DTU Technical University of Denmark
Department of Engineering Technology and Didactics

Implementation of data acquisition and application function on Raspberry Pi for district heating system.

Diploma Bachelor Thesis

Diploma Bachelor Thesis

Implementation of data acquisition and application function on Raspberry Pi for district heating system.

November, 2022

By
Anders Dahlerup Johansson
Andreas Silla

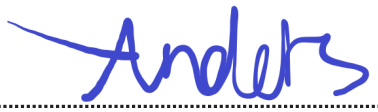
Cover photo: Vibeke Hempler, 2012
Published by: DTU, Department of Engineering Technology and Didactics, Lautrupvang 15, 2750 Ballerup Denmark
www.engtech.dtu.dk

Approval

This thesis has been prepared over twelve weeks at the Department of Engineering Technology and Didactics, at the Technical University of Denmark, DTU, in partial fulfilment for the degree Bachelor of Engineering, BEng.

It is assumed that the reader has a basic knowledge in the areas of computer science, statistics, IoT, deep learning, Linux OS and embedded systems.

Anders Dahlerup Johansson - s184813



Signature

20-11-2022

Date

Andreas Silla - s165214

Signature

20-11-2022

Date

Acknowledgements

Anders Dahlerup Johansson, BEng, IT Electronics, DTU
Andreas Silla, BEng, IT Electronics, DTU

Guangya Yang, Senior Researcher, Supervisor
Thanks to Guangya for being a very helpful and engaged supervisor. Always being quick to set up meetings and offer feedback, which was a great help throughout.

Tomasz Blaszczyk, Lektor, DTU Supervisor
Thanks to Tomasz for being our DTU supervisor for our Diploma Bachelor Project.

Ana Turk, PostDoc, Co-supervisor
Thanks to Ana for all the guidance regarding model construction and how to improve it.

Anders Larsen, R & D Scientist, Neogrid
Thanks to Anders for providing support to the Neogrid aspects both device and API.

Abstract

Inconsistency has always been an issue that has plagued renewable energy sources such as wind and solar energy. This lack likely accounts for why other means are prioritized higher due to their consistent energy generation and conventional employment independent of their environment. In order to make renewable energy a more attractive choice, the inconsistency aspect must be addressed by developing strategies to manage its fluctuating output. One of such strategies is to establish a framework from where it is possible to accurately predict the ebbs and flows of the output, which enables power grids to plan for every ebb and more effectively utilize every flow.
This project seeks to explore the complication mentioned earlier by proposing a reproducible solution that addresses the uncertainty aspect of renewable solar energy. The solution is meant to provide a Deep Learning model capable of forecasting energy fluctuations, running on a lightweight hardware platform that fetches input data from the model. Predictions by the model should then be made accessible for power grids to adjust accordingly by either preparing other means of energy production at lows or favoring energy from solar panels at highs.

This project was successful in delivering a fully functional model incorporated into a docker container running on a Raspberry Pi platform. Additionally, a MQTT connection was established between the Raspberry Pi and a subscriber, which enabled the platform to retrieve data from a temperature sensor. MQTT connection was established to be capable of local and remote server access. The deep learning model is set to forecast the expected power output four hours into the future, and this job will be run once per hour. The predictions will be saved in a file called “predict.txt”. Before executing that job, temperature data is funneled into another file called “data.csv”. This system is meant to be versatile in its employment of MQTT connection since temperature data could easily be replaced with any other type of input. In conclusion, this system provides the core functionality requested by RE-EMPOWERED, but as with most things, it is viable for future improvements.

Contents

1 Introduction

This chapter introduces the reader to RE-EMPOWERED, Neogrid, Bornholms Energi & Forsyning, and the technical collaboration between the three. Furthermore, an introduction to the project description and objectives, problem definition, and an overview of why the project scope had to be changed will be discussed.

1.1 Case description

Solar energy has proven to be a convenient asset for the eventual transition to renewable energy that our modern society will hopefully undergo in the decades to come. Unfortunately, as is true with most things, novel things come with their own set of challenges. In this case, it is maintaining a stable and predictable flow of energy to the connected power grids since the amount of energy generated by solar panels is primarily dictated by weather conditions. At present, there is little to be done if the weather turns cloudy, which will lead to a significant drop in power output. This will leave us with a cycle of 'feast or famine' regarding energy output, as the fluctuating weather conditions will dictate how long the panels will directly be exposed to the sun. Therefore, since weather manipulation is not an option just yet, we are left to predict an outcome and try to work from that point. This is where one of the critical parts of this project comes into the picture, as it seeks to tackle this instability. Reliability is key, which is why the delegated task is to make a reasonable estimation of the expected output from power grids benefiting from solar panels outputs. The more reliable a prediction is, the greater power manageability is achieved.

Water can usually, by default, be heated by a wood chip boiler, but alternatively, this can also be done through electric boilers. These electric boilers are powered through, for instance, solar panels. When an excess of energy is gathered through these means, it would be favorable to find scenarios where this excess could be put into use. Predicting when this excess may arise would enable the broader water heating system to adjust for this change and then produce less hot water through wood chip power and more tempered water from the electric boilers.

The strategy is to provide reliable predictions in order for the power grid to utilize the energy contributed by solar panels. This way, less energy will go to waste, and thus the system will be more energy efficient.

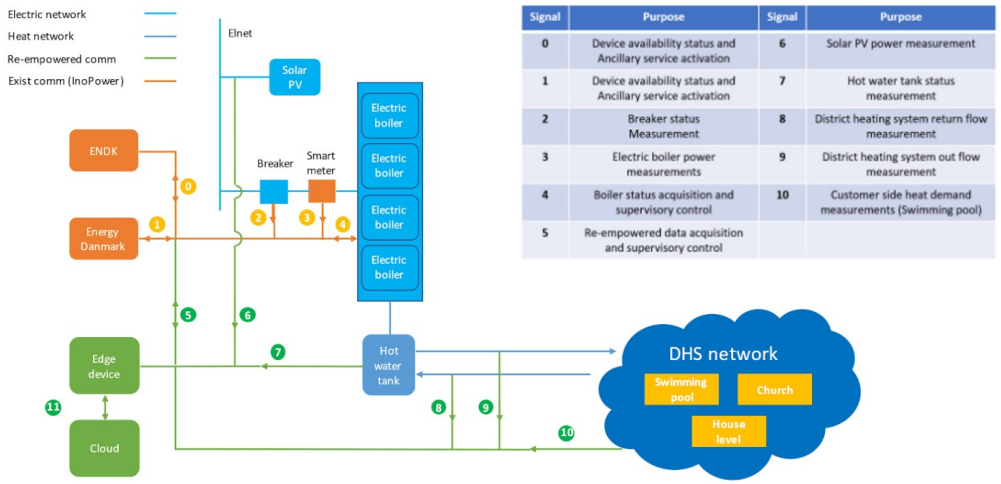
Project RE-EMPOWERED[1] and Neogrid[2] are cooperating in order to work on this issue. RE-EMPOWERED aims to develop better strategies for existing power grids on islands to more effectively utilize said power grids and thus lead to less energy waste. Neogrid Technologies ApS is a cleantech company that works with intelligent energy visualization, monitoring, and control. Here, knowledge and experience from wireless communication technology are used to develop Smart Grid solutions. In addition, Neogrid offers hardware and data connectivity for the project to enable development into new power grid strategies. The data is collected by Bornholms Energi & forsyning.

1.2 Components

- Raspberry Pi 4 Model B 8GB
- Deep Learning
- Docker
- ESP8266
- DHT11 Temperature sensor
- Relays

1.3 Problem definition

RE-EMPOWERED requires an Internet of Things(IoT) Communication system, which is integrated with a forecasting deep learning model.



Signal	Purpose	Signal	Purpose
0	Device availability status and Ancillary service activation	6	Solar PV power measurement
1	Device availability status and Ancillary service activation	7	Hot water tank status measurement
2	Breaker status Measurement	8	District heating system return flow measurement
3	Electric boiler power measurements	9	District heating system out flow measurement
4	Boiler status acquisition and supervisory control	10	Customer side heat demand measurements (Swimming pool)
5	Re-empowered data acquisition and supervisory control		

Figure 1.1: Structure of project development

The task involves:

- **Build and test communication to acquire data (Direct Heating System(DHS) data, boiler, households) into Raspberry Pi.**
  Establishing a reliable connection to acquire data will be paramount to this project since generating predictions will be impossible without any input for the model. It is also of importance to the project that said predictions are accessible for power grids to alter their heating strategy more accordingly.
- **The communication between a Neogrid device and a Raspberry Pi must be established.** Since the Neogrid will serve as a gateway for the Raspberry Pi to access data from Bornholm and data is essential for Deep Learning model development, it would be reasonable to assume this task as the initial step to take when commencing this project.

- **Data acquisition from different energy carriers and data correlation**
  It should be possible to acquire data from different carriers, as this will demonstrate versatility in data acquisition in the final product. In addition, this will enable the product to more convincingly be defined as an universal platform to be freely deployed on various locations rather than a platform bound by a few carriers.
- **Data analysis with deep learning models and use of a forecasting algorithm for heat demand based on machine learning (Python))**
  Constructing a deep learning model is broadly utilized in fields using AI pattern recognition to forecast future events or outputs for the benefit of developing more efficient strategies. Generating reliable predictions can be considered another core part of this project, as predictions should contribute a basis for more effective power use.
- **Implement/load the forecasting algorithm in Raspberry Pi by using the Docker containers**
  It is necessary to implement the model into the Raspberry Pi as it will serve as a data endpoint. This will more tangibly involve the model in the overall product and ensure that the physical product will be a singular instance.

It is expected that these tasks will be completed within the project deadline. Depending on the project results, further optimization could be done. This would be:

- **Mutiple clients connected over a broad network.**
  This attribute would assist in providing a picture of how scalable the connectivity aspects of this project would be. Eventual fixes and remedies could then be applied or suggested to support further scaling.
- **Prediction visualization through plots**
  This would make the predictions outputted by the model a little more intuitively comprehensive and a nice-to-have. This could for example be achieved with the help of a dashboard with plotting capabilities.
- **Custom defined hourly forecast** This is another nice-to-have, as it would give the user input as to how far the model is to predict. This would provide the user with more control as to how the model is desired to work.
- **Time Scheduler for automation**
  This would be beneficial as no human intervention is needed for the cycle of commands to run and would introduce an element of automatization
- **Neogrid API Python library utilization**
  This library would be a convinient asset for the project as it allows for data to be retrieved within the model code. This would enable the code to be more independent, as it would be able to provide itself historical data for model training and inputs for predictions.

RE-EMPOWERED prioritizes both an IoT connection between cloud and edge devices and a functioning deep learning model implemented in Docker containers. This is with the intent to be able to replicate and scale their system to other households in the future.

1.4 Project method - changes to the project scope

Changes can often be expected when working on a project. Therefore, one is bound to find another answer that will illustrate a reasonable solution, given the circumstances, when obstacles are encountered during the project run-time.

We encountered several risks when reviewing our project, which required us to make adjustments. (see Project Management section 2.3).

To complete the tasks, we therefore, modified the scope of our project:

- To illustrate the DHS data, we switched to receiving data from a DHT11 temperature sensor, which we implemented and tested connectivity with the system. We also added relays that functioned similarly to mechanisms used to adjust the motorized control valves. Hardware was an essential part of our Diploma Bachelor Thesis requirements. Therefore, we decided to implement our own hardware system to compensate, as the project scope was mainly software-based.
- Data acquisition from different energy carriers was planned for us to access Bornholms data. Still, because of GDPR reasons which never got resolved, we had to find a way to get data ourselves.

Figure 1.2: Projec Method Development Process

The Theory
This chapter lays the foundation for the techniques and tools we will use for creating the solution. Different IoT communication models will be discussed, and a primary model will be chosen for future development. Furthermore, fundamental knowledge of deep learning is explored, and how this relates to the project scope. The following subsections outline the overarching concepts relevant to model construction and optimization, with emphasis on those involved in this project's model implementation.

IoT Integration
The IoT integration chapter will give the reader an idea of the design and implementation of our system. Then, step by step, the reader will be guided through each system upgrade, starting with simple client and broker communication and going further with upgrading the system with temperature readings and controllable relays.

Deep Learning
In this chapter, data processing as a concept will be outlined, whereafter, as how this concept is applied to preparing the data for appliance. Then, the various data processing methods applied will be described, and finally, two model configurations will be presented in a section where each configuration will be compared. This comparison will conclude with results from each configuration being compared and discussed. The chosen configuration is then optimized further, and the results will be presented afterward. Finally, this chosen configuration will define the model that is to be implemented into the Docker

containers.

Docker Implementation
This chapter will shortly describe the necessary concepts employed in the following section describing how the Deep Learning Model compatible environment was established. Particularly cumbersome errors and their potential fixes will be presented. Lastly, how to set up the new compatible environment will be delineated with code examples.

Final Implementation
This chapter outlines the final features that need to be implemented to create a connection between IoT, Deep Learning, and Docker. The employed methodology used to save MQTT data to CSV, executing of the bash-script, usage of the time scheduler, and at last, a flowchart that creates an overview of the entire solution and how it is all connected.

2 Project Management

We have applied project management principles throughout the entire project. This chapter discusses those principles and how various techniques and methods were implemented. Project management is a large portion of our project, which is why we assigned it a chapter in this diploma thesis.

2.1 Project Management Theory

The following are the primary sources of project management theory:

- ISO21500 Project - Context and concepts[3]
- ISO21502 Project - Guidance on project management[4]

The above ISOs are well-standardized, and they give a variety of ideas about what effective project management should cover. The methods in the book are based on ISO21500 and ISO21502, which draw on accepted ideas and common terminology. We decided to use the techniques from these documents to work through the project in a more clarified and organized approach. The four pillars that make up the project management approaches are:

- Purpose
- People
- Complexity
- Uncertainty

Of the four pillars, two, in particular, were of great use for the process of the project. Those two pillars being **Purpose** and **Uncertainty**.

2.2 Purpose

Undertaking a new project can be very resource and time-consuming; therefore, this project's fundamental reasons and intentions must be clearly defined. Not only for the project to commence with no ambiguation but also to secure a stable trajectory from start to finish. First, the project's whole raison d'etre is put into question with a simple ”why”. This could ordinarily be put into words such as ”Why is this project being done?”. Putting the fundamental parts of the project into question should open up for an even more numerous amount of questions by asking, ”How will this project be carried out”, ”What is the intention of this project?”, ”What is the project producing?” etc. The Golden Circle[5] is a tool specifically designed to outline this aspect of the project by presenting three ways of questioning the project. These three styles being the ”why”, the ”how”, and the ”what”.

As previously mentioned, the ”why” is to clarify the purpose or the fundamental beliefs supporting the project's conception. Therefore, the ”why” cannot point to anything result-based such as making money, since results come later and are not directly tied to the fundamental ”why”.

The ”how” is to mostly question the practicalities, such as: how the project will be conducted, how the project will achieve its predetermined goals, and how specific tasks will be

completed in time. It refers to the specific actions that will be done after the fundamental ”why” has been realized.

The ”what” is the final step and can be articulated through what is being or has been done, along with what the project achieved. This is where results can be involved and expressed through sales figures or other metrics. This question can naturally vary throughout the process, for instance, by asking ”what am I doing”. Answering this could simply be the current specific task at hand.

The Golden Circle model[6] was proposed by Simon Sinek[5], who rationalizes that the ”why” and ”how” appeals to the limbic brain. It has been claimed that this part of the brain controls behavior and decision-making and would therefore be engaged with the ”whys” and ”hows”. This should establish a gut feeling for the project. The final ”what” is claimed to appeal to the neocortex, which is a section of the brain that’s said to control senses and analytical thinking. The result of this is to establish a robust rationalization for the project.

When the aforementioned questions have been answered in relation to the project, it can be visualized within the Golden Circle model as such:

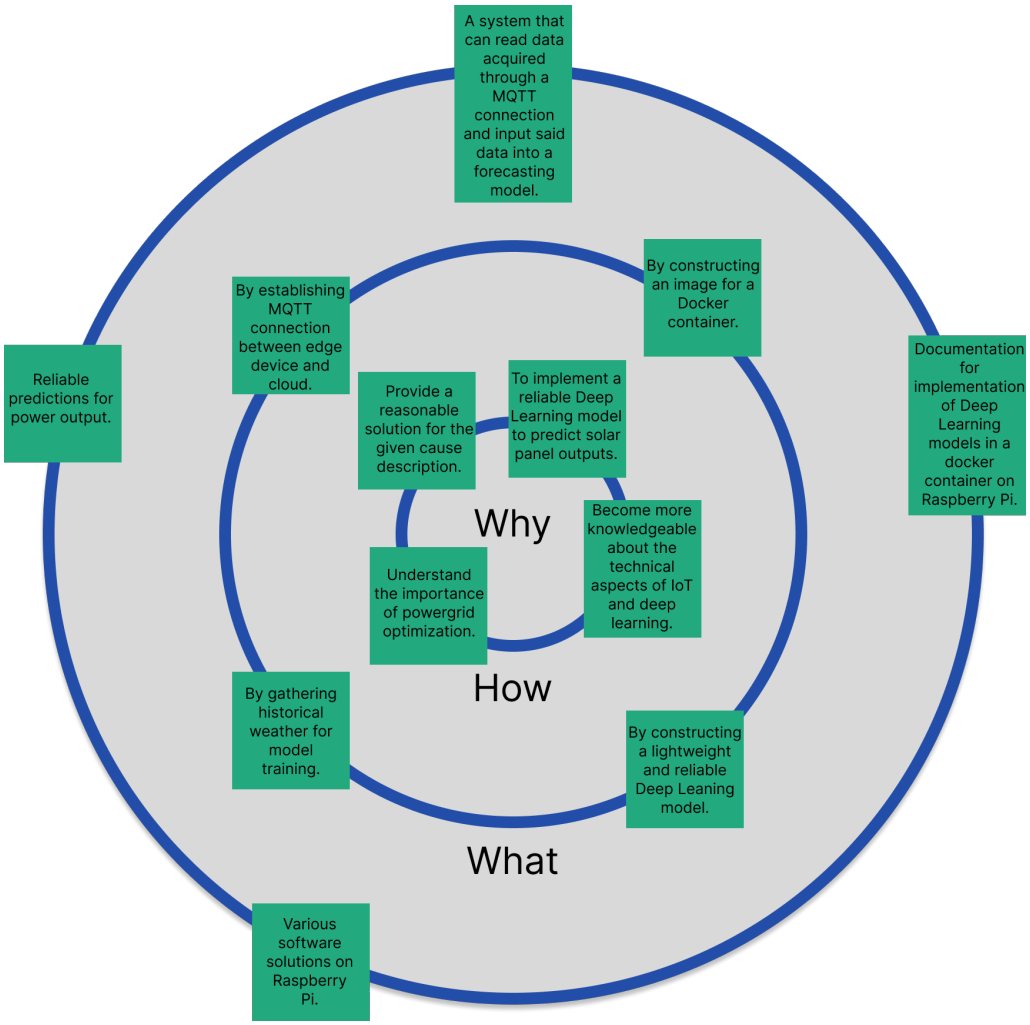


Figure 2.1: Golden Circle with What, How and Why displayed

In order to visualize this aspect of the project(as seen in figure 2.1), the three question styles have been explored:

Why (why are we doing this project?)

- To implement a reliable Deep Learning model to predict solar panel outputs.
- Become more knowledgeable about the technical aspects of IoT and deep learning.
- Understand the importance of powergrid optimization.
- Provide a reasonable solution for the given cause description.

How (will we realize the project?)

- By constructing a lightweight and reliable Deep Learning model.
- By gathering historical weather for model training.
- By establishing a connection between the edge device and the cloud.
- By constructing an image for a Docker container.

What (is the outcome of this project?)

- Reliable predictions for power output.
- Documentation for implementation of Deep Learning models in a Docker container on Raspberry Pi.
- Various software solutions on Raspberry Pi.
- A system that can read data acquired through a connection and input said data into a forecasting model.

2.3 Uncertainty

When discussing uncertainty in project management, we will look into risks and unknown factors that might occur throughout the project runtime. Of course, we cannot predict if a risk actually will materialize, but we can identify its characteristics and take precautions against it. The characteristics of risks can be outlined:

- It can be anticipated and identified.
- It can be combated by reducing the likelihood that it will materialize.
- Countermeasures can be implemented to lessen the harm caused by it.

Identifying as many potential risks as possible and developing thoughtful, reliable, and adaptable plans to deal with these uncertainties is crucial. In order to satisfy the project objectives, three components had to be implemented in a functional cohesion. This introduces three elements of uncertainty, as all three are codependent and equally necessary for the final project. One must assume that unforeseen events will inevitably occur, and there are three steps in the risk assessment process used to break down risks:

- Identify risks
- Evaluate risks
- Treat risks

The various risks associated with this project that could materialize during its execution are listed in table 2.1.

ID	Risk	Impact (0-5)	Probability (0-5)	Score (impact * probability)
R1	Docker not working	5	1	5
R2	Unable to produce Bash Script	2	1	2
R3	Deep learning model unable to compile	5	2	10
R4	No historical data for deep learning model	4	3	12
R5	Docker dependencies not successfully implemented	4	4	16
R6	No connection to the MQTT broker	5	2	10
R7	Hardware sensor component malfunction (DHT, Relays etc.)	2	2	4
R8	Raspberry Pi malfunctioning	5	2	10
R9	Long model compilation/prediction time	3	3	9
R10	Neogrid device not working	5	1	5
R11	Team members get Covid19	3	2	6

Table 2.1: Brainstorm for risks and risk score

A risk matrix was created after outlining the various risks, along with their impact and probability:

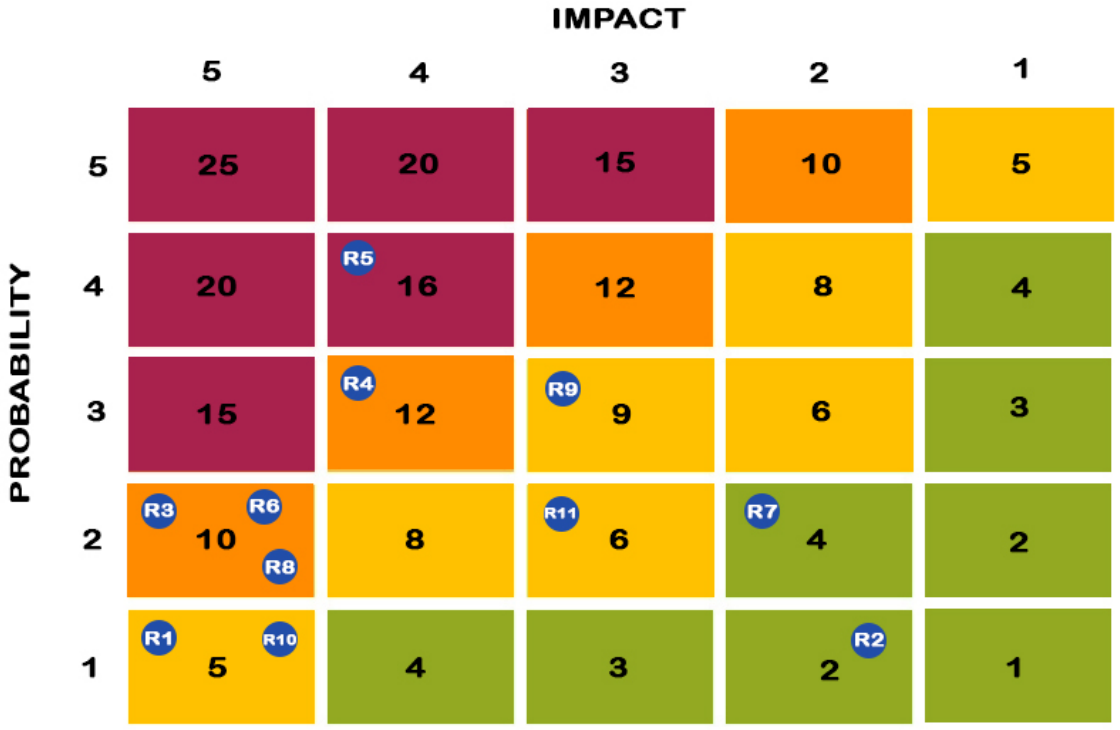


Figure 2.2: Risk Matrix with Risk plots

After brainstorming and mapping the risks on the Risk Matrix, it was noticed that many risks were placed in the orange and red fields, which was worrying. As a result, concentrating on developing an emergency plan for risks rated ten or above was conducted.

Neogrid Gateway device is not working
Establishing a link between the edge device and the Neogrid gateway was one of the project's goals in order to guarantee a reliable cloud connection. As the device is provided to us by Neogrid, we can contact Anders if any issue with the device should occur.

Raspberry Pi Malfunctioning
For this assignment, we were given a Raspberry Pi 4 Model B; fortunately, we had backups in case the main Raspberry Pi malfunctioned.

