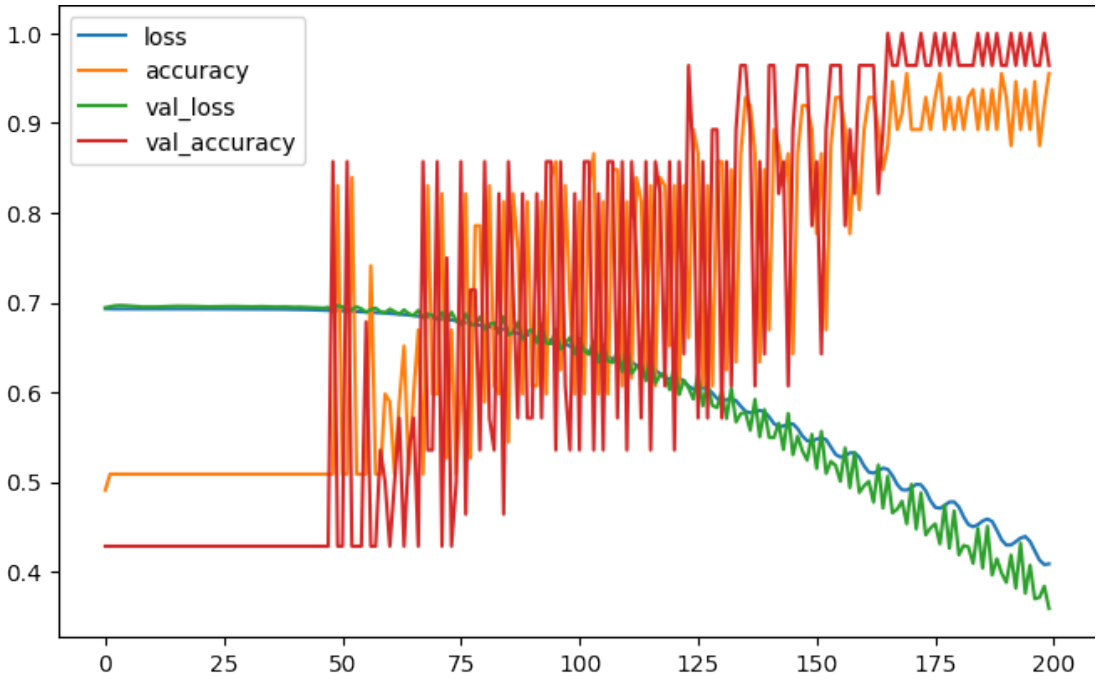


Figure 4.3: Plot of accuracy and loss on training and validation set for 2 device classifications. 88.33% accuracy

4.3.1 Preprocessing

The preprocessing steps are the same as the earlier ones. To overcome the imbalanced dataset, I decided to replace the voltage values with moving average values with a sliding window of 10. The moving average function takes the average of the first 10 values and replaces it with the first value and then slides down one step, takes the average of the next 10 values, replace it with the second value and that is how it slides over the entire dataset and replace the original value with the moving average values. This is how we removed some noise from the signal and smoothed the curves a little. From figure 4.4 and figure 4.5 the difference can be observed. With the normal voltage, we have varying voltage differences in the different devices but after replacing the values with moving average values the voltage differences across all the measurements came down to the -1.0 to 1.0 range.

A function has been written to check whether there are any incomplete datasets. After running the function, I discovered the number of incomplete datasets was reduced but not completely gone. To resolve this issue, I determined to fill up or pad those empty cells with the values from the cell above. After the padding, there were finally no incomplete datasets remaining. During the test of 2 device classification, the selected image size was 14 x 100 pixels but for this 6-device classification, I needed to reduce the image size to 10 x 100 pixels. To comprehend the reason if we take a look at figure 4.6 it can be seen that some measurements from Device_5 have less bit count than 14. Therefore, to make all the images the same size I selected a 10 x 100-pixel size. The changes in the coding for this method are shown in the listing 4.3

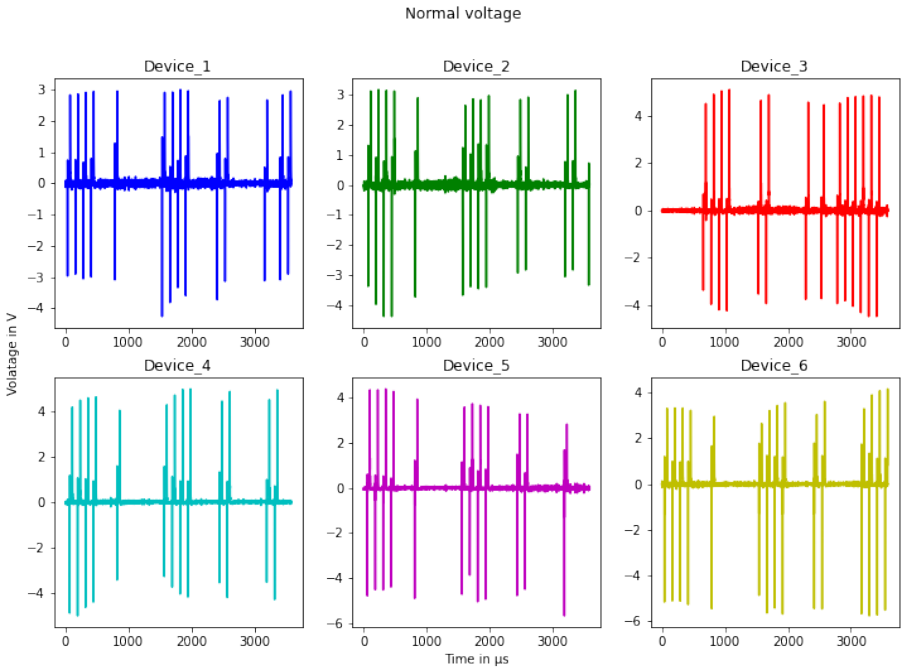


Figure 4.4: A plot of the voltage difference across different devices with original voltage value

```
device_list, device_label = [],[] #compressed

for device_name, folder in file_name_dict.items():
    for file in folder:
        file_df = pd.read_csv(str(file))
        file_df.columns = ['time', 'voltage']

        #Replacing originial values with moving average values
        file_df['MA'] = file_df['voltage'].rolling(10,min_periods=0).mean()#
            moving average
        file_df= file_df.drop('voltage',axis=1)
        file_df.rename(columns={'MA':'voltage'},inplace=True)

        #Split and shift function
        threshold = file_df['voltage'].diff().gt(1)
        group = (threshold&~threshold.shift(fill_value=False)).cumsum().add(1)
        time= lambda i: i['time'].groupby(group).apply(lambda j: j- j.iloc[0])
        df_2 = (file_df.assign(bit=group,time=time).pivot(index='bit',
            columns='time', values='voltage'))
        df_3 = df_2.copy()
        df_3.reset_index(drop=True, inplace=True)
        df_3 = df_3.rename_axis(None, axis=1)

        #Compressing to 10 rows 100 columns
        df_4=df_3.iloc[:10, :100]
```

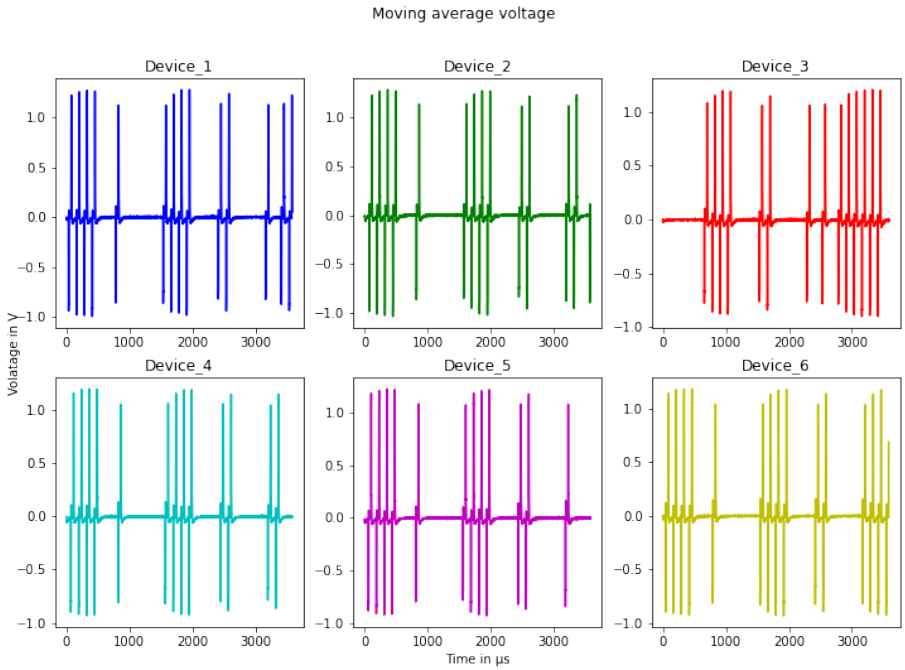


Figure 4.5: A plot of the voltage difference across different devices with moving average voltage value

```
24
25          #Filling out empty cells
26          df_4= df_4.fillna(method='pad')
27          device_list.append(df_4)
28          device_label.append(file_label_dict[device_name])
```

Listing 4.3: Preprocessing of the 6 device classification

4.3.2 Training and testing

A similar model configuration as the previous one has been used for this 6-device classification. The model summary is shown in figure 4.7. Once the model was trained I tested the model on test data and after many tests run the maximum accuracy I got is **56.11%**. Figure 4.8

shows the accuracy and loss curves on training and validation data. I checked if the model performance could be increased. I tried several different configurations which include increasing and decreasing the number of filters, convolutional layers, dense layers, number of epochs, and batch_size. I have also tried with different optimizers such as Adam, SGD, RMSprop, Adadelta, and Adagrad and tried to tune the learning rate. The model performance was checked after every change in the model configuration. After many tests run the maximum accuracy received from the model is 56.11%.

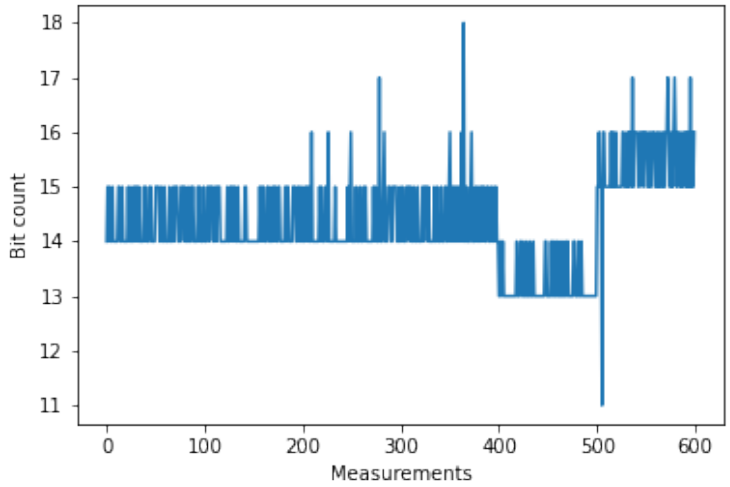


Figure 4.6: A plot of bit count of all the devices

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 10, 100, 8)	80
max_pooling2d (MaxPooling2D)	(None, 5, 50, 8)	0
conv2d_1 (Conv2D)	(None, 5, 50, 16)	1168
max_pooling2d_1 (MaxPooling2	(None, 2, 25, 16)	0
conv2d_2 (Conv2D)	(None, 2, 25, 32)	4640
max_pooling2d_2 (MaxPooling2	(None, 1, 12, 32)	0
flatten (Flatten)	(None, 384)	0
dense (Dense)	(None, 64)	24640
dense_1 (Dense)	(None, 128)	8320
dense_2 (Dense)	(None, 6)	774

Figure 4.7: The model summary of 6 device classifications.

