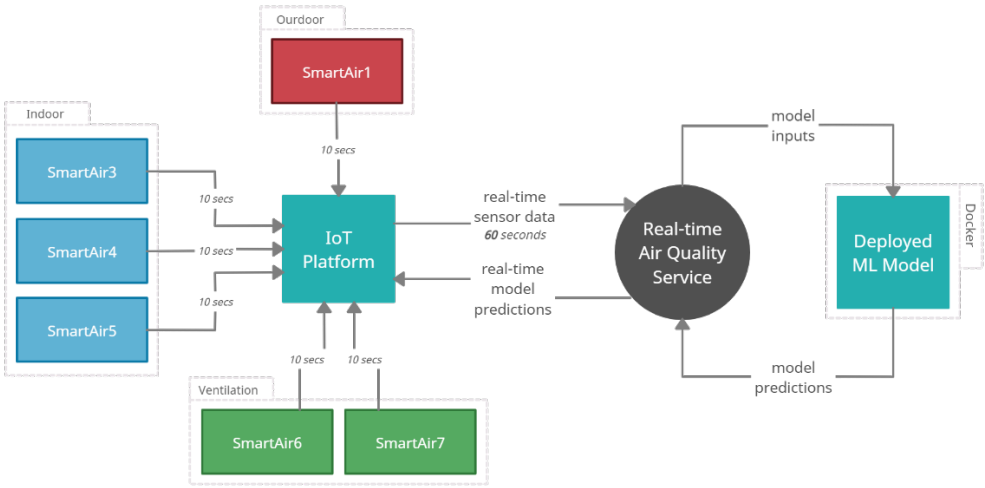
INTERNATIONAL HELLENIC UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF INFORMATION AND ELECTRONIC ENGINEERING

BACHELOR'S THESIS

«Machine learning assisted real-time air quality evaluation for Smart Homes»



Of the student
Tsokaktsi Stavroula
Reg. Number: 154557

Supervisor Professor:
Name Chatzimisios Periklis
Rank Professor

Date 15/09/2021

Title of Dissertation Machine learning assisted real-time air quality evaluation for Smart Homes
Code of Dissertation 20170
Student's full name Tsokaktsi Stavroula
Supervisor's full name Chatzimisios Periklis
Date of undertaking 29-04-2020
Date of completion 15-09-2021

I hereby affirm the authorship of this paper as well as the acknowledgement and credit of whichever assistance I received in its composition. I have, furthermore, noted the various sources from which I extracted data, ideas, visual or written material, in paraphrase or exact quotation. Moreover, I affirm the exclusive composition of this paper by myself only, for the purpose of it being a dissertation, in the Department of Information and Electronic Engineering of the I.H.U.

The approval of this dissertation by the Department of Information and Electronic Engineering of the International Hellenic University, does not necessarily entail the adoption of the author's views, on behalf of the Department.

Preface

The Internet of Things and the possibilities it provides are the future of humankind and technological evolution. Machine Learning, respectively, is attracting a lot of attention when it comes to automating human involvement in simple or more complex procedures. The combination of the two technologies have come very far achieving outstanding results in various aspects of our lives and will surely be responsible, to a very large extent, for shaping our daily lives in the years to come. The objective through the elaboration and application of this work is to study and understand these technologies, and their joint contribution to the global technological scheme, while implementing a project that demonstrates their great potential.

As long as technology is evolving, humans will be shifting their behaviour accordingly. People tend to spend most of their time indoors nowadays and this new aspect of our lives introduces new challenges. One of these challenges involves the deterioration of Indoor Air Quality and its direct impact on our health and quality of life. This work commits to tackling this invisible yet critical encounter, as it threatens us and our environment, by witnessing these novel technologies joining forces.

This thesis was implemented as part of a six-month internship of the undergraduate student Tsokaktsi Stavroula at the University of Bristol in 2019.

Abstract

The concept of Internet of Things is rapidly growing, enabling a variety of applications that enhance the quality of everyday life. IoT is a product of the convergence of multiple technologies, such as Machine Learning, real-time analytics, and wireless sensor networks. To this end, in this dissertation, we examine the potential of combining the aforementioned mechanisms having as a core idea the real-time evaluation of Indoor Air Quality. A thorough study is being conducted on related works and Machine Learning algorithms and implementation tools. Thereinafter, we utilise IoT sensors, monitor their measurements and apply data manipulation techniques on the gathered information aiming to prepare them for Machine Learning. A Neural Network is being tuned and trained, taking advantage of state-of-the-art tools, and, finally, achieving a rather low error rate. As a final phase, the model is deployed into production on a real-life server using a Docker container and provides real-time air quality evaluation predictions on an ongoing basis. Those predictions are visualised into a web-based IoT dashboard, offering a cross-platform end application with continued access to the present state of the air quality in the specified environment. According to this information, this system could potentially provide maintenance services to uphold a healthy environment of the surrounding space.

Today, Smart Cities and Smart Homes are becoming more and more popular, in an era that everyone is trying to automate procedures and simplify our everyday lives. To this end, this concept is implemented as an end-to-end Smart Home case study to demonstrate the great potential of utilising Machine Learning for home automation. This project is an attempt to take one step further and prove that major technologies, as in this case Internet of Things and Machine Learning, can accomplish even greater things together.

Acknowledgements

This thesis would not have been possible without Erasmus+ and Mr Chatzimisios, so I would like to thank him for his support and help in the elaboration and implementation of this work.

Furthermore, the University of Bristol and High Performance Networks group gave me the opportunity to take advantage of the software, hardware and Internet of Things sensory equipment, which in turn enabled me to implement this end-to-end experiment that would not have been conceivable otherwise. I should also not disregard this six-month experience, the people and knowledge I gained, that will stay with me for a lifetime and for which I am grateful.

Last but not least, I would like to thank the Department of Information and Electronic Engineering (IEE), International Hellenic University (IHU) and its professors for the acquisition of knowledge throughout these years that will help me establish my presence in this industry.

Table of Contents

Table of Figures

List of Tables

Abbreviations

AF	Activation Function
AI	Artificial Intelligence
ANN	Artificial Neural Networks
ASIC	Application-Specific Integrated Circuits
BLE	Bluetooth Low Energy
BP	Backpropagation
CAFFE2	Convolutional Architecture for Fast Feature Embedding
CCA	Canonical Correlation Analysis
CDR	Call Detail Record analysis
CI	Confidence Interval
CNTK	Microsoft Cognitive Toolkit
CTF	frequency-domain Channel Transfer Function
DNS	Domain Name System
DPI	Deep Packet Inspection
DT	Decision Trees
ELM	Extreme Learning Machines
ESN	Echo State Networks
FAIR	Facebook's AI Research lab
FANN	Fast Artificial Neural Network library
FCF	Frequency Coherence Function
FTP	File Transfer Protocol
GA	Genetic Algorithm
GSoC	Google Summer of Code
HBM	High Bandwidth Memory
HPN	High Performance Networks research group
HVAC	Heating, Ventilation, and Air Conditioning system
IAP	Indoor Air Pressure
IDS	Intrusion Detection System
IEQ	Indoor Environmental Quality
INRIA	Institut national de recherche en informatique et en automatique
IoT	Internet of Things
IPS	Intrusion Prevention System
ISP	Internet Service Provider
k-NN	k-Nearest Neighbour
LDA	Linear Discriminant Analysis
LoRa	Long-Range, low power wireless transmission protocol
LoRaWAN	Long-Range Wide-Area Network
LPWAN	Low-Power Wide-Area Network
LSTM	Long Short-Term Memory neural network