No historical data for Deep Learning Model
Accurate historical data is essential for the model to shape itself in any constructive way. Therefore, it is extremely important to acquire historical weather data since there is no model to be built without it. If it is impossible to acquire any historical data, other weather data sources should be consulted to secure data for the model. Other sources could be publicly accessible weather data collection sites or APIs. Since there could be a real chance of either not accessing the data due to, for example, GDPR restrictions, or the data could be acquired late in the project, making it challenging to use optimally.

Long model compilation/prediction time
This is a risk worth considering while constructing the model. Since the Raspberry Pi is a somewhat limited platform, it is important to adjust the complexity of which the model will possess. Ensuring a simple structure with reasonable capabilities should be kept in mind during the model construction. Parameters, layers, and further configurations should be used with consideration.

Docker dependencies not successfully implemented
When creating a successful Docker image that is able to run deep learning models, one needs to be very careful about which libraries and dependencies that are used in building the Docker environment. If it is impossible to run the model on Raspberry Pi within a compatible Docker environment, other measures, such as running the model online or separate from the Docker, were considered.

2.4 Time Management
When keeping an overview of the project, we used a management tool called Asana[link], a web platform that helps organize, track and manage projects and workflows. The Asana board was categorized into different parts, where IoT, Deep Learning, and Raspberry Pi(Docker) were the three most essential parts filled with many tasks. We even had a daily board for tasks that needed to be done immediately.

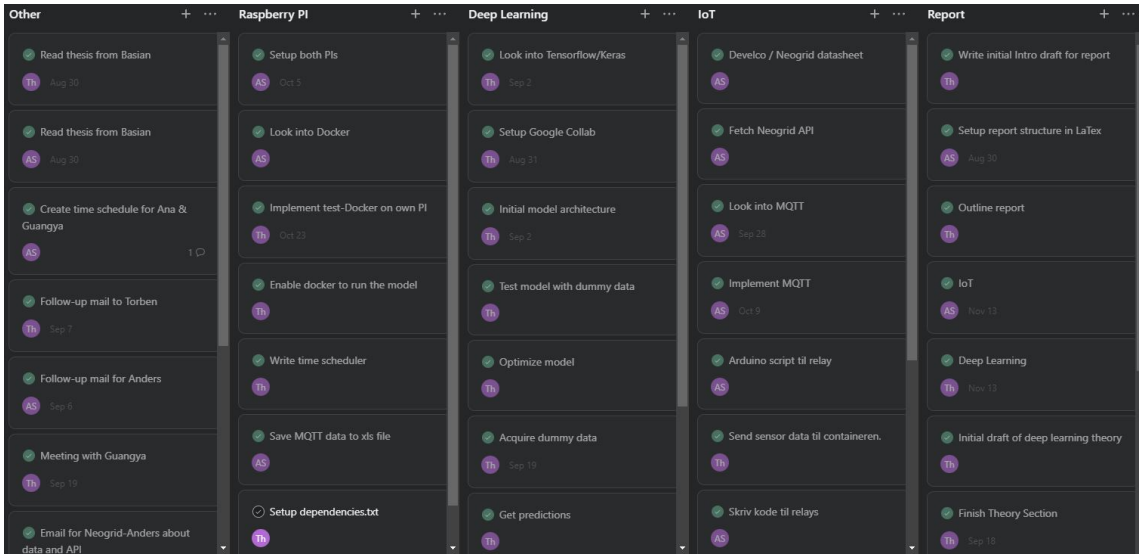


Figure 2.3: Asana work management program | overview of the project

In order to secure time to review and adjust the report thoroughly, it was from the beginning decided within the team that the final product, as well as a comprehensive draft of the report, should be produced a week prior to the official deadline.

Our meeting times were fixed to 9 am, and sometimes we also worked during weekends to catch up on tasks proving to be more time-consuming than expected.
An example of this occurrence could be implementing the Deep Learning model into Docker containers, which proved to be much trickier than previously thought. Another significant bottleneck in the project was communication through emails back and forth. Communication through this medium took considerable time to reach any conclusion and hence slowed down the project's progression.

3 Theory and Research

The following chapter will contain a thorough review of the research, theory, and techniques that were studied which gave us the tools used to develop our solution. The reader is presented with the fundamental theories and architecture of IoT, MQTT, Raspberry Pi, deep learning, and fundamental concepts in model construction.

3.1 Internet of Things

Information Technology(IT) has been at the front line for creating, accelerating, and automating how businesses are run today. Therefore, IT is an essential key factor in any business, and with the new advances coming to market, the IoT is one of the technologies that have gained tremendous exponential growth. This has been achieved by allowing Machine to Machine (M2M) communication and thus bringing users and smart devices closer.

As Oxford dictionary defines IoT:

"IoT is network of interconnected computing devices which are embedded in everyday objects, enabling them to send and receive data."[7]

As of the current time, there are approximately more than 13 billion devices online. However, the number of devices is expected to reach a stunning 30 billion in the near future.[8]

The primary characteristics of IoT can be defined by 7 traits[9]::

- Connectivity
- Identity
- Scalability
- Data/Intelligence
- Safety
- Self Configuring
- Architecture

Connectivity
In IoT infrastructure, establishing a connection anywhere at any time is essential. With connectivity between the internet and things like cell phones, routers, sensors, and other gadgets, it is necessary to have a strong and stable connection.

Identity
A device will have a unique identity when creating an IoT device and connecting it. This form of identification is often helpful in tracking and identifying one's equipment.

Scalability
Scalability is an absolute necessity for a successful IoT integration. While the number of devices connected to the "IoT Zone" is rapidly increasing, one needs a scalable IoT infrastructure to handle a growing number of devices.

Data/Intelligence
Gathering and processing data is also a significant aspect of IoT and the first step towards creating intelligent decisions/actions with regard to artificial intelligence.

Safety
When all of the users' devices are connected to the internet, there is a risk that their sensitive personal information will be compromised. Consequently, the main difficulty is data security. The safety of the equipment is, therefore, important.

Self Configuring
As software often needs updates to newer versions, it is important for IoT devices to upgrade their software to meet standards with the least amount of user involvement.

Architecture
The IoT architecture focuses on being inclusive and open to all devices and users, meaning ordinary people are able to do their own IoT projects if needed. Further exploration into IoT architecture is done in the following section.

An IoT device does not need to fulfill all the characteristics above. However, most of them need to be satisfied as IoT devices are computing devices such as computers or microcontrollers that are able to connect wirelessly to a network and which are capable of sending and receiving data.

3.1.1 Archeticture
IoT Architecture consists of four primary building blocks, Sensors, Processors, Gateways and Applications[10].

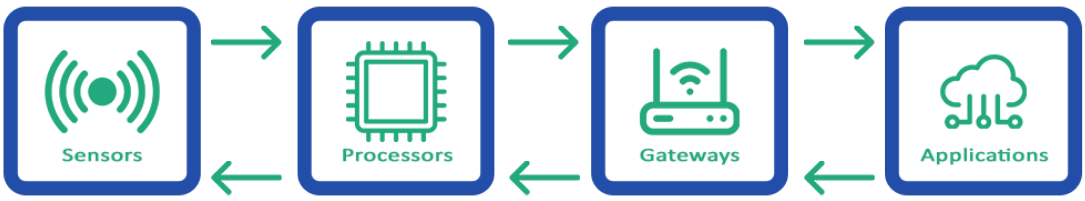


Figure 3.1: IoT Building Blocks

Sensors
In IoT, the sensors are referred to as "things", which make up the front end. Their primary responsibility is to gather essential information from the environment and transfer it to processing systems. Because they are the primary front-end interface in the vast network of other devices, they must be able to be located and identified. In addition, sensors can collect real-time data and operate automatically or under user instructions. Examples of sensors would be temperature sensors, humidity sensors, etc.

Processors
Processors act as the brain of the IoT system, like computers and other electrical systems. However, the primary responsibility of processors is to transform the raw data generated by the sensors into helpful information for the end user. In short, it provides data insight.

Microcontrollers and embedded circuitries are examples of processors that are able to translate the given sensor data to create data insight.

Gateways
The primary function of gateways is to transport processed data to appropriate databases or online storage for proper use. In other words, a gateway facilitates data transfer. For IoT devices to function, communication and network connectivity are necessary.

Gateways include Local Area Network(LAN), Wide Area Network(WAN), Personal Area Network(PAN), etc.[11]

Applications
Another component of an IoT system is applications. Applications effectively gather data and allow the user to engage with relevant information. These programs, which may be cloud-based, are in charge of displaying the gathered data.

Smart home apps such as Phillips hue are an example of an application. The application is able to display the state of the light bulbs and control all light integrated within the system.

To summarize the four build blocks, the sensors collect and send raw data to the processors. Then, by using gateway device connectivity, processors convert unstructured data into useful information before transferring it to distant cloud-based apps or database systems. The data can subsequently be transferred to the applications for proper data insight and big data analysis.

3.1.2 Communication Models
During the research phase, four primary IoT communication models were examined. These models of communication were as follows[12]:

Request and Response Model
To obtain information, the client must submit a request, which typically is encoded to secure a level of security. The server receives the request and reacts accordingly by retrieving data resources or declining unauthorized access. When the client has access, the server will fetch the appropriate data and send it back to the client as a response. An example would be how a printer operates by doing certain tasks as per request by the connected client.

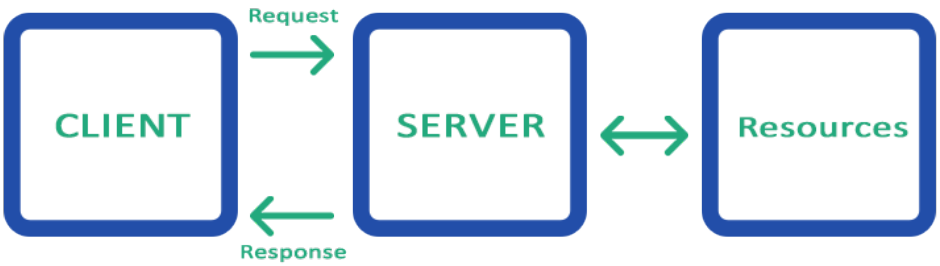


Figure 3.2: Request and Response Model

Push-Pull Model
The Push-Pull model consists of 3 units, publishers, queues, and consumers. Data is released by publishers and posted online. On the other hand, consumers pull the online data. Queues serve as the buffer between the two endpoints. This buffer handles mismatches between the publisher push rate and the consumer pull rate.

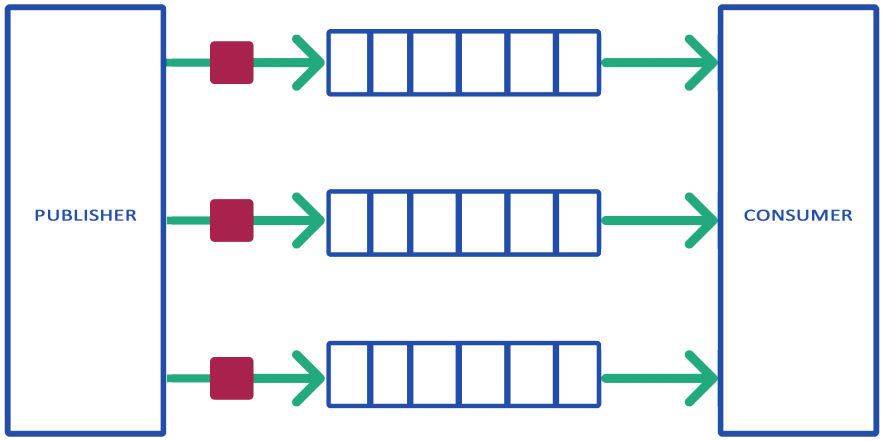


Figure 3.3: Push-Pull Model

Exclusive Pair
This concept incorporates complete dual communication between client and server and is an IoT bidirectional communication model. Up until the client sends a request to cancel the connection, nothing happens to the connection; it stays open. An example would be Websocket-based communication[13], which is a method to open a two-way communication with two or more parties.

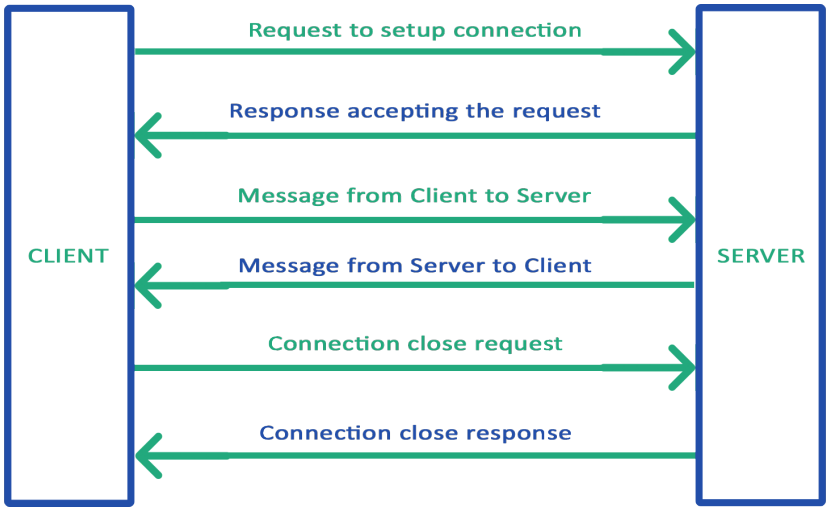


Figure 3.4: Exclusive Pair

Publisher-Subscriber Model
The Publisher-Subscriber communication model is employed when using the Message Queuing Telemetry Transport(MQTT). This model will be further explored in the following section about MQTT.

3.2 MQTT

MQTT is a lightweight publish/subscribe messaging protocol[14] that is favorably employed in situations with constrained bandwidth resources. An alternative server-client architecture is the publish/subscribe model; in server-client models, the client connects directly with an endpoint, as described under the communication models section. The publish/subscribe model, on the other hand, differs in that the publisher and subscriber never get in touch with one another. A third device, the broker, instead handles the link between the two. The following image shows the MQTT communication:

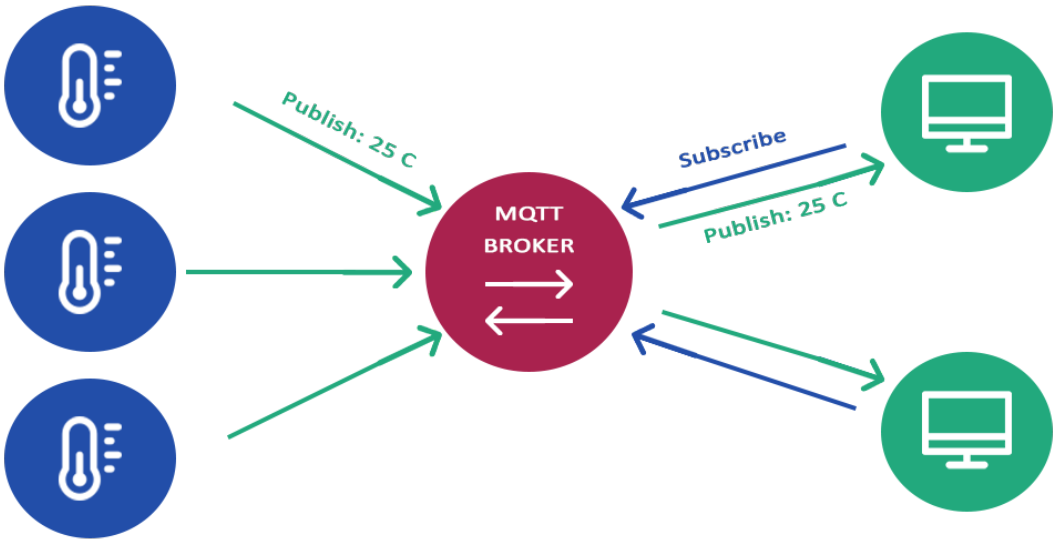


Figure 3.5: MQTT

In the IoT space, MQTT is a commonly used messaging protocol, and in order to fully comprehend its theory, one must understand its publish/subscribe pattern[15]:

3.2.1 Publish

After an MQTT client is connected to a broker, it can start publishing messages. The messages on the broker are filtered using a topic-based approach by MQTT. Every message must have a subject so the broker knows to which client to forward it to. Each message typically has a payload that contains the information to send in byte format. MQTT is not data-specific. How the payload is structured depends on the client's use case.

Details about a few characteristics of a MQTT Publish message[16] will be discussed:

Figure 3.6: MQTT Publish package attributes

Packet ID
A message is uniquely identified as it travels between the client and broker by the packet identifier(as seen in figure 3.6). Only Quality of Service(QoS) levels greater than zero make use of the packet identification. The client library and the broker set this internal MQTT packet identifier.

Topic Name
The topic name is a string that has a hierarchy built into it. Such a string might look like ”esp32/dht/temperature”, which is a topic level with a slash separating each level. A client can subscribe directly to the temperature topic, and while subscribed, they are able to obtain the temperature readings from a given temperature sensor. It is a good idea to keep in mind that topics alter depending on how it’s written; therefore, ”Esp32/Dht/Temperature” and ”esp32/dht/temperature” are two different topics. Wildcards are also possible, where a client can connect to multiple topics by using ””. So an example would be there were two topics ”/esp/dht/temperature” and ”/esp/dht/humidity”, to subscribe to both, one might use wildcards as ”/esp/dht/” to subscribe to both.

QoS
A crucial component of the MQTT protocol is QoS. Different levels from 0 to 2 will indicate the degree of guarantee that message will reach its intended destination. These QoS levels are described as such:

- 0 - Level message is sent with no acknowledgement from the broker
- 1 - Level message is sent with acknowledgement from the broker
- 2 - Level message is sent with a handshake (4 message between sender and receiver)

Retain Flag
The flag designates whether the broker saves the message as the last value for a certain topic. This last flagged value will be published to a new client when they subscribe to it.

Payload
The payload references to an actual message that has been sent. The payload can be text, images, encrypted data, or even data in binary. Anything contained in a message will be sent as a payload over the MQTT connection.

DUP Flag
The duplicate flag(DUP flag) denotes a duplicated message delivered after the intended receiver (a client or broker) failed to recognize the first message.

3.2.2 Subscribe

A message that has been published is not worth much if there is no one to read it. In other words, the data would be useless if no clients subscribed to the message topic since the data would not be stored anywhere. Therefore, in order to subscribe to messages on specific topics, the client sends a subscribe message with a topic destination to the MQTT broker. The broker will then allow the client to retrieve the temperature data from the requested topic.

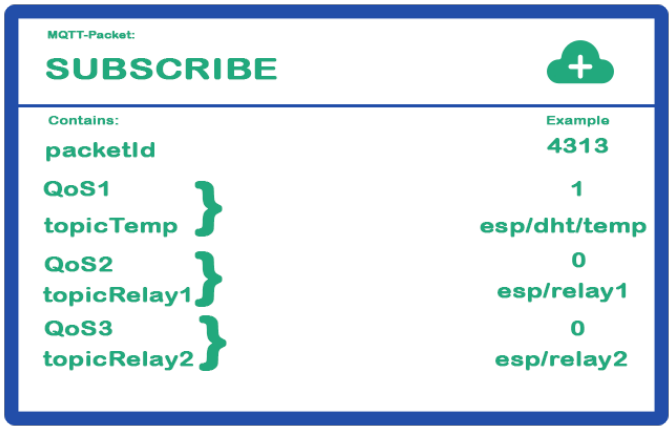


Figure 3.7: MQTT Subscribe package attributes

In the SUBSCRIBE message(See figure 3.7). Finally, an ordinary subscription would consist of a packet ID, topic, and a QoS level, to be able to subscribe to the MQTT broker.

3.3 Deep Learning

3.3.1 Description

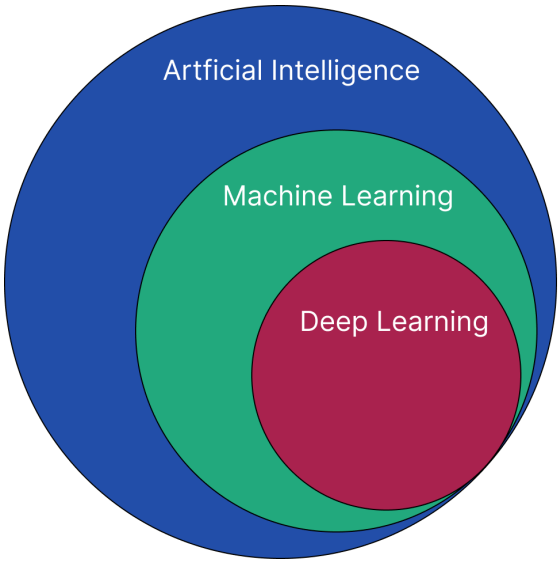


Figure 3.8: Overview of Artificial Intelligence, Machine Learning and Deep Learning

Deep Learning (DL) is a somewhat new addition to the burgeoning world of Artificial Intelligence (AI). An important distinction to emphasize is that while DL is AI; AI is conversely not DL but the overarching subject in which DL dwells.[17] It is a subset of Machine Learning (ML) to specify even further. It differs from previous methods by utilizing an artificial neural network to emulate the learning process of a conventional brain. This approach enables learning with no human interference and no explicitly ascribed patterns. A point that separates DL from ML is the amount of data these two approaches may require. ML can be fed a comparatively sparse amount of data, while DL necessitates much larger quantities and a diverse set of data. With a large amount of data, DL algorithms can improve through repetitions instead of ML, where human intervention is required to bolster precision. DL can be considerably heavy weight due to the amount of data and model complexity that it employs. Under the umbrella term DL, the models employed can contain layers of algorithms and computing units, referred to as the hidden layer/layers or a neural network. The hidden layers are located between the input and output layers of a given DL model.

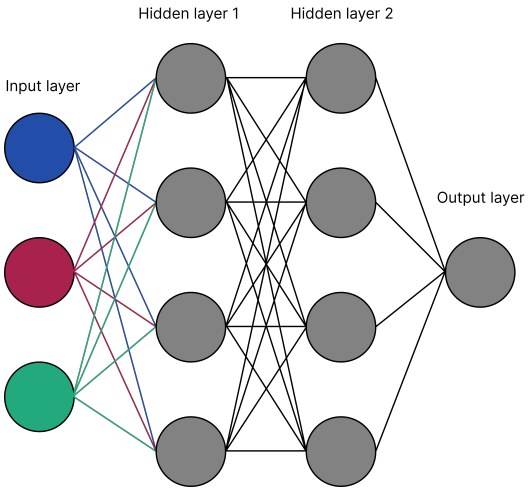


Figure 3.9: Overview of a neural network

Some models introduce an even greater amount of complexity, like Recurrent Neural Networks(RNN), where feedback cycles are created between nodes. This could, for instance, occur between two hidden layers, where the outcome of the current data point is affected by the previous data point. This is especially useful when working with dependent data since it allows the model to retain some memory while processing new information, thus outputting a more cohesive pattern. [18]RNNs are not the only architectures; in fact, there is a great variety of other structures, such as Deep Reinforcement Networks, Convolutional Neural Networks, and Deep Neural Networks.

3.3.2 Application

[19]Deep Reinforcement Network is applied in healthcare, natural language processing, and even in the world of trading and finance. IBM uses a Deep Reinforcement Network based platform capable of trading and calculating loss or profit per transaction. [20]Convolutional Neural Networks(CNN) are most commonly used for image classification, such as satellite image analysis or handwriting recognition. It can also be used for speech and language recognition. [21]Deep Neural Networks (DNN) is favorably employed to analyze time series data and can be used for forecasting. DNNs have also shown promising results in neighboring fields such as image classification and automatic speech.

3.3.3 Relevance to task

On this basis, it becomes apparent which network to employ for this project. Since there is going to be a series of predictions on a continuous stream of time series data, DNN becomes a promising candidate. At this point, the most suitable DNN architecture is to be narrowed down and described later in this report.

3.3.4 DNN outlining

[18]This group of neural networks always consists of the following components: neurons, synapses, weights, biases, and functions.

Neurons are nodes to pass on information through the network. The information passed can be described with a real number value, usually in the range between 0 and 1. Neurons may also be ascribed a weight, which can vary through the learning process. This weight influences the strength of the neurons' output signal that is sent downstream. Synapses

exist between neurons and are used to transmit signals that also can be weighted through the network. [22]Weights are made to value signal strength above others and vice versa inside the hidden layer. Weight is a parameter that is acquired through learning, and its effect is multiplicative. [23]Bias differentiates itself from weight by having an additive effect, e.g. Adding a flat value that is applied to the signal strength passing through a synapse. Bias is another parameter that is acquired through learning and is added after weight is multiplied by the signal strength. Biases are not influenced by the previous layer either. A high value of bias means that the model is making assumptions about the form of the output, whereas a low bias suggests a lower degree of assumption.

A more technical way to understand how this network plays out is to dive into the mathematics of how this complex network transmits and calculates signals.