4.4 Classification of 6 devices: Method-1. Different structure.

4.4.1 Preprocessing

This new approach is an extension of the previous one. As has been mentioned earlier that after the split and shift function there were many imbalanced data sets, which is shown in figure 4.1. This new approach starts from that phase. The idea is to append all the rows and all columns together so that it becomes one row and at least 900 columns and there are no empty cells between. In another term, the 10 rows become one row and there are no empty cells and this one row represents one image. The appending will be done row after row. Then the values of this one row will be reshaped to a square-sized image. Thus, we can achieve

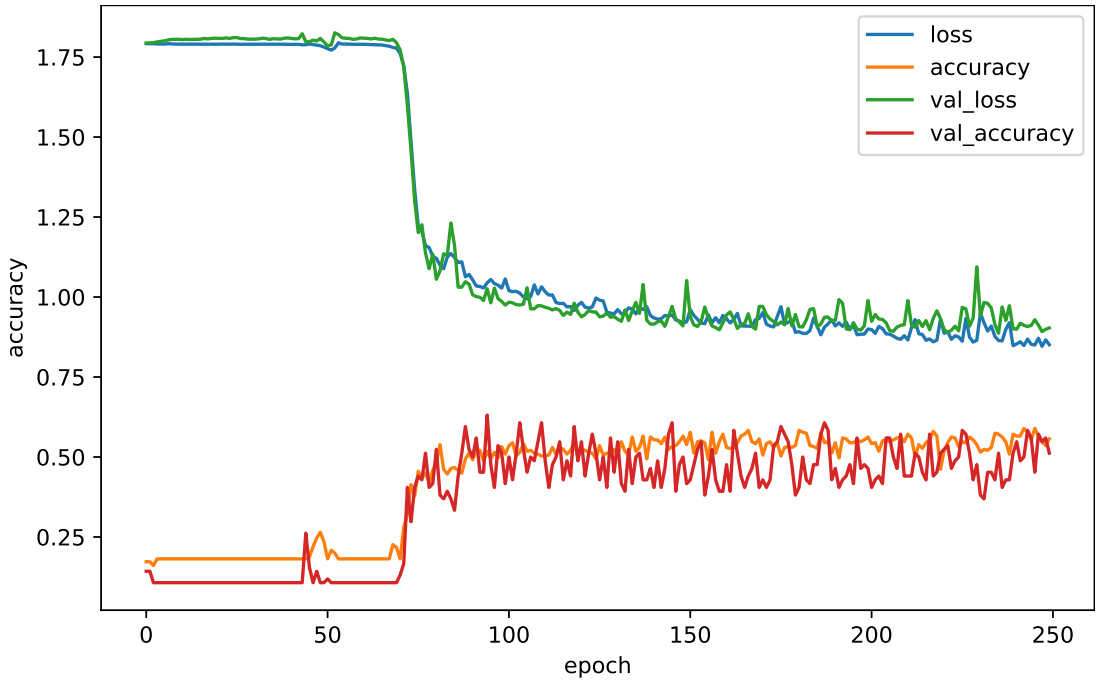


Figure 4.8: Plot of accuracy and loss on training and validation set for 6 device classifications. 56.11% accuracy.

two things: one, there is no imbalanced dataset anymore and, in this way, we do not have pad any value from above or below cell which can confuse the model, and two, we can have a square-shaped image. The standard method for CNN is to take a square-shaped image. It is simpler and easier for the network and also could improve the efficiency of the model.

I decided to take the first 10 rows and 120 columns each from each measurement and then take 900 values out of it so that I can reshape it to a 30x30 pixel. The reason I did not choose 10x90 is that there would be many imbalanced datasets or empty cells which means the number of non-empty cells or real values would not be 900 but less than that. Therefore, I decided to take more columns so that after stacking all the values in one row I will have more than 900 values and out of those values I can simply take the first 900 values and make it a 30x30 pixel image.

Before reshaping a single row into a 30x30 pixel image I will add all 600 measurement values to a single dataframe, add the corresponding label of those measurements with the dataframe and then shuffle the dataframe. In other words, there will be a dataframe that consists of 600 rows and 900 columns where each column represents a single image. In addition to that, the label column would be added which represents the device label for that specific measurement. Shuffling is important for the training process because if a model is trained continuously on the sample of one device or one label then there is a possibility that the model would be biased. Therefore, it is a good practice to shuffle the sample so that there are mixed samples from

different devices instead of training one complete device sample and then another. Figure 4.9 shows 1 row with 900 columns which stand for one image and figure 4.10

```
1 device_list[432]
```

	0	1	2	3	4	5	6	7	8	9	...	890	891	892	
0	31.063877	31.099281	31.080736	31.099281	31.086131	31.095909	31.082663	31.077786	31.065001	31.051739	...	32.354581	32.305016	32.262532	32.212

1 rows × 900 columns

Figure 4.9: 1 row with 900 columns represents one image

	label	0	1	2	3	4	5	6	7	8	...	890	891	892	893
0	4	33.147615	33.112212	33.097039	33.079337	33.064670	33.044777	33.017562	32.994622	32.967789	...	33.205272	33.149638	33.097039	33.043428
1	3	31.063877	31.063877	31.060506	31.061349	31.055785	31.050390	31.042202	31.033532	31.024540	...	32.343454	32.299959	32.255452	32.207910
2	1	33.420726	33.395438	33.366779	33.347391	33.329689	33.307773	33.284893	33.257618	33.230785	...	33.440957	33.392404	33.331712	33.281136
3	1	32.621623	32.611508	32.594649	32.578633	32.560932	32.542387	32.523361	32.495183	32.473266	...	33.421738	33.367116	33.311482	33.256860
4	5	30.891918	30.886861	30.885175	30.876746	30.873711	30.870002	30.860128	30.850193	30.841342	...	32.916988	32.870458	32.819882	32.766271
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
595	1	31.286413	31.276297	31.269554	31.263653	31.262136	31.257753	31.248842	31.239630	31.226845	...	33.762621	33.703953	33.647308	33.587628
596	1	33.006002	32.975656	32.968913	32.947839	32.927103	32.909907	32.884619	32.860596	32.841910	...	33.417692	33.361047	33.312494	33.255848
597	5	32.419319	32.394030	32.365371	32.356098	32.336374	32.316480	32.305161	32.275176	32.255227	...	32.960483	32.902827	32.856297	32.800663
598	5	30.982956	30.982956	30.982956	30.977898	30.976886	30.974526	30.967060	30.962725	30.954858	...	33.260906	33.209318	33.154696	33.100074
599	0	30.780651	30.730075	30.743562	30.737661	30.736144	30.728389	30.718515	30.713638	30.701977	...	32.630727	32.575093	32.526540	32.475964

600 rows × 901 columns

Figure 4.10: Shuffled dataframe with the label. Each row represents one image

shows the complete dataframe with 600 rows and 900 columns with the respective device label. Now the data is ready for the final steps. I scaled the data for normalization and then reshaped it to a 30x30x1 pixel image where 1 represents the color channel. To have an idea how what the image looks like, I plotted the image using `matplotlib` and the figure 4.11

shows a subplot that includes multiple random images. Afterward, the data is split into train and test split same as before where training data is 50%, validation data is 20%, and test data is 30%. This listing 4.4 shows the complete code of processing that has been mentioned above.

```
1  device_list, device_label = [],[] #compressed
2
3  for device_name, folder in file_name_dict.items():
4      for file in folder:
5          file_df = pd.read_csv(str(file))
6          file_df.columns = ['time', 'voltage']
7          file_df['MA'] = file_df['voltage'].rolling(10,min_periods=0).mean()#
               moving average
8          file_df= file_df.drop('voltage',axis=1)
9          file_df.rename(columns={'MA':'voltage'},inplace=True)
10
11         threshold = file_df['voltage'].diff().gt(1)
12         group = (threshold&~threshold.shift(fill_value=False)).cumsum().add(1)
13         time= lambda i: i['time'].groupby(group).apply(lambda j: j- j.iloc[0])
```

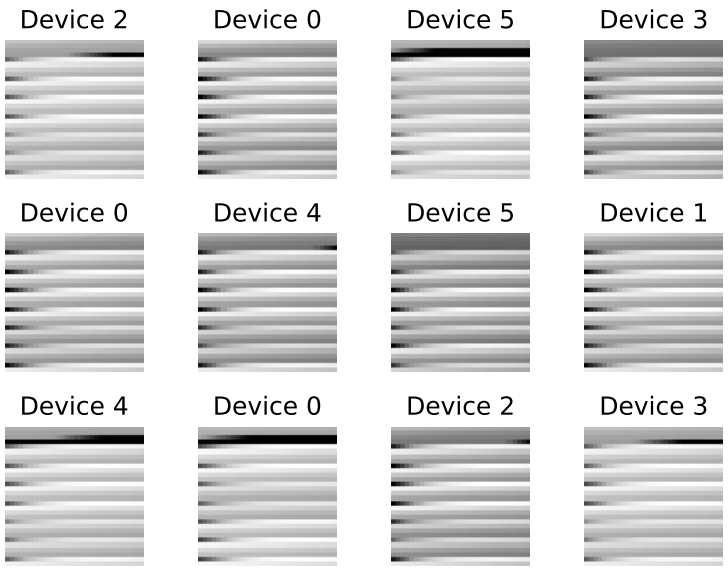


Figure 4.11: A subplot of some random images with their labels after reshaping into 30x30 pixel

```
        df_2 = (file_df.assign(bit=group,time=time).pivot(index='bit',
            columns='time', values='voltage'))
        df_3 = df_2.copy()
        df_3.reset_index(drop=True, inplace=True)
        df_3 = df_3.rename_axis(None, axis=1)

        #Compressing to 10 rows 100 columns. Stacking all the values,
            reshaping into a single row.
        df_4=df_3.iloc[:10, :120]
        df_5= df_4.stack().reset_index(drop= True)
        df_6 = pd.DataFrame(df_5.values.reshape(1,-1))

        #Compressing to all row 900 columns
        df_7 = df_6.iloc[:, :900]
        device_list.append(df_7)
        device_label.append(file_label_dict[device_name])

#Merging all dataframe
df_merged= pd.concat(device_list)
df_merged= df_merged.reset_index(drop=True)

# adding device label and exporting dataframe to csv format
df_with_label_2= df_merged
df_with_label_2.insert(0,'label',device_label)
df_with_label_2.to_csv('merged_with_label.csv', index= False)

df_1 = pd.read_csv('merged_with_label.csv')
#Shuffling the data
```

```
df = df_1.sample(frac = 1).reset_index(drop=True)

#Split label from dataframe. Converting to to numpy array.
y= np.array(df.label)
df.drop('label', axis=1,inplace= True)

#Data scaling# MinMax scaler
scaler = MinMaxScaler()
X = scaler.fit_transform(df)

#Reshaping each single row into 30x30x1
X = np.array(df).reshape(df.shape[0],30,30,1)

#Train and test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
    0.30,random_state= 42)
```

Listing 4.4: Preprocessing of the different method for 6 device classification

4.4.2 Training and testing

Now it is time to create a model, train, and test the model on processed data. This model contains 4 convolutional layers, where each convolutional layer is followed by a max pooling layer and 2 dense layers at the end. I used optimizer `Adam` with a learning rate of 0.001, the number of epochs is set to 250, batch size 128 and for validation, I used 20% of the data. Figure 4.12 shows the model summary. After training the model I tested the performance on test data and the maximum accuracy I got was 58.33%.

Figure 4.13 shows the accuracy and loss curves on training and validation data. To improve the performance of the model I tried several different configurations of the model the same as I did with the earlier one. After several runs on each different model configuration, the maximum accuracy was 58.33%.

4.5 Classification of 6 devices: Method-2. Partial image

4.5.1 Preprocessing

The concept of this method is much simpler than the earlier ones. The idea is to take the time series data and plot a time vs voltage figure which has been shown in figure 3.4. Now this

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 30, 30, 16)	160
max_pooling2d_16 (MaxPooling	(None, 15, 15, 16)	0
conv2d_17 (Conv2D)	(None, 15, 15, 32)	4640
max_pooling2d_17 (MaxPooling	(None, 7, 7, 32)	0
conv2d_18 (Conv2D)	(None, 7, 7, 64)	18496
max_pooling2d_18 (MaxPooling	(None, 3, 3, 64)	0
conv2d_19 (Conv2D)	(None, 3, 3, 128)	73856
max_pooling2d_19 (MaxPooling	(None, 1, 1, 128)	0
flatten_4 (Flatten)	(None, 128)	0
dense_12 (Dense)	(None, 256)	33024
dense_13 (Dense)	(None, 128)	32896
dense_14 (Dense)	(None, 6)	774

Total params: 163,846
Trainable params: 163,846
Non-trainable params: 0

Figure 4.12: Model summary of the 6 device classification

same figure will be supplied as an image to the model to train the model. The processing will be done in two parts. In the first part, each CSV file containing time series data will be read, an image will be plotted from the data and then each image will be saved in a folder respective to their device name. In such a way, plotted images from all 6 devices will be saved to 6 different folders with the respective device name. Afterward, this image will be used to train the model. The code written to perform this operation is shown in listing 4.5.

```
data_dir_3 = (r'E:\Rostock-Masters\Thesis\Test')
data_dir_3 = pathlib.Path(data_dir_3)

#Creating a file name dictionary
file_name_dict_3 = {
    'Device_1' : list(data_dir_3.glob('Device_1/*.csv')),
    'Device_2' : list(data_dir_3.glob('Device_2/*.csv')),
    'Device_3' : list(data_dir_3.glob('Device_3/*.csv')),
    'Device_4' : list(data_dir_3.glob('Device_4/*.csv')),
    'Device_5' : list(data_dir_3.glob('Device_5/*.csv')),
    'Device_6' : list(data_dir_3.glob('Device_6/*.csv'))
}