MAE	Mean Absolute Error
MAP	Maximum A Posteriori hypothesis
MCU	Microcontroller Unit
MILA	Montreal Institute for Learning Algorithms
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NB	Naive Bayes
NFV	Network Functions Virtualisation
NLP	Natural Language Processing
NN	Neural Networks
NumPy	Numerical Python extensions
OAP	Outdoor Air Pressure
ONNX	Open Neural Network Exchange
P2P	Peer to Peer
PCA	Principal Component Analysis
PM	Particulate Matter
QoS	Quality of Service
RBF	Radial Basis Function kernel function
ReLU	Rectified Linear Unit activation function
RF	Radio Frequency
RH	Relative Humidity
RMSE	Root Mean Squared Error
RSS	Received Signal Strength
SBS	Small Cell Base Station
SciPy	Scientific Python
SCN	Small Cell Network
SDN	Software-Defined Networking
SGD	Stochastic Gradient Descent
SOM	Self-Organising Map
SVM	Support Vector Machines
SVR	Support Vector Regression
T	Temperature
TFX	TensorFlow Extended
TPU	Tensor Processing Units
UTM	Unified Threat Modelling
VAP	Ventilation Air Pressure
VM	Virtual Machine
VOCs	Volatile Organic Compounds
VR	Virtual Reality
WHO	World Health Organisation

Chapter 1 Fundamentals

1.1 Introduction

In this chapter, we undertake a literature study to identify similar works in the field of Machine Learning (ML) and Internet of Things (IoT). To this end, we analyse associated previous works in terms of ideas and concrete results. To further enhance our vision in the search of shaping our proposed application, we perform the following study that describes ML techniques with IoT equipment. Our primary goal is to identify potential techniques and suggestions that will shape and inspire the guidelines of the current project. A project that will take advantage of these leveraging technologies, the available sensory equipment, the acquired knowledge, and the willingness to advance and improve quality of life. Consequently, we will describe in greater detail the shaped idea and objective, the guidelines and motivation, and the structure that this thesis followed.

1.2 Literature Review

This section focuses on studying and presenting scientific papers and works that implement Machine Learning techniques and/or utilise Internet of Things technology and equipment. The objective here is to get inspired by these use cases and form and finalise the theme, details, and main goals of this project. We analyse one paper at a time and underline its concept, key features, results and the technical approach on the matter.

In [1], the authors focus on Software-Defined Networking (SDN), which has started to gain interest in the field of mobile networks. SDN performs resource allocation efficiently and dynamically relying on quick and accurate identification of traffic flows in the network. Originally, in terms of traffic classification, there are three methods: port-based, payload-based and machine learning-based. The latter seems to be the most suitable in this task since it leads to lower computational costs and identifies encrypted traffic quickly. The paper's results show that machine learning can achieve overall accuracy of over 95%. Hence it is suitable for various SDN applications. More specifically, as proved in [2], the Bayesian classification method can separate several applications successfully with 86.5% average accuracy.

The authors consider two ML algorithms for traffic classification, the supervised Support Vector Machines (SVM) and the unsupervised K-means clustering. The experiments have been carried out using traffic traces collected from a real research facility. Before the traffic classification, SVM model tuning and feature selection have been performed to optimise the performance and minimise the computational costs. Through the classification process, five-fold cross-validation has been used. Eventually, the SVM model, based on the Radial Basis kernel function, managed to provide the best performance with 98% overall accuracy. While this kernel contained only 13 attributes, it was still more computationally efficient than the other kernel functions. In contrast to SVM, K-means clustering achieves an average of 80% accuracy. Comparing the two, SVM seems to perform with higher precision

on all the different application types and is more stable in any circumstance. On the contrary, supervised learning methods (like SVM) have higher computational overhead because of training with pre-classified and labelled instances. K-means classifier can be dramatically improved by applying more traffic flows and co-training. While, at the same time, SVM can maintain a reasonably high performance of approximately 95% by applying only 5% of the training flows.

In conclusion, the traffic classification and application characterisation problems are becoming increasingly important with many applications in future 5G and SDN networks. Machine learning approaches overcome some of the limitations of traditional classification methods in Internet traffic characterisation. Based on the experimental results, the SVM model gives higher precision than the approaches based on unsupervised learning. The accuracy of the K-means classifier is highly affected by the size of the training dataset. At the same time, the SVM model can maintain relatively high classification accuracy by using a small portion of the training dataset. This exemption is advantageous as it has important implications for real-time applications. Nevertheless, something that should have been implemented differently in [1] is that the classification system should have been applied to real SDN traffic data. That could lead to entirely different, and probably more reliable, results and conclusions.

[3] is discussing the major influence of indoor air quality on the comfort and health of building residents. Good air quality contributes to productivity and learning in offices and classes, respectively. The concept of Smart Homes is gaining a lot of interest lately, in an effort of reducing human involvement in regulating and making HVAC (Heating, Ventilation, and Air Conditioning) systems more efficient. Crowd sensing has arisen as an IoT solution since it is scalable, cost-effective and easy to deploy. A solution like this is part of a Smart Home sensing system, which monitors and processes air quality data to further prompt an HVAC unit for necessary actions aiming at improved air quality and comfort. The authors have developed a resource-efficient intelligent mobile air quality sensing system, based on a microcontroller board equipped with sensors and an artificial intelligence engine. Utilising obtained concentrations of air pollutants and air quality parameters, the engine provides recommendations in real-time. The system is able to determine the current indoor air condition and advise appropriate countermeasures to the HVAC system for indoor air quality refinement.

More specifically, the artificial intelligence analytic engine is based on an Artificial Neural Network (ANN), which is receiving three air quality measurements as input and returns the comfort level for each of them as output. The authors utilised the Fast Artificial Neural Network library (FANN) in C/C++ environment. The whole deployed system consists of a low-power embedded processor, and temperature, humidity and CO_2 sensors, which communicate wirelessly. This sensor node is comprised of Arduino UNO, MQ135 (CO_2), HM-10 (Bluetooth), and DHT-11 (humidity/temperature) boards. This smart engine continuously evaluates these three measurements, which are being normalised, and shows whether they are within/exceed/below the specified comfort ranges, implemented as a regression problem. The received output is being used so as to perform an air quality improvement and is also being displayed in the deployed mobile app, along with the measurements. The final architecture of the neural network consists of three layers with the hidden layer having six neurons, chosen as a minimum number of neurons given the restricted power and memory resources of the Arduino UNO. The accuracy of the engine is 0.0002 MSE (Mean Squared Error) using the library's iRPROP (improved Resilient back Propagation) learning algorithm. The implemented ANN is deployed on the Arduino UNO board from

scratch using a customised Arduino C++ library. The final computation time of the network is around 1.4 seconds for one pair of inputs.

Taking everything into consideration, the authors of this paper managed to develop a resource-constrained mobile air quality platform, implemented as a Smart Home use case. The system continuously monitors indoor air, based on the real-time measurements of temperature, humidity and CO_2 concentrations, and determines if an air adjustment is required so as to improve air quality. Initially, the analytic ANN engine was developed individually from scratch, based only on the Arduino C++ library. The deployed network was successfully trained, achieving a 0.0002 MSE, 1.4 seconds response time and a lower memory and storage consumption of the Arduino UNO board. The only weakness of this experiment is that the engine was trained until a 0.0002 error was reached, requiring 400.000 epochs to complete. The resulting network may be successful based on the simplicity of the problem, but this approach might not prove to be applicable in more complex real-life scenarios.