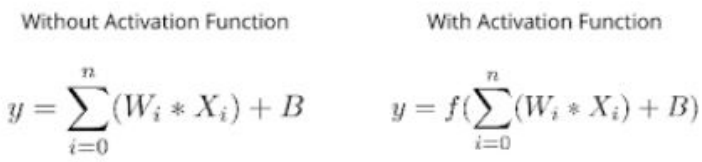


Figure 3.10: System with and without activation function

W_i represents the weight of each individual neuron output from the previous layer.
X_i represents each neuron's actual value outputted from the previous layer.
B represents the bias added after the weight multiplied by the neuron signal has been calculated.
f represents the Activation Function and can be configured when constructing a nerual network architecture.

[24]When a neuron takes input from two or more sources, this source could be neurons from the previous layer; the input values are simply added together as such(See figure 3.11):

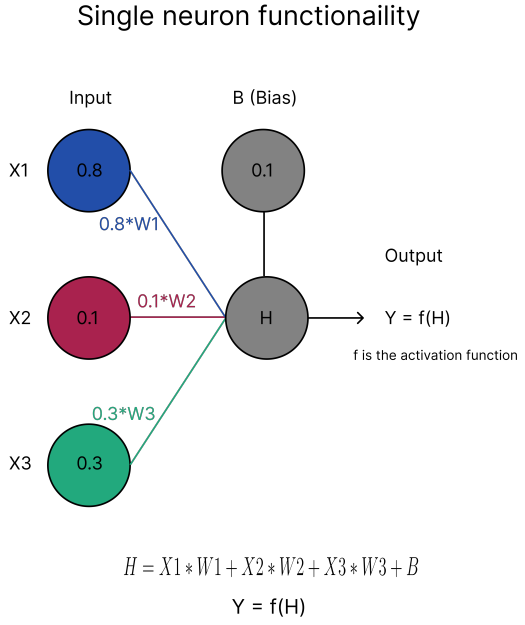


Figure 3.11: Neuron in DNN

Commonly, multiple neurons will exist in each of their respective layers. For example, eight neurons from layer one could be connected to each of the following eight neurons in layer two. Layer two could then produce the final result that is to be outputted from this hypothetical model. In this instance, vectors are usually employed to calculate the output of this model with its added complexity. This can be done as such:

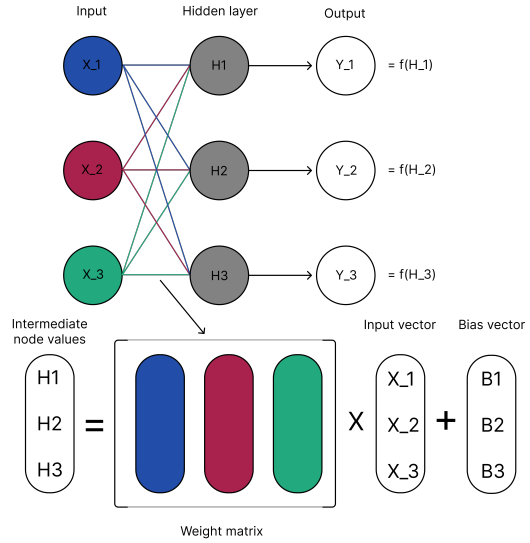


Figure 3.12: Input passing through layers in DNN

The weight vectors can be joined in a ”weight matrix”, which is then much like in the same fashion as previously demonstrated, multiplied by the current input vectors, whereafter the bias vector is finally added.

[25]Here the output will be in the form of a vector; this vector can contain one or more variables that are being predicted for. For instance, this could be determined not only to predict the color of a given fruit but also what type of fruit it is predicted to be. This is referred to as a multi-output model, but it is most common for the model to output a single value by having the final layer’s vector condensed into one single value(See figure 3.13):

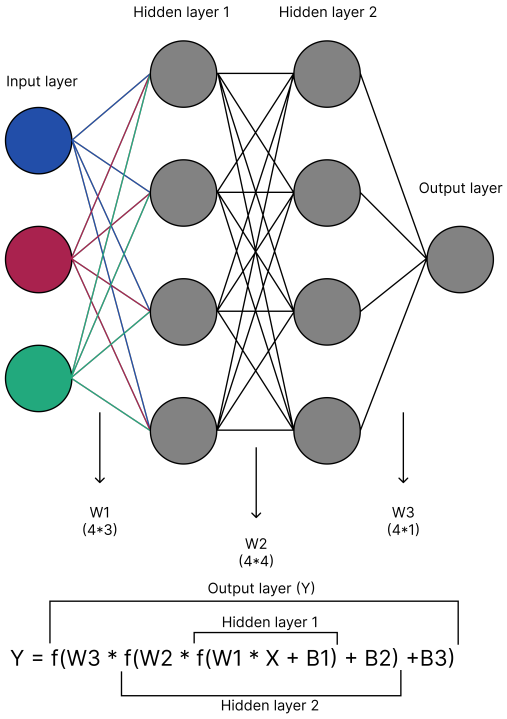


Figure 3.13: Whole DNN

3.4 Overarching concepts

3.4.1 Activation functions

[26]The activation function dictates how the summed input from one neuron to another is transformed. This transformation typically occurs between the layers, for example, from layer one to layer two in a neural network. The activation function may also be referred to as “The Transformation Function”, and it can come in a variety of different shapes. One could also refer to these functions as “The Squashing Function” when the function range is considered limited, or if the function is nonlinear, it could be referred to as the nonlinearity in the network design. For the sake of simplicity, these functions will only be referred to as activation functions.

[24]The Sigmoid function is a simple and common activation function used in instances where the output is desired to be squashed into a less polarized range. The output will be a value in the range of 0 to 1, where very low or negative input values will be transformed into 0, and very high values will be 1. [26]This function can be referred to as a squashing function, as it squashes all real numbers between -∞ and +∞ into a value between 0 and 1.

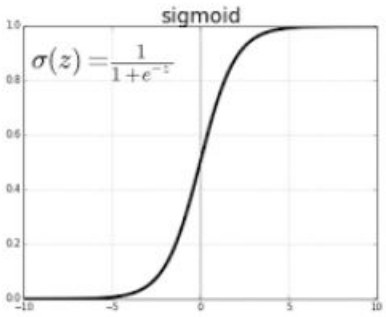


Figure 3.14: Plot of sigmoid activation function

The Sigmoid Function paired with a hidden layer can enable a neural network to learn nonlinearly separable problems. This means that the model will be equipped with a greater sense of nuance when grouping classifications and even taking account of divergence within the classifications. This could be illustrated as such:

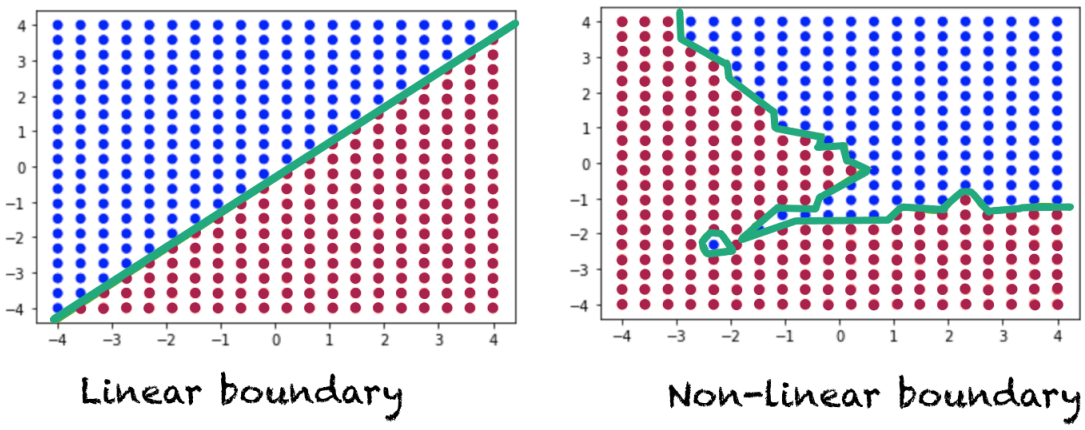


Figure 3.15: Equation of signal strength

With a linear boundary, there is an apparent distinction between the two groups and not much room for outliers within these groups. However, with nonlinear learning, the model is able to recognize even more complex patterns to classify. This is a reason as to why the Sigmoid Function is so common in neural networks.

[27]The Hyperbolic Tangent Function can be an alternative to the aforementioned Sigmoid function and is likewise a nonlinear activation function. It is similarly simple and is also a common activation function for neural networks. It does not quite squash the input to the same extent as the Sigmoid but outputs in the range between -1 to 1.

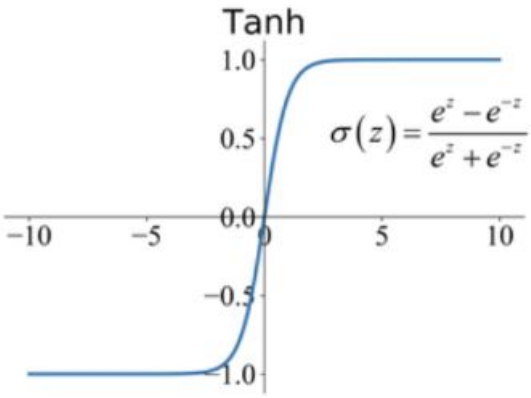


Figure 3.16: Plot of sigmoid activation function

The main difference between the two activation functions is the behavior of the gradient. The gradient can be derived and plotted as such(See figure 3.17):

$$s'(x) = s(x) * (1 - s(x))$$
$$tanh'(x) = 1 - tanh^2(x)$$

Figure 3.17: Equation for sigmoid and tanh

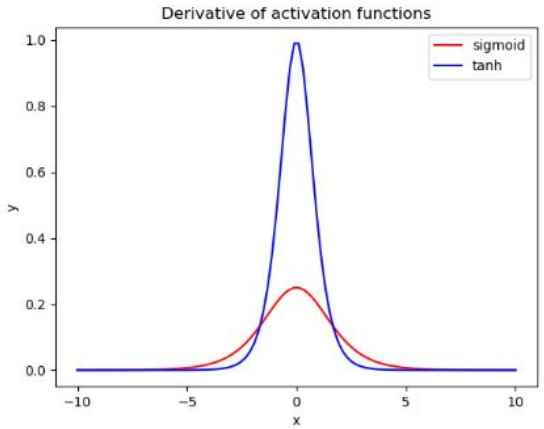


Figure 3.18: Plot of sigmoid and tanh gradient

[28]When either of the activation functions are employed in a neural network, the data values will usually be centered around 0, as seen in the plot above. The hyperbolic tangent gradient appears to be considerably greater than the gradient of its counterpart. This means that higher gradient values are attained during training, which in turn means that bigger learning steps are acquired with this activation function.

Both activation functions, unfortunately, suffer from an issue described as the Vanishing Gradient Problem. This problem alludes to an instability within the network during training. It refers to a situation where a recurrent neural network or deep multilayered feed-forward network fails to propagate useful gradient information. This issue thwarts the models' capability to properly learn from a given dataset and leads to poor results.

This leads to a third and also commonly used activation function called the Rectified Linear Unit, or reLU in short. This function seeks to overcome the vanishing gradient problem and thus bolster models' performance and learning capabilities.

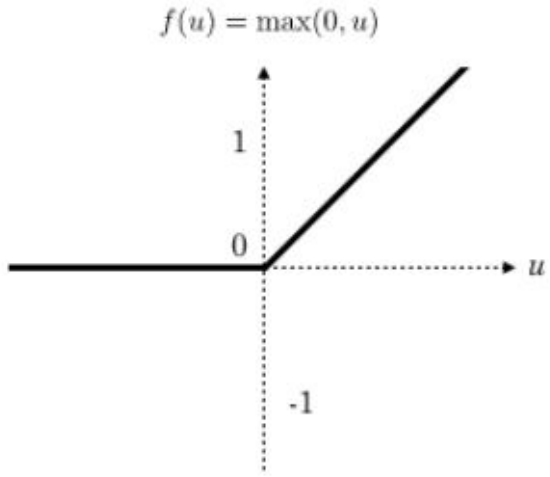


Figure 3.19: Plot of reLU activation function

This function only squashes any negative value into 0, while any positive value is returned just the same. For example, -3 becomes 0, and 10 will be returned with no transformation applied. This function is recommended for multilayered architectures such as CNN and Multilayer Perceptrons. ReLU suffers from an entirely different problem than the two previously mentioned functions(Sigmoid / tanh), which is described as the 'exploding gradient problem'. [29]This problem is the opposite of the vanishing gradient problem in the sense that gradient values become oversaturated. More specifically, error gradients are calculated during training, and the larger the values, the larger updates are made to the network weights. At some point, the updates will become too radical, and the performance will worsen. At extremes, the weight values will become so bloated that they will overflow and return NaN. Error gradients are calculations of the direction and magnitude within the network, and they are supposed to update network weight throughout a training process. This exploding gradient problem arises due to reLU not squashing positive values, which means that exponential growth to the weights can occur when gradient values above 1 are repeatedly multiplied through the network.

[29]One way to relieve this problem is to initialize the network with small weight values.

3.4.2 Optimization algorithms

Another important step in achieving a capable model for reliable predictions, beyond choosing a suitable architecture and suitable activation functions, is to find a fitting optimization algorithm.

[30]An optimization algorithm is supposed to minimize errors occurring when mapping input to output. By minimizing errors, the overall performance and accuracy of the model are expected to improve. This kind of algorithm is applied during training and will modify various attributes of the network. This could, for example, be by modifying the weight of each epoch in a way to improve the model. [31]An epoch is when the entire dataset is passed forward and backward through the entire network one time. Entire datasets can be passed through the network multiple times to train the model, but there is no universally correct amount of times this is to be done. The right amount of epochs, therefore, varies from context to context. Generally speaking, using one epoch could lead to underfitting the model, while using an exorbitant amount will lead to overfitting. The sweet spot between these two states, called optimum, is what a model should aim for.

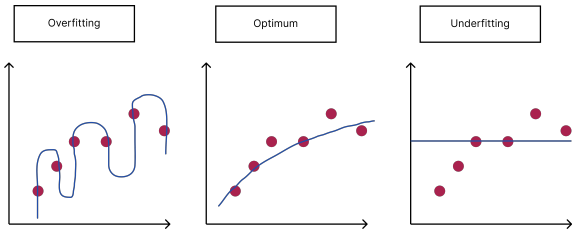


Figure 3.20: Model overfitting, optimum and underfitting plotted

There is a variety of optimization algorithms, for instance:

The Gradient Descent (GD)

Modifications are consistently made to values throughout training in order to achieve a local minimum. [32]In concept, the algorithm wants to step in the opposite direction of the gradient since this direction leads to the steepest descent; hence the name GD. If the algorithm were to step in the same direction as the gradient, it would be called Gradient Ascent.

$$x_{new} = x - \alpha * f'(x)$$

This is how the GD is calculated. Alpha represents the step size of how many steps will be taken against the gradient per iteration. This algorithm has been a reliable optimization algorithm over the years, but it does have its downsides. If the dataset is vast, which a lot of neural networks benefit from, it can become considerably expensive to run.

Stochastic Gradient Descent (SGD)

[30]This algorithm works the same way as the aforementioned GD algorithm, but it addresses its inherent costliness by introducing a stochastic element into the equation. Stochastic means ‘random’, so instead of running through the entire datasets, only random batches of the data are processed as in GD. Firstly initial parameters are selected weights and learning rate, whereafter, the dataset is shuffled and randomly selected in batches per iteration. Through this process, a local minimum will be approximated.

$W_i\ w := w - \eta * \nabla * Q_i(w)$
One weakness of this algorithm is that due to the random nature of how batches of data are picked, the path of which the algorithm takes is usually full of noise compared to the more robust manner of the normal GD algorithm.

Adam
[30]The curious name, Adam, is derived from Adaptive Moment Estimation and can be considered to be a further optimization of SGD. Instead of maintaining a constant learning rate like SGD, Adam updates the network learning rate through the training. Adam comes with a variety of benefits, which is why it is recommended as a default optimization algorithm as a starting point. It has a comparatively shorter running time, requires less tuning than other algorithms, and has lower memory requirements.

$\mathsf{m}_t = \beta_1 * m_{t-1} + (1 - \beta_1) * [\frac{\delta * L}{\delta * w_t}]$
$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * [\frac{\delta * L}{\delta * w_t}]^2$

This does not mean that Adam is the best optimizer algorithm out there since the best choice depends on the context. For example, Adam is focused on computation time, while the costlier SGD generally is better focused on data points. Therefore the optimization algorithms should be picked depending on the requirements of the model, rather than merely defaulting to just one every scenario.

4 IoT Integration

This chapter will focus on the IoT Integration process. First, a description of the system design from Bornholm will be outlined, and a rough draft of this project's system will be described. Moreover, the MQTT connection will be established. Then, step by step, the system will be upgraded, starting with a clear connection with the client and broker, adding the temperature sender, and creating a two-way communication with the relays.

4.1 Design

4.1.1 ECL210

At Bornholm, the Direct Heating system(DH) used is the ECL Comfort 210 (ECL210) from Danfoss[33], which is a DH controller. The ECL210 is an electronic heat controller which can regulate how much DH is used by how much DH systems for heating and cooling consume in up to two sections. For example, the ECL210 setup can be seen in Figure 4.1.
The different measurements in the ECL210 application:

- S1 Outdoor temperature
- S2 Return temperature for room heating
- S3 Room heating forward temperature
- S4 Forward temperature, (DHW)
- S5 Temperature of S5 DH return (Room heating)
- S6 Temperature of S6 DH return (DH)
- P1 and P2 Pumps for circulation
- M1 and M2 Motorized control valves

Section 1(Blue) of the heat exchangers helps by heating the water for room heating, while Section 2(Green) heats water for Dometic Hot Water(DHW). Room heating and the DHW how their own sections and are isolated from the DH supply by heat exchangers, which is a system used to transfer heat between two or more fluids of different temperatures. The system is autonomous and can change the temperature set-point for the room heating measurement (S3) in response to various factors, such as the outside temperature measurement.

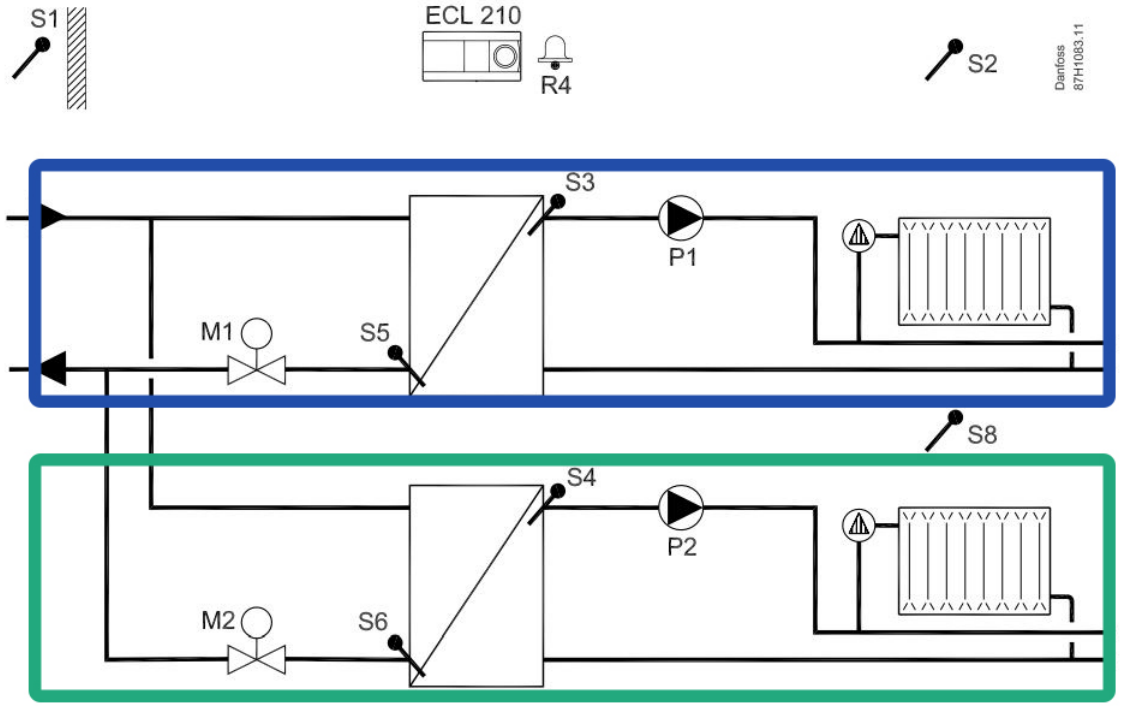


Figure 4.1: ECL210 with 2 Circuits outlined, picture taken from [34]

Two motorized valves (M1, M2) situated at the return point of the two heat exchangers regulates how much DH water is allowed to pass through the heat exchanger. When the DHW temperature sensor (S4) detects a value below the appropriate set-point, M1 will open further. When the temperature is above the appropriate set-point, M1 will close further. The ECL210 controller operates the secondary side circulation pumps P1 and P2 as well as the DH input valves M1 and M2 based on the measurements S1 to S6.

4.1.2 ECL210 Emulate

Due to difficulties in accessing the ECL210, it was decided that an emulation of the system would be created to test and satisfy connective capabilities between sensors and the Raspberry Pi.

In the project system, a DHT11 sensor will be used as one of the temperature sensors(S1 - S6). Two relays will function as the motorized control valves (M1 and M2).

4.2 Setting up the Raspberry Pi

4.2.1 Raspberry Pi

For this project, a Raspberry Pi 4 model B was provided, which is a microprocessor with a 1.5 GHz 64-bit ARM CPU, 8 GB RAM, and the capability to connect with Wifi, Bluetooth, or Ethernet.

The Raspberry Pi is set up with a Debian 10 64-bit image. This is done by loading the image onto a SD card; in this case, it was done with the Raspberry Pi Imager program downloaded from Raspberry Pi's homepage[35] When the SD card is inserted into the Raspberry Pi, VNC is used to connect to it. VNC is a tool that can be used to access a Raspberry Pi's graphical user interface(GUI) remotely through a Secure Shell(SSH) connection. This made the process of working on the Raspberry Pi more convenient, as opposed to only working in the Command Line Interface(CLI).

4.2.2 Mosquitto Broker

For this project, Eclipse Mosquitto[36] was employed, which is an open-source message broker that uttilizes the MQTT protocol.
Using a publish/subscribe approach, the MQTT protocol offers a simple way to conduct messaging. As a result, it can be used for IoT messaging with low-power sensors or portable devices like smartphones, embedded computers, or microcontrollers.

After preparing the Raspberry Pi running headless with VNC access, the Next step is to install the Mosquitto Broker by entering the following command in the terminal:

```
sudo apt install -y mosquitto mosquitto-clients
```

When mosquitto is installed, it is supposed to start on every Raspberry Pi boot-up. This is done by enabling the ”mosquitto.service” command that secures initialization at each start-up.

```
sudo systemctl enable mosquitto.service
```

Moveover, when all that is done, we can test if the broker har running by writing:

Moreover, when enabled, one can test if the broker is running by writing:

```
mosquitto -v
```

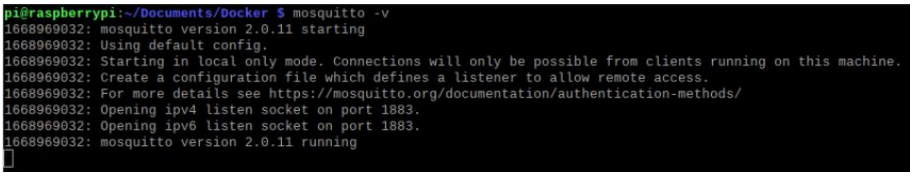


Figure 4.2: mosquitto -v command

4.3 Testing Broker and Client Connection on Raspberry Pi

After connecting the Raspberry Pi, one can check for a clear connection by setting up a mosquitto MQTT broker topic, then opening a new terminal and writing the following line:

```
mossquitto_sub -d -t test-sub/pub
```

After that, subscribers are able to connect to the topic, where they can post messages that are visible to other subscribers. For example, a string message is sent using the following command:

```
mossquitto_sub -d -t test-sub/pub -m "Hello from DTU"
```

By setting up the subscriber in one terminal, listening to the following topic ”test-sub/pub” (Window 2), and writing a message from the other terminal(Window 1), a test was completed to successfully show a connection between the two.

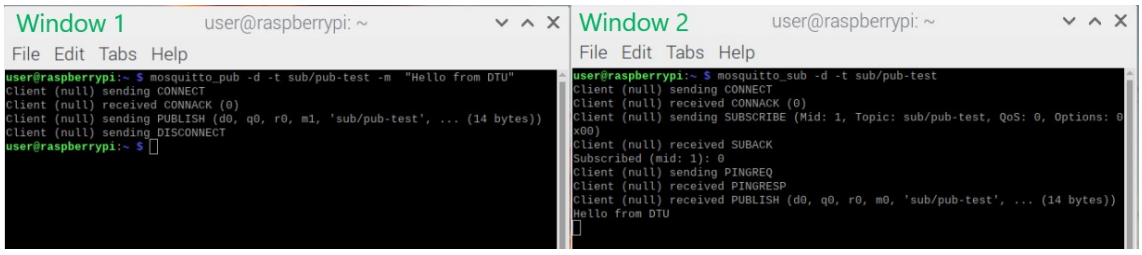


Figure 4.3: MQTT

The ”Hello from DTU” message will be delivered to the subscribed client since its subscrbied to the ”sub/pub-test”-topic.