```

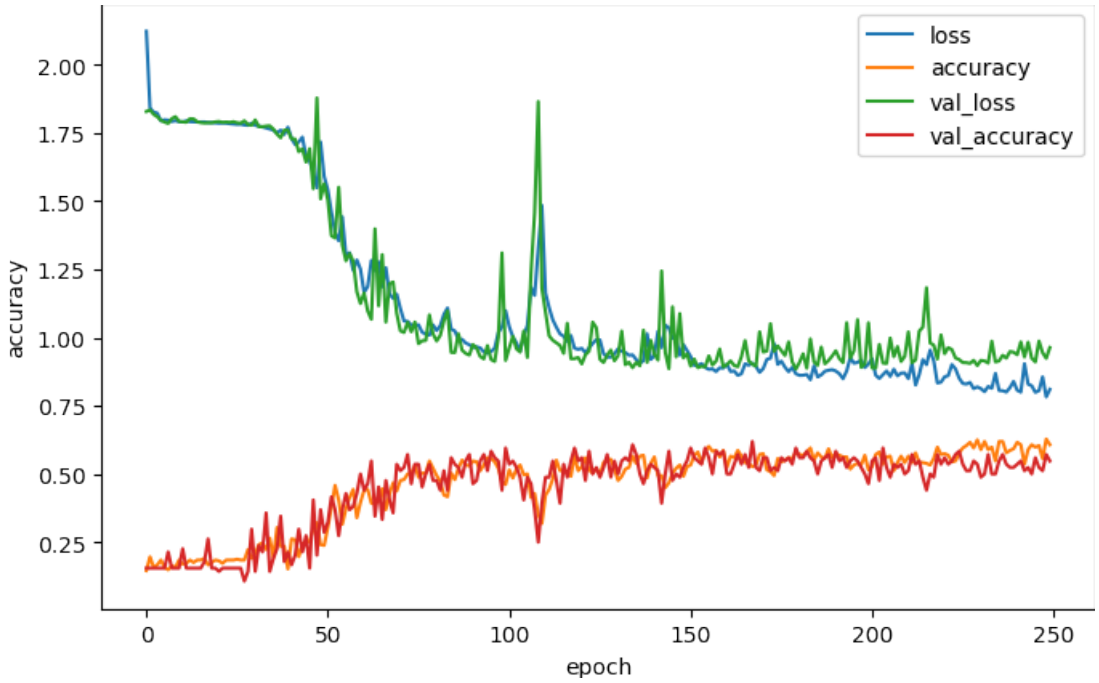


Figure 4.13: Plot of accuracy and loss on training and validation set for 6 device classifications. 58.33% accuracy.

```
#Function to read and process file
def csv_plot(device, fold_name, img_name):
    df_shortened_list = []
    for csv_file in file_name_dict_3[device]:
            csv_file_df = pd.read_csv(str(csv_file))
            csv_file_df.columns = ['time', 'voltage']
            #Compressing dataframe to 2000 rows
            df_short = csv_file_df.iloc[:2000, :]
            df_shortened_list.append(df_short)

    #Saving plotted images
    for j in range (len(df_shortened_list)):
        plt.figure(100,figsize=(1,1))
        plt.axis('off')
        plt.plot(df_shortened_list[j]['time'],
            df_shortened_list[j]['voltage'])
        plt.savefig(r'E:\Rostock-Masters\Thesis\Data\images_2000\{}/{}-{}.jpg'
        .format(fold_name,img_name,j),bbox_inches = 'tight', pad_inches = 0)
        plt.close()

#plotting images from a different folder and saving them to a different
    folder based on their name
csv_plot('Device_1', 'Device_1', 'device_1')
csv_plot('Device_2', 'Device_2', 'device_2')
csv_plot('Device_3', 'Device_3', 'device_3')
```

```
csv_plot('Device_4', 'Device_4', 'device_4')
csv_plot('Device_5', 'Device_5', 'device_5')
csv_plot('Device_6', 'Device_6', 'device_6')
```

Listing 4.5: A code to plot images and save them to different folders

In this process, a part of the telegram will be plotted instead of plotting a whole telegram to check if it is possible to classify the device from this part of the plotted telegram. Each CSV file contains 7169 rows of time and voltage values, out of these 7169 rows only the first 2000 values will be taken to plot a smaller part of the telegram. Figure 4.14 shows an example of plotted figure from the first 2000 values. Following this, the function will be run separately for each device to plot and save the images to previously created folders.

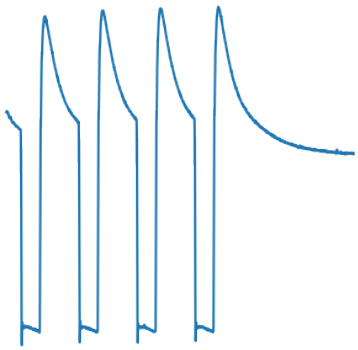


Figure 4.14: A plot of a small part of the telegram

In the second part of the processing, the saved image will be taken from the folders made in the earlier part and will go through some preprocessing. In the beginning device name dictionary and the device, label dictionary has been created similar to the previous methods then I took each image and read the image using `OpenCV python`. It is a library that allows performing different image processing and computer vision tasks. Using this library, the images will be converted to their respective RGB values which range from 0 to 255 and will be stored in a numbered array. Then the images are resized to a fixed 30x30 pixel image.
As mentioned earlier, shuffling the sample data is more effective for the training process than training all samples from the same class therefore to shuffle the data I have reshaped the 3-dimensional numbered array into 1-dimension and stored them in a list of dataframe. Later on, that dataframe in the list has been merged, the label has been added as a column and then the complete dateframe has been shuffled. The procedure has also been performed during the preprocessing of the previous method. The figure in 4.15shows some random sample images with their label. After that, the data and label have been converted to a NumPy array, train and test split are specified and finally, it has been scaled from 0 to 1 by dividing the data by 255. Now the data is ready to be trained and followed by a test. The code to perform the processing is shown in the listing 4.6.

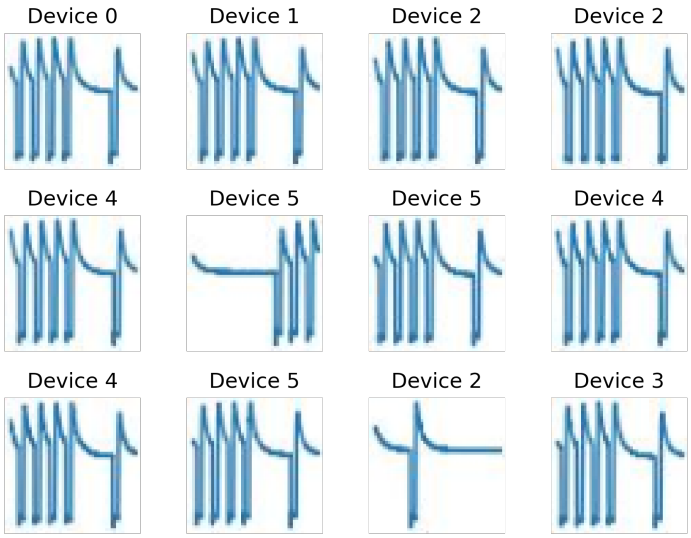


Figure 4.15: A subplot of some random time vs voltage plot with their labels

```
data_dir = (r'E:\Rostock-Masters\Thesis\Data\images_2000')
data_dir = pathlib.Path(data_dir)

#Device name and corresponding file dictionaray
device_image_dict = {
    'Device_1' : list(data_dir.glob('Device_1/*')),
    'Device_2' : list(data_dir.glob('Device_2/*')),
    'Device_3' : list(data_dir.glob('Device_3/*')),
    'Device_4' : list(data_dir.glob('Device_4/*')),
    'Device_5' : list(data_dir.glob('Device_5/*')),
    'Device_6' : list(data_dir.glob('Device_6/*'))
}

#Device name and corresponding label dictionaray
devices_label_dict = {
    'Device_1' : 0,
    'Device_2' : 1,
    'Device_3' : 2,
    'Device_4' : 3,
    'Device_5' : 4,
    'Device_6' : 5,
}

#Saving image data to a dataframe
device_list, label_list = [],[]
for device_name, folder in device_image_dict.items():
    for image in folder:
        img = cv2.imread(str(image))
```

```
        resized_img = cv2.resize(img,(30,30))
        flattened = resized_img.reshape(-1)
        df_flat= pd.DataFrame(flattened.reshape(1,-1))
        device_list.append(df_flat)
        label_list.append(devices_label_dict[device_name])

#Merging, Adding label, shuffling
df_merged= pd.concat(device_list)
df_merged= df_merged.reset_index(drop=True)
df_with_label= df_merged
df_with_label['label'] = label_list
df_with_label.to_csv('device_with_label.csv', index=False)
#Shuffling
df_1 = df_with_label.sample(frac = 1).reset_index(drop=True)

#Converting to numpy array
y= np.array(df_1.label)
df_1.drop('label', axis=1,inplace= True)
X = np.array(df_1).reshape(df_1.shape[0],30,30,3)

#Train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
    0.30,random_state=42)

#scaling data from 0 to 1
X_train_scaled = X_train/255.0
X_test_scaled = X_test/255.0
```

Listing 4.6: Preprocessing of method 2 for 6 device classification

4.5.2 Training and testing

The model will be trained and tested on the processed data. This model contains 4 convolutional layers, where each convolutional layer is followed by a max pooling layer and 2 dense layers. I used optimizer Adam with a learning rate of 0.001, the number of epochs is set to 250, batch size 128 and for validation, I used 20% of the data. The listing 4.7 shows the model.

```
#Building a model
tf.random.set_seed(42)
model_2 = tf.keras.Sequential([

    tf.keras.layers.Conv2D(filters=8, kernel_size=3, strides=1,
        padding="same", activation='relu',
```

```
    kernel_initializer='he_normal',input_shape=(X_train_scaled[0].shape)),
    tf.keras.layers.MaxPool2D(pool_size=2),

    tf.keras.layers.Conv2D(16,3, padding="same",
        activation='relu',kernel_initializer='he_normal',),
    tf.keras.layers.MaxPool2D(pool_size=2),
    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Conv2D(32,3,
        padding="same",activation='relu',kernel_initializer='he_normal'),
    tf.keras.layers.MaxPool2D(pool_size=2),
    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Conv2D(64,3,
        padding="same",activation='relu',kernel_initializer='he_normal', ),
    tf.keras.layers.MaxPool2D(pool_size=2),
    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128,kernel_initializer='normal',activation='relu'),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.Dense(256,kernel_initializer='normal',activation='relu'),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.Dense(6, activation= 'softmax') #Output layer
])
model_2.compile(loss="sparse_categorical_crossentropy",
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=["accuracy"])

history= model_2.fit(X_train_scaled, y_train, epochs=250, batch_size=128,
    validation_split=0.20, shuffle = True)
history
loss, accuracy= model_2.evaluate(X_test_scaled, y_test)
print(f'Loss: {loss}, Accuracy: {accuracy}')
print('Test accuracy : ',accuracy*100,'%')
```

Listing 4.7: A TensorFlow Keras model for classification of 6 devices

After training the model on the data it has given very good accuracy on the training data but poor accuracy on the test data. Which means the model is overfitting. Overfitting refers to a machine learning problem where the model learns the data very tightly that it learns the details and noise of the data therefore it performs very well on the training set and performs poorly on the test data or unseen data. To prevent overfitting, I used some dropout layers in

this model. During the training season, the dropout layer drops the specified number of neurons in each iteration. Hence, the neuron cannot depend on one input because it might be dropped out at random in some iteration thus it reduces the bias and bias is the major reason for overfitting. Now the model is again trained and then tested to check the performance. Figure 4.16 shows the accuracy and loss curves on training and validation data. I tried sev-

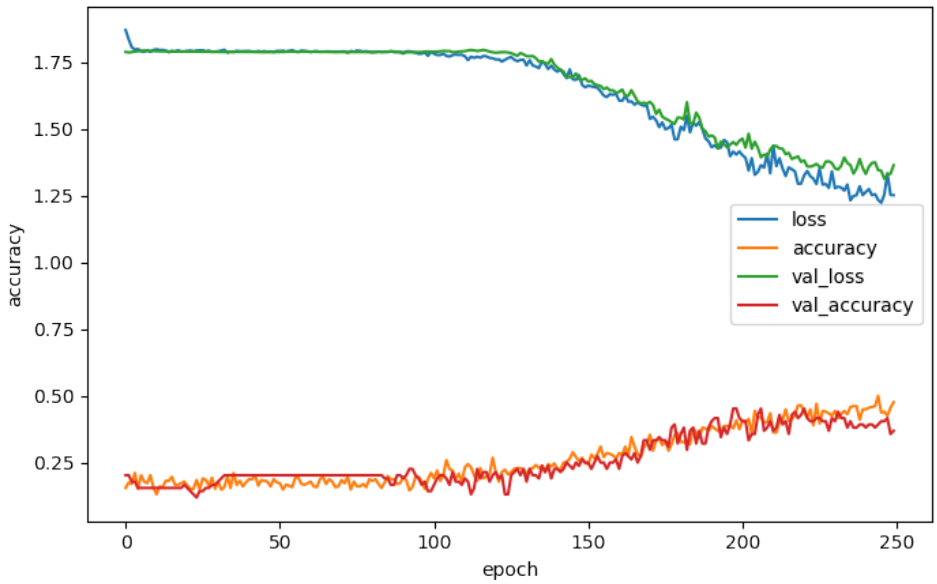


Figure 4.16: Plot of accuracy and loss on training and validation set for 6 device classifications. 44.44% accuracy

eral different configurations of the model to get the best accuracy. After several runs on each different model configuration, the maximum accuracy obtained is 44.44%

4.6 Classification of 6 devices: Method-2. Full image

4.6.1 Preprocessing

This approach mostly follows the previous one with a little change. In the processing, everything remains the same except for the number of voltage values. In the last method, I selected the first 2000 voltage values out of 7169 values. In this method, all 7169 values will be taken to plot the figure. It will look like as is shown in figure 4.17. Then this image will be resized to a 35 x 35-pixel image. This is the only change made differently from the previous approach. Figure4.18 shows some random images after processing. This image will be used to train the model.