What is being discussed in [4] is the Network Functions Virtualization (NFV), which manages to improve the flexibility, efficiency and manageability of networks. However, its implementation presents issues such as the management of bottlenecks, the introduction of new software components and the monitoring of the hidden traffic. Hidden traffic refers to the amount of traffic, in NFV, that is invisible using traditional monitoring strategies and communicates among Virtual Machines (VMs) running inside a physical host. It can lead to difficulties in diagnosing network performance, detecting malicious agents and handling errors. That is where autonomic management, and of course, Machine Learning, assist in coping with these challenges by automating and distributing decision making. ML algorithms have been successfully applied to deal with problems such as the accurate classification of traffic on traditional networks. As proved in [5], [6], [7], the supervised algorithms have excellent behaviour in this task. However, there is only a small number of works that focus on classifying traffic in NFV. This classification is different from regular networks because, in NFV, the traffic either flows by physical links or, occasionally, flows by virtual links exclusively.

In this work, the authors perform a benchmarking to analyse the behaviour of several supervised ML algorithms in the IP traffic classification in NFV-based networks. The results reveal that the Naive Bayes algorithm has the best performance in traffic classification in this kind of network. More specifically, it achieves a good trade-off between classification precision and time response, with 99,9% precision in 1,1 sec. The authors tested three supervised ML algorithms: J48, Naive Bayes and BayesNet. While they also performed a comparison between their performance in traditional networks and NFV-based ones. The results show that the overall precision in NFV is higher than 79.5%, being similar to the one achieved by the same algorithms in the traffic classification in traditional networks. This fact proves that they are the right choice for the classification in virtual networks. On the one hand, J48 analyses and selects the features with the highest gain of information, which makes it more precise with values between 88.8% and 99%. While Naive Bayes and BayesNet do longer and more complex data processing than J48. Furthermore, the Naive Bayes algorithm presents the smallest variation between the response times in the classification. So, although J48 and BayesNet achieve a precision greater than 88%, they have significant variations in their response times. This phenomenon happens because they depend on the amount of data contained in the dataset. While Naive Bayes reaches a precision greater than 79.5%, its response times are more stable. Therefore, J48 is the most accurate and Naive Bayes is

the most constant in time. Nevertheless, Naive Bayes manages to compensate that difference of 5.5% in precision, without reducing the precision but reducing the response time by about 6 seconds.

To sum up, a benchmarking of the behaviour of some supervised algorithms (J48, Naive Bayes and BayesNet) is used to classify traffic in NFV-based networks. The results reveal that J48 and Naive Bayes have a range of precision from 80% to 99% in both traditional and NFV-based networks. The new features identified for the NFV-based networks allow the algorithms to improve their classification models and lead to improvements in precision and reduction in response times. In this sense, the Naive Bayes is the most efficient classifier in NFV-based networks, since it has a precision value range upper than 79.5%, similar to J48, and between response times it has minimal variations. To verify the behaviour of the supervised algorithms in more extensive networks, the authors should have had generated and collected traffic data on a broader environment, with more types of services and data.

The authors in [8] present a comparison of internet traffic identification on machine learning methods. Traffic classification is an essential part of computer networks since it enables managing the network better, filtering the insecure network flows and providing better network services. Furthermore, it is a critical component in abnormal detection, intrusion detection, firewalls and control of the Quality of Service (QoS). Conventional traffic classification methods (such as port-based method and Deep Packet Inspection - DPI) are unable to work well because of the change in the network environment and the rising flow complexity. Machine learning methods have become the most efficient way of solving this kind of problem and have been utilised in traffic classification to solve new challenges and achieve good results. It mainly aims at training the classification model based on internet traffic features. The main idea of traffic application identification is to inspect the features of internet traffic and identify the application categories of network flows. Traditional methods commonly rely on the port numbers of TCP or UDP packets (port-based) or inspect the payload content to find out the signatures (DPI). However, these methods are facing many problems. Ports can recognise the application running in two endpoints, so port numbers can be utilised to identify the application categories. Though this method is fast and easy to realise, many applications today use dynamic ports to transfer data to the other end, or they do not even use well-known port numbers. This fact results in the failure of the method based on ports. In like manner, DPI searches the specific bytes in the packet (known as signatures) to identify flow applications. However, this method needs to keep pace with the change of protocols. It puts much effort into analysing the structure of the network packets, and it is challenging to identify encrypted traffic. Nowadays, the statistical characteristics of network flow are used to identify the traffic application. ML techniques overcome the faults of the other methods and become an essential direction in these tasks.

ML algorithms were introduced in traffic identification in intrusion detection systems many years ago. In [8], classic algorithms, in supervised and unsupervised learning methods, are being discussed and establish a model for abnormal traffic detection. Data normalisation had been performed beforehand. So, the convergence rate and the accuracy of the model are expected to be improved a lot. K-means is a classic unsupervised algorithm whose core idea is to find the centroids of the clusters. It can be well used to cluster the internet traffic into disjoint groups based on similar flow features and identify the specific application of clusters. K-means can classify the data into different disjoint classes, and then the application of the classes can be identified based on the majority label of each cluster. SVM is a supervised learning algorithm, and its target is to establish a model that is capable of predicting the test data to an application class. In, the most commonly found, nonlinear classifying problems, different

kernels can be used to transform the initial features to a higher dimensional space. According to the results of [9], Linear and RBF (Gaussian) kernels have higher accuracy. As a consequence, K-means and SVM with Linear and RBF kernels are being compared. The results reveal that, after the feature normalisation, the K-means' training time is 111.4 seconds, the prediction time is 0.25 seconds, and the accuracy is 83.12%. For the Linear kernel, SVM's training time is 39.8 seconds, the prediction time is 55.03 seconds, and the accuracy is 91.53%. Lastly, for the RBF kernel, SVM's training time is 80.99 seconds, the prediction time is 168.5 seconds, and the accuracy is about 91.48%. Comparing the results, K-means is slower than SVM during the training phase but much faster than SVM in the testing phase. At the same time, the accuracy of SVM is higher than the K-means algorithm in both Linear and RBF kernels.

To sum up, K-means is slower in training the model, but it is faster in identifying the application of the traffic. So, it is more suitable for a fast, abnormal report. In comparison, SVM is better in conducting precise classification and identification. In some cases, the combination of these two approaches can actually improve the efficiency and the accuracy of traffic identification. That is something that could have been implemented in [8], and it could have achieved a much more efficient identification of internet traffic.

In [10], the main subject is the wide-spreading usages of the internet and increases in access to online content, which leads to an increasing cybercrime rate. The first step in warding off security breaches is intrusion detection. Hence many security solutions, such as Unified Threat Modelling (UTM), Intrusion Detection System (IDS) and Intrusion Prevention System (IPS), are getting more attention in studies. IDS collects information and can distinguish attacks from various network sources and systems. Following that, it analyses the information for potential security breaches. Till today, anomaly-based detection is lagging behind the detection that works based on signatures. Hence, anomaly-based detection still remains a significant area for research. One of the challenges is that it needs to deal with a novel attack, for which there is no prior knowledge to identify the anomaly. The system needs to be intelligent enough to determine which traffic is anomalous or malicious and which one is harmless. So, for that, ML techniques are being explored by researchers over the last few years, and currently, anomaly-based detection is a significant focus area of research and development. Several anomaly-based techniques have been proposed including Linear Regression, Support Vector Machines (SVM), Genetic Algorithms, k-Nearest Neighbour algorithm (k-NN), Naive Bayes classifier and Decision Trees (DT) [11], [12]. The most commonly used learning algorithm is SVM, as it has already established itself on numerous kinds of problems [13]. All the aforementioned techniques can detect novel attacks, but they all suffer a high false alarm rate in general. The cause of this is the complexity of generating profiles of practically normal behaviour by learning from the training datasets [14]. Additionally, today, Artificial Neural Networks (ANN) are often trained by the backpropagation algorithm, which is the reverse mode of automatic differentiation [15].