Its demonstrated that MQTT works, and other devices will then be added (such as the ESP8266 or ESP32, etc.), which may subscribe to the same topic to receive messages.

4.4 Setting up the ESP8266

The ESP8266 is a low-cost system on a chip(SoC), which has a Wi-Fi transceiver implemented that uses the TCP/IP model[37].

Its 11 GPIO(General Purpose Input/Output) pins make it ideal for sensor connections and data transferring. Due to its low cost and good functionalities, it is often used in prototyping different IoT devices. Furthermore, its cross-functionality with Arduino makes it easy to program with the Arduino IDE[38].

When programming this ESP8266, the Arduino IDE will be used.

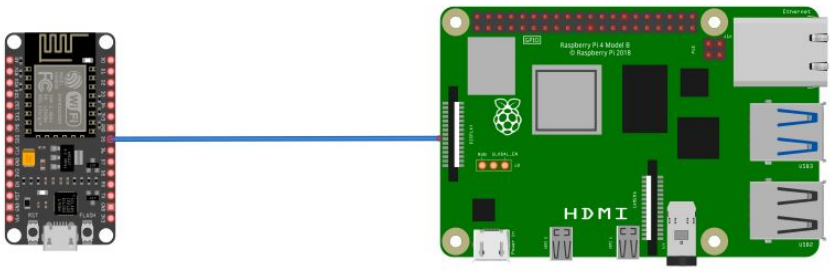


Figure 4.4: MQTT connection | ESP8266(left) as client, Raspberry Pi(right) as Broker

4.4.1 ESP8266 Code

Firstly the project-specific libraries should be included. These libraries contain the Wi-Fi library (because the ESP8266 connects to the internet), and the PubSubClient library, which creates the MQTT messaging protocols.

```
// Start by including the libraries
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
```

Next, the Wifi name and password are set to reflect the current network credentials. Then, to connect the ESP8266 to the required network, the wifi credentials will be modified to the corresponding Raspberry Pi, which functions as the MQTT broker. Finally, change the mqtt_server variable to the Raspberry Pi's IP address and let the port stay the same (because an MQTT broker's default port is 1883).

```
const char* wifi_name = "network name";
const char* wifi_password = "network password";

const char* mqtt_server = "10.0.0.13";
const char* mqtt_port = "1883"
```

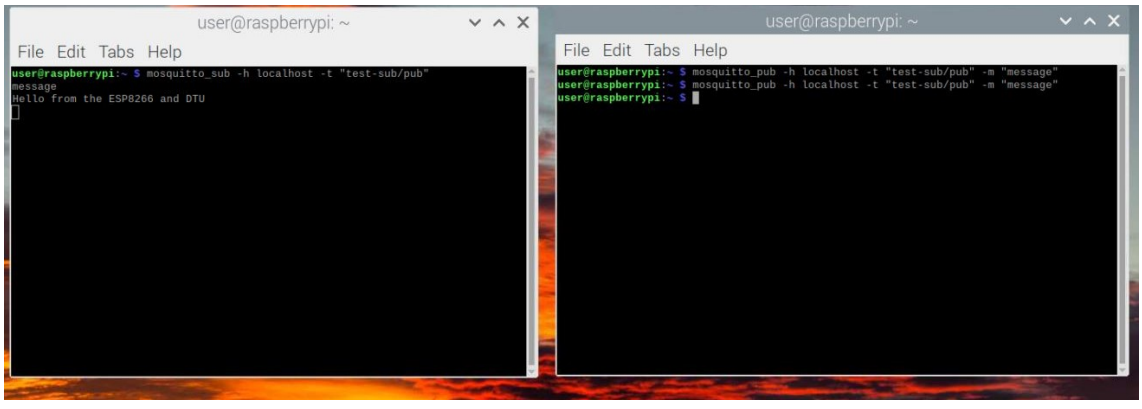


Figure 4.5: Able to send message from the ESP8266 to the Raspberry Pi Broker

4.5 Setting up the Temperature Sensor

One of the temperature sensors in the emulated ECL210 system will be the DHT11 temperature sensor, which will be able to use the ESP8266 as a processor to post temperature information from this DHT11 sensor to the MQTT broker.

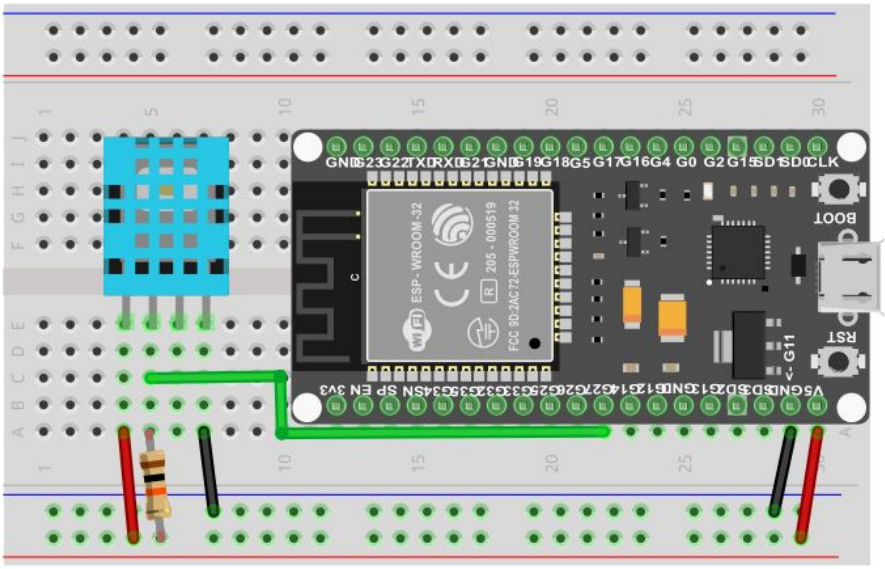


Figure 4.6: DHT setup

4.5.1 DHT Code

A necessary DHT library[39] has been added, supporting low-cost temperature/humidity sensors such as DHT11.

```
// Start by including the libraries
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"
```

```
#define DHTTYPE DHT11 // DHT 11
#define TEMP "/esp/temp"
const int DHTPin = 14; // DHT Sensor
DHT dht(DHTPin, DHTTYPE); //Initialize the DHT Sensor
```

In order to transfer data from the DHT sensor to the MQTT broker, being able to read the temperature is needed. Adding a void function called ”readTemp”, to the code ensures that the ESP8266 is able to retrieve data and use the ”/esp/temp” topic to publish said data. This function is on a loop and will continue to gather temperature data.[40]

```
void readTemp(){
 now = millis();
  // Publishes new temperature and humidity every 10 seconds
  if (now - lastMeasure >= readInterval) {
    lastMeasure = now;
    // Read humidity
    float h = dht.readHumidity();
    // Read temperature as Celsius
    float t = dht.readTemperature();

    // Error if the sensor fail reads.
    if (isnan(h) || isnan(t)) {
      Serial.println("DHT sensor faild to read!");
      return;
    }

    // Computes temperature values in Celsius
    float tempCalculate = dht.computeHeatIndex(t, h, false);
    static char temperatureTemp[7];
    dtostrf(tempCalculate, 6, 2, tempTemp); //turns floats into strings

    // Publishes Temperature as that is the thing we need.
    client.publish(TEMP , tempTemp);

    Serial.print("Temperature: ");
    Serial.print(t);
    Serial.print(" *C ");
  }
}
```

As seen below(Figure 4.8), the MQTT connection is established as messages from the ESP8266 are sent to the Raspberry Pi Broker. A new data reading is sent every 10 seconds and will update the ”/esp/temp” topic. In a real-world scenario, the reading would not be done every 10 seconds but more every 5 - 10 minutes, giving the user 144 to 288 temperature readings daily.

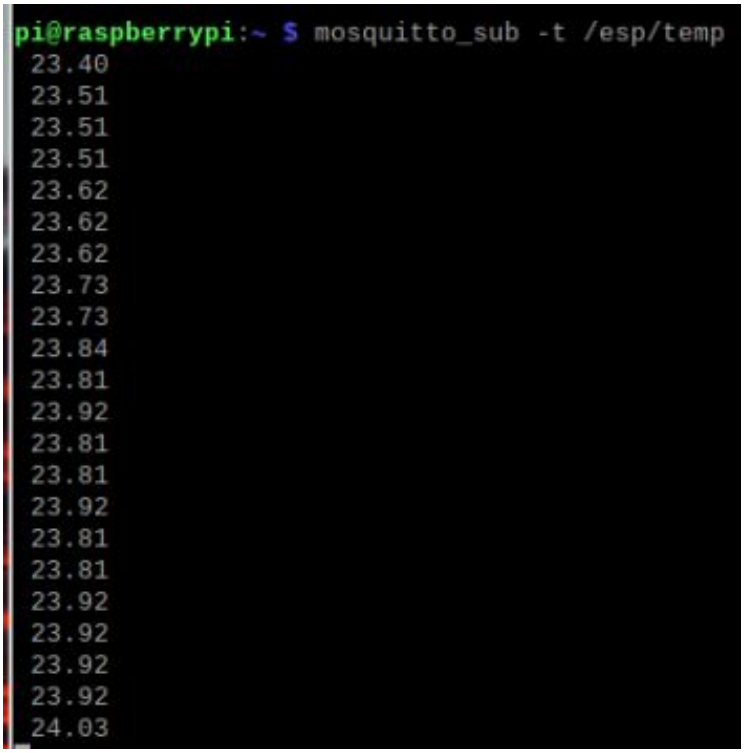


Figure 4.7: MQTT temp-data shown on the Raspberry Pi

4.6 Setting up the Relays

Relays are electrical switches which, when electricity is allowed to flow through them, are able to change state between on or off. The Relays illustrate the motorized control valves. At this stage, the setup for the publisher and the broker has been completed. Setting up the subscriber is now the final step. A new terminal is opened on the Raspberry Pi to act as a subscriber since it will function as an independent subscriber instead of setting up another device.

To expand on the previous code. Under the reconnect function, connections to the relays are established by subscribing to each relay topic.

```
1       client.subscribe("esp8266/relay1");
2       client.subscribe("esp8266/relay2");
```

As the subscriber needs to be able to control something through the MQTT connection, a callback function is added to create the required communication with the relays. When connected to the topic, sending either a ”1” or ”0” will change the state of the Relay to ”ON” or ”OFF”. ”ON” means when the motorized valve opens, and ”OFF” means when it closes.

```
void callback(String topic, byte* message, unsigned int length){
  Serial.print("You got a message on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String msgTemp;

  for(int i = 0; i < length; i++){
    Serial.print((char)message[i]);
    msgTemp += (char)message[i];
  }

  if(topic=="esp/relay1"){
    Serial.print("Changing Relay1 state to ");
    if(msgTemp == "1"){
      digitalWrite(relay1, HIGH);
      Serial.print("ON");
    } else if (msgTemp == "0"){
      digitalWrite(relay1, LOW);
      Serial.print("OFF");
    }
  }
    if(topic=="esp/relay2"){
    Serial.print("Changing Relay2 state to ");
    if(msgTemp == "1"){
      digitalWrite(relay2, HIGH);
      Serial.print("ON");
    } else if (msgTemp == "0"){
      digitalWrite(relay2, LOW);
      Serial.print("OFF");
    }
  }
}
```

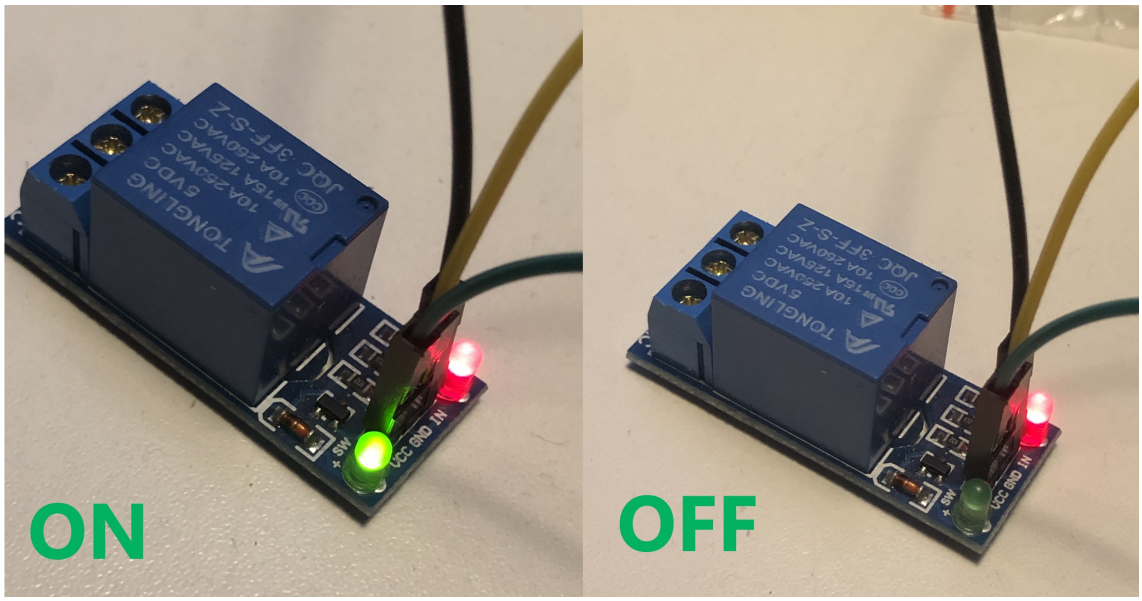


Figure 4.8: Illustration on and off capability

4.7 Finalizing the build

Every component has been combined to create the following final build(See figure 4.9 and 4.10). This build is able to send temperature data with the DHT sensor to another device.

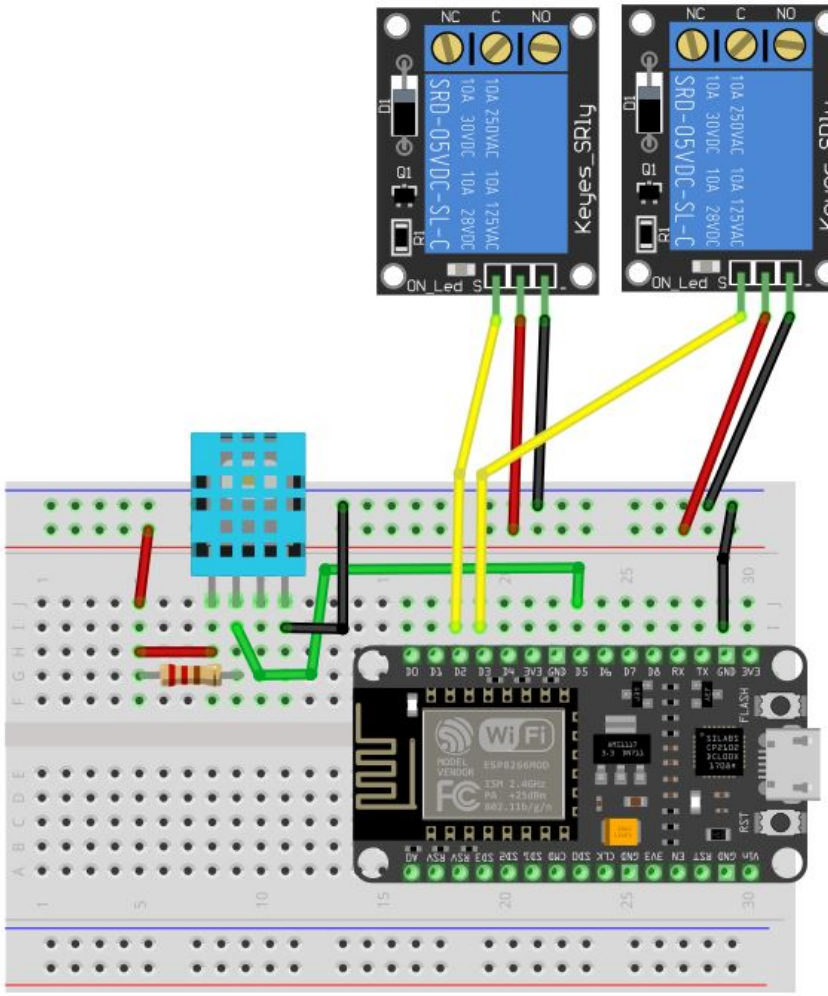


Figure 4.9: Final build

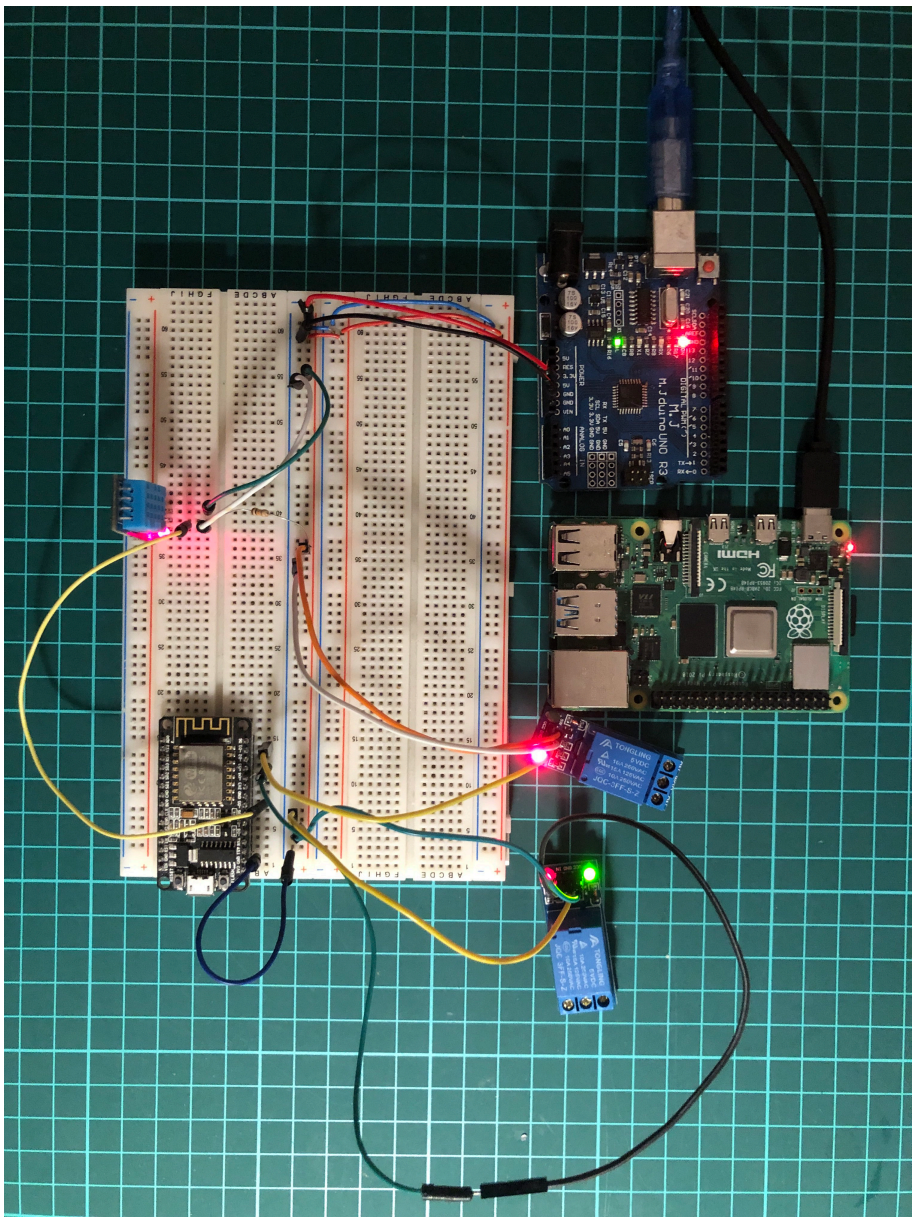


Figure 4.10: An overview of the final build - Real life

4.8 Creating the cloud connection

The MQTT connection was also able to establish a connection to a remote server instead of a local server. HiveMQ provided tools to test the MQTT broker through WebSockets[41].

```
const char* mqtt_server = "10.0.0.13";
\\changed to:
const char* mqtt_server = "final-diplom.mqttdashboard.com";
```

So by changing the mqtt_server variable to the HiveMQ hostname, a connection was able to be made. This showed that the Raspberry Pi broker could establish a connection

to local and remote devices. While both concepts are possible, there is still room for improvement.

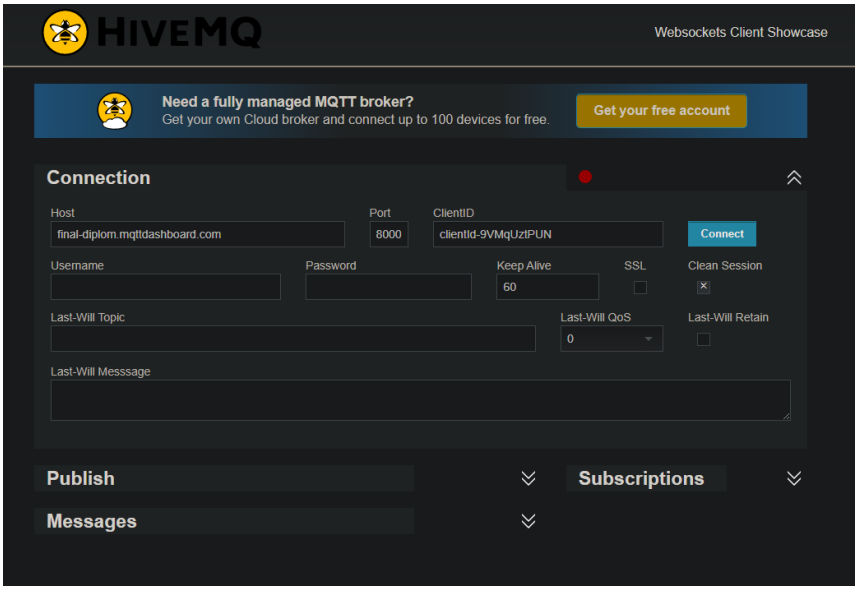


Figure 4.11: HiveMQ Dashboard where information can be added to establish connect

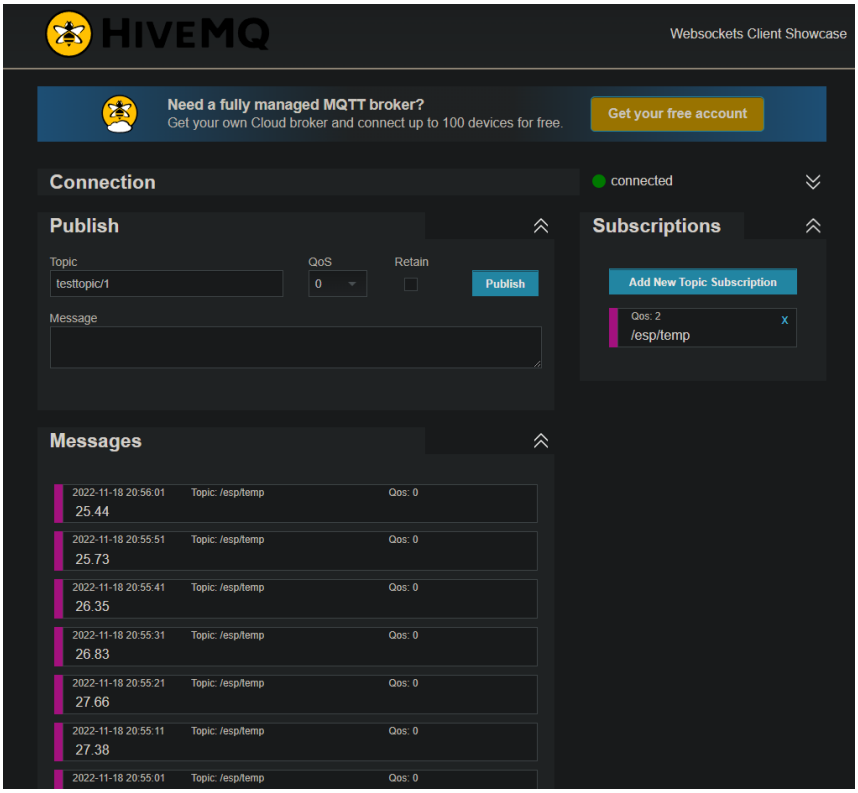


Figure 4.12: Anders receiving Data from DHT sensor placed in Andreas Local Network

5 Deep Learning implementation

This chapter will focus on the Deep Learning modeling process. A description of data preprocessing will explain its importance, as well as the role it plays in Deep Learning model construction. After that, features from the dataset will be discussed, which in turn will lead to a detailed description of how the project data was processed. Two variations of the chosen Deep Learning model will be compared to establish the most favorable variation. Lastly, the chosen model will then be outlined further in great detail.