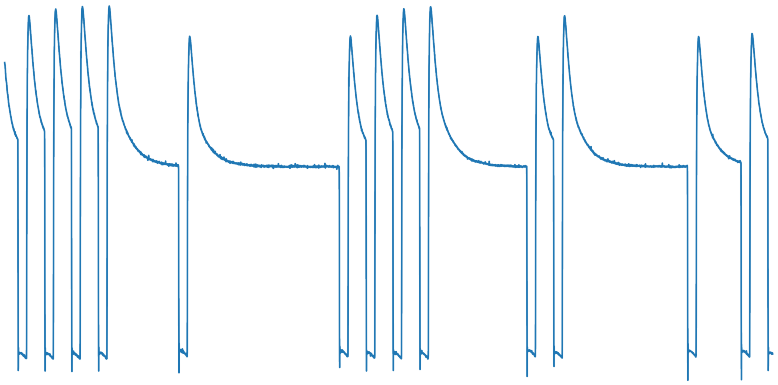


Figure 4.17: A sample time vs voltage plot with all 7169 values.

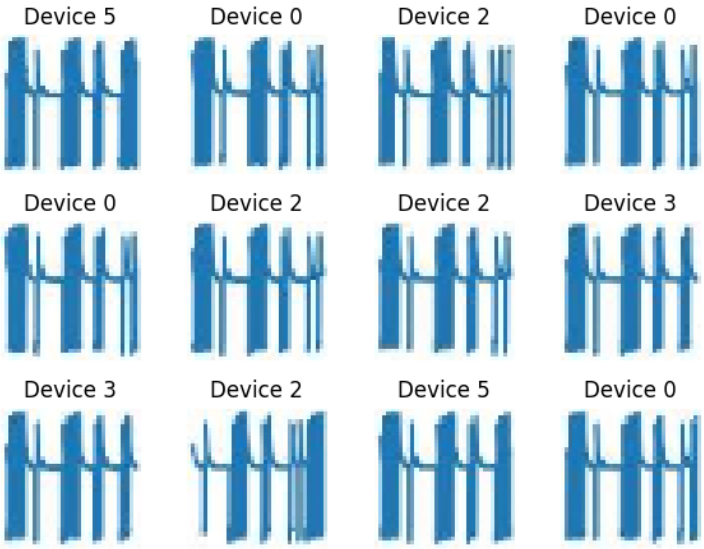


Figure 4.18: A subplot of some random images with their label

4.6.2 Training and testing

The training and testing procedure is also almost the same as the last one. In the last model, some dropout layer has been used to prevent overfitting. In this, I did not encounter an overfitting issue still kept those dropout layers so that the model cannot memorize the pattern or learn the data too tightly which can create bias. The number of dropout layers is reduced to 20% from 30% and the number of epochs is reduced to 200. The summary of the model is shown in figure 4.19. The accuracy and loss curves of training are shown in figure 4.20. Once the training ended the model was tested on test data and I got an excellent accuracy of 98%. To test the model further the test data has been changed a few times still the accuracy remained almost the same with a little fluctuation from 96-99%.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 35, 35, 8)	224
max_pooling2d_4 (MaxPooling2	(None, 17, 17, 8)	0
conv2d_5 (Conv2D)	(None, 17, 17, 16)	1168
max_pooling2d_5 (MaxPooling2	(None, 8, 8, 16)	0
dropout_5 (Dropout)	(None, 8, 8, 16)	0
conv2d_6 (Conv2D)	(None, 8, 8, 32)	4640
max_pooling2d_6 (MaxPooling2	(None, 4, 4, 32)	0
dropout_6 (Dropout)	(None, 4, 4, 32)	0
conv2d_7 (Conv2D)	(None, 4, 4, 64)	18496
max_pooling2d_7 (MaxPooling2	(None, 2, 2, 64)	0
dropout_7 (Dropout)	(None, 2, 2, 64)	0
flatten_1 (Flatten)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32896
dropout_8 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 256)	33024
dropout_9 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 6)	1542

Figure 4.19: Model summary

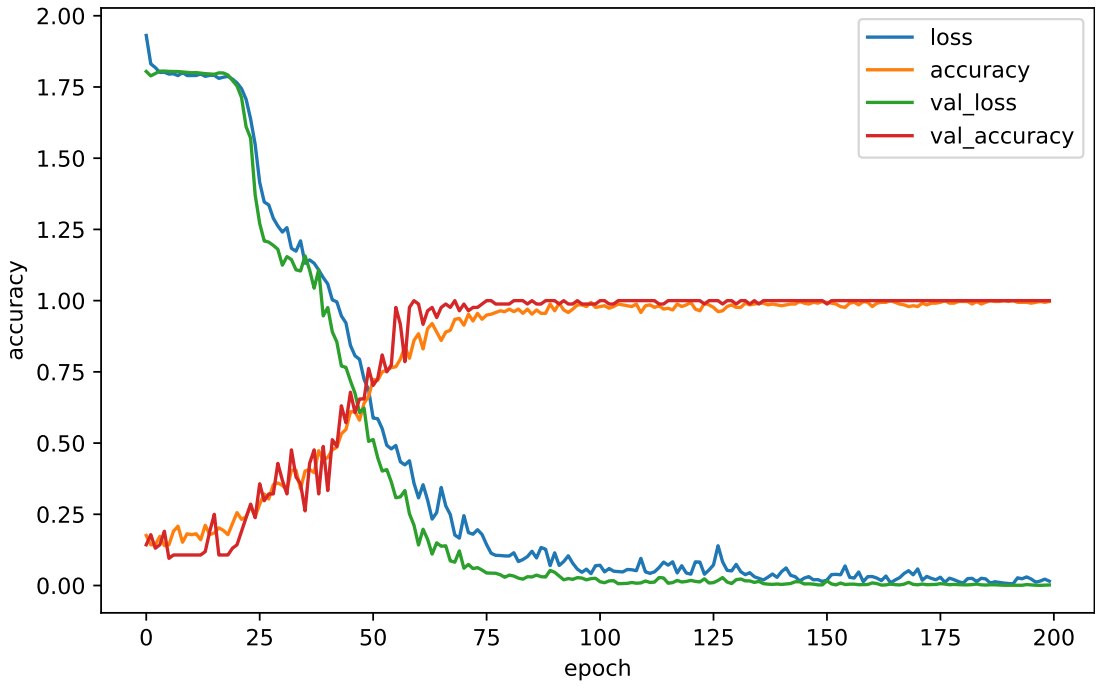


Figure 4.20: Plot of accuracy and loss on training and validation set. 98.44% accuracy

Chapter 5

Evaluation

In this chapter

The implementation that has been conducted in the previous chapter will be evaluated in this chapter. In the beginning, the methodology and some metrics will be described which will assess the classification task of different methods and their models. Afterward, each of the implementations will be evaluated in order to find out how well the specific model performed. The chapter ends by discussing whether the presented concept is appropriate to detect anomalies in the physical layer of the fieldbus network.

5.1 Standard evaluation

5.1.1 Evaluation Metrics

There are different evaluation metrics available to check the model performance, among those some well-known evaluation metrics to evaluate the classification problem will be described here [37] [38].

True Positive (TP): An outcome where the model correctly predicts the positive class.

False Positive (FP): An outcome where the model incorrectly predicts the positive class.

False Negative (FN): An outcome where the model incorrectly predicts the negative class.

True Negative (TN): An outcome where the model correctly predicts the negative class.

Accuracy: It is the most basic and primarily used metric to measure the performance of a model. It is the ratio between the true negative and true positive of all observations. It represents the correctly predicted or true predicted values divided by all items.

Confusion matrix: To summarize the performance of a classification algorithm, a confusion matrix is used. When a model makes a prediction, the confusion matrix shows how the classification model is confused. It gives insight into errors that the classifier makes but more importantly where it makes the error and the types of error. The classification accuracy alone is not good enough to evaluate the model performance when there is an unequal number of samples in each class or if there are more than two classes in the datasets. Accuracy hides the details which are needed to understand the model's performance better. For example, when there are more than two classes the model could provide very good accuracy but it is not possible to know if the model is predicting well, equally on all classes or if it is neglecting some classes. If each class does not have an equal number of observations the model could be predicting the most common class value which will give very good accuracy but at the same time, the model might be performing poorly. The structure of the confusion matrix is shown in figure 5.1. On the X-axis it shows the predicted output while in Y- the axis it shows the true output.

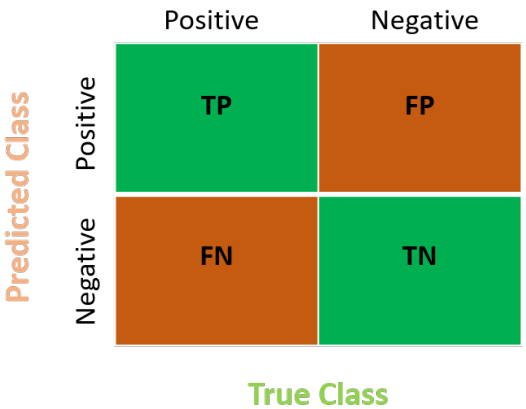


Figure 5.1: Structure of confusion matrix

Precision: Precision is the number of true positives divided by the number of total positive predictions. It indicates the percentage of actual positive results that match the positive predictions. The precision is defined as follows

$$Precision = \frac{TP}{TP + FP}$$

Recall: Recall is true positive divided by the number of true positive and false negative or in other words the total number of the items that belong to the positive class because false negative means they were positive class but predicted as negative. It is the ability of the model to find out the relevant items within a data set. The formula of recall is shown below

$$Recall = \frac{TP}{TP + FN}$$

F-score (F1): The F-score also known as F1-score is a way to combine the precision and recall of the model in such a way that it defines the harmonic mean of the model's precision and recall. The F-score is expressed as follows

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

5.1.2 Results evaluation

In this section, the previously implemented methods' results will be analyzed and evaluated. The evaluation metrics have already been selected and discussed. Using those metrics, it will be checked how well a specific method and the corresponding model performed. Two Scikit-learn libraries named `classification_report` and `confusion_matrix` have been used which generate the evaluation report in a nice format. The classification report constructs precision, recall, and F1-score report together in a table format. Each model's result will be evaluated separately.

5.1.2.1 Classification of 2 devices

Confusion matrix

The structure of the confusion matrix has already been discussed briefly. In figure 5.2 the confusion matrix of this model is shown. The diagonal elements in the confusion matrix represent the label that is predicted and true while other elements except for the diagonal ones

indicate wrongly predicted labels. A higher number of diagonal values in the confusion matrix indicates more correct predictions and specify better performance of the model. From the figure 5.2 it can be seen that this model predicted 26 observations as Device_1 which is correct and 5 observations as Device_2 which is incorrect. For Device_2 it is 27 correct predictions while 2 wrong predictions.

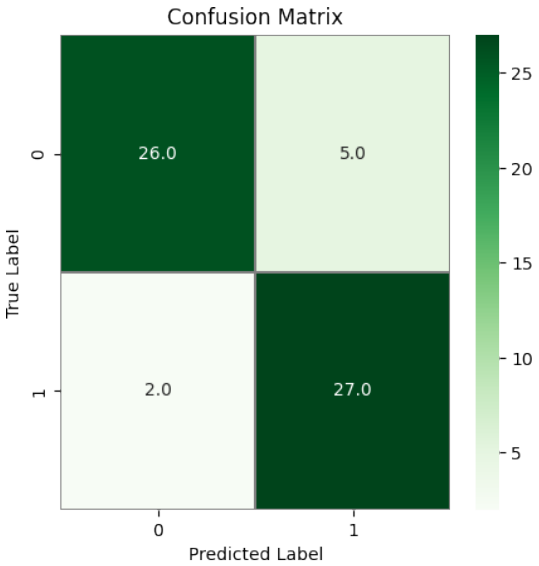


Figure 5.2: Confusion matrix. Classification of 2 devices

Classification report

Figure 5.3 shows the classification report of this model. The precision, recall and f1-score of the Device_1 is 93%, 84%, 88% respectively while for Device_2 it is 84%, 93%, and 89%. The accuracy of the model is 88%. From the result above it can be said the model for the classification of 2 devices is performing well. It correctly predicted most of the observations for both of the classes.