Here the authors used the two supervised learning algorithms, SVM and ANN, on the NSL-KDD dataset, which is a popular benchmark dataset for network intrusion. As mentioned before, in SVM, a separating hyperplane defines the classifier depending on the problem type and the available dataset. In contrast, Artificial Neural Network is another powerful tool used in Machine Learning. It is a system inspired by the human brain, and it replicates the way it learns. It uses a technique called backpropagation to adjust the outcome to the expected result or class. Furthermore, cross-validation was performed to avoid

overfitting and underfitting. Regarding feature selection, the Chi-Square algorithm retained 35 features as the most relevant to the resultant class, whereas the Correlation-based feature selection retained only 17 features. For the classification, SVM and ANN were applied to each type of feature selection method. Eventually, the detection accuracy that was accomplished using the SVM with 17 features was 81.78%, and with 35 features, it was 82.34%. While for the ANN with 17 features it was 94.02% and 83.68% having 35. Multiple experiments have been performed with different values on the configuration parameters, such as the learning rate, the number of instances and the hidden layers. This experimentation led to obtaining the optimum results.

In [10], the authors presented two different machine learning methods using two feature selection techniques to find the best model. The analysis of the results shows that the model built using ANN with wrapper feature selection outperformed all the other models in classifying network traffic correctly, having a 94.02% detection rate. Implementing the solution on a large-scale network would have definitely had much more realistic and reliable results. Unfortunately, for the time being, the intrusion detection system that exists can only detect known attacks. Because of the high false-positive rate of the existing systems, tracing new or zero-day attacks still applies as a common research subject.

The authors in [5] focus on network traffic classification as it is an important topic nowadays in the field of Computer Science. It actually is essential for Internet Service Providers (ISPs) to manage the overall performance of a network and be aware of which types of network applications flow in it. Traffic classification is the first step to analyse, identify and classify different types of applications flowing in a network. It plays a very vital role in network security and management, such as Intrusion Detection and Quality of Service (QoS). Through this technique, network operators can take some actions, such as blocking some flows and managing resources. In the last twenty years, several network traffic classification techniques have been presented to categorise unknown classes. The first one was the port-based technique that searches for the port number, which is firstly registered in IANA. However, it failed due to the increase of Peer to Peer (P2P) applications, which use dynamic port numbers. The second one is payload-based, which generally gives accurate results in network traffic classification and is also called Deep Packet Inspection (DPI) technique. The problem with this is that it cannot be used for encrypted data network applications since numerous applications use encrypted techniques to protect data from detection. So, this technique failed as well. The Machine Learning technique though gives promising accuracy results in network traffic classification and is able to classify internet traffic as well as to know what type of applications flow in the network.

A comparative analysis of four machine learning classifiers, Support Vector Machine, C4.5 decision tree, Naive Bayes and Bayes Net, are being discussed here. First, network traffic had been captured using the packet capturing application Wireshark. After that, using the NetMate tool, 23 features had been extracted from the captured traffic and then four machine learning classifiers were applied to classify WWW, DNS, FTP, P2P and Telnet applications. After applying 10-fold cross-validation, the experimental results show that the C4.5 classifier gives high accuracy of 78.91% in comparison to other machine learning classifiers. In particular, in terms of accuracy, SVM achieves 74.05%, BayesNet 68.10% and Naive Bayes 71.89%. It is clear that the C4.5 classifier gives a better accuracy result than the other applied ML classifiers. In terms of recall and precision, C4.5 obtains an excellent result compared to the others, with approximately 80.88% recall and 49.99% precision. Comparing the precision and recall results of the five captured (WWW, DNS, FTP, P2P and TELNET) applications, it is clear that DNS and WWW recall and precision are very poor. When at the same time, in FTP, P2P

and TELNET applications, all the classifiers achieved roughly the same optimum results of precision and recall.

Concluding, machine learning algorithms are being applied extensively in the field of network traffic classification. They are able to manage the performance of the network and identify unknown applications. A comparative analysis of four machine learning classifiers (Support Vector Machine, C4.5 decision tree, Naive Bayes and Bayes Net) on different types of applications is performed. The results reveal that the C4.5 decision algorithm gives higher accuracy than the rest of the applied ML methods and can achieve the best results in terms of precision and recall among the different applications and classifiers. The problem in [5] is that the duration of the traffic capture was only 1 minute and cannot be considered as a very representative and competent sample to be depending on.

In [16], the subject that is being discussed is the evolving Internet of Things (IoT) applications, that revolutionised the field of telecommunications. These applications use smart sensors that efficiently work together to offer the desired service. The effective deployment of IoT-based sensors mostly relies on enabling the modification of sensor power consumption on the Radio Frequency (RF) propagation channel. The type of the surrounding indoor environment dictates this fact. The performance of this task is significantly improved by classifying the type of the surrounding indoor environment. This classification is critical when operating the deployed IoT sensors, as it leads to efficient power consumption. A Machine Learning approach is proposed for indoor environment classification based on real-time data measurements of the radio frequency (RF) signal in a realistic environment. This approach is achieved by exploring the use of three RF signatures: Received Signal Strength (RSS), frequency-domain Channel Transfer Function (CTF) and the autocorrelation of the CTF. The latter is denoted by the Frequency Coherence Function (FCF), along with different ML algorithms such as Decision Trees (DT), Support Vector Machines (SVM), and k-Nearest Neighbour (k-NN).

First of all, the Decision Tree Machine Learning method is widely used for classification and is based on a binary DT that is constructed by the training data. It consists of three different types of nodes: root node, internal and leaf. The algorithm stops splitting into subtrees when the uncertainty is minimum. Moreover, SVM is widely used for classification as well. Its objective is to find the optimal margin classifier, and it can be formulated as an optimisation problem. The k-NN is a nonparametric method used for classification. The test sample is assigned to the class, which is the majority of its neighbours. According to the results, the classification accuracy using RSS can only reach up to 42.7% maximum in the case of SVM with Gaussian kernel function. With CTF, the classification accuracy increases and reaches up to 79.9% in the case of weighted k-NN (k = 10). When FCF is utilised, the classification accuracy keeps improving, reaching up to 83.4% in the case of k-NN (k = 1). Combining RSS with CTF enhances the accuracy of classification compared to the individual performance, but it does not surpass the accuracy obtained with FCF. Unlike the combination with CTF, when RSS is combined with the FCF, the classification accuracy can reach up to 93.4%. The combination of CTF and FCF actually has the highest classification accuracy of 99.3%, which is obtained as well when all three features are combined. Adding RSS to the combination of FCF and CTF features does not significantly increase accuracy. Hence, it can be concluded that a combination of only the CTF and FCF features, is sufficient for accurate classification of indoor environments. Moreover, successful real-time deployment of ML algorithms requires minimum computational time. It was found that the prediction time for the k-NN (k = 1) algorithms is below 10 μs. This time indicates that the adopted algorithms are very applicable in

practical real-time use-cases. Furthermore, while the classification accuracy of k-NN degrades significantly at k = 100, in the case of the weighted k-NN, the accuracy is significantly high with a slight variation and does not go below 98.5%.

The authors in [16] focused on a ML approach for indoor environment classification based on real-time measurements of the Radio Frequency signal. Several ML classification methods were tested using different RF features. The results obtained reveal that the weighted k-NN method, utilising CTF combined with FCF, outperforms the other methods in identifying the type of indoor environment, with an accuracy of 99.3%. Additionally, the prediction time was below 10 μs, which verifies that this algorithm is a successful option for real-time deployment scenarios. The results facilitate the efficient deployment of IoT applications in dynamic channels and highlight the benefits of using ML as a versatile tool for indoor environment classification. This work obtained the results based on measurements in a stationary environment with a distinct line-of-sight component. A very intriguing addition to this would be to explore other scenarios, including non-line-of-sight and time-varying channels.