5.1 Importance of data Preprocessing

Data preprocessing is arguably the most important step for constructing a reliable prediction model, as underlined by the common saying in computer science: “garbage in, garbage out”[42]. This phrase emphasizes the importance of productive and relevant data since it is used during the training phase and thus dictates how a Deep Learning model’s internal structure initially will form. Data preprocessing can refer to a variety of modifications done to the model-related data, whether it be training data, test data, or data used to make new unsupervised predictions. A common modification is to remove outliers from the data, this is especially important when dealing with the training data since outliers can skew the model predictions greatly. One should also be mindful of data that is either out of range, misleading or erroneous. For example, these obstacles could be overcome by removing negative values describing house incomes or discarding a picture of a boat, when one wishes to train the model to recognize cars. This process can be referred to as removing noise from the samples or cleaning up the data. Data quality assessment should go even further and also account for mismatches in values, such as if there is an arbitrary switch between ‘woman’ and ‘female’ in the data. Other potential errors to account for should be missing values and a mismatch between data types (e.g. using floats and strings in the same data feature).

When the data is all clean and coherent, there are still some final steps before the model is to be trained. Feature selection refers to how finding the model’s input features is conducted. There are two ways to do this; either through manual selection of perceived relevant features or through a pattern-searching algorithm.[43] The manual way can be referred to as supervised learning, while unsupervised learning refers to the alternative of features being highlighted through automatic means. This is a crucial step in constructing an efficient model since the selected features will be used to train the model and serve as inputs post-training phase. It is important to note that more added features do not equate to a more precise model. Feature characteristics may overlap, and the training process will be longer for each feature added. Sometimes this can lead to less accurate results. Normalization[44] is the process of scaling the numerical values into a tighter range, which will help the model compare values more accurately. There is a variety of different normalization techniques, some more suitable for handling outliers, while others can adjust the data into a range between -1 and 1.

Additionally, the data can be split into two different batches; a training batch and a validation batch. The training batch should contain the majority of the total data and will serve as the data used for training a model. The remaining data is stored in a validation batch, which is used to test the accuracy of the same trained model.

5.2 Model Implementation

5.2.1 Data features

Keeping the constrained platform in mind, the most correlated features for the model were incorporated through selection. In the training data following features were present:

- DiffuseSunPower kW/m²
- SunAltitude °
- DirectSunPower kW/m²
- DirectSunPowerVertical kW/m²
- SunAzimuth °
- Temperature °C
- DewPointTemperature °C
- WindGust m/s
- WindSpeed m/s
- Pressure hPa
- WindDirection °
- Cloudiness %
- LowClouds %
- HighClouds %
- Fog %
- Precipitation mm
- MediumClouds %
- Humidity %

Two features, in particular, were noted since they described the amount of power generated by a given solar panel; these two features are “DiffuseSunPower” and “DirectSunPower ”. DiffuseSunPower describes the generated power when incoming sun rays are obfuscated, while DirectSunPower adversely describes when there is no obfuscation. This diffusion of sun rays is most likely caused by the clouds in the sky, which to varying degrees, will dampen the panel’s power transmission. DiffuseSunPower was chosen due to having the strongest correlation with another feature.[45] Since Denmark’s sky, on average, is more cloudy than clear, this seems to be the most relevant feature to use as output. Out of the 17 features, “SunAltitude” appeared to be the most correlated feature, with a correlation value of 0.86. Followed by “DirectSunPower ” with a value of 0.68 and the third, “DirectSunPowerVertical ” with 0.61 value.

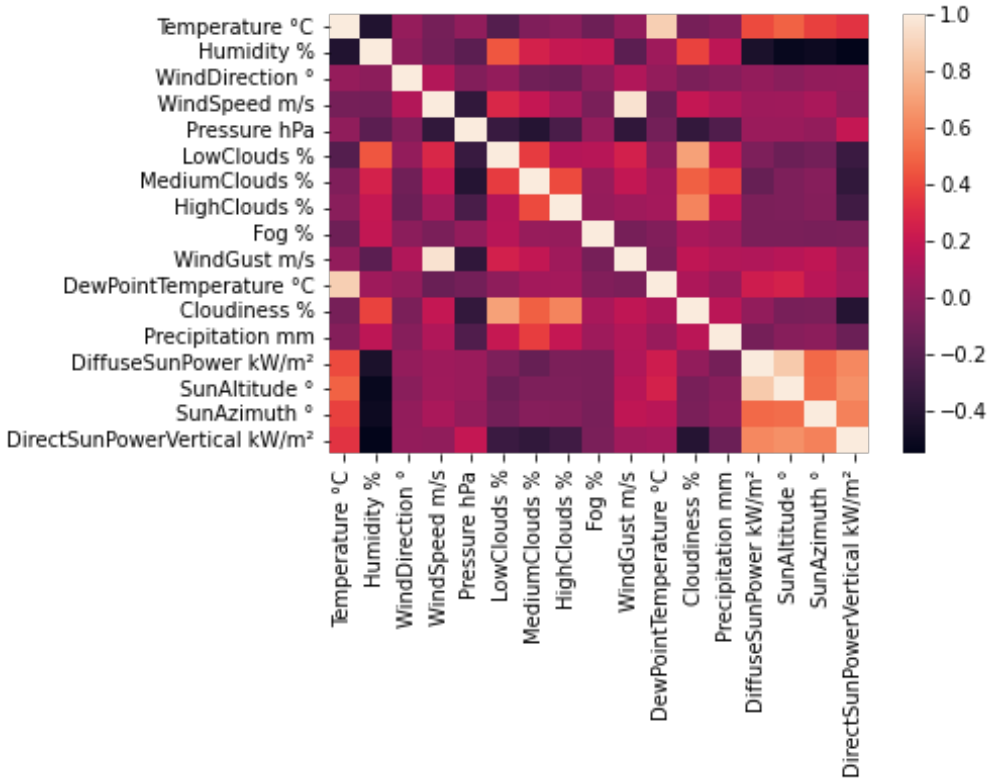


Figure 5.1: Picture of correlation meat map between variables

5.2.2 Data Preprocessing

Data inspection

The data does not appear to have any absurd values, nor are there any missing values. This means there is no need to remove rows with a missing data point or attempt to replace any missing data points.

Feature extraction

Since ”SunAltitude” is the most correlated, a new batch of data is created with only SunAltitude and the planned output feature (DirectSunPower).

	SunAltitude °	DiffuseSunPower kW/m²
0	0.512254	0.12
1	0.510858	0.15
2	0.466701	0.14
3	0.385194	0.12
4	0.275238	0.08
...	...	...
15248	0.157603	0.01
15249	0.039444	0.00
15250	-0.065275	0.00
15251	0.000000	0.00
15252	0.000000	0.00

Figure 5.2: Picture of SunAltitude and DiffuseSunPower dataset

Conversion
The ”SunAltitude” feature describes the angle at which the sun shines onto the solar panel; this value is presented in units of degrees. Degrees of units is converted into radians, as it is said to be easier for the model to work with.

Normalizing
To normalize the data, MinMaxScaler is used to adjust the data values. [46]MinMaxScaler is a scaling function from Sklearn's library. This approach scales the data into a number between 0 and 1. This is done through this equation:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Figure 5.3: Equation to calculate the MinMax transformation

It is a relatively easy process in python since it is done through an object. In this code segment, the object is instantiated, whereafter it is fitted with the data and then used to transform said data with the aforementioned equation:

```
inputTransformer = MinMaxScaler()
inputTransformer = inputTransformer.fit(train)
norm_train = inputTransformer.transform(train)
```

The object is handy since it is just as easy to reverse the transformation. This is used to inverse transform the model output predictions before saving them:

```
predictionInverse = outputTransformer.inverse_transform(prediction)
```

Split
The data collection is split into two batches; one batch for training, consisting of the initial 90%, and another for validation consisting of the remaining 10%. Training data is used during the model's training phase, while the validation batch is used to compare the accuracy of the trained model's outputs.

Reshape
Lastly, the data batches should be reshaped to conform to the model's expected input shape. This shape can vary from a vector of one to multiple dimensions. For example, a CNN, which typically is used for image classification, could expect a two-dimensional vector shape as input. Since this project involves time series forecasting, a RNN is utilized. More specifically, a GRU is used, which requires the input shape of fx. (none, 24, 2). This shape dictates how far back the model gets to look per iteration (24) and the number of expected features (2). The “none” indicates the number of input vectors, or in other words, the batch size is undefined. Undefined means that it could be set to anything, so if none = 20, it would mean that the input would consist of 20 vectors with a length of 24 and features of 2(see figure 5.4).

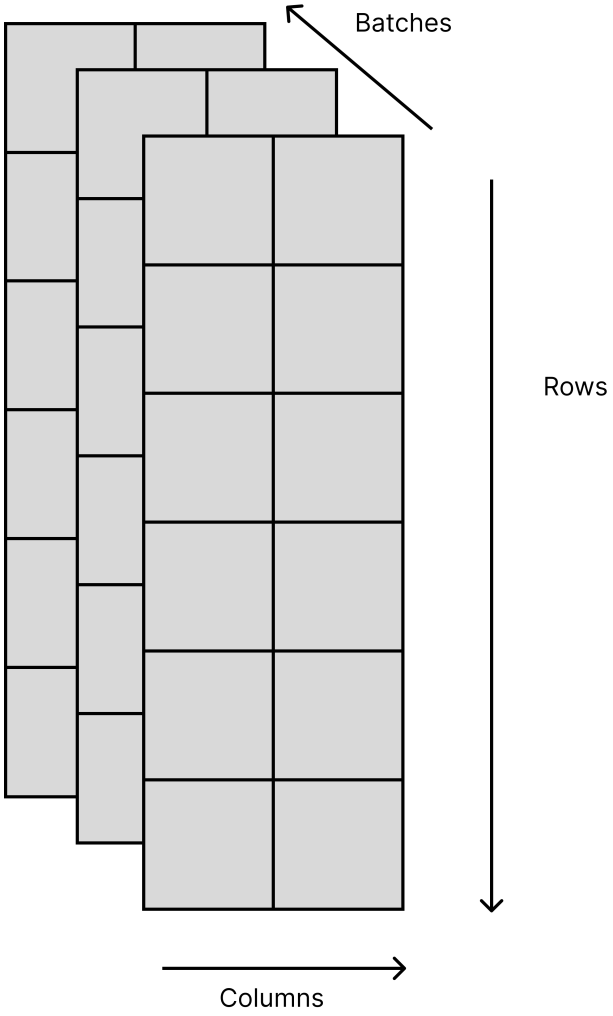


Figure 5.4: Illustration of Data batches

5.2.3 Model Construction

Model comparison

Initially, two models were considered for this project. These two models were the LSTM (Long Short-Term memory) and the GRU (Gated Recurrent Unit), which are commonly employed for Deep Learning forecasting tasks. [47]LSTM is a common RNN model to use for predicting time series; this can be due to its ability to process a whole data sequence. [48]Another RNN model referred to as a GRU, will also be considered since it is very similar to a LSTM, except it is more lightweight. This is due to the GRU having fewer parameters since it lacks an output gate. Though it is a simpler model, it has shown better performance “on certain smaller and less frequent datasets”, than the slightly more complicated LSTM. At the very beginning, these two models were compared with a variety of configurations, and at every configuration point, the GRU would consistently perform better and sometimes similar to the LSTM. Another important factor is also to consider model complexity with regard to the somewhat limited platform that is to run the model. As described earlier, the GRU is a lighter version of the LSTM. Based on these comparatively minor differences in performance and the GRU being of a lighter structure, the GRU seems to be the obvious choice for the project.

Another consideration was if there were any additional parameters that would improve performance. Time of day was proposed as a possible additional parameter since it would be related to when the sun is out. The time of day is converted into a sinus and cosinus wave in order to maintain data type uniformity in the data, which in this case, is a float value. This is achieved with the following operations:

```
dateColumn = pd.to_datetime(data['DATE'], format='%d-%m-%Y %H:%M:%S')  #
    convert the index to datetime
hours = np.array([x.hour for x in dateColumn]) #Get all hours
minutes = np.array([x.minute for x in dateColumn]) #Get all minutes
float_time = hours + minutes/60 #Set value expression for time of day
n_hours = np.unique(hours).shape[0] #Get all unique hour values
```

Yielding sin and cos values as can be seen on following plot:

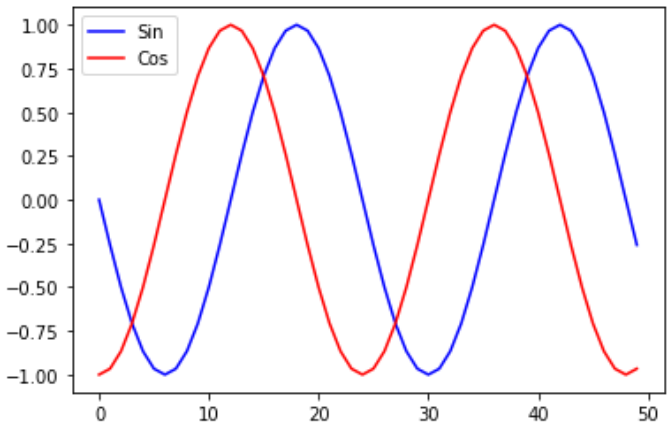


Figure 5.5: Sin and Cos wave representing time of day

Doing this would constitute two additional inputs into the model since both sin and cos waves would be needed as input, to give the model a clear picture of the time of day. Adding two additional variables for the model to consider will add more complexity and thus, longer computation time. It is, therefore, important to test how great the payoff will be.

The following section will compare the original GRU model with two inputs versus the potentially improved version with four inputs. Overview:

	GRU	New GRU
Input	Sun altitude Sun power	Sun altitude Sun power Sin value Cos value

This comparison will mainly be based on Mean Average Error(MAE) and r-squared values, as well as how much the models add to the project. The comparisons will be made with three different training configurations. These three rounds have been run multiple times to test for any general trends.

Following configurations that will be adjusted per round is:
conf_epochs - Describes the maximum amount of epochs the training-phase will be done for.
lag - Describes how many datapoints backwards the model is allowed to look through training.
n_ahead - Describes how many datapoints (hours) the model must predict for.
conf_batch_size - Describes the size of data batches the model will be loaded with.

1'st round Configuration:

```
optimizationAlgorithm = tf.keras.optimizers.Adam()
patience = 2
conf_epochs = 100
lag = 48
n_ahead = 4
model_units = 128
activation_function = 'sigmoid'
conf_batch_size = 32
```

GRU with 4 inputs yielded following results:

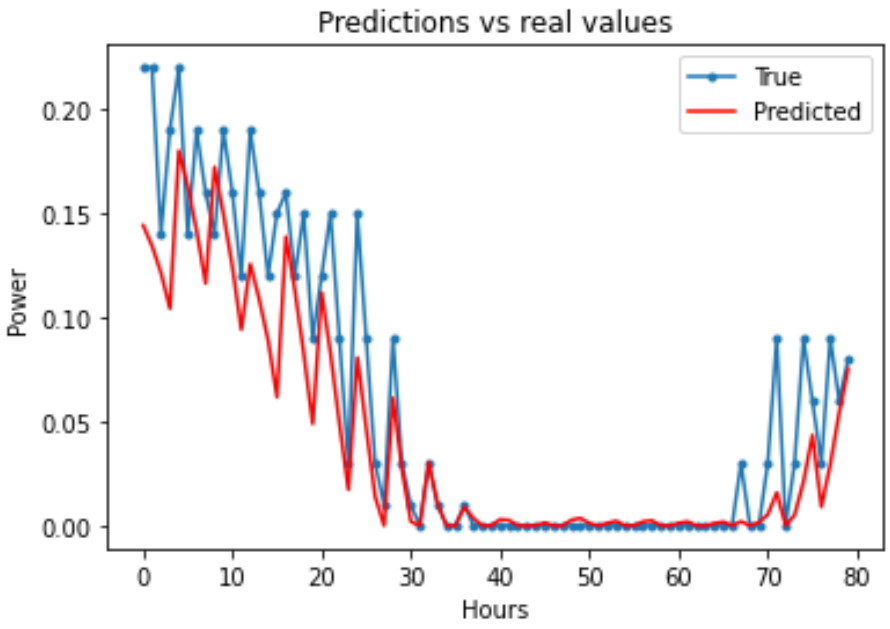
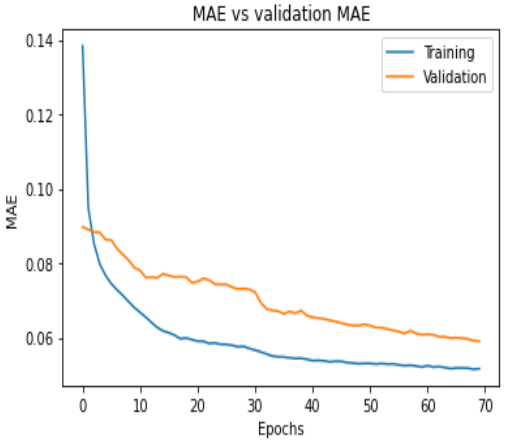
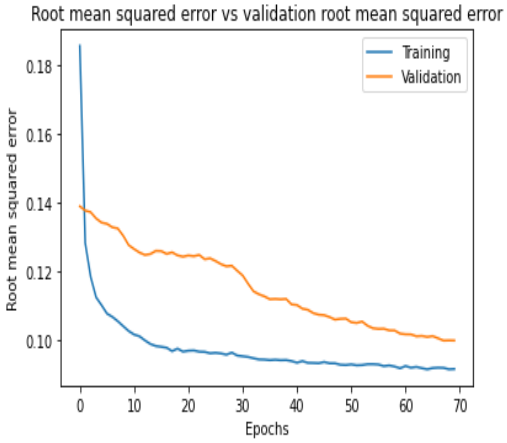


Figure 5.6: Round 1. GRU with 4 inputs: predictions vs real values

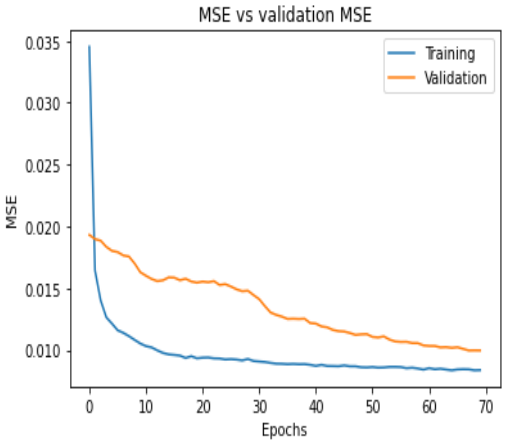
(a) Round 1. GRU with 4 inputs: loss



(b) Round 1. GRU with 4 inputs: MAE



(c) Round 1. GRU with 4 inputs: root mean squared error



(d) Round 1. GRU with 4 inputs: MSE

Figure 5.7: Round 1. GRU with 4 inputs: training metrics

Results of sklearn.metrics on prediction performance:

```
MAE: 0.11809979371226527
MSE: 0.030684987526327714
RMSE: 0.17349415176976546
R-Squared: 0.6404413012035481
MAPE: 32038222791767.684
ME: -0.07335055562876479
```

GRU with 2 inputs yielded following results:

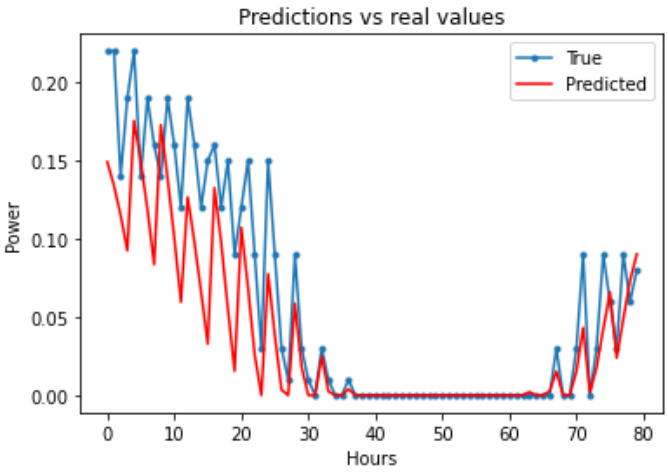
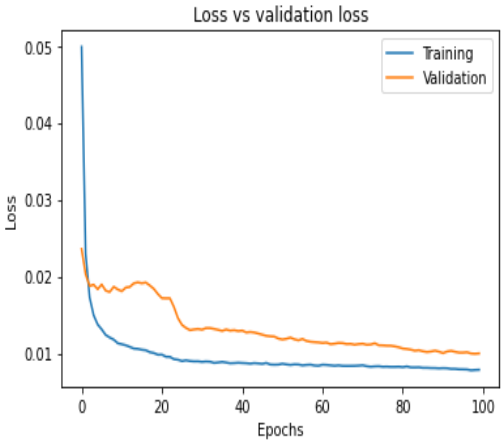
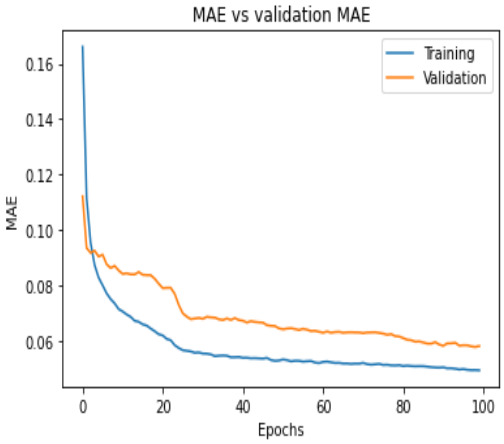


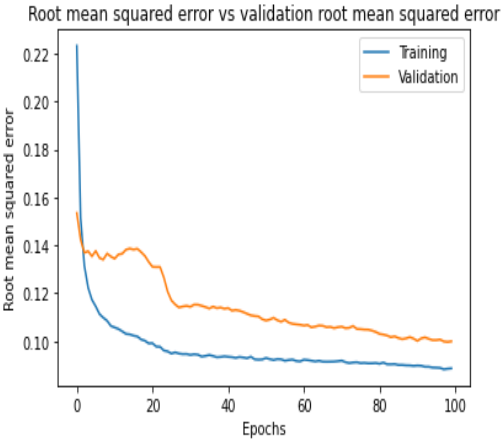
Figure 5.8: Round 1. GRU with 2 inputs: predictions vs real values



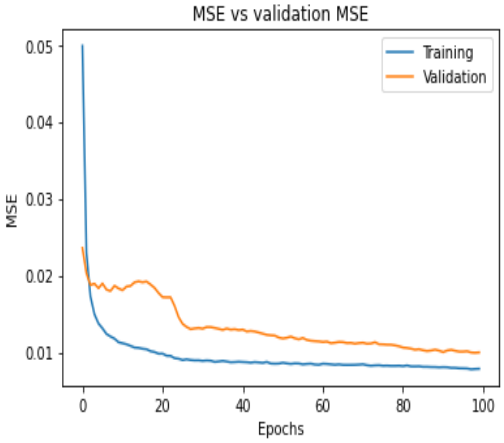
(a) Round 1. GRU with 2 inputs: loss



(b) Round 1. GRU with 2 inputs: MAE



(c) Round 1. GRU with 2 inputs: root mean squared error



(d) Round 1. GRU with 2 inputs: MSE

Figure 5.9: Round 1. GRU with 2 inputs: training metrics

Results of sklearn.metrics on prediction performance:

```
MAE: 0.1216890417399118
MSE: 0.03311438031377552
RMSE: 0.17935476232549444
R-Squared: 0.6119277893666595
MAPE: 32174190739656.234
ME: -0.08365573973804782
```

2'nd round Configuration:

```
optimizationAlgorithm = tf.keras.optimizers.Adam()
patience = 2
conf_epochs = 50
lag = 24
n_ahead = 4
model_units = 64
activation_function = 'sigmoid'
conf_batch_size = 24
```

GRU with 4 inputs yielded following results:

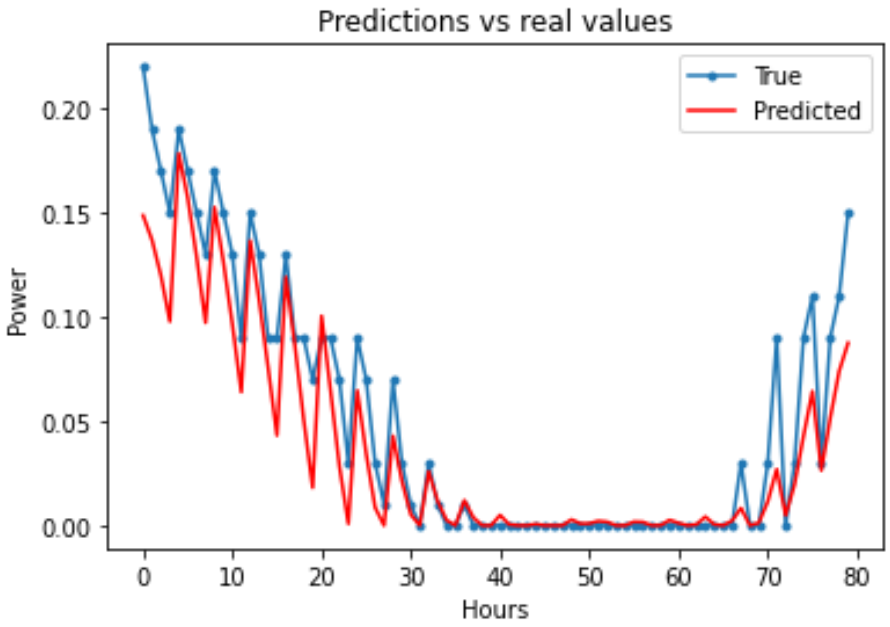
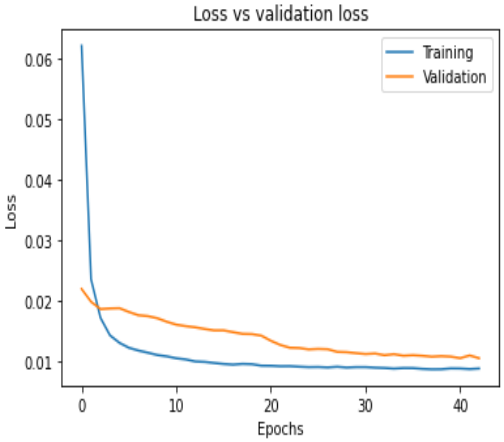
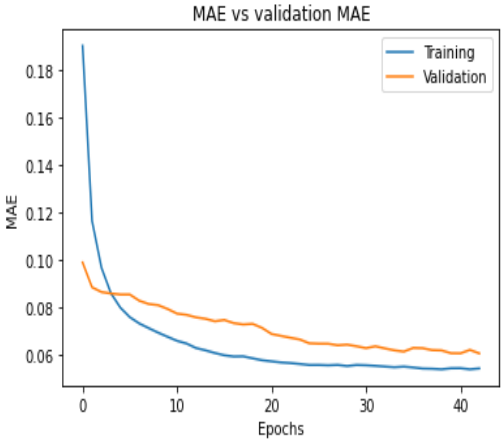


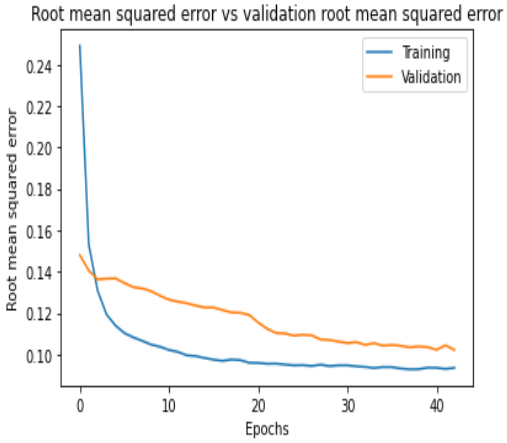
Figure 5.10: Round 2. GRU with 4 inputs: predictions vs real values



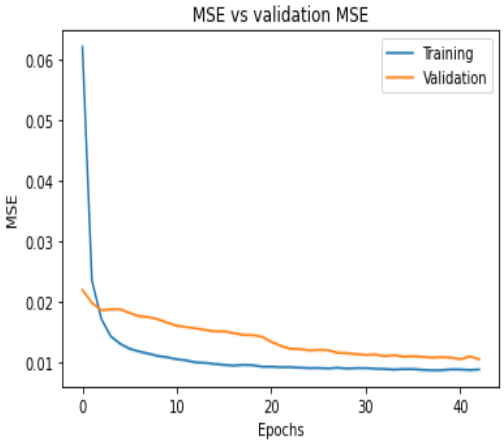
(a) Round 2. GRU with 4 inputs: loss



(b) Round 2. GRU with 4 inputs: MAE



(c) Round 2. GRU with 4 inputs: root mean squared error



(d) Caption 2

Figure 5.11: Round 2. GRU with 4 inputs: training metrics

Results of sklearn.metrics on prediction performance:

```
MAE: 0.1209800695500187
MSE: 0.032532196059952556
RMSE: 0.1779578831942411
R-Squared: 0.6194143452205336
MAPE: 33100385236305.84
ME: -0.0769599121734152
```

GRU with 2 inputs yielded following results:

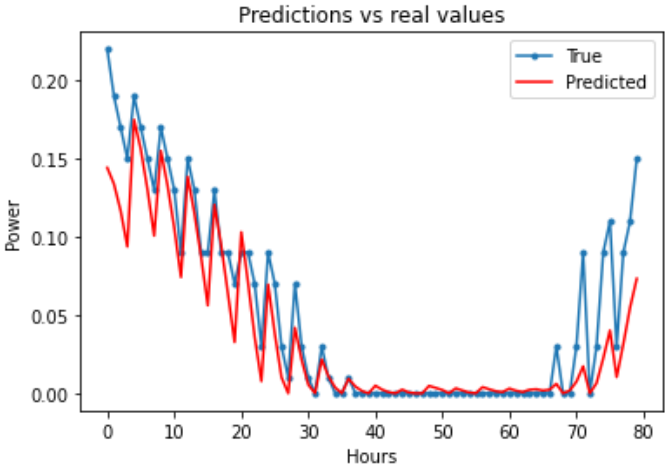
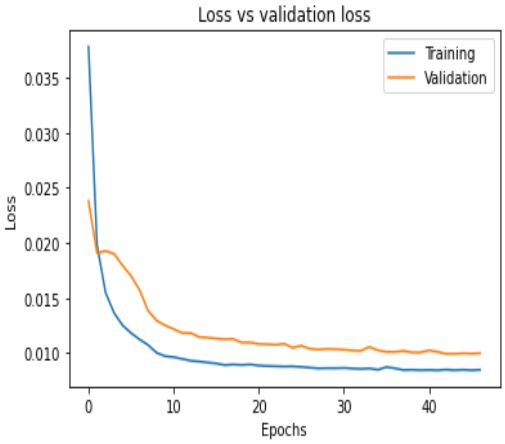
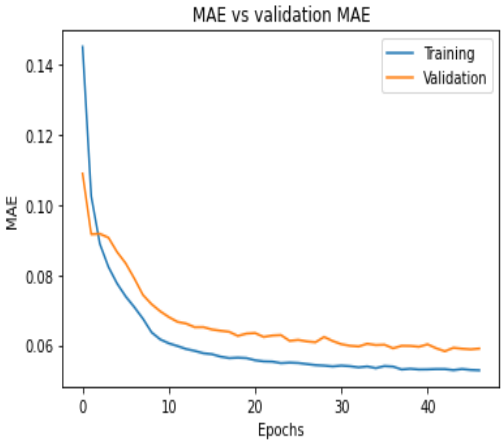


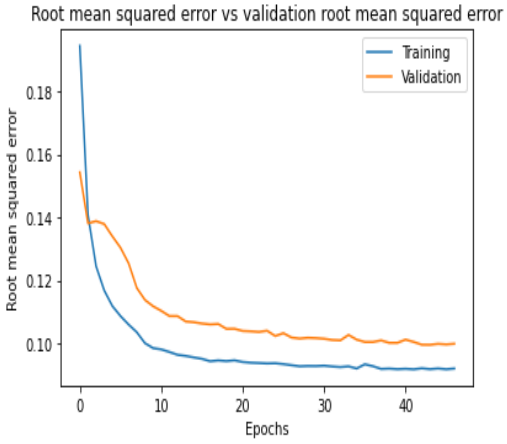
Figure 5.12: Round 2. GRU with 2 inputs: predictions vs real values



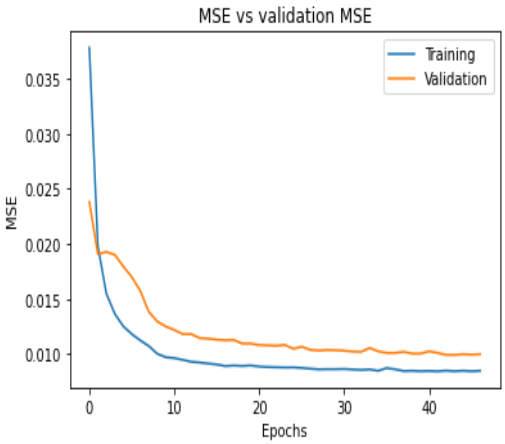
(a) Round 2. GRU with 2 inputs: loss



(b) Round 2. GRU with 2 inputs: MAE



(c) Round 2. GRU with 2 inputs: root mean squared error



(d) Round 2. GRU with 2 inputs: MSE

Figure 5.13: Round 2. GRU with 2 inputs: training metrics

Results of sklearn.metrics on prediction performance:

```
MAE: 0.11591526431797884
MSE: 0.03129297682723125
RMSE: 0.17422960139571572
R-Squared: 0.6338997598755505
MAPE: 30259745048999.98
ME: -0.07145309067630921
```

3’rd round

Configuration:

```
optimizationAlgorithm = tf.keras.optimizers.Adam()
patience = 2
conf_epochs = 50
lag = 24
n_ahead = 4
model_units = 64
activation_function = 'sigmoid'
conf_batch_size = 24
```

GRU with 4 inputs yielded following results:

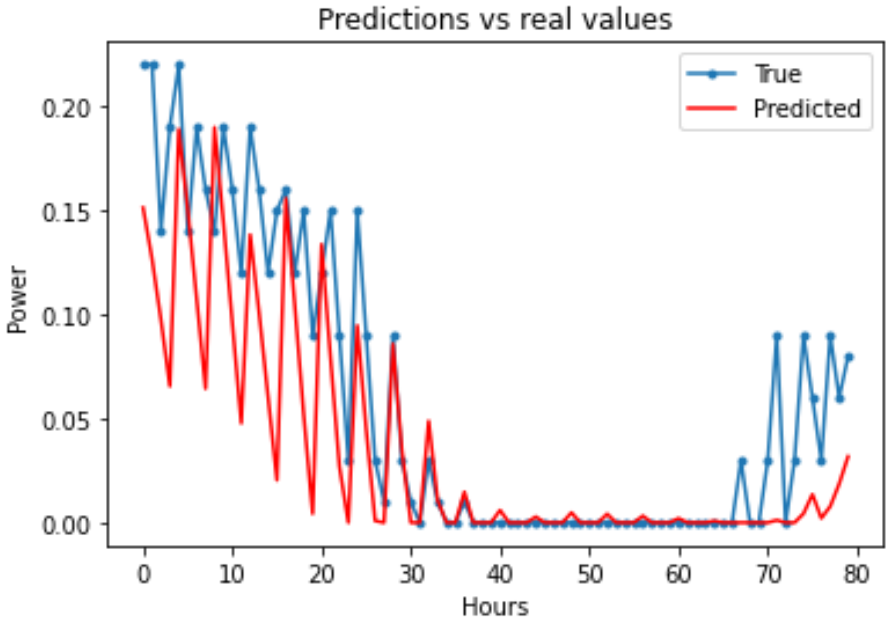
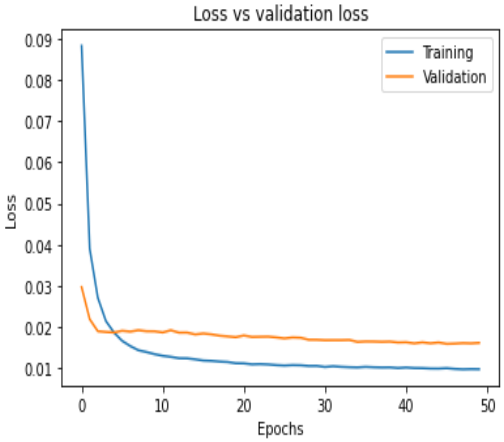
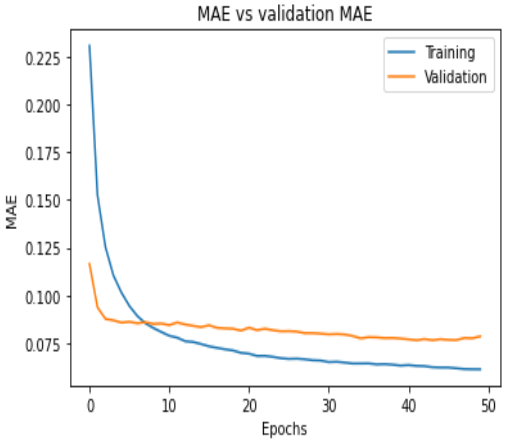


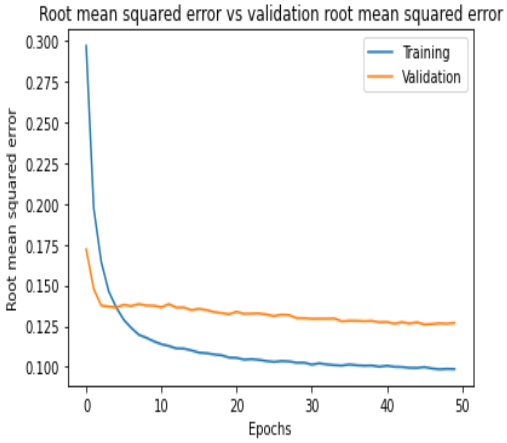
Figure 5.14: Round 3. GRU with 4 inputs: predictions vs real values



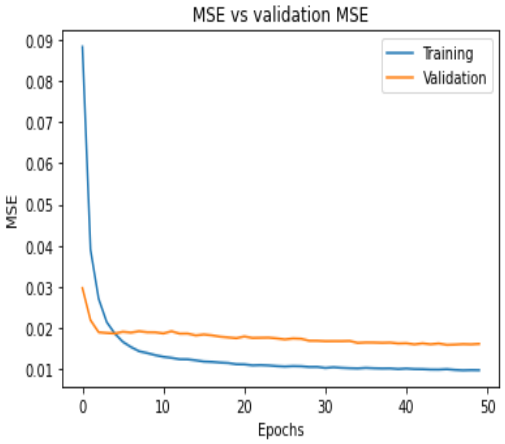
(a) Round 3. GRU with 4 inputs: loss



(b) Round 3. GRU with 4 inputs: MAE



(c) Round 3. GRU with 4 inputs: root mean squared error



(d) Round 3. GRU with 4 inputs: MSE

Figure 5.15: Round 3. GRU with 4 inputs

Results of sklearn.metrics on prediction performance:

```
MAE: 0.16171356591463218
MSE: 0.05218047676064913
RMSE: 0.2200495924187461
R-Squared: 0.38827790853270006
MAPE: 72625249496671.3
ME: -0.1306682009668937
```

GRU with 2 inputs yielded following results:

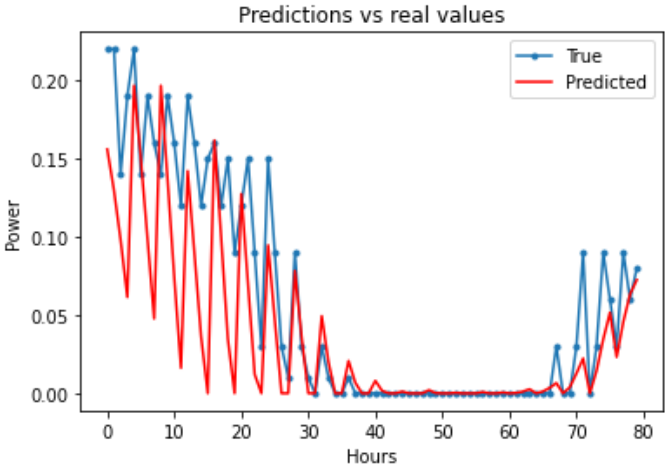
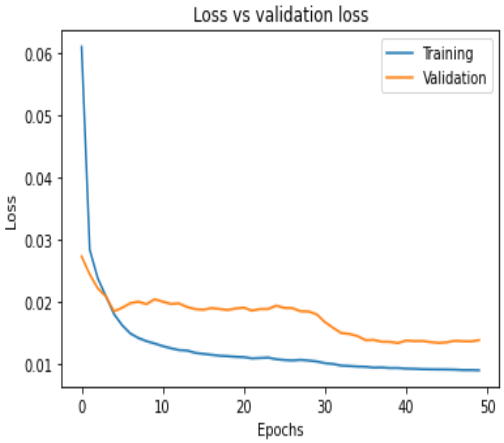
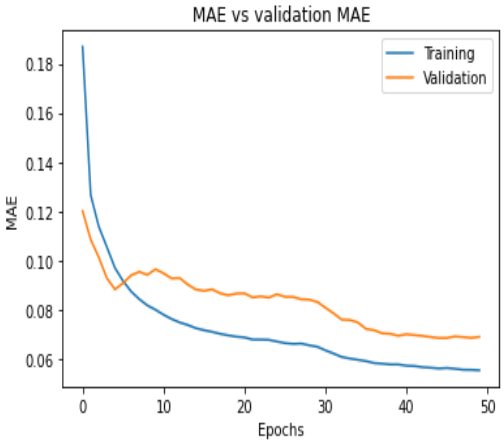


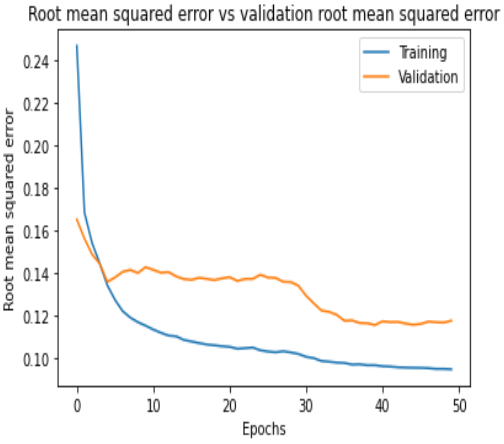
Figure 5.16: Round 3. GRU with 2 inputs: predictions vs real values



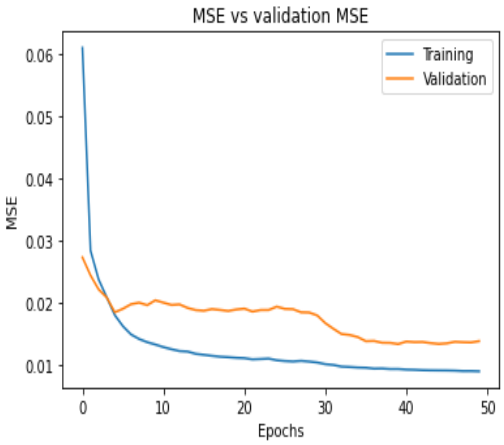
(a) Round 3. GRU with 2 inputs: loss



(b) Round 3. GRU with 2 inputs: MAE



(c) Round 3. GRU with 2 inputs: root mean squared error



(d) Round 3. GRU with 2 inputs: MSE

Figure 5.17: Round 3. GRU with 2 inputs

Results of sklearn.metrics on prediction performance:

```
MAE: 0.14135855747250337
MSE: 0.046972894807293886
RMSE: 0.20691418355769026
R-Squared: 0.44927990410987734
MAPE: 42739276347795.11
ME: -0.10194253634635324
```

The data gathered from these three test rounds do not point to any significant changes in performance between the two versions. However, the general trend through the multiple iterations of the three rounds seemed to point at the two input model version having a slight edge. Considering the increased complexity and the lack of any noticeable performance-wise improvements would indicate that keeping the model with two inputs might be the right decision. Therefore the final implementation will be done with two inputs.

The final model implementation will have the following configurations:

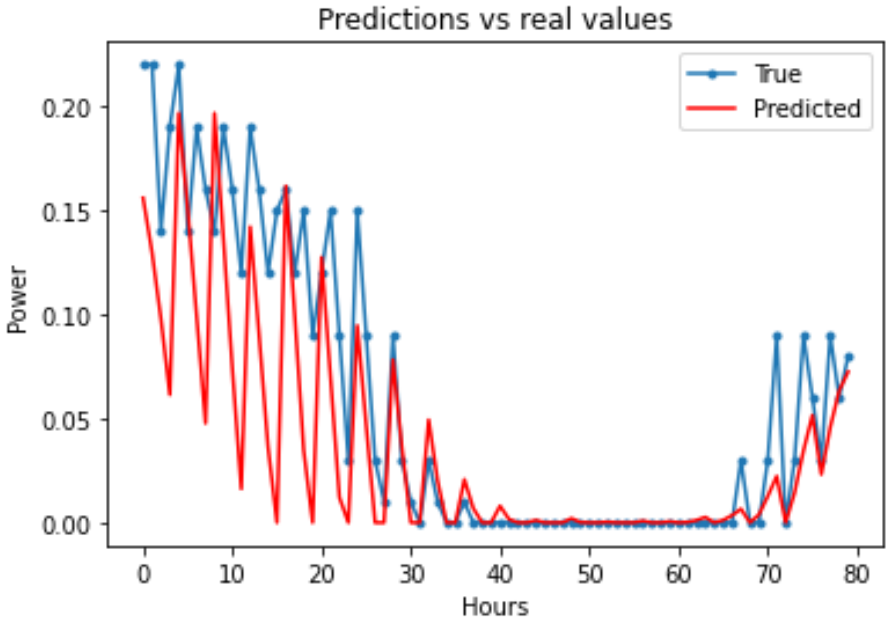


Figure 5.18: Final model: predictions vs real values

Results of sklearn.metrics on prediction performance:

```
MAE: 0.1066777097006566
MSE:0.026878709707441814
RMSE: 0.1622383852980655
R-Squared: 0.6855723399995535
MAPE: 33426117301290.523
ME: -0.055229311086515991 3
```

Model outlininig:
Model: GRU, unidirectional
Layers: GRU -> Dropout layer -> Dense layer
Optimization algorithm: Adam
Epochs: 60
Lag: 24
Hours ahead: 4
Model nodes: 32
Activation function: sigmoid
Batch size: 24

5.2.4 Model outlining

This section will explore the selected model's internal structure and how information is passed through each neuron.

[49]GRUs differentiate themselves from LSTMs by only having two gates; the update gate and the reset gate. These two gates can also be referred to as the Short Term Memory (reset gate) and the Long Term Memory (update gate). GRUs also do not have a separate cell state (C_t) as an ordinary LSTM would. GRUs only have a hidden state (H_t). This contributes to a much simpler architecture, as well as faster computation. The below figure 5.19 is a side-by-side comparison of an LSTM and GRU neuron.

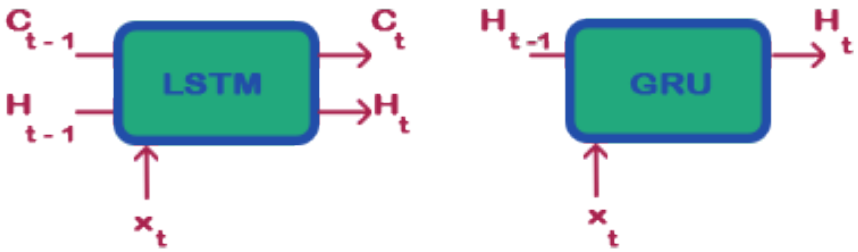


Figure 5.19: Simple visualization of a LSTM and GRU

Inside these two neurons, a variety of operations are performed on the incoming data; this could be whether this data is relevant or how the data will be squashed while passing through the neuron. In a more detailed representation, this is how a GRU neuron would look:

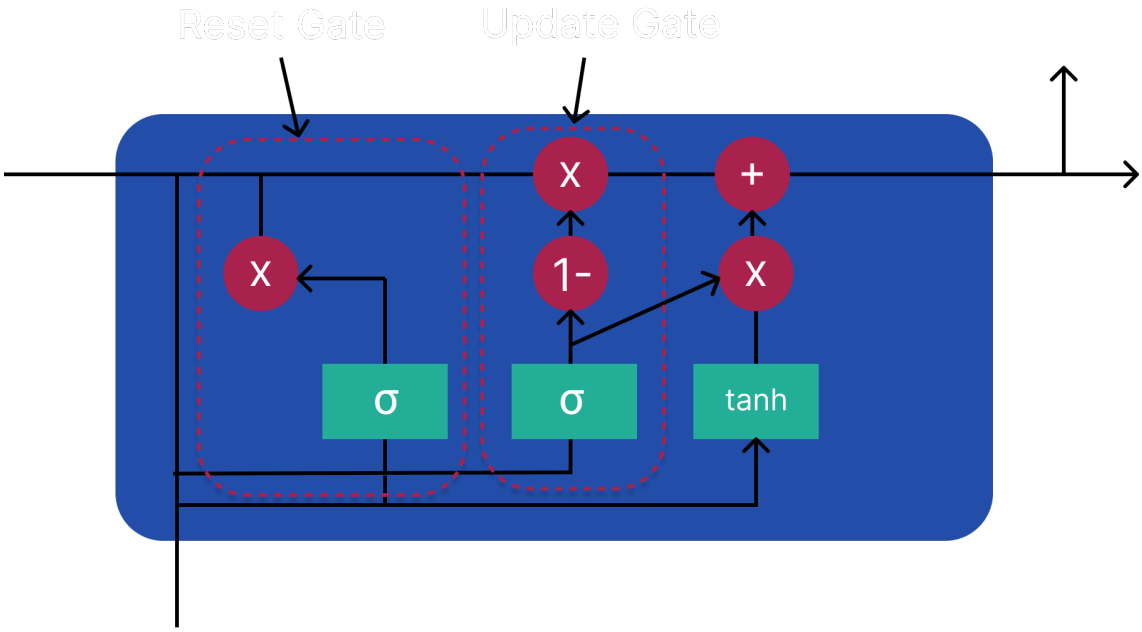


Figure 5.20: Detailed visualization of a GRU

[50]The update gate functions similarly to how the forget gate and input gate would in a LSTM architecture. Its function is to regulate what information to integrate and what to disregard. [49]The equation for this gate can be written as such(See figure 5.22):

$$z_t = \sigma_g * (W_z * x_t + U_z * h_{t-1} + b_z)$$

Figure 5.21: Equation for finding the update gate vector

z_t is the update gate vector.
x_t is the input signal.
U_z and W_z are the update gate weight matrices.
h_{t-1} is the hidden state value from the previous timestep.
b_z is the bias.
σ represents the sigmoid function, which means that the final value(s) of z_t will be between 0 and 1. Since the sigmoid function squashes any value within that range.