5.1.2.2 Classification of 6 devices: Method-1

Confusion matrix

From the confusion matrix shown in figure 5.4 it can be observed, out of 30 test observations for each class the model made a correct prediction of 20, 19, 6, 15, 12, and 29 for Device_1 to Device_6 respectively. It made an excellent prediction on Device_6 (29 correct predictions out of 30) input and a very poor prediction on Device_3 (6 correct predictions out of 30) input. It predicted incorrectly most of the Device_3 data as Device_5 data. The prediction score of Device_3, Device_4, and Device_5 is lower than Device_1, Device_2, and Device_6.

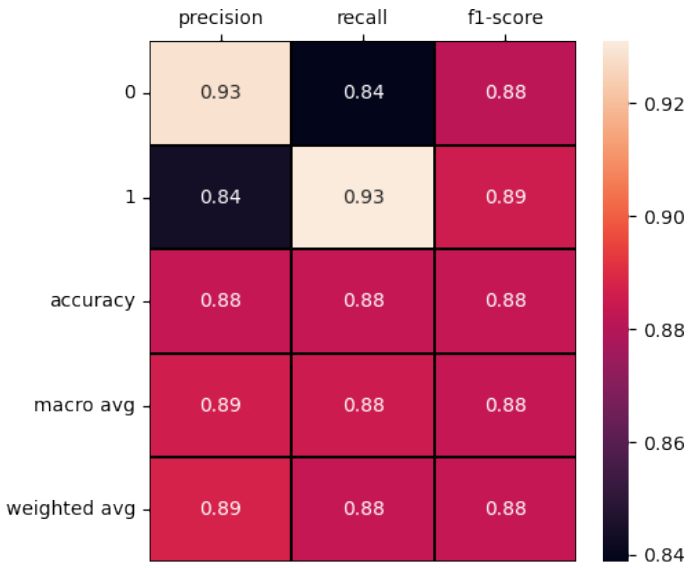


Figure 5.3: Classification report. Classification of 2 devices

Classification report

The poor performance on Device_3 prediction is also reflected in the classification report. Figure 5.5 shows the classification report of this model. The precision, recall, and f1-score of Device_3 are 40%, 20%, and 27% respectively. Device_4 also has a poor score which is 30%, 40%, and 34% respectively. Device_6 has the best score in all of those three metrics which is obvious because of a high number of accurate predictions among all observations.

5.1.2.3 Classification of 6 devices: Method-1. Different structure.

Confusion matrix

This model used the same data as the previous one but the data structure was different. Figure 5.6 shows the confusion matrix of this model. The number of correct predictions made by this model from Device_1 to Device_6 is 25 out of 29, 16 out of 26, 5 out of 25, 15 out of 34, 17 out of 37, and 27 out of 29 respectively. If we take a closer look then we can see the prediction score is kind of similar to that earlier one. Device_3 has the lowest correct prediction and Device_5 has the most correct prediction. Device_1 and Device_5 have more correct predictions than the earlier ones.

Classification report

Figure 5.7 shows the classification report of this model. Device_3 has low precision, recall, and f1-score which is 33%, 20%, and 25% respectively while Device_4 and Device_5 have closely similar scores in all three metrics. The precision score of Device_6 is a little lower than that of Device_1 and Device_2. Device_2 has a 62% recall score which is fairly lower than that of Device_1 and Device_6 which is 86% and 93% respectively. The f1-score of Device_1, Device_2, and Device_6 is 78%, 68%, and 76% respectively.

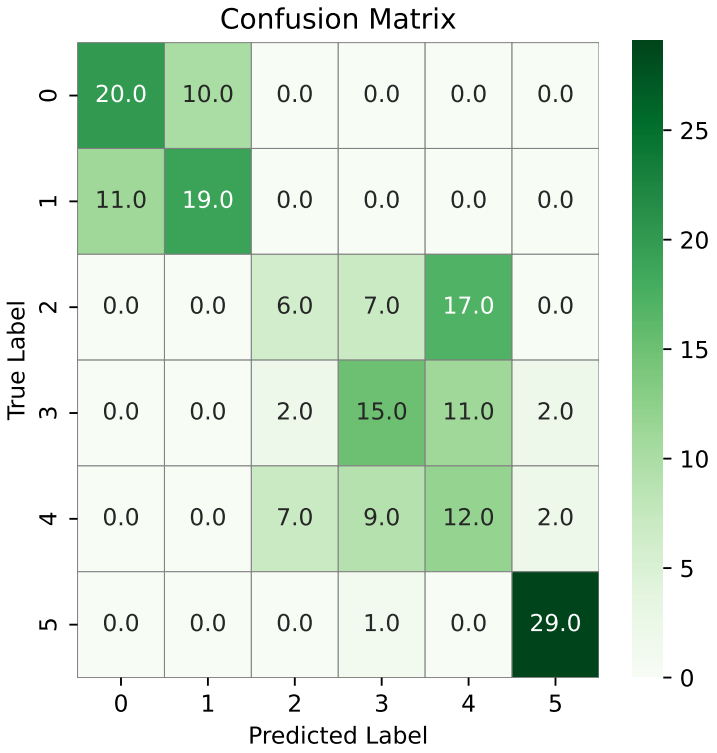


Figure 5.4: Confusion matrix. Classification of 6 devices: Method-1

5.1.2.4 Classification of 6 devices: Method-2. Partial image

Confusion matrix

The confusion matrix of this model is shown in figure 5.8 The number of correctly predicted observations from Device_1 to Device_6 is 20 out of 28, 30 out of 37, 0 out of 29, 6 out of 28, 7 out of 23, and 17 out of 35 respectively. In this prediction, the model could not correctly classify any observation from Device_3 out of 29 observations. In the last two models, the number of accurate predictions of Device_3 was also significantly lower among all the other observations. Based on the data the model had it mostly incorrectly predicted Device_3 data as Device_5 data, the reason might be that the voltage signature of this device is very similar to the Device_5. A poor number of correct predictions is seen in Device_3, Device_4, and Device_5 while Device_2 has the most correct prediction followed by Device_1 and Device_6.

Classification report

The classification report of this model can be seen in figure 5.9. Device_1 and Device_2 have better scores in precision, recall, and f1-score among the others. Device_3 has a 0% score in all three metrics since it could not correctly classify any test observations. Among Device_4, Device_5, and Device_6, Device_6 has a better score in all three while Device_4 and Device_5 have a little bit similar score.

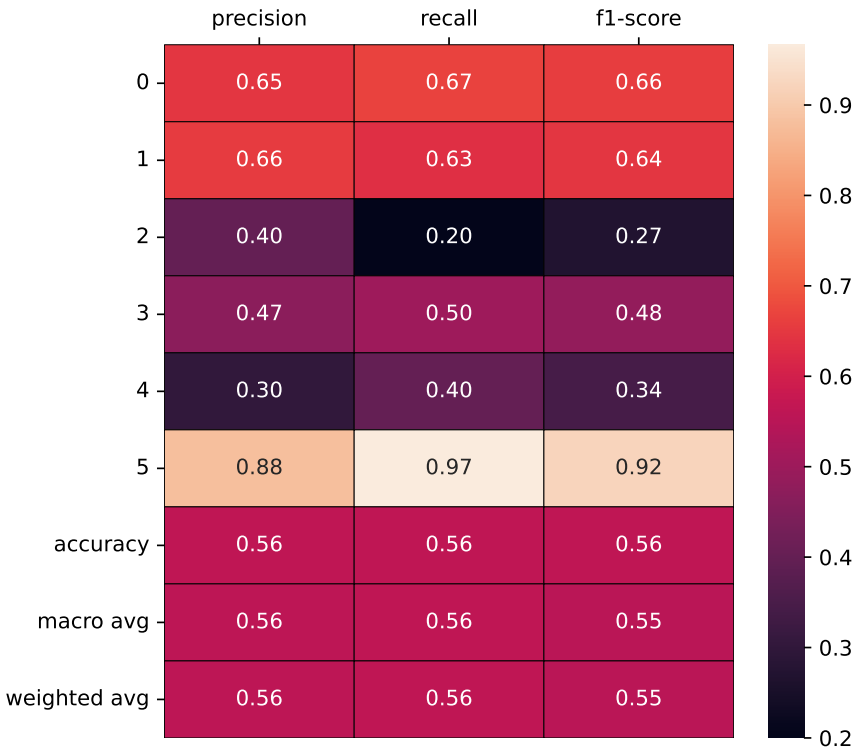


Figure 5.5: Classification report. Classification of 6 devices: Method-1

5.1.2.5 Classification of 6 devices: Method-2. Full image

Confusion matrix

Figure 5.10 shows the confusion matrix of this model. It is evident from the result that we got almost a perfect result from this model. Out of the 30 observations in each class the model has been able to correctly classify all classes except the one in Device_6.

Classification report

The classification report of this model is shown in figure 5.11 Since 99% of predictions were correct therefore the precision, recall, and f1-score of most of the classes are 100% apart from Device_2 and Device_5. The precision and f1-score of Device_3 are 97% and 98% while the recall and f1-score of Device_6 are 97% and 98%.

5.1.2.6 Evaluation summary

A simple evaluation summary of all the models is shown in table 5.1. It is clear from the table that the last method and the corresponding model has the highest score in term of accuracy, precision, recall, and F1 score. The first method has the next best score, which was the classification of only two classes. The rest three methods and their corresponding models' scores do not seem to be very promising.

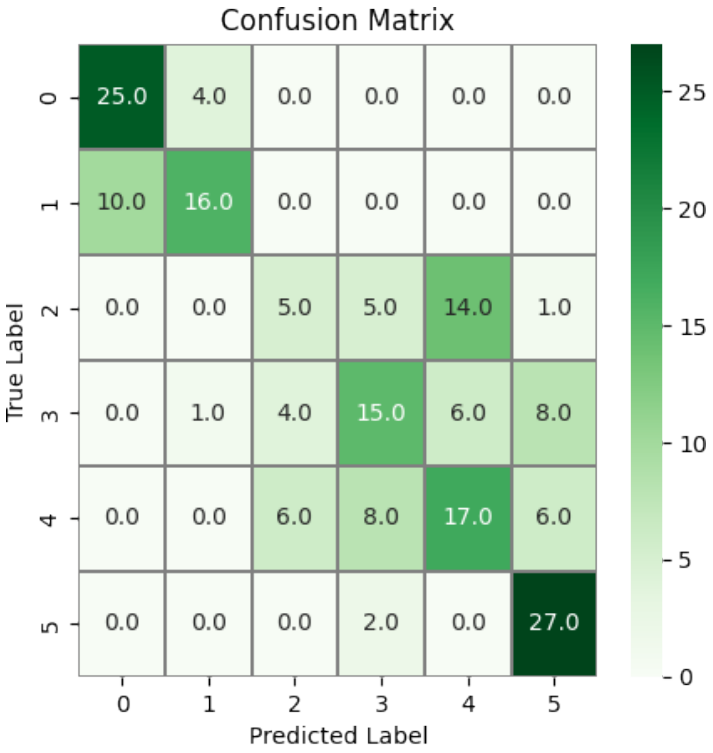


Figure 5.6: Confusion matrix. Classification of 6 devices: Method-1. Different structure.