The study in [17] focuses on the negative effects of the concentration of PM2.5 particulate matter on human health. These concentration levels are determined by many nonlinear and uncertain factors, thus predicting them cannot be done effectively using traditional approaches. The authors suggested a system that monitors the number of air pollutants outdoors utilising LoRa wireless connectivity and a cloud-based model. By forecasting the changes of PM2.5, this system can assist people in planning in advance and preserving their health during periods of increased air pollution. Based on related works, a Long Short-Term Memory (LSTM) neural network was chosen to perform the predictions of the PM2.5 values that change over time. LSTMs can achieve a much lower percentage error compared to traditional neural networks and SVMs when predicting future values over time-series scenarios. The proposed system consists of four elements, a sensing device (Arduino Mega board, PM2.5 sensor & LoRa shield), a LoRa gateway, a server and a final application.

The terminal sensing device is used to sense and measure the PM2.5 values and their timestamp. All datasets are, then, packaged and transmitted through a LoRaWAN connection to the LoRa gateway every 1 hour, which in turn delivers them to the cloud server via WiFi. There, the datasets are sorted by time and device ID and are being displayed on an internet page so as to check whether the device is functioning properly and the measurements are valid. These three components comprise the preparation phase and implement the "detection functionality", in which they collect and store the acquired measurements. The sensing interval is 12 seconds and the total data collection period is 50 days, from January 1st to February 20th 2019. The last element, the application, analyses and forecasts the PM2.5 values, by requesting the stored datasets from the server. It visualises the data and analyses the air quality situation, while, using an LSTM deep learning model, it predicts the next few values of PM2.5. During the pre-processing phase of the data, the valid and accepted entries are being normalised by the min-max scale method. Additionally, since the time step can greatly affect the accuracy of the model, the experiment was conducted by comparing three different time step values, 12 hours (h), 24 h and 48 h. Regarding the analysis of the air pollution levels, the pie chart concluded relatively good results, with 90.17% acceptable air pollution levels (40.66% Excellent, 35.49% Good, 14.02% lightly polluted). The performance of the LSTM model during training was found to be the best in the case of the 48 hours time step, with 11.50 MAE (Mean Absolute Error) and 18.65 RMSE (Root Mean Squared Error). The other two cases resulted in 12.58 MAE and 22.93 RMSE for the 24h and 14.11 MAE and 25.29 RMSE

for the 12h. With the 48 hours time step, the forecasted values have been found to be leaning very close to the real data.

Concluding the study of [17], the paper implemented a forecasting system of PM2.5 values utilising LoRa wireless connectivity and LSTM neural network predictions. The authors deployed the experiment in a real outdoors environment and successfully forecasted and analysed the PM2.5 measurements for the next 12h, 24h and 48h. The best performance was found in the 48h time step experiment with 11.5 MAE and 18.65 RMSE. For the implementation of the system, a sensing device and LoRa gateway was used through a LoRaWAN connection and a cloud server through a WiFi connection. Finally, an application was developed that analyses and visualises the results, but the authors proposed the development of a mobile app that warns about extreme air pollution levels in real-time as a future plan. [17] is a notable work and the only suggestion is that there should be more sensing devices deployed, with the intention of covering a broader area and possibly a greater amount of different air conditions.

In [18], the flourishing technology of IoT is being introduced, while it is rapidly changing the way of living. There is an exponentially increasing number of intelligent and connected edge devices in everyday life. Nevertheless, battery life and communication over long distances are major obstacles for off-grid IoT applications. For data processing, the most widespread method has been the use of the Cloud for data analysis needs, where raw data are transmitted there for analysis and processing. Raw data transmission is considered a very power-hungry activity for any device. For that reason, the transmission of a large volume of data necessitates a continuous power source or regular charging, narrowing the scope of Cloud-based IoT. Moreover, there is a gradually increasing necessity for IoT applications to be able to transmit data over long distances. Thus, the ability to transmit data over long distances while maintaining power efficiency is an essential factor in IoT devices. Those off-grid devices also necessitate edge computing so as to be exempted from the high transmission bandwidth need and the reliance on the Cloud as a data analysis platform. Low-power edge computing devices that cut the transmission payload and integrate Low-Power Wide-Area Network (LPWAN) technologies are in great demand. These devices can provide a long battery life while still offering wide range connectivity. One of the most promising LPWAN technologies is LoRa, a long-range, low power wireless transmission protocol. The use of long-range small-bandwidth protocols, like LoRa, can evolve the efficient use of data compression and local data analysis.

The use of LoRa, coupled with ML on an embedded system (edge device), and an analysis of various factors affecting the battery life, is demonstrated by the authors. By implementing embedded ML with LoRa, they could compress the transmitted data by 512 times and extend the battery life by three times. All the network parameters and the system architecture, where the experiment and the measurements had taken place, conform to the LoRaWAN specifications. Activity classification happens on the end device. The authors focused on the k-Nearest Neighbour (k-NN) algorithm mainly because of its ease and simplicity of implementation and high enough accuracy of 96.2%. The accuracy of the offline testing attained by the algorithm was 96.2% and the offline training was 91.4%. While, in the online classification testing, the labels matched for 95.5% of the samples. Additionally, the wakeup time for the microprocessor was less than 50 μs, which is much smaller than the data capture time, which is of the order of milliseconds.

The authors in [18] designed an activity classification IoT system, which benefits from the use of LoRa and an embedded k-NN ML algorithm to save on its energy requirements. The use of embedded ML resulted in 512 times compression in data. The integration of LoRa into the system and the combination of activity data before transmission lowered the energy consumption by 3 times, ending up with 39 days of battery life. They actually achieved a very low energy expenditure of 5.1 mJoule per classification. Finally, they propose an optimisation methodology to achieve further savings in power to extend the battery life to 331 days and achieve an energy expense of 0.597 mJoule per classification. This considerable power saving could be accomplished by putting the Microcontroller Unit (MCU) in a low power "stop mode" when no capturing or any processing is going on. When for the moment, the MCU always stays in active mode. Even though more data collection is required for accurate performance, the aim was to show the flexibility of the system in adapting to various applications with distinct features for the ML model and a different number of classes.

The authors in [19] focused on the security of the Android operating system, which occupies a significant share in the mobile application market. Android mobiles have turned into easy prey for attackers. The main reason is user ignorance in the process of installing and using the apps. Android malware apps can be identified against genuine apps based on the permissions that the app requests during installation. Several ML algorithms are employed for the detection of android malware based on the list of permissions enabled for each app. There is a need to identify the most performing technique in the identification of malware in android app data. That is the reason why the performance of several ML algorithms is being compared and evaluated in [19]. Some of them are Naive Bayes, J48, Random Forest, Multi-class classifier and Multi-layer Perceptron (neural network). Google Play Store 2015 and 2016 app data are used for regular apps, and standard malware datasets are used in the evaluation.

The proposed method has mainly three stages. Firstly, the permission fields are extracted from the android manifest file of the apps. Second, a database of all the permissions for both regular and malware data is established and, finally, the ML algorithms are used to classify and identify the malware in android applications [20], [21], [22], [23]. By observing the experimental results of all the five classifiers on four different datasets, the Multi-class classifier has performed better over the other classification algorithms with an average accuracy of 99.81% (in 0.23 seconds). However, Naive Bayes classifier performed better as far as computational complexity is concerned (0.065 seconds) having an average accuracy of 99.19%. The Multi-layer Perceptron (MLP) performed pretty close to the Multi-class classifier with 99.68%, but its computational complexity was inferior (356.8 seconds).