The reset gate is used to determine how much information the model should forget. When triggered, it classifies data deemed to be unrelated and then tells the model to forget said data and move forward without it. This is a great feature to help the model distinguish important data from 'noise'. The vector value produced from this reset gate can also be used to calculate the network's hidden state (h_t). In more specific terms, it is calculated as such:

$$r_t = \sigma_g * (W_r * x_t + U_r * h_{t-1} + b_r)$$

Figure 5.22: Equation for finding the reset gate vector

r_t is the reset gate vector.
x_t is the input signal.
U_r and W_r are the reset gate weight matrices.
h_{t-1} is the hidden state value from the previous timestep.
b_r is the bias.
σ represents the sigmoid function, which again means that the final value(s) of r_t will be between 0 and 1.

In order to calculate H_t the model needs to know the value of r_t and employ following two-step process:

$$\hat{h}_t = \phi_h * (W_h * x_t + U_h * (r_t \odot h_{t-1}) + b_h)$$

Figure 5.23: Equation for finding the candidate activation vector

x_t is the input signal.
U_r and W_r are the reset gate weight matrices.
r_t is the reset gate vector.
h_{t-1} is the hidden state value from the previous timestep.
b_h is the bias.
Φ is the hyperbolic tangent function. This means that the final result will be squashed into a value between -1 and 1.

It is worth noting how important the reset gate vector is in this equation since it essentially determines how much influence the previous hidden state will have on the current hidden state. If r_t = 0, then the previous hidden state will be entirely ignored, and conversely, if r_t = 1, then 100 % of the previous hidden state will be considered. This process can be referred to as calculating the candidate activation vector. $\hat{h}_t$ is then used, with varying effects, to calculate the hidden state value from the current timestep:

$$h_t = z_t \odot \hat{h}_t + (1 - z_t) \odot h_{t-1}$$

Figure 5.24: Equation for finding the hidden state value

This time the update gate vector plays an essential to how h_t will take shape. If u_t = 0, then the first term of the equation will be 0, and thus will h_t be set to $\hat{h}\_t$. This makes the hidden state value solely derived from the hidden activation vector, while information

from the previous hidden state vector will be totally disregarded. Conversely, the equation outcome can also be solely determined by the previous hidden vector value if (1-u_t) in the second term equals 0. Therefore, in order for both terms to have an impact, u_t must not be on either extreme of its range, neither 0 nor 1 but in between.

This hidden state value is then passed on as input into another neuron in the network. This process can visually be represented as such:

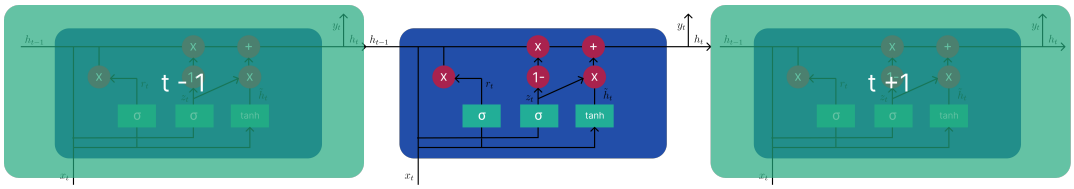


Figure 5.25: Timestep between GRU units

The internal structure can be illustrated like this:

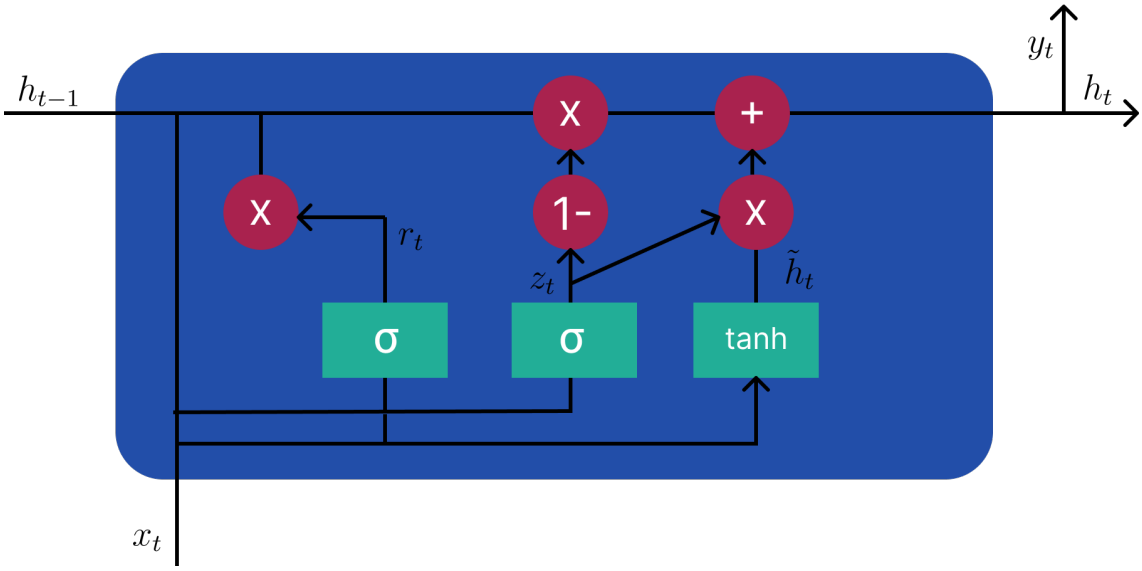


Figure 5.26: GRU structure with signal variables

With the current configuration, the GRU layer will output into a Keras dropout layer. [51]The idea with this layer is to alleviate overfitting by randomly setting input units to 0 with a fixed rate. This happens every step during training. Those input units that are not set to 0 will be scaled up by 1/(1-rate) in order to preserve the sum of all input units.

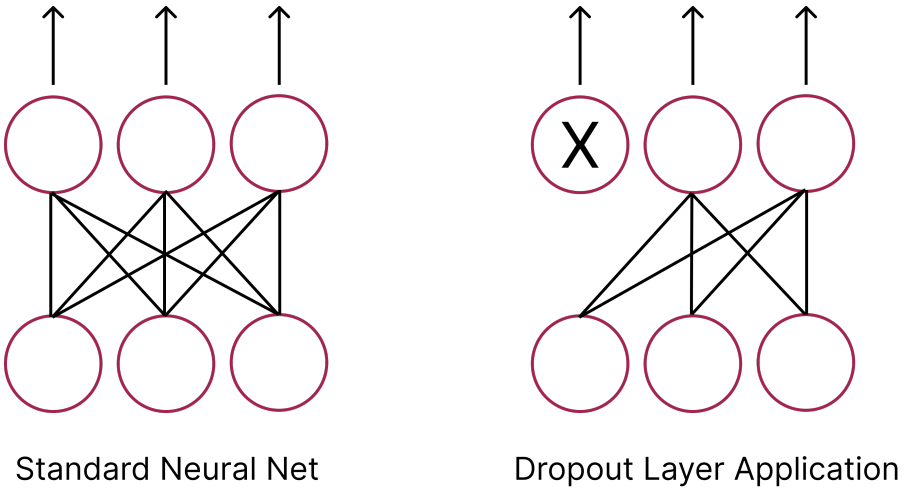


Figure 5.27: Standard net and dropout layer application

[52]The final layer added to the model network is a dense layer used as an output layer for the model. Its input will be a TensorFlow object that it passed from a previous layer. The input will be passed through the dense layer, where weights and biases are applied to process the input further. Then lastly, an activation will be applied before outputting the data. The process can be described as follows: activation(dot(input, kernel) + bias)

Where activation is the activation function like sigmoid, linear and so on. “dot(input, kernel)” describes the dot product of the input and weight matrix created by the layer. Bias is, much like the name indicates, the bias created by the layer.

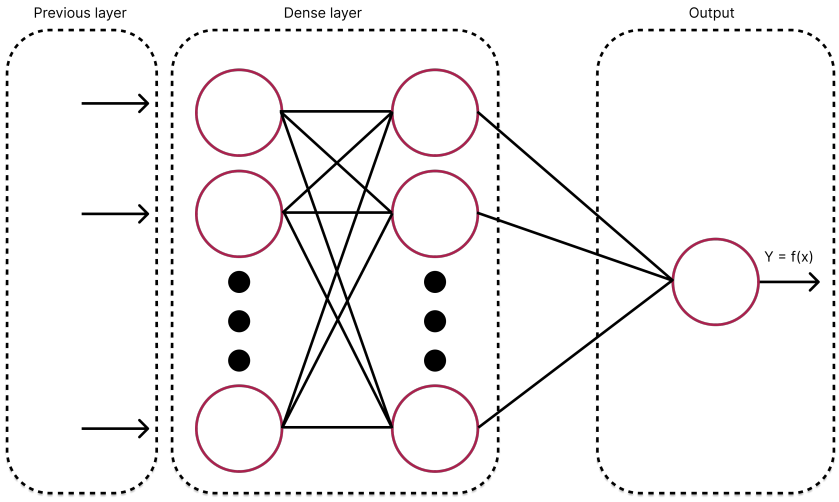


Figure 5.28: Dense layer illustration

6 Docker

This chapter will focus on the Docker implementation on Raspberry Pi. This will involve a short introduction to Docker and key concepts. Whereafter a detailed walkthrough of how the Docker image was constructed will follow. How to use said image is then showcased, which will lead into the next section, where the system as a whole is outlined.

6.1 About Docker

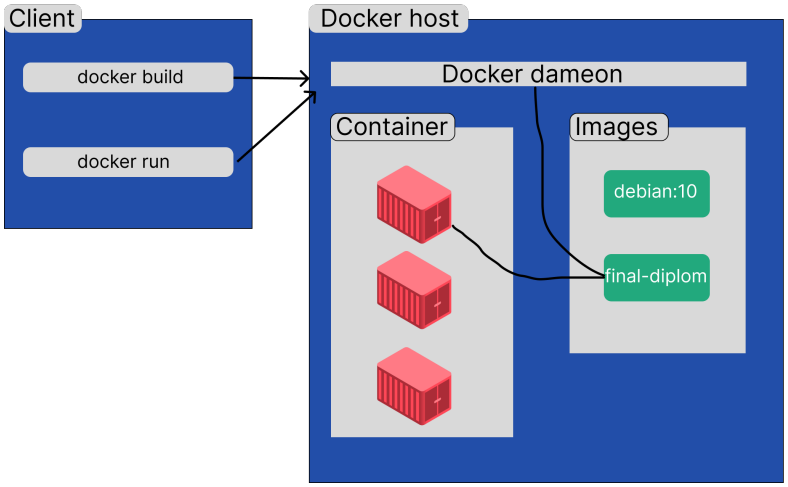


Figure 6.1: Docker process visualization

[53]Docker is an open-source platform that helps to create, deploy, and manage containers, e.g., small virtual machines (VMs). A container is a small virtual environment on which images can be loaded onto. The container will usually be assigned an ID and is able to run as a background process. The image is a definition of how a virtual environment should be, in terms of what packages and files will be present, as well as what operating system everything will be run on. Images are used to define the container's internal environment, while the container itself is simply an instantiation of the image. In this project, a new Docker image is created and then loaded into a container, which will be prompted to run once per hour. The container is supposed to run the predictions model code and output forecast predictions into a .txt file on the local system (Raspberry Pi system).

6.2 Image Setup

As a starting point, Github is installed to pull the model code from our repository as such:

```
$ sudo apt-get update
$ sudo apt-get install libcurl4-gnutls-dev libexpat1-dev gettext libz-dev
    libssl-dev asciidoc xmlto docbook2x
$ git clone https://github.com/TheAnders121/TestRepo
```

The code pulled from Github is retrieved in jupyter notebook format, which can be demanding to run on Linux through CLI. The next step is, therefore to convert it into a more manageable python-format:

```
$ sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-
    generic
```

A reboot might be necessary to get the jupyter to python-format conversion command to work properly, but if jupyter is not found then this fix might work:

```
$ pip install ipython
$ pip install nbconvert
$ pip install ipynb-py-convert
```

When these packages have successfully been installed, the .ipynb file should be able to be reformatted: jupyter nbconvert –to python ProjektKode.ipynb

Now it is time to install Docker, which is installed as such:

```
$ sudo apt-get install docker
```

First and foremost, the Docker container is initialized with a custom-made Dockerfile made explicitly for this project. The Dockerfile content:

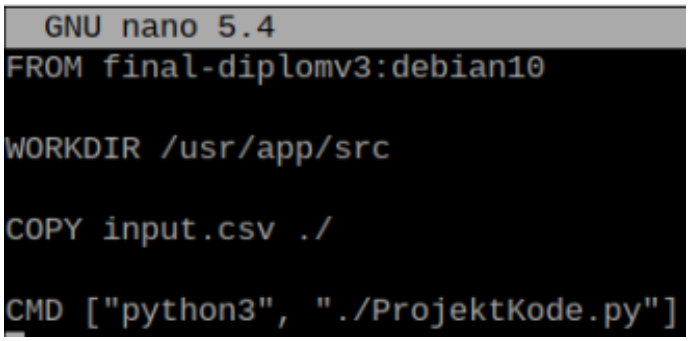


Figure 6.2: Inside the Dockerfile

Debian 10 is used as an OS platform and is set to execute the model code, which is inside the .py file transferred from the local system into the container. Debian 10 is used since it comes with python3.7.3 pre-installed, and this version is the first prerequisite for running the model code successfully. The Docker container is built by running the following commands: (Note that the commands must be run in the same folder containing the reformatted model code, Dockerfile, and dependencies.txt)

Build container:

```
$ sudo docker build -t docker_container .
```

Additional changes need to be made in order to run the model code. To enter the container:

```
$ sudo docker run -it docker_container /bin/bash
```

Get pip3

```
$ apt install python3-pip
```

Get following dependencies:

```
$ apt-get install libhdf5-dev libc-ares-dev libeigen3-dev
$ pip3 install Cython==0.29.28
$ apt-get install libatlas-base-dev libopenblas-dev libblas-dev
$ apt-get install liblapack-dev
```

While inside the container gdown needs to be installed in order to download the TensorFlow wheel from google drive:

```
$ pip3 install gdown
$ gdown https://drive.google.com/uc?id=1x2nxbi8rLvF7WzYkuwt9SOuDvsOUytjE
```

Afterwards the wheel is installed as such (This might around 40 minutes):

```
$ pip3 install tensorflow-2.4.0-cp37-cp37m-linux_aarch64.whl
```

It is important to notice that this TensorFlow wheel is of a particular version. That being Tensorflow 2.4.0 for python 3.7 with Linux aarch64 compatibility. This is very important to get right since there are a lot of different distributions to be found. Other versions of Tensorflow have not been tested for this specific project.

When the installation has been successfully completed, it can be a nice thing to confirm that TensorFlow indeed has been installed and is reachable through python code. The following commands can be run to check this. Enter python environment:

```
$ python3
```

While inside the environment, these two lines are used to confirm importation and version.

```
import tensorflow as tf
tf._version_
```

This output should be displayed:

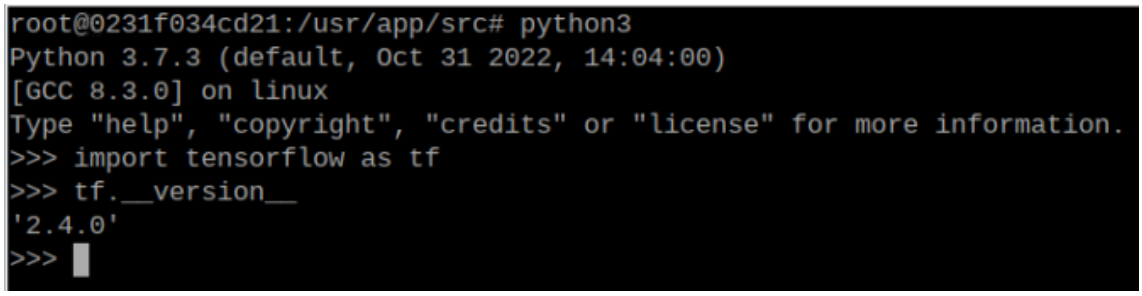


Figure 6.3: Display python version

At this point, all the additional dependencies from dependencies.txt can be installed. All packages and versions are specified within the txt file. Since several packages are to be installed and wheels to be built, this step will take some time. When all these packages have been installed, the container should be able to run the model code.

```
$ pip3 install -r dependencies.txt
```

While installing the dependencies, two errors needed to be addressed before finalizing the installation process. The first error occurred when trying to install NumPy and was fixed as such:

```
$ pip3 install cython
$ apt-get install gcc python3.7-dev
$ pip3 install pycocotools==2.0.0
$ pip3 install --upgrade numpy==1.20.1
```

The second error occurred when trying to install h5py and was fixed as such:

```
$ pip3 uninstall h5py
$ apt-get install libhdf5-dev
$ pip3 install h5py
$ pip3 install --upgrade h5py==2.10.0
```

Finally when running the code, another error occurred:

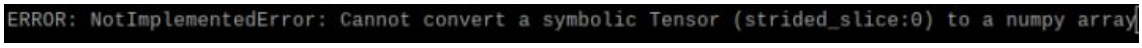


Figure 6.4: Tensor error on model construction

This error could be caused by different failings, but what solved the error for this particular project was to make some minor changes to one of the installed TensorFlow libraries. First, a text editor is needed to be installed; in this case, “nano” is installed as such:

```
$ apt-get install nano
```

Then the library is opened in nano as such (note that paths may vary):

```
$ nano /usr/local/lib/python3.7/dist-packages/tensorflow/python/
ops/array_ops.py
```

Following changes were made to the python code

Firstly a new import is called within the code:

```
from tensorflow.python.ops.math_ops import reduce_prod
```

Secondly a line within the “_constant_if_small” method is replaced:

```
def _constant_if_small(value, shape, dtype, name):\
  try:\
    if np.prod(shape) < 1000: ## --> it to if reduce_prod(shape) < 1000:\
      return constant(value, shape=shape, dtype=dtype, name=name)\
  except TypeError:\
    # Happens when shape is a Tensor, list with Tensor elements, etc.\
    pass\
  return None
```

“np.prod(shape)” is replaced with “reduce_prod(shape)”. After saving changes and exiting the file, the model codes executed as intended.

```
$ python3 ProjektKode.py
```

Figure 6.5: Model output

Important package versions to take note of:
Python 3.7.3
Tensorflow 2.4 for aarch64 (compatible with armv7)
Numpy 1.20.1
Pandas 1.3.5
h5py 2.10.0
pycocotools 2.0.0
scikit-learn 1.0.2
Keras 2.4.0

At this point, the container is set up to run the model code, and now it only needs input from a CSV file to make predictions. The model code will look for a CSV file named “input.csv”; therefore, a file named as such is created in the same directory as where the model code is located.

Before saving the image to the local image pool, it is important to run the model code first. This is because the code will generate a model and save it inside its environment. This model construction is only done, when no model is found in the environment; meaning this process is ideally only done once. If a model exists within the environment, it will be employed to make predictions. Running the model for the first time is shown in figure 6.6.

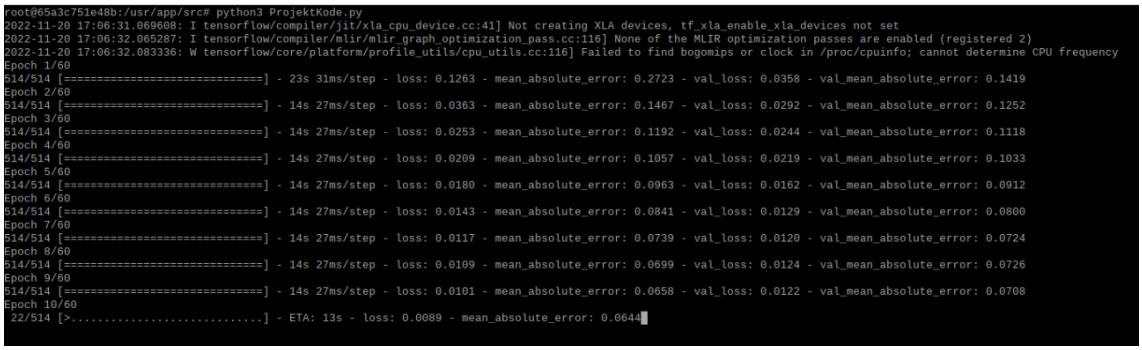


Figure 6.6: Tensor error on model construction

The container should then be saved as a Docker image. To do this, another terminal in the Raspberry Pi system is opened, and following command is used:

```
$ sudo docker commit "New image name" "ContainerName or ID":"tagname"
```

“New image name” can be any desired name. “ContainerName or ID” is either the name or ID of the currently running container that is being committed. These values can be found through

```
$ sudo docker ps -a
```

“Tagname” is voluntary and can be used to specify the image version if one is inclined to create several images with the same name.

After the image has been committed, it is safe to exit the container without losing all progress. Now an image with all the necessary configurations is saved within the Docker image system. A new container can be loaded with this particular image by calling its name as such:

```
$ docker run -it "image "name
```

This will run the model code within the instantiated container and then exit after job completion. This whole process proved to be very time-consuming since there was a significant number of packages necessary to run the deep learning model. Getting the correct version of a particular package was vital for the whole python environment to work in accord. When running a container with the above command, it is essential to be inside the same working directory as previously described, Dockerfile, since this file is necessary to build the container as intended in this project. This is mainly because it is used to provide the model with its necessary input; otherwise, with no input, it would not do much. The input takes the form of a CSV file that exists in the same working directory as the Dockerfile and is meant to be updated once every hour. Every update is meant to be done right before it is copied into a new container. Every update to input.csv and instantiation of a new container is made through a script called main.sh. How this process works will be outlined further in the ”Main bash script”-section below.

7 Finalizing the build

In this chapter, all the scripts used to connect the three main aspects of the project are outlined. Firstly, how data is acquired from MQTT messages and saved on the Raspberry Pi. Secondly, how the main.sh work and how it ensures sequential cohesion between the various tasks running on Raspberry Pi. Its content will also be presented and explained. Then the time scheduler used to run main.sh once per hour will be described, and lastly, a flowchart will give an overview of the final build.

7.1 Updating the CSV file

Working with remote sensors makes it difficult to keep track of measurements and gather data in an efficient manner. As the MQTT messaging protocol was being worked on, a solution for how to save the data in a CSV file became necessary since it could be used for the deep learning model.

To achieve this, a Python program able to connect to the broker and one or more topics was developed. When connected, it would subscribe to the topic (/esp/temp), and take the temperature readings before importing the data into a CSV file.

```
m2csv = ConvertMQTT("10.0.0.13","input.csv")

m2csv.setReadings(5) #5 readings, can be changed to more if needed

# More topics can be added here
m2csv.addTopics("/esp/temp")

m2csv.start()
```

Figure 7.1 below displays how the getData.py reads the temperature data found on the ”/esp/temp” topic in Window1.

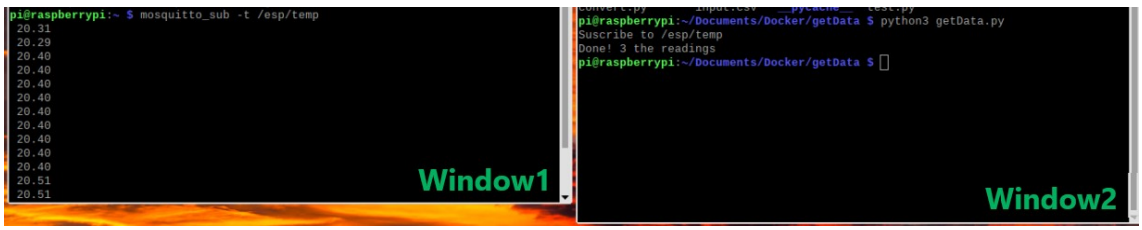


Figure 7.1: 2 Terminals: Window1 displaying the temperature and Window2 running the getData.py script

7.2 Main Bash Script

The main bash script is made to ensure that each individual job on the Raspberry Pi will run in a coherent and sequential manner.