Table 5.1: Evaluation summary of all models

Methods	Accuracy	Precision	Recall	F1-score
Classification of 2 devices	88%	89%	88%	88%
Classification of 6 devices: Method-1	56%	56%	56%	55%
Classification of 6 devices: Method-1. Different structure	58%	57%	58%	57%
Classification of 6 devices: Method-2. Partial image	44%	37%	42%	39%
Classification of 6 devices: Method-2. Full image	99%	99%	99%	99%

5.2 Anomaly detection

5.2.1 Anomaly detection using dummy data

It has been shown that using the full image method the model can classify the devices very accurately. Now a method needs to be devised on how can we use the prediction score to detect anomalies in the fieldbus layer. A possible technique will be discussed here by which it is possible to detect anomalies using the prediction score. To test this technique, I saved the previous model and made five CSV files containing some dummy data which was produced using the MS excel `RANDBETWEEN` function. I have filled up the voltage column with some random data from 20 to 37. Except for those five CSV files, I have taken 30 CSV files for each device and those CSV files were not used for training. Then each CSV files were processed and converted into an image to be used for testing. Consequently, I loaded the saved model

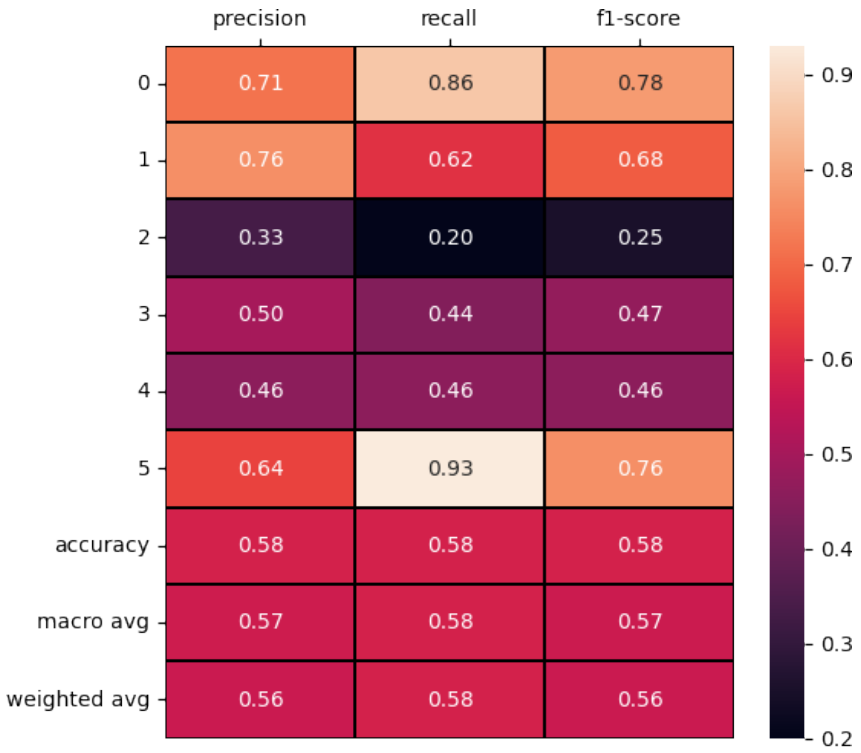


Figure 5.7: Classification report. Classification of 6 devices: Method-1. Different structure.

and supplied the processed images to the model to predict.

It is necessary to know how a model delivers the prediction score. When the test data is supplied to a model to make a prediction it provides a prediction score for each class. Listing 5.1 shows an example of prediction scores for six different observations. Each prediction score consists of six different numbers since the model was trained for six devices and each number corresponds to the probability score for corresponding Devices starting from Device_1 to Device_6. If the first array is taken as an example, it can be seen, the first number 0.9905557 is a big number while the other numbers are much smaller than that. It means the model predicts that there is about 99% probability that this input belongs to Device_1 while the possibility of this input belonging to other devices is very much slim. The first four arrays have one big number in a different position where the position indicates the index number of the Device. That means the first two input belongs to Device_1, the third one belongs to Device_6 and the fourth one belongs to Device_5. In the last two arrays, the maximum number is 0.54. That means these two inputs have a maximum of 54% probability of being one of the six devices.

```
array([
       [0.9905557, 0.0054009, 0.00396093, 0.0001309, 0.00002221, 0.00004718 ],
       [0.9998623, 0.00002246, 0.00011524, 0.00000002, 0.   , 0.       ],
       [0.0000029, 0.00000012, 0.     , 0.       , 0.       , 0.9999995 ],
       [0.9999809, 0.0000003, 0.00001881, 0.    , 0.       , 0.       ],
       [0.00001175, 0.      , 0.5446801, 0.00000028, 0.00000496, 0.    ],
       [0.00000006, 0.00000002, 0.    , 0.       , 0.       , 0.5482906 ]
```

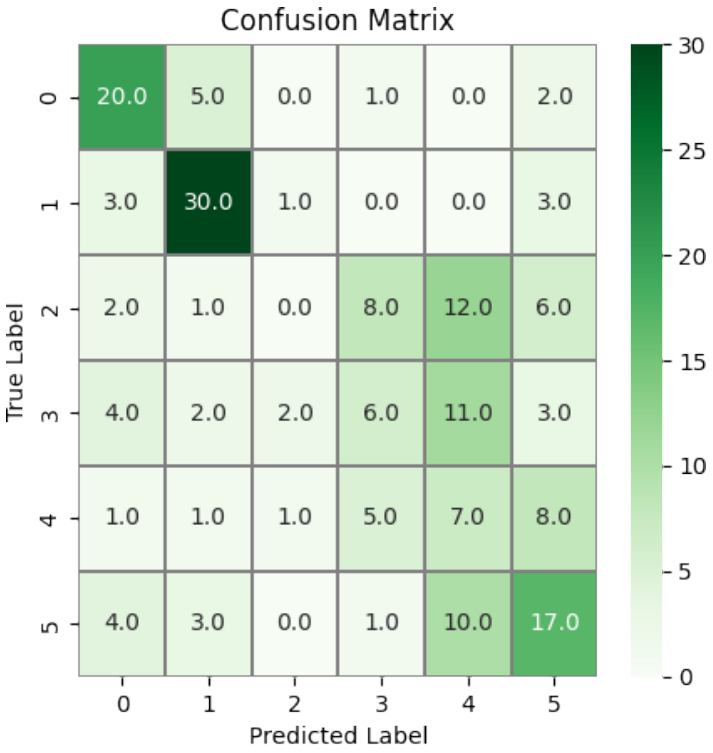


Figure 5.8: Confusion matrix. Classification of 6 devices: Method-2. Partial image

```
8       ],)
```

Listing 5.1: An example of prediction score for different input

Later on, I wrote a small code that will go through all the arrays where each array consists of six values, and take the maximum value from each array. If that maximum value is more than 0.70 then this input belongs to a normal observation and if it is less than that then it is an abnormal observation. In other words, if there is equal or more than 70% probability that this input belongs to any of the six devices then it is normal, if the probability is less than that the input is abnormal and it prints a line that anomaly detected and shows the total number of abnormal observations. According to the necessity and model's performance, the threshold value can be adjusted. Figure 5.12 shows a scatter plot of normal and abnormal observations after running the function. From the figure, it can be seen the model predicted 5 observations as an anomaly which is the exact number of CSV files I created previously.

5.2.2 Anomaly detection using real data

In this part, the model will be tested on some real data but the model has never seen this data before. This data belongs to five different and new devices and they are not the same device on which the model was trained. Since the test data is entirely unknown to the model, it should categorize all the data as an anomaly, or in other words, the prediction scores for

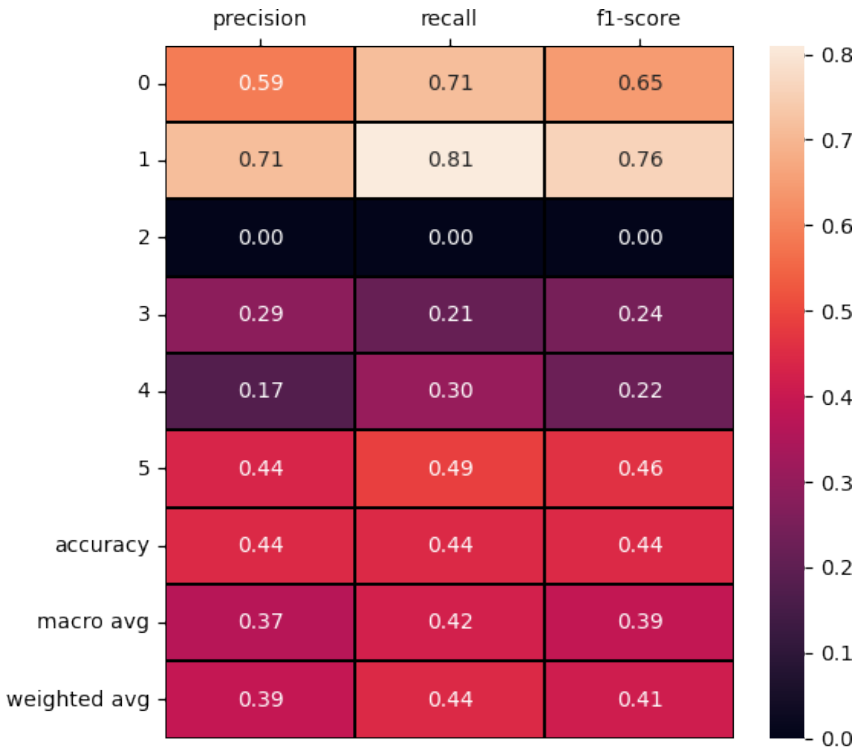


Figure 5.9: Classification report. Classification of 6 devices: Method-2. Partial image

this observation should be much lower. Because I choose the threshold level as 0.70, if the probability is less than 70% then it is an anomaly. To test the model, I took ten files each from five new devices. Subsequently, the data was processed, converted into images, and supplied to the model for prediction.

Figure 5.13 shows a scatter plot of normal vs abnormal observations after the prediction was done. From the figure, it can be seen that out of 50 data points, only 6 were categorized as an anomaly. However, the model should have categorized all of them as anomalies since those data points are completely unknown to the model. If we take a look at the Y-axis which shows the prediction score, it can be seen the model has predicted that among those data points most of them are near 0.99 and some of them are between 0.70 - 0.90. Put differently, the model predicts there is more than 70% probability that 88% data belongs to one of the six devices that I used for training, and out of this 88% data, again 88% data has more than 90% probability that it belongs to one of the six devices, which is incorrect.

5.2.2.1 Testing model robustness

In this segment, the model used earlier for anomaly detection will be tested with the same data but with a little added noise. All the data except the first cell has been shifted one step down or if it is considered as an image then one pixel to the right and the last cell data is transferred to

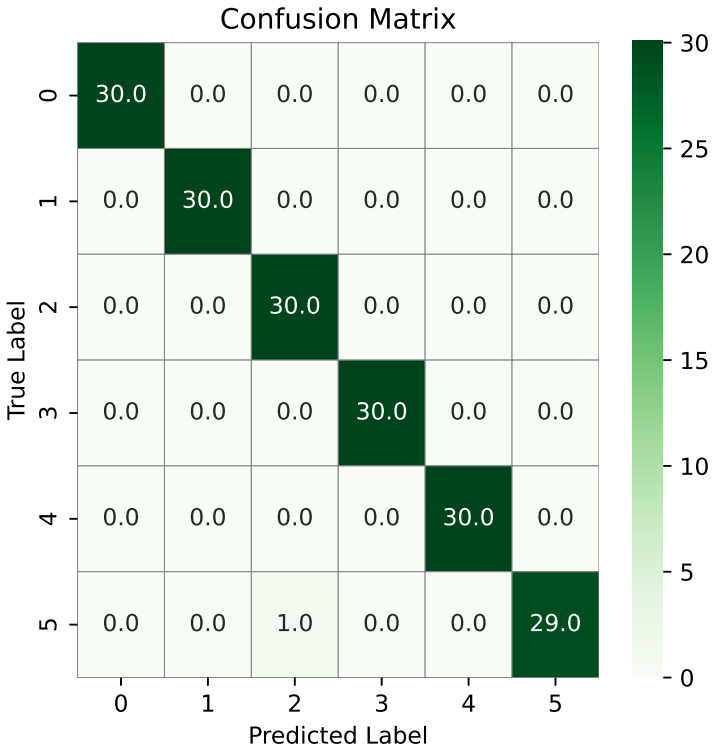


Figure 5.10: Confusion matrix. Classification of 6 devices: Method-2. Full image

the first cell. The motive is to check the model's robustness and how it performs when some noise is added to the model. If the model performance changes significantly then the model is not performing very well on the other hand if the performance changes only a little then it is performing well. Figure 5.14a and 5.14b show the data before and after adding noise. All the data goes through the processing same as before to be tested by the same saved model.

Once the processing is finished the data was tested and a plot is drawn from the result. Figure 5.15 shows the plot after the test. If the figure 5.15 and 5.13 are compared then it can be noticed there are slight changes after the noise has been added. It detected 8 anomalies with the predefined threshold limit whereas in the data without noise it was 6. Most of the data points are concentrated between 0.9-1.0 whereas, in the previous one, some data points existed between 0.7-0.8. From the test, it can be assumed the model is performing well enough even after the noise is added to the data.