In conclusion, in [19], different ML algorithms were used to detect android malware and evaluate the performance of each of them. A framework was implemented for classifying android applications with the help of ML techniques to check whether it is a malware or standard application. For validation purposes, 3258 android applications were collected and used for training the models. Then, the evaluation was made with the help of the classification accuracy and the model's response time. From the experiment, it was found that the Multi-class classifier has performed better than other methods, regarding classification accuracy. In comparison, Naive Bayes was found to be the fastest, as far as computational complexity is considered. Lastly, feature reduction could significantly improve the performance of a classifier [24], and that would have been a remarkable enhancement of [19].

In [25], the authors introduce the problem of strictly energy-constrained IoT sensing devices, such as wearable sensors. They require much energy for the data generation and their reliable wireless

communication to a central location. This makes maintaining them a challenging task, in terms of cost-effectiveness, due to their dependence on constant recharging of their battery. To effectively achieve their purpose, sensing infrastructures should require close-to-zero maintenance. These sensing systems generate raw data that is processed into valuable knowledge by reasoning techniques and ML algorithms. The benefits of embedded ML are being investigated since the output knowledge can be represented in fewer bytes than the original raw data. Therefore, it is beneficial for the battery life to execute the knowledge extraction on the wearable sensor, instead of communicating numerous raw data over the low power network. By utilising embedded machine learning, a great reduction in the radio and processor duty cycle can be achieved, hence considerably prolonging the battery lifetime of resource-constrained wearable sensors.

The authors implemented a residential monitoring system that makes use of wearable sensors to track the physical activity levels of house occupants. The system attempts to periodically categorise the user's activity levels into three types: sedentary, moderate, and vigorous activities. For the classification, the SVM classifier was employed based on the RBF (Radial Basis Function) kernel (configured with a box constraint $c = 100$ and a scaling factor $\sigma = 1$). In the raw data approach (the traditional way), the average classification accuracy was 93.23% with a standard deviation of 6.57%. Assuming an energy budget of 1000 Joule (standard for the size of a wearable battery), the energy consumption rate, in this instance, is equivalent to approximately 13 days of battery life. In the case of optimising the data generation (optimisation of data sampling), the classification task can be effectively executed (more than 90% accuracy rating) with a sampling frequency of as low as 0.39 Hz and a 5 bits resolution. Its energy consumption rate is equivalent to a battery life of roughly 771 days. With the feature extraction being embedded, the energy consumption rate corresponds to approximately 989 days. While when the whole process of the classification is embedded, the battery life is estimated to be approximately 997 days. This stage has decreased the radio duty cycle by nearly an order of magnitude. The overall improvement is minuscule, however, because of the idle power, which severely limits the performance, and emphasises the necessity for more energy-efficient components on the wearable sensor. Regardless, with embedded ML, the battery lifetime of the wearable sensor is significantly extended from weeks to years.

[25] investigates the advantages of using embedded ML as a method of prolonging the battery lifetime of critically energy-constrained embedded systems. By optimising the data collection to a particular application and moving the knowledge extraction closer to the data source, the duty cycle of the radio and processor have been reduced by several orders of magnitude. It effectively extends the battery lifetime of the wearable sensor to years. The raw data approach, though, has significant advantages that should not be neglected. Instead, the designers of monitoring systems should consider the benefits of both approaches and make a decision based on their design priorities. In essence, this is a compromise between versatility and efficiency.

[26] examines the problem of resource management in a network, where wirelessly connected Virtual Reality (VR) users communicate over Small Cell Networks (SCNs). The idea is that in order to enable genuinely immersive VR applications, one can deploy wireless VR systems that use reliable wireless connections, such as those provided by cellular networks. Specifically, VR systems can utilise the wireless connectivity of emerging Small Cell Networks (SCNs), in which Small Cell Base Stations (SBSs) can act as the VR control centres which connect directly to the VR devices over wireless cellular links. Consequently, the SBSs will collect the tracking information from the VR user devices over the

cellular uplink. Once this information is collected, the SBSs will then send the VR three dimensional images (and surround stereo audio) to the VR user devices over the wireless cellular downlink. To capture the VR users' quality-of-service (QoS) and effectively quantify it for all users, a VR model, based on multi-attribute utility theory [27], is proposed. This model considers equally VR metrics such as transmission delay, processing delay and tracking accuracy. While for the resource allocation problem in VR wireless networks, it must also consider both the uplink and downlink.

The problem is formulated as a noncooperative game, in which the players are the SBSs. Each SBS looks for an optimal spectrum allocation scheme so as to optimise a utility function that captures the VR QoS. In this regard, a distributed algorithm based on the ML framework of Echo State Networks (ESNs) [28] is proposed to find the solution to this game. [26] focuses on recreational VR applications, like playing immersive games and watching immersive videos. This kind of application has much more stringent requirements in terms of data rate, delay and reliability than traditional multimedia services. In the simulation performed, an SCN is deployed within a circular area with a 100m radius, 25 users and 4 SBSs uniformly distributed. The Oculus VR device is used. Moreover, for comparison purposes, a baseline Q-learning algorithm is implemented. The wireless transmission is simulated, whereas the users' localisation data is gauged from a real wired Oculus VR device. A large number of independent runs was executed and all the statistical results are averaged. The results reveal that the VR QoS depends on both delay and tracking. This fact means that if something goes wrong with one of them, it will affect the other. Furthermore, tracking accuracy affects the processing delay as well. As the amount of SBSs grows so does the interference from the SBSs to the user. Moreover, the proposed algorithm achieves up to 18.6% gain in average delay compared to the Q-learning algorithm in the case of 6 SBSs. At the same time, it can yield up to 16.1% of gain in an average of the total VR QoS utility compared to the Q-learning in the case of 5 SBSs. The ESN-based learning algorithm also allows the wireless VR transmission to meet the typical delay requirement of VR applications. These gains stem from the fact that the proposed algorithm uses the past ESN information to find a better solution for allocating the resources for the proposed game. Last but not least, the proposed algorithm achieves a 25% gain in terms of the number of iterations needed to reach convergence compared to Q-learning.

In [26], the authors developed a multi-attribute utility theory-based VR model that can capture the delay and tracking components of VR QoS. Based on this model, they suggested an innovative resource allocation framework for optimising the QoS of VR for all users. In order to improve the overall QoS, a novel algorithm has been developed based on the ML tools of Echo State Networks (ESN). This algorithm enables each SBS to decide on its actions autonomously according to the users' and networks' states. Furthermore, the proposed learning algorithm only needs to update the mixed strategy throughout the training process. Therefore, it can swiftly converge afterwards. The results of the experiment have proven that the suggested VR model can capture the VR QoS in wireless networks. Furthermore, the results prove that, compared to conventional approaches, the proposed model has significant performance gains regarding total VR QoS utilities for all users. The only thing that is questionable in [26] is that the wireless transmission was simulated and not actually performed as an experiment, which requires further investigation for its results to be undeniably accepted.

These were the related works on Machine Learning and Internet of Things that were studied in order to create a vision of what is being implemented in research and industry scenarios. As noticed, there is a vast variety of fields of application creating a great number of options. This study is going to assist in forming an idea and problem that will best address the shortcomings of these works and essentially help

create an application that truly enhances our quality of life. The subject is going to be a matter of interest and, of course, the accessible hardware equipment, while we also aim to implement an end-to-end application that reflects the needs of today's lifestyles and hazards.