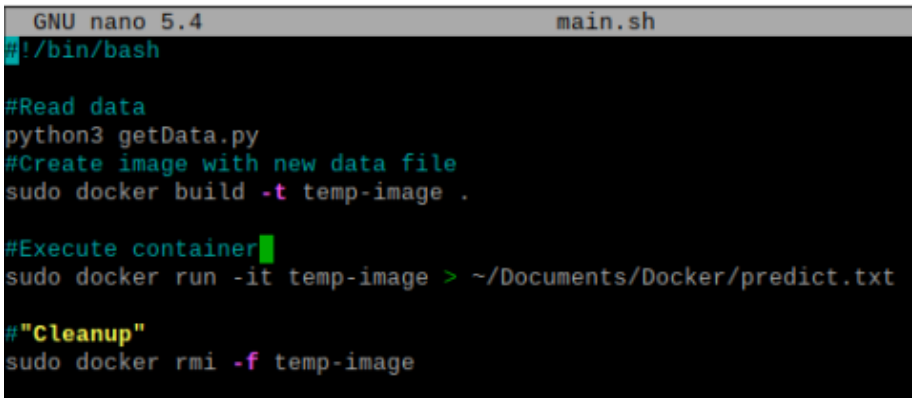


Figure 7.2: Main bash script content

To begin with, the script calls a python script which is used to initialize a MQTT connection to the broker, where a subscription is established to retrieve data from the temperature sensor. The retrieved data is then written into input.csv.
In principle, this sets the foundation for what is required to run the model. Since there now is a new input, a new image is created with the same configurations as the one built earlier, but the difference is that the new image is built with a different input.csv file. The temperature sensor values are, by design, not directly used for the model, as the model requires other variables to work with. The new image is only meant to be temporary and is therefore deleted after usage to avoid any lasting consumption of memory; hence the name temp-image. When the temporary image has been built it is used to run a container and then direct the container output into a text file called predict.txt. Predict.txt is overwritten with new predictions for every run. As a final step, the temporary image is removed from the Docker's local image pool. This enables the script to run again undisturbed by using the exact same image name when constructing a slightly different version.

7.3 Time Scheduler

The model needs to output new predictions every time new data is collected; therefore, the model is run with a time scheduler on the Raspberry Pi. Every hour a file containing weather should be updated with a new entry, whereafter this data should be used to make a new prediction. Crontab is used to call the model python script once an hour while also cycling through other tasks for the system as a whole can work. All of these tasks are invoked through the main bash script.

Crontab uses a special syntax that is used to describe how often a job will run. The time is scheduled through 5 subsequent values, where each value will be a determining factor for the schedule as a whole. First: Minutes.
Second: Hours.
Third: Days.
Fourth: Months
Fifth: Weekdays.

These values are usually described through integers, but one can also employ a star value to mean every minute, day etc.

Therefore in order to run model predictions every hour, the crontab script will look as such:

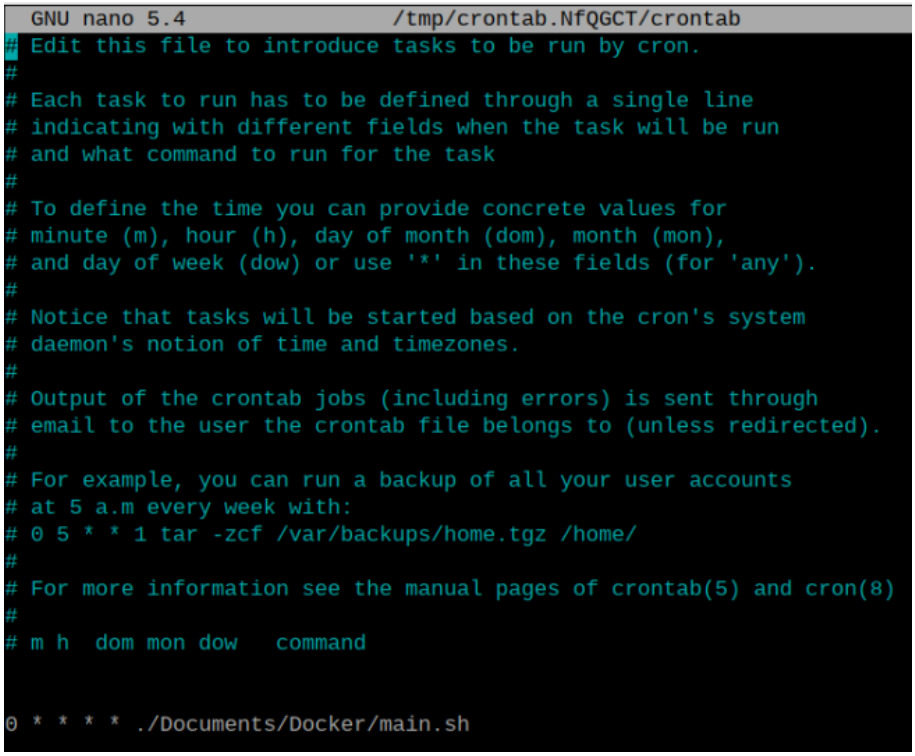


Figure 7.3: Crontab content

”0 * * * *” will ensure that the crontab job run at every instance where minutes = 0, which essentially means once an hour at for instance 14:00, 15:00, 16:00 and so on.

7.4 Flowchart

Here is a flowchart of how the different processes within the system interact with each other. It can be seen that crontab, which is running on the Raspberry Pi, is used to call main.sh, which in turn is used to gather data from the sensors and run a container with a newly made image containing input.csv. The container output is then lastly piped into predict.txt. (See figure 7.4)

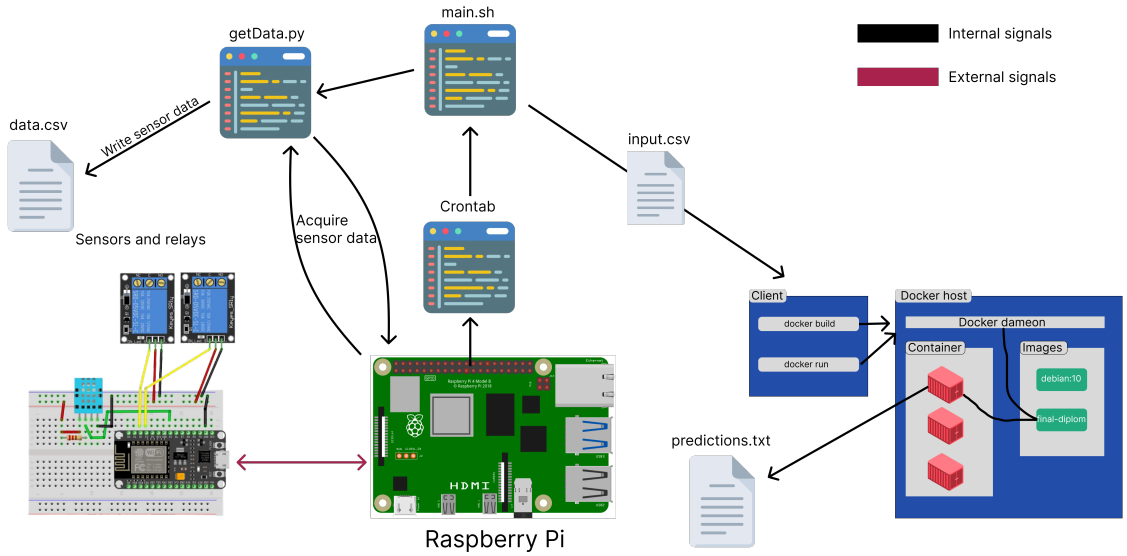


Figure 7.4: FlowChart

8 Conclusion

This chapter will contain a conclusive look at how the project scope was realized. Each objective found in chapter 1, section 1.3 will be examined based on how these specific tasks were accomplished. Furthermore, how time management was actually realized and what issues were faced.

The specified objectives can be put into three categories: Deep Learning model for forecasting, implementing said model into a docker container, and acquiring data through MQTT connection.

Building and testing the communication to acquire data was completed to the degree that the data was not required from the DHS but instead from the DHT11 temperature sensor. In chapter 1, section 1.4, it was from the project scope declared that these changes were to be implemented instead. In general, implementing any sensor in this system would be a relatively non-invasive process, as minimal configuration would be required on the MQTT broker.

Communication between the Neogrid device and the Raspberry Pi was unfortunately not established. In the middle of the project, it was communicated that the connection between the Neogrid device and Raspberry Pi was not needed. Instead, a connection through an API would be more sufficient. However, this was not realized due to technical complications and an impending deadline.

Data analysis has thoroughly been described in both purpose and intent in chapter 5, section 5.2. Moreover, a final model was produced through extensive testing and comparison between configurations. The final model is able to forecast power output for the following four hours, with a certain degree of deviation - as is expected with most models. While some predictions might not be 100% correct every time, the general trend of highs and lows is captured. This means that rises and falls in production are expressed very well. This output can be used to estimate when to draw more power generated by solar panels. It can thus be concluded that a capable deep learning model has been constructed, and the task “Data analysis with deep learning models and use of a forecasting algorithm for heat demand based on machine learning (Python))” has been completed.

Implementing the forecast algorithm in Raspberry Pi into Docker containers was fully completed. It proved to be a much more challenging task than initially estimated and was therefore, considerably more time-consuming. An image, fully capable of running the model, was finally constructed and saved onto the Raspberry Pi. This image resides inside the Dockers’ local image pool and can be instantiated through a simple line of Docker CLI commands. As documented in this thesis (chapter 6, section 6.2), it can be concluded that this objective has been achieved.

Referring back to the three previously mentioned categories, it can be reasoned that all Deep Learning aspects were accomplished. Any Docker-related task was also produced as requested. The MQTT connection data acquisition was successful in gathering temperature data from a sensor through a publish/subscriber model. Since the connection between the Neogrid device and the Raspberry Pi was not realized, this category is only partially accomplished.

References

[1] RE-EMPOWERED. *RE-EMPOWERED homepage)*. 2022. URL: https://reempowered-h2020.com/.

[2] Neogrid. *Neogrid homepage)*. 2022. URL: https://neogrid.dk/.

[3] ISO.org. *ISO 21500 Project, programme and portfolio management — Context and concepts*. 2021. URL: https://www.iso.org/standard/75704.html.

[4] ISO.org. *ISO 21502 Project, programme and portfolio management — Guidance on project management*. 2020. URL: https://www.iso.org/standard/74947.html.

[5] Dave Chaffey. *Golden Circle Model*. 2022. URL: https://www.smartinsights.com/digital-marketing-strategy/online-value-proposition/start-with-why-creating-a-value-proposition-with-the-golden-circle-model/#:~:text=The%20Golden%20Circle%20theory%20explains,if%20they%20start%20with%20why.

[6] Ruslan Kildeev. *Golden Circle Model*. 2022. URL: https://miro.com/miroverse/the-golden-circles-plan-with-meaning/.

[7] MDN contributors. *The WebSocket API (WebSockets)*. 2022. URL: https://www.oxfordlearnersdictionaries.com/definition/english/internet-of-things.

[8] Lionel Sujay Vailshery. *Number of Internet of Things (IoT) connected devices worldwide*. 2022. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[9] Rajiv. *Characteristics of Internet of Things*. 2022. URL: https://www.rfpage.com/characteristics-of-internet-of-things/.

[10] Laura Vasilov. *Knowledge Byte: Building Blocks Of IoT Architecture*. 2022. URL: https://www.cloudcredential.org/blog/knowledge-byte-building-blocks-of-iot-architecture/#:~:text=An%20IoT%20system%20comprises%20four,processors%2C%20gateways%2C%20and%20applications..

[11] Lawrence Williams. *Types of Computer Network*. 2022. URL: https://www.guru99.com/types-of-computer-network.html.

[12] Ninad Ingale. *Internet of Things: Communication Models*. 2021. URL: https://technophileholmes.hashnode.dev/internet-of-things-communication-models-and-apis.

[13] MDN contributors. *The WebSocket API (WebSockets)*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API#:~:text=The%20WebSocket%20API%20is%20an,the%20server%20for%20a%20reply..

[14] MQTT. *MQTT.org)*. 2022. URL: https://mqtt.org/.

[15] Steve. *MQTT Publish and Subscribe Beginners Guide*. 2021. URL: http://www.steves-internet-guide.com/mqtt-publish-subscribe/.

[16] Mathworks. *Publish MQTT Messages and Subscribe to Message Topics)*. 2022. URL: https://www.mathworks.com/help/supportpkg/raspberrypi/ref/publish-and-subscribe-to-mqtt-messages.html.

[17] Coursera. *Deep Learning vs. Machine Learning: Beginner's Guide*. 2022. URL: https://www.coursera.org/articles/ai-vs-deep-learning-vs-machine-learning-beginners-guide.

[18] Wikipedia. *Deep learning*. 2022. URL: https://en.wikipedia.org/wiki/Deep_learning.

[19] Derrick Mwiti. *10 Real-Life Applications of Reinforcement Learning*. 2022. URL: https://neptune.ai/blog/reinforcement-learning-applications.

[20] Christopher Thomas. *An introduction to Convolutional Neural Networks*. 2019. URL: https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7.

[21] Jack Kelly and William Knottenbelt. *Neural NILM: Deep Neural Networks Applied to Energy Disaggregation*. 2015. URL: https://arxiv.org/pdf/1507.06594.pdf.
[22] deepai. *Weight (Artificial Neural Network)*. ???? URL: https://deepai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network.
[23] https://machine-learning.paperspace.com/. *Weights and Biases*. 2021. URL: https://machine-learning.paperspace.com/wiki/weights-and-biases.
[24] Harsha Bommana. *Introduction to neural networks part 1*. 2019. URL: https://medium.com/deep-learning-demystified/introduction-to-neural-networks-part-1-e13f132c6d7e.
[25] Willy Ngashu. *Multi-Output Classification with Machine Learning*. 2022. URL: https://www.section.io/engineering-education/multi-output-classification-with-machine-learning/#conclusion.
[26] Jason Brownlee. *How to Choose an Activation Function for Deep Learning*. 2021. URL: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/#:~:text=for%20Output%20Layers-,Activation%20Functions,a%20layer%20of%20the%20network.
[27] Sefik Ilkin Serengil. *Hyperbolic Tangent as Neural Network Activation Function*. 2017. URL: https://sefiks.com/2017/01/29/hyperbolic-tangent-as-neural-network-activation-function/.
[28] Panagiotis Antoniadis. *Activation Functions: Sigmoid vs Tanh*. 2022. URL: https://www.baeldung.com/cs/sigmoid-vs-tanh-functions.
[29] VIJAYSINH LENDAVE. *Can ReLU Cause Exploding Gradients if Applied to Solve Vanishing Gradients?* 2021. URL: https://analyticsindiamag.com/can-relu-cause-exploding-gradients-if-applied-to-solve-vanishing-gradients/.
[30] Ayush Gupta. *A Comprehensive Guide on Deep Learning Optimizers*. 2022. URL: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/.
[31] SAGAR SHARMA. *Epoch vs Batch Size vs Iterations*. 2017. URL: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9#.
[32] Wikipedia. *Gradient descent*. 2022. URL: https://en.wikipedia.org/wiki/Gradient_descent.
[33] Danfoss. *ECL Comfort 210)*. 2022. URL: https://store.danfoss.com/dk/da/Climate-Solutions-for-heating/Elektroniske-regulatorer-og-overv%C3%A5gningsl%C3%B8sninger/ECL-Comfort-regulatorer/ECL-Comfort-210%2C-LCD-dot-matrix%2C-Forsyningssp%C3%A6nding-%5BV%5D-AC%3A-207---244/p/087H3020.
[34] image. *Picture of the ECL210*. 2022. URL: https://res.cloudinary.com/evoleska/image/upload/v1524410840/product/1618614.pdf.
[35] Raspberry Pi. *Raspberry Pi imager software)*. 2022. URL: https://www.raspberrypi.com/software/.
[36] Mosquitto. *Eclipse Mosquitto open source message broker*. 2022. URL: https://mosquitto.org/.
[37] Guru99. *TCP/IP Model: What are Layers Protocol? TCP/IP Stack*. 2022. URL: https://www.guru99.com/tcp-ip-model.html.
[38] Arduino. *Arduino IDE software)*. 2022. URL: https://www.arduino.cc/en/software.
[39] Arduino. *Arduino DHT Libraries)*. 2022. URL: https://www.arduino.cc/reference/en/libraries/dht-sensor-library/.
[40] MisterBotBreak. *How to Use Temperature and Humidity (DHT) Sensors*. 2019. URL: https://create.arduino.cc/projecthub/MisterBotBreak/how-to-use-temperature-%20and-humidity-dht-sensors-9e5975.

[41] HiveMQ. *HiveMQ Websockets Client Showcase*. 2022. URL: http://www.hivemq.com/demos/websocket-client/.
[42] wikipedia. *GIGO redirects - Garbage in, garbage out*. 2022. URL: https://en.wikipedia.org/wiki/Garbage_in,_garbage_out.
[43] Tobias Geisler Mesevage. *Data Preprocessing*. 2021. URL: https://monkeylearn.com/blog/data-preprocessing/.
[44] Deepchecks. *Normalization in Machine Learning*. 2021. URL: https://deepchecks.com/glossary/normalization-in-machine-learning/.
[45] Weatherspark. *Climate and Average Weather Year Round in Copenhagen*. 2022. URL: https://weatherspark.com/y/74001/Average-Weather-in-Copenhagen-Denmark-Year-Round#Figures-CloudCover.
[46] scikit-learn.org. *sklearn.preprocessing.MinMaxScaler*. 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.
[47] Wikipedia. *Long short-term memory*. 2022. URL: https://en.wikipedia.org/wiki/Long_short-term_memory.
[48] Wikipedia. *Gated recurrent unit*. 2022. URL: https://en.wikipedia.org/wiki/Gated_recurrent_unit.
[49] Shipra Saxena. *Introduction to Gated Recurrent Unit (GRU)*. 2021. URL: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-gated-recurrent-unit-gru/.
[50] Ajay Ohri. *Understanding GRU Networks In 2021*. 2021. URL: https://www.jigsawacademy.com/blogs/data-science/gru/.
[51] scikit-learn.org. *Dropout layer*. 2022. URL: https://keras.io/api/layers/regularization_layers/dropout/.
[52] scikit-learn.org. *Dense layer*. 2022. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense.
[53] scikit-learn.org. *Docker*. 2022. URL: https://www.ibm.com/cloud/learn/docker.

A Appendix

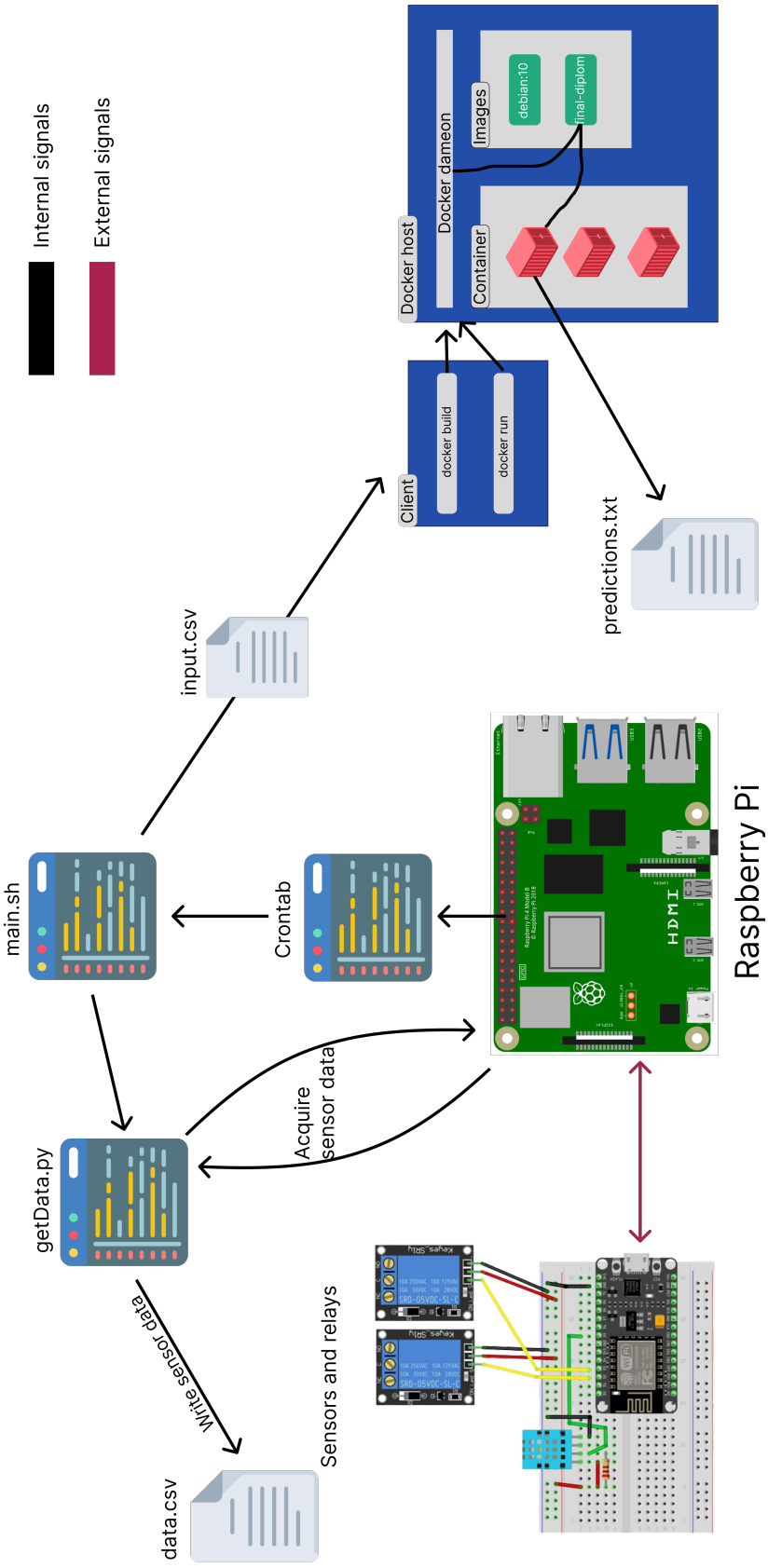


Figure A.1: Overview of the Flowchart system

Github

Here is the github repository containing all our code:
https://github.com/TheAnders121/FinalDiplom

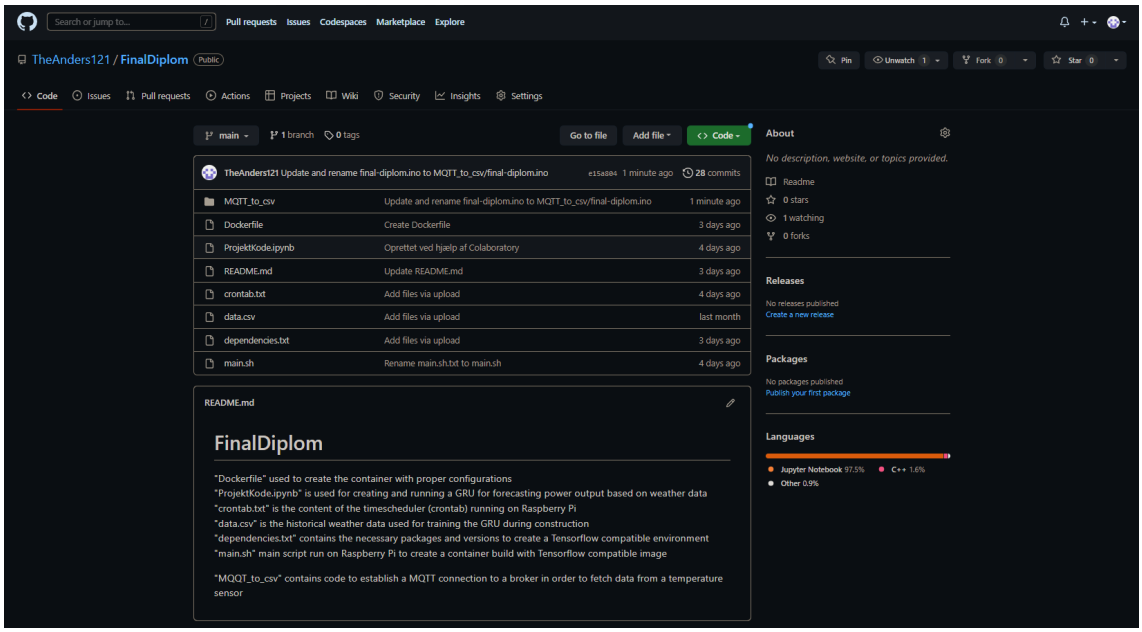


Figure A.2: Github for our project

List of Figures

Technical
University of
Denmark

Lautrupvang 15
2750 Ballerup
Tlf. 4525 1700

www.engtech.dtu.dk