5.2.3 Anomaly detection: 7 output layer method

In this method, the same data and model which has been used in the full image method will be used with the addition of 1 extra output layer. The objective is to classify the abnormal observation as layer number 7. The first 6 layers will be used to classify from Device_1-Device_6 same as earlier and all abnormal observations will be classified as 7. To do this, number 7 has been trained with some made-up data which I have created using MS excel

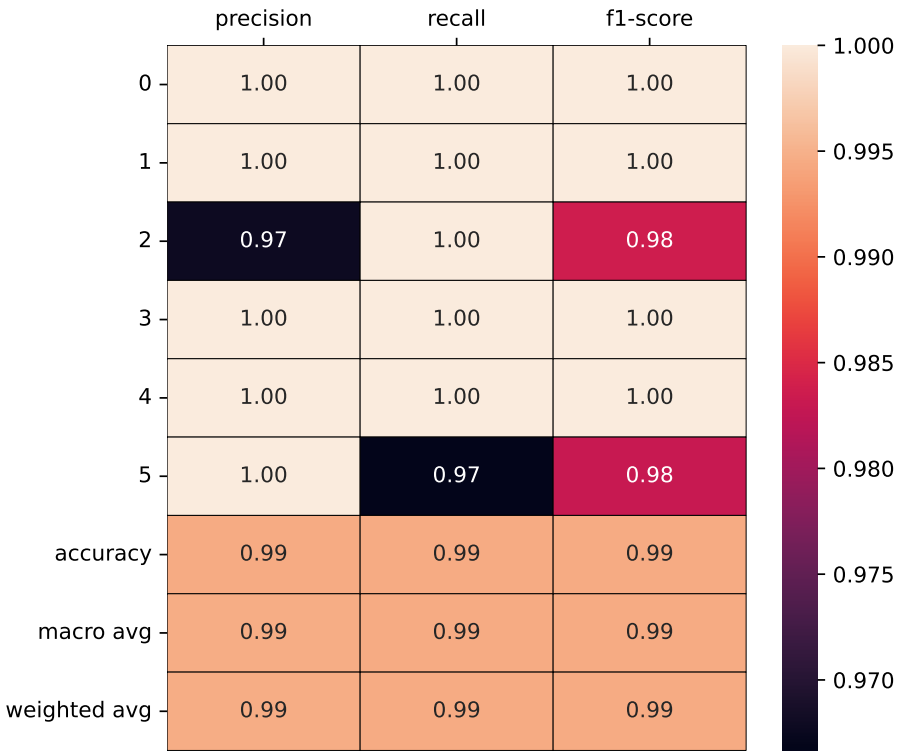


Figure 5.11: Classification report. Classification of 6 devices: Method-2. Full image

with some random values ranging from 20-37 and some unrelated device data which does not belong to any devices of Device_1- Device_6. For example, those data belong to Device_10, Device_11, Device_12... etc. The idea is, if the incoming data point or test data has very less probability of being one of the six devices then the model would classify those data points as 7, and the number 7 will be considered an abnormal observation. The data preprocessing and training procedure were the same as the full image implementation method.

Once the data was trained that model was saved and then loaded to carry out the test. For the test a total of 30 data points were supplied to the model. 6 of them were made up of random data created with MS excel, and the rest of the data belongs to five different devices and they are Device_10, 15, 17, 19,20. The test result is shown in figure 5.16 as a confusion matrix. It shows good accuracy from Device_1 – Device_6 (0 is Device_1 and 5 is Device_6) but out of 30 observations in 7 (labeled as 6), it only predicted 8 observations as anomalies while 4 observations predicted as Device_1, 6 observations as Device_2, 8 observations as Device_3 and 4 observations as Device_5. Out of the 8 correctly predicted samples, 6 of them were made up of random data which means the model only predicted 2 unrelated device data as an anomaly while the rest of them were classified as normal observations. It suggests the model predicts there is a high probability that those unseen device samples are pretty much similar to the known sample. Thus, those unrelated real device data are classified as one of the six devices or as a normal observation while all of the made-up random data has been classified as anomalies since the model has already learned the pattern of those data and they are very much different

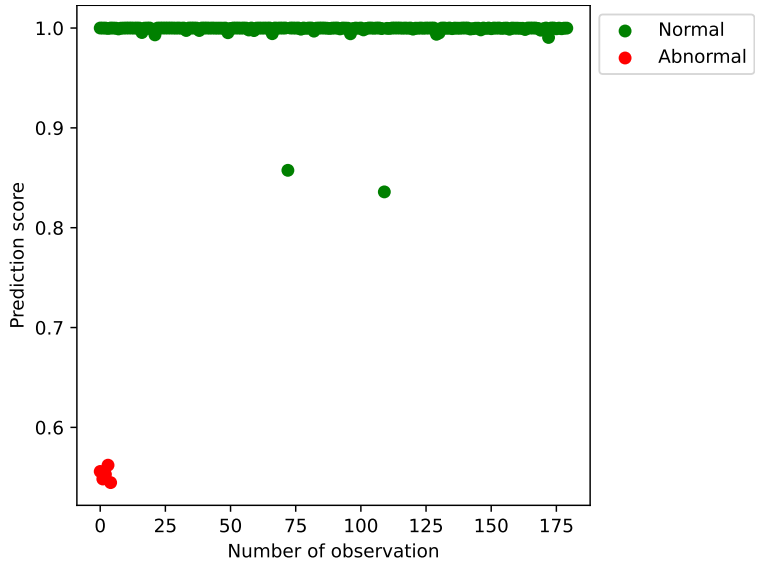


Figure 5.12: A scatter plot with normal and abnormal observations using dummy data

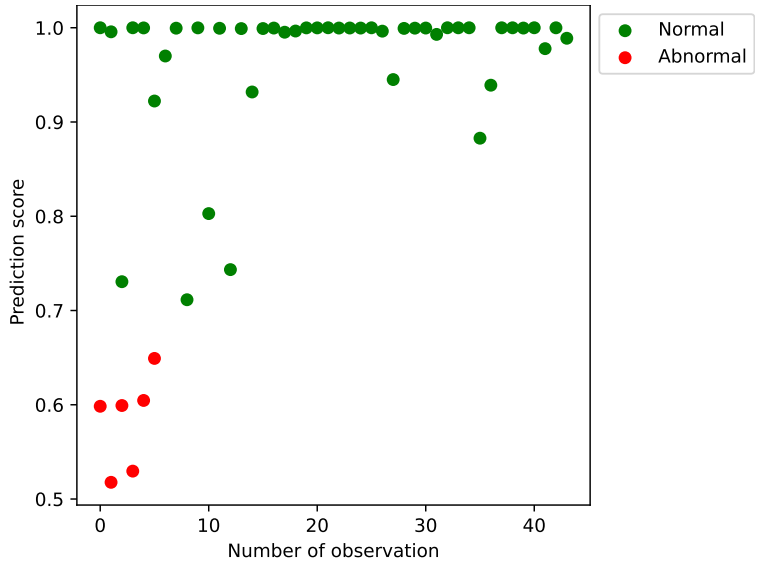


Figure 5.13: A scatter plot with normal and abnormal observations using real data

than the real data.

5.3 Discussion

Some different test has been run in the implementation section to find out if it is possible to classify the devices using a part of the telegram and a full telegram. The tests did not show very good results except for the full image method. Therefore, it can be said using those parts of the telegram it may not be possible to classify the devices. With the full image method or using the full telegram, good accuracy was achieved but in the evaluation section it has been observed when the model gets real data input that is out of observation or on which the model

	time	voltage
0	0.0	30.679499
1	0.5	30.649153
2	1.0	30.639038
3	1.5	30.639038
4	2.0	30.649153
...	...	...
7163	3581.5	34.867205
7164	3582.0	34.877320
7165	3582.5	34.907666
7166	3583.0	34.917781
7167	3583.5	34.857089

7168 rows × 2 columns

(a) Data without noise

	time	voltage
0	0.0	34.857089
1	0.5	30.679499
2	1.0	30.649153
3	1.5	30.639038
4	2.0	30.639038
...	...	...
7163	3581.5	34.846974
7164	3582.0	34.867205
7165	3582.5	34.877320
7166	3583.0	34.907666
7167	3583.5	34.917781

7168 rows × 2 columns

(b) Data after adding little noise. All the data shifted down one step and the last cell data is transferred to the first cell.

Figure 5.14: Data before and after adding noise

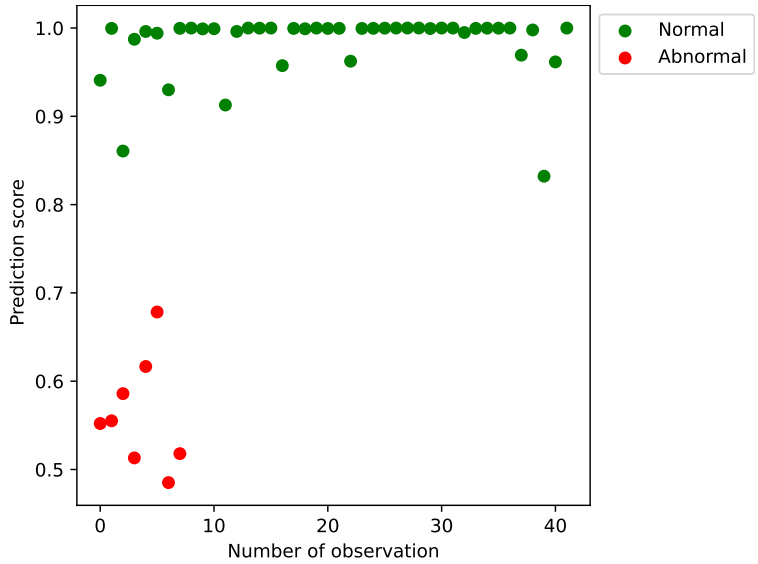


Figure 5.15: A scatter plot of normal vs abnormal observations after adding noise to the data

was not trained, it misclassifies them as not anomaly.

Figure 5.17 shows two different telegrams from two different devices. It shows both of the telegrams look almost the same with very little difference. To investigate further both of the device's voltage reading was checked and it was noticed the voltage reading of the devices are not the same rather there are some differences in the voltage reading. An argument can be made from this figure and voltage reading. When the model was trained on the respective device data it learned the small differences in the curves, which is why it can classify correctly the devices on which it was trained but when there is an observation on which the model was not trained, it classifies it as one of the devices which looks mostly alike. From the tests in the evaluation part with the random data, it has been observed if there are significant differences between real device data and anomaly data then it is possible to classify the anomaly but if the malicious devices output the same voltage or almost the same voltage reading as the normal devices then it may not be feasible to classify between normal and abnormal data.

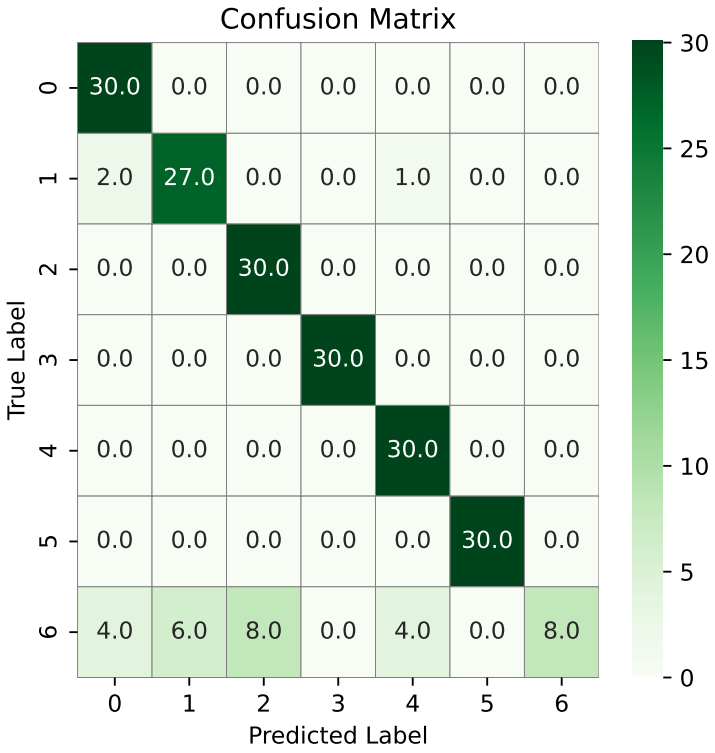


Figure 5.16: A confusion matrix of 7 output layer method

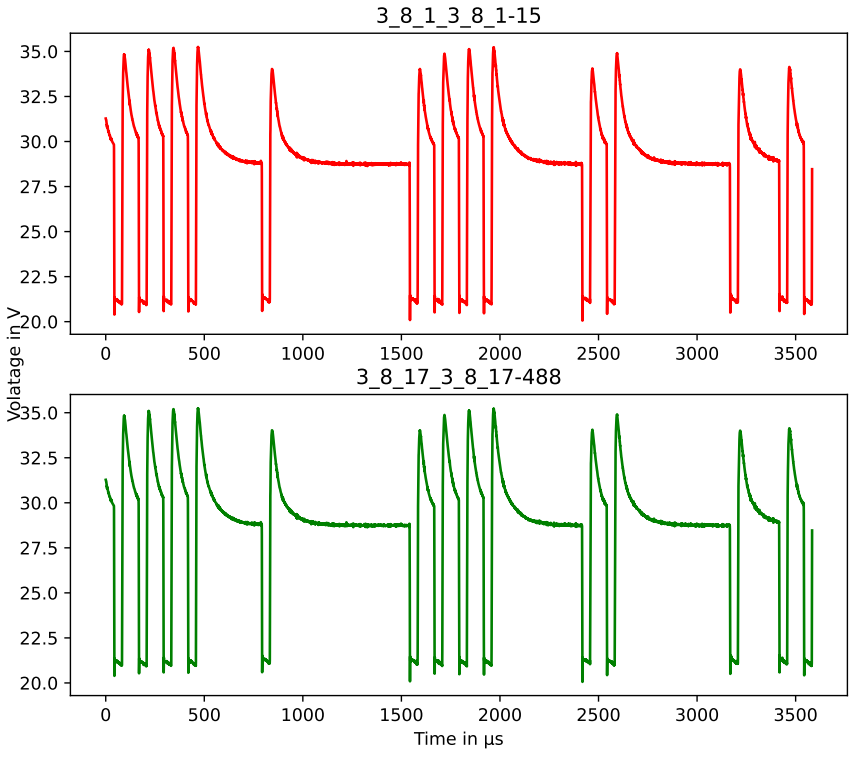


Figure 5.17: A comparison of two different telegrams from two different devices

Chapter 6

Conclusion and future work

Building automation systems are becoming more and more popular due to the fact they help to reduce energy consumption and lower operating and maintenance costs. It has already been observed that existing protocols do not offer any security features such as encryption, authentication, and data freshness. Hence, it is important to secure those protocols using any other means possible. In this thesis, some experimentation has been conducted on physical layer data using machine learning to inspect whether it is possible to use an IDS that can detect anomalies on the network.