1.3 Main Objective and Motivation

The concept of Internet of Things is rapidly growing, enabling a variety of applications that enhance the quality of everyday life. IoT is a product of the convergence of multiple technologies, such as Machine Learning, real-time analytics, and wireless sensor networks. Organisations in a range of industries are gradually using IoT to improve decision-making, function more efficiently, and understand their customers better to provide greater customer services, thus increasing the value of the business. IoT aids people in gaining total control over their lives, by helping them work and live smarter. Besides offering smart devices for home automation, IoT is also vital to businesses. It delivers insights from the performance of machines to logistics operations and supply chain and provides them with a real-time view of how their systems truly work. A home automation business can surveil and operate electrical and mechanical systems in a building by utilising IoT. Smart Homes that are outfitted with connected lighting, heating and electronic devices, and smart appliances and thermostats can be remotely controlled through smartphones and computers. The aim of all these is to make users' lives easier and more comfortable, and that is the reason why this technology has been becoming more and more popular lately. For instance, smart buildings can cut energy costs using sensors that detect the number of occupants in a room. The temperature can be automatically adjusted, by lowering the heat if everyone in the office has gone home or turning the air conditioner on if sensors detect a conference room is full. On an even broader scale, Smart Cities can help citizens reduce waste and energy consumption. IoT can also make use of Artificial Intelligence (AI) and Machine Learning to aid in making data collecting processes easier, more dynamic, and automatic, or even turn any kind of procedure into a smart one, as in non-human intervention required for them to operate. As such, IoT is one of the most important technologies of everyday life, and it will continue to pick up steam as more people and businesses realise the potential of interconnected devices to keep them competitive and simplify their lives.

Meanwhile, Machine Learning algorithms are responsible for the vast majority of the AI advancements and applications that everyone hears about. Abilities like emails detecting spam, webpages offering advertisements of the products we are interested in, speech and text recognition, are some of the powerful examples of our daily lives being powered by Machine Learning. Machine Learning has the power to reshape humanity for the better in several aspects of our lives. Nowadays, it is being used to a great extent in various industries, for instance in medicine, genetics, finance and automobiles to automate processes, eliminating the possibility of human error while also decreasing the processing time. Additionally, it helps in analysing at scale, thus helping decision-making become even quicker and better. As of 2017, 25% of organisations are spending over a sixth of their IT budget on Machine Learning capabilities [29]. Machine Learning, in the right setting, have introduced a new level of efficiency for humans, business professionals and individuals to produce their best work and for companies to thrive, inspiring organisations to achieve their full potential. ML-assisted Smart Homes can provide many useful features, like tracking when the owner last walked the dog or notifying him when kids come home from school. Some up-to-date systems are even able to call for emergency services autonomously. It enables convenient automations in homes and combined with appliances, it

could make household management seamless. Another advantage of Smart Homes can be the reduction of household waste and automated recycling, putting it in better balance with the ecosystem. Improved sustainability, stress reduction and time-saving are some of the benefits of releasing people from housekeeping chores. Although Machine Learning is not part of the original concept of Smart Homes and the Internet of Things, it can get the most value out of these deployments through analysing IoT data, extracting hidden information, and automating the prediction of control decisions. Moreover, next-generation and 5G networks can also be used to achieve the high communication requirements of IoT and ML (or Deep Learning) and connect a large number of IoT devices, even when they are on the move. Thereby, they can further improve and enhance these capabilities, especially when dealing with Big Data.

After the literature study in the previous section and taking into consideration the available equipment in the office, we shaped an end-to-end application that is going to be described here. Since in this thesis we wanted to examine the potential of combining the aforementioned technologies, the proposed idea is to utilise Machine Learning in order to evaluate the Indoor Air Quality in real-time by using Internet of Things wireless sensor equipment. Nowadays, the quality of the air we breathe even inside our homes or working environments is getting more polluted day by day. That is because of the high emission of air pollutants created by excessive industrialisation that has disturbed the balance of the natural environment. This phenomenon results in a reduction of air quality, causing a deterioration in the health of citizens. The official scientific data that prove this deterioration are so alarming that everyone should be aware and, thus, protect themselves and those around them.

Continuous monitoring of air quality, measuring and observing its components, as well as the variations they show, is necessary in order to improve the quality of daily life. A very effective, and low-priced solution, is proven to be the use of IoT systems. Low cost and high precision systems that are able to provide reliable continuous air quality monitoring data. These data can be used both for real-time air quality monitoring and for the creation of historical data warehouses, which can be used at a research level to draw conclusions and conduct statistical analysis respectively. Implementing Machine Learning models is also a common practice to predict future trends in air pollution and possibly prevent disasters or outbreaks. IoT wireless sensor networks, installed in multiple locations, can combine their measured sensory data to power data-processing and decision-making ML models. The goal of IoT for monitoring air quality is the efficient storage, transmission and processing of data generated by their sensors, combining low energy consumption and low maintenance costs. Additionally, many web services and mobile applications are being used globally to inform citizens daily about the air quality, which are often combined with indoor air purifiers, ionizers, and ventilators (or in general HVAC systems, as in Heating, Ventilation, and Air Conditioning), improving the quality of everyday life. These applications are based on IoT systems in scenarios such as Smart Homes or Smart Buildings, often assisted by Machine Learning models for complete automation.

Utilising all this information as the driving force and motivation for the elaboration of this thesis, we have come to a final form of the proposed application. The main objective is to deploy a Machine Learning model into production, which is responsible for the real-time evaluation of the air quality of the HPN (High Performance Networks) office space in the University of Bristol. Initially, this will be done by working on and installing IoT sensors throughout the space and storying the measurement sensory data into a database. When enough information is acquired, data manipulation techniques will be applied so that we create a dataset shaped in the right form in order to be fed into a supervised

Machine Learning model. After this process, the training phase will be performed, and parameter tuning and model optimisation will take place until an acceptable performance threshold is met. At this point, it should be mentioned that an additional guideline that has been set to be followed is that we wanted to take advantage of only state-of-the-art frameworks and tools because an underlying goal is to implement this thesis in such a way that it could potentially be applied in a real-world Smart Home scenario. That is also the reason why we present an extensive study on how Machine Learning is applied in terms of algorithms, libraries, and frameworks, and why the Machine Learning part of this project is the main concern and focus since it is not a standardised process. Finally, the trained and optimised model will be deployed on a Docker container in a real-life server, from where it will be accessible from anywhere for real-time inference on the evaluation of the current air quality in the office. Afterwards, a script will be implemented as a service that fetches real-time predictions from the model and, using this information, a real-time IoT dashboard will be populated with the air quality evaluation. This idea is implemented as an end-to-end Smart Home case study to demonstrate the great potential of utilising Machine Learning and IoT for home (or building) automation.

To sum everything up, in an era that everyone is trying to automate procedures and simplify our everyday lives, Smart Cities and Smart Homes are finally becoming a reality, while the quality of the air we breathe is rapidly declining. This thesis is an attempt to take this vision one step further and prove that major technologies, as in this case Internet of Things and Machine Learning, can accomplish great things together and effectively improve our quality of life. This project utilises Internet of Things equipment and Machine Learning techniques to predict and estimate the real-time Indoor Environmental Quality (IEQ) of the specified office space. Based on this air quality evaluation, it could potentially provide maintenance services to uphold a healthy and safe environment of the surrounding space.

1.4 Thesis Structure

In the 1st chapter of the thesis, there is a summary of the proposed problem and its primary objectives. A literature review on Machine Learning and Internet of Things is being presented based on different approaches regarding these leveraging technologies in order to shape our idea and the guidelines that will be followed. The motivation that formed the theme and subject of IEQ is also described in detail and an outline of the experiment implementation is being introduced.

The 2nd chapter is dedicated to the theoretical study and background of the Machine Learning algorithms and tools that were used more in the related scientific papers in the previous chapter. This study is performed in order to understand the way these algorithms operate and the capabilities of the available libraries and frameworks. The objective is to determine which of them suits better the requirements and scenario of the current dissertation, and thus which of them are going to be utilised.