The tests revealed that it may not be possible to classify the normal and abnormal data using the physical layer data. Though to further investigate, a different part of the telegram could be used for experiments, for example in the conducted tests, mostly the first part of the telegram was used, but some different tests could be run on the middle part or last part of the telegram to see if it is possible to classify the devices.

The graph of the whole telegram method provided a promising result in classifying the devices but was unable to classify the abnormal observations due to the very much similarity with the normal observations. But if it is possible to identify abnormal data and train the model with abnormal data then there is a high chance the model would be able to classify normal and abnormal observations with higher accuracy. If we consider using the voltage values of a whole telegram then 'Classification of 6 devices: Method-1. Different structure' method could be used to examine if it can classify the normal and abnormal data since it will be using the real voltage values and it has been seen before even the graph of two measurements from two different devices looks the same but their voltage reading is not. Although it would make the image size much bigger (approximately 84x84 pixels) it might deliver some optimistic results. It might also be possible to take a smaller part of the telegram and doing so will make the image size smaller. In addition, some preprocessing can be done on the data by applying

a low-pass or high-pass, or band-pass filter on the data and that could likely produce some satisfactory results.

To further evaluate, it is also worth trying some different algorithms on the data because some algorithms may produce satisfactory output for the specific data type. RNN (Recurrent neural network) is also a neural network algorithm that provides good results on sequential or time series data.

List of Figures

List of Tables

Listings

Bibliography

[1] V. Graveto, T. Cruz, and P. Simões, "Security of Building Automation and Control Systems: Survey and future research directions," *Computers & Security*, vol. 112, p. 102 527, Jan. 2022, ISSN: 0167-4048. DOI: 10.1016/J.COSE.2021.102527 (cit. on pp. 3, 4).

[2] M. Peters, J. Goltz, S. Wiedenmann, and T. Mundt, "Using Machine Learning to Find Anomalies in Field Bus Network Traffic," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11611 LNCS, pp. 336–353, 2019, ISSN: 16113349. DOI: 10.1007/978-3-030-24907-6_26 (cit. on pp. 4, 28, 30).

[3] T. Mundt and P. Wickboldt, "Security in building automation systems - a first analysis," in *2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security)*, 2016, pp. 1–8. DOI: 10.1109/CyberSecPODS.2016.7502336 (cit. on p. 4).

[4] T. Mundt, A. Dähn, and S. Sass, "An intrusion detection system with home installation networks," *International Journal of Computing and Digital Systems*, vol. 3, no. 01, 2014 (cit. on p. 4).

[5] "Context aware intrusion detection for building automation systems," *Computers & Security*, vol. 85, pp. 181–201, Aug. 2019, ISSN: 0167-4048. DOI: 10.1016/J.COSE.2019.04.011 (cit. on p. 4).

[6] A. Zdziarstek, W. Brekenfelder, and F. Eibisch, "Using the physical layer to detect attacks on building automation networks," *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 336, pp. 372–390, 2020, ISSN: 18678211. DOI: 10.1007/978-3-030-63095-9_24/FIGURES/9. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-63095-9_24 (cit. on p. 4).

[7] W. Granzer, F. Praus, and W. Kastner, "Security in building automation systems," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3622–3630, 2010, ISSN: 02780046. DOI: 10.1109/TIE.2009.2036033 (cit. on pp. 5, 40, 41).

[8] P. Ciholas, A. Lennie, P. Sadigova, and J. M. Such, "The Security of Smart Buildings: a Systematic Literature Review," Jan. 2019. DOI: 10.48550/arxiv.1901.05837. arXiv: 1901.05837. [Online]. Available: https://arxiv.org/abs/1901.05837v3 (cit. on p. 5).

[9] Mead O'Brien, *The Basics of a Fieldbus Control Network*, 2017. [Online]. Available: https://www.antaira.com.tw/Insight/Detail/6 (visited on 07/06/2022) (cit. on p. 10).

[10] M. Rahman, A. Desalegn Fentaye, V. Zaccaria, I. Aslanidou, E. Dahlquist, and K. Kyprianidis, "A Framework for Learning System for Complex Industrial Processes," in *AI and Learning Systems - Industrial Applications and Future Directions*, IntechOpen, Feb. 2021. DOI: 10.5772/intechopen.92899 (cit. on p. 12).

[11] KNX.org, "KNX Basics," *Knx*, 2016. [Online]. Available: http://www.knx.org/media/docs/Flyers/KNX-Basics/KNX-Basics_en.pdf (cit. on pp. 12–15, 17–20, 23).

[12] S. Cavalieri and F. Chiacchio, "Limiting the loss of information in KNXnet/IP on congestion conditions," *Computer Networks*, vol. 73, pp. 154–172, Nov. 2014, ISSN: 13891286. DOI: 10.1016/j.comnet.2014.08.012 (cit. on p. 20).

[13] P. Ehrlich, *Building automation*. 2014, vol. 31, p. 20, ISBN: 9783319732220. DOI: 10.1515/9783035612912-008 (cit. on pp. 20, 21).

[14] *A guide to using KNX over IP — Ivory Egg (AU)*. [Online]. Available: https://www.ivoryegg.com.au/case_studies/a-guide-to-using-knx-over-ip (visited on 07/12/2022) (cit. on p. 22).

[15] H. J. Liao, C. H. Richard Lin, Y. C. Lin, and K. Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, Jan. 2013, ISSN: 1084-8045. DOI: 10.1016/J.JNCA.2012.09.004 (cit. on pp. 23, 27).

[16] M. Martellini and A. Malizia, *Cyber and Chemical, Biological, Radiological, Nuclear, Explosives Challenges: Threats and Counter Efforts*, 1st. Springer Publishing Company, Incorporated, 2017, ISBN: 3319621076 (cit. on p. 24).

[17] T. Mundt, S. Wiedenmann, J. Goltz, J. Bauer, and M. Jung, "Detecting intrusive behaviour of people in a building through data analysis and anomaly detection in home automation systems," in *2019 10th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2019 - Proceedings and Workshop*, Institute of Electrical and Electronics Engineers Inc., Jun. 2019, ISBN: 9781728115429. DOI: 10.1109/NTMS.2019.8763844 (cit. on pp. 24, 27).

[18] Z. Yu, J. J. Tsai, and T. Weigert, "An adaptive automatically tuning intrusion detection system," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 3, no. 3, Aug. 2008, ISSN: 15564665. DOI: 10.1145/1380422.1380425. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1380422.1380425 (cit. on p. 24).

[19] D. M. Farid and M. Z. Rahman, "Anomaly network intrusion detection based on improved self adaptive Bayesian algorithm," *Journal of Computers*, vol. 5, no. 1, pp. 23–31, 2010, ISSN: 1796203X. DOI: 10.4304/jcp.5.1.23-31 (cit. on pp. 24, 26).

[20] "Preventing System Intrusions," in *Network and System Security: Second Edition*, Syngress, Jan. 2013, pp. 29–56, ISBN: 9780124166899. DOI: 10.1016/B978-0-12-416689-9.00002-2 (cit. on p. 25).

[21] "Introduction to Intrusion Detection Systems," in *Cisco Security Professional's Guide to Secure Intrusion Detection Systems*, Elsevier, 2003, pp. 1–38. DOI: 10.1016/b978-193226669-6/50021-5 (cit. on pp. 25–27).

[22] H. Jiang, *Machine Learning Fundamentals : A Concise Introduction*. 2021, ISBN: 9781108837040 (cit. on pp. 27, 29).

[23] M. Batta, "Machine Learning Algorithms - A Review," *International Journal of Science and Research (IJ*, vol. 9, no. 1, pp. 381–386, 2020. DOI: 10.21275/ART20203995 (cit. on p. 28).

[24] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, *Convolutional neural networks: an overview and application in radiology*, Aug. 2018. DOI: 10.1007/s13244-018-0639-9. [Online]. Available: https://link.springer.com/article/10.1007/s13244-018-0639-9 (cit. on pp. 30, 33, 35, 49).

[25] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, Jun. 2021, ISSN: 2162-237X. DOI: 10.1109/tnnls.2021.3084827. arXiv: 2004.02806 (cit. on pp. 30, 34).

[26] J. Goltz, "Securing Building Automation Systems," in *2021 11th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2021*, Institute of Electrical and Electronics Engineers Inc., Apr. 2021, ISBN: 9781665443999. DOI: 10.1109/NTMS49979.2021.9432650 (cit. on pp. 37, 43).

[27] KNX Association, "KNX Secure," 2020. [Online]. Available: https://www.knx.org/wAssets/docs/downloads/Marketing/Flyers/KNX-Secure-Position-Paper/KNX-Secure-Position-Paper_en.pdf (cit. on pp. 38, 42).

[28] M. Kapsalakis, "Passive Situational Awareness and Threat Detection in Building Automation Networks," Ph.D. dissertation, 2017 (cit. on p. 39).

[29] M. Burgess, *Austrian hotel Romantik Seehotel Jaegerwirt was hit by a cyberattack — WIRED UK*, 2017. [Online]. Available: https://www.wired.co.uk/article/austria-hotel-ransomware-true-doors-lock-hackers (visited on 07/29/2022) (cit. on p. 39).

[30] N. Falliere, L. O. Murchu, and E. Chien, *"W32. stuxnet dossier". In: White paper, Symantec Corp., Security Response 5.6*, 2011 (cit. on p. 39).

[31] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus, "Security in networked building automation systems," *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, pp. 283–292, 2006. DOI: 10.1109/WFCS.2006.1704168 (cit. on p. 39).

[32] G. Johannes, M. Thomas, and S. Wiedenmann, "Risk analysis in fieldbus networks using the example of knx," in *International Conference on Information Networking*, vol. 2019-Janua, IEEE Computer Society, May 2019, pp. 310–315, ISBN: 9781538683507. DOI: 10.1109/ICOIN.2019.8718149 (cit. on pp. 39, 41).

[33] A. Judmayer, L. Krammer, and W. Kastner, "On the security of security extensions for IP-based KNX networks," *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, 2014. DOI: 10.1109/WFCS.2014.6837593 (cit. on p. 41).

[34] A. Antonini, F. Maggi, and S. Zanero, "A Practical Attack Against a KNX-based Building Automation System," BCS Learning & Development, Sep. 2014. DOI: 10.14236/ewic/icscsr2014.7. [Online]. Available: https://www.scienceopen.com/hosted-document?doi=10.14236/ewic/ICSCSR2014.7 (cit. on p. 41).

[35] *KNX IP Secure becomes new ISO Standard KNX Association [Official website]*. [Online]. Available: https://www.knx.org/knx-en/for-professionals/newsroom/en/news/KNX-IP-Secure-becomes-new-ISO-Standard/ (visited on 08/01/2022) (cit. on p. 42).

[36] Willi Brekenfelder, "Automatisierung der Messdatenerfassung für die Überwachung des Physical Layers in Feldbussen," Bachelor Thesis, University of Rostock, 2020 (cit. on p. 43).

[37] R. N. Sucky, *Learn Precision, Recall, and F1 Score of Multiclass Classification in Depth — by Rashida Nasrin Sucky — Towards Data Science*. [Online]. Available: https://towardsdatascience.com/precision-recall-and-f1-score-of-multiclass-classification-learn-in-depth-6c194b217629 (visited on 11/12/2022) (cit. on p. 79).

[38] K. Leskovar, *Machine Learning — How to Evaluate your Model? — by Karlo Leskovar — Towards Data Science*. [Online]. Available: https://towardsdatascience.com/machine-learning-how-to-evaluate-your-model-1dabbdc849a4 (visited on 11/12/2022) (cit. on p. 79).

Declaration

I, Nazmul Alam, hereby declare that I have written this master's thesis independently and have not used any sources or aids other than those specified.

All passages that are taken literally or analogously from publications are marked as such.

The work has not yet been published and has not yet been submitted in a similar or the same way as an examination for recognition or assessment.

Rostock, December 19, 2022