In the 3rd chapter is being presented the whole experiment process and methodology. This procedure starts from one end, data collection and analysis, to the other, the deployment of the trained Machine Learning model. It covers all the technical details of the use-case's design providing helpful diagrams and graphs to make them more understandable and familiar. The reason why the data was shaped in that form, the way the dataset was constructed, what adjustments had been made during the training phase

and how the model's deployment took place, are only some of the topics that are going to be touched on in this chapter.

Through the 4th and last chapter, the goal is to list the conclusions of the implementation of this thesis and discuss the performance and results of the conducted experiment. Among others, things that could have been done differently, alternative routes that could have been taken, any issues that were faced, improvements, and cases that the proposed application might apply and be useful in, are being discussed there. That segment gives closure to the current dissertation in the hope that this work meets the expectations of this assignment.

1.5 Conclusion

This chapter constituted the introduction of the subject that is going to be presented and implemented in the thesis. A detailed study on related works has been carried out and helped guide and form the objective and theme of the current project. The reason why and the motivation that justified our vision and proposed application has also been showcased. The main objective and idea have been described based on the main guidelines that had been set to be followed. Finally, the structure of the chapters, that follow the Machine Learning and Internet of Things aspects of the problem, have been presented.

Chapter 2 Machine Learning over a real-world network

2.1 Introduction

The main focus in this chapter is Machine Learning, and the theoretical and technical background information required to comprehend before utilising and applying it over a real-world network. After presenting similar works and describing the main objective of the thesis, here we are going to show in great detail the available tools, regarding Machine Learning algorithms, libraries and frameworks, in order to implement the idea. The goal is to describe and study the way someone can apply Machine Learning to real-world applications and not just on experimental scenarios and environments. To this end, a detailed study on the available algorithms and frameworks is being carried out with the intention of assisting in the search for the most suitable tools in this particular use case. Besides, the main concern about the project's implementation is the Machine Learning aspect, and not the Internet of Things details, because it is not such a standardized process and, performance-wise, it is the most critical part that has the greatest impact on the results and the outcome.

2.2 Machine Learning Algorithms study

As it can be easily inferred from the studied related works on the field, some Machine Learning algorithms have been quite popular regarding the pick rate. To this end, the most frequently picked models are studied extensively in this section in order to present the advantages and disadvantages of each one. The purpose of this research is to bring to light their attributes, features and capabilities as an ulterior motive to assist in deciding which one is the most suitable for this specific application.

2.2.1 Support Vector Machines

The Support Vector Machines (SVM) is a supervised learning algorithm that given a set of labelled data, it produces input-output mapping functions [30]. These mapping functions can either be a regression function or a classification function. For classification, nonlinear kernel functions are usually used to transform input data into a high-dimensional space in which the input data become more distinctively separable compared to the original input space. In other words, given labelled training data, the algorithm outputs an optimal hyperplane that categorises new samples successfully. On nonlinear mapping, different types of kernels can be used depending on the specific problem. Its objective is to maximise the distance between the two different classes. That way, maximum-margin hyperplanes are then created between them. Hence, also known as the maximum margin classifier. Therefore, the model produced relies solely on a subset of the training data near the class boundaries. That data subset, the so-called support vectors, are crucial to the correct operation of the algorithm.

Similarly, the model produced by Support Vector Regression (SVR) depends only on a subset of the training data and ignores any training data that is sufficiently close (below a defined threshold) to the model prediction. The goal, in this case, is to minimise the cost produced when the predictions deviate

a lot from the original values. Additionally, SVM and SVR are, of course, able to solve both linear and nonlinear separable problems with quite the same performance results. SVMs are trained by solving a constrained quadratic optimisation problem, well known in mathematics [31]. Among others, this implies that there is a unique optimal solution for each choice of the SVM parameters. This fact is unlike other machine learning algorithms, such as standard Neural Networks trained using backpropagation.

Some of the main issues regarding the design and use of SVMs are the choice of the kernel function, the regularisation and gamma parameters. Kernel functions, as referenced above, are responsible for training and defining the maximum-margin dividing hyperplane by transforming the input data. By applying the kernel trick, apart from the linear problems, nonlinear ones are then solved successfully. Many kernel functions exist, such as Polynomial, Gaussian Radial Basis Function (RBF) and Hyperbolic tangent, but some of them are more suitable in each specific scenario. Some characteristic diagram examples are shown in Figure 2.1. Moreover, the regularisation parameter (often termed as C parameter) tells the SVM optimisation how much you want to avoid misclassifying each training sample. For large values of C, high misclassification avoidance, the optimisation chooses a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a small value of C, high misclassification tolerance, causes the optimiser to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. Lastly, the gamma parameter defines the reach of the influence of a single training sample. With low gamma values, points far away from the conceivable separation line are also considered in the calculation for the separation line. Whereas high gamma means only the points close to the conceivable line are considered in the calculation. The best combination of C and gamma is often selected by a grid search with exponentially growing sequences of C and gamma values.

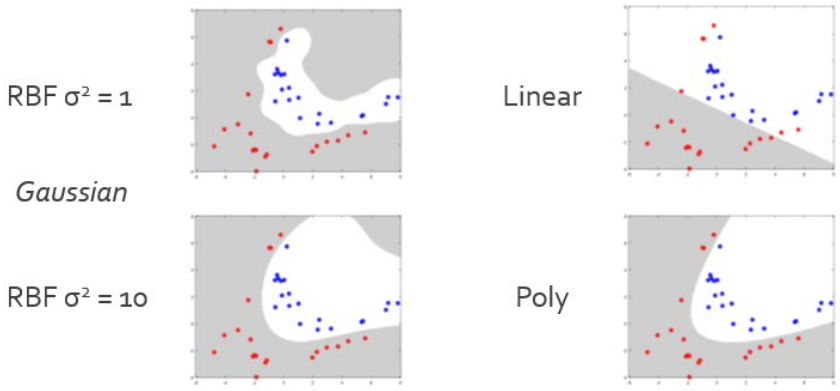


Figure 2.1: SVM kernel function examples [32]

Furthermore, unlike Perceptron, SVM comes up with the optimal solution (optimal margin) instead of finding a random hyperplane that simply solves the specific problem. SVMs have shown highly competitive performance in numerous real-world applications (linear and nonlinear) [33], as depicted above, such as medical diagnoses, face recognition and text and image processing. This fact has established SVMs as one of the state-of-the-art tools for machine learning and data mining. Some studies also show that training many local SVMs instead of a single global one can lead to extensive improvements in the performance of a model [31]. Some potential issues of the SVM are the difficulty of tuning the parameters to achieve the best results, and the parameters of a solved model are usually difficult to interpret as well. Last but not least, SVM is only directly applicable to two-class problems.

Therefore, algorithms that reduce and split the multi-class task into several binary classifications have to be applied beforehand.

2.2.2 Neural Networks

Artificial Neural Networks (ANN), or just Neural Networks (NN), mainly form the base of Deep Learning (DL), which is a subfield of Machine Learning. The structure of the human brain inspires this algorithm. Neural Networks, usually used in supervised learning, take input information, train themselves to recognise patterns found in the samples, and then autonomously predict the output for a new set of unknown data. Hence, a Neural Network can be thought of as a system that mimics the behaviour of the human brain to answer complex data-driven problems. A massively parallel distributed system made up of simple processing units that have a natural tendency for storing experiential knowledge and providing it later for further use. It resembles the brain because the network acquires knowledge from its environment through a learning process. While inter-node connection strengths, known as synaptic weights, are used to store the acquired knowledge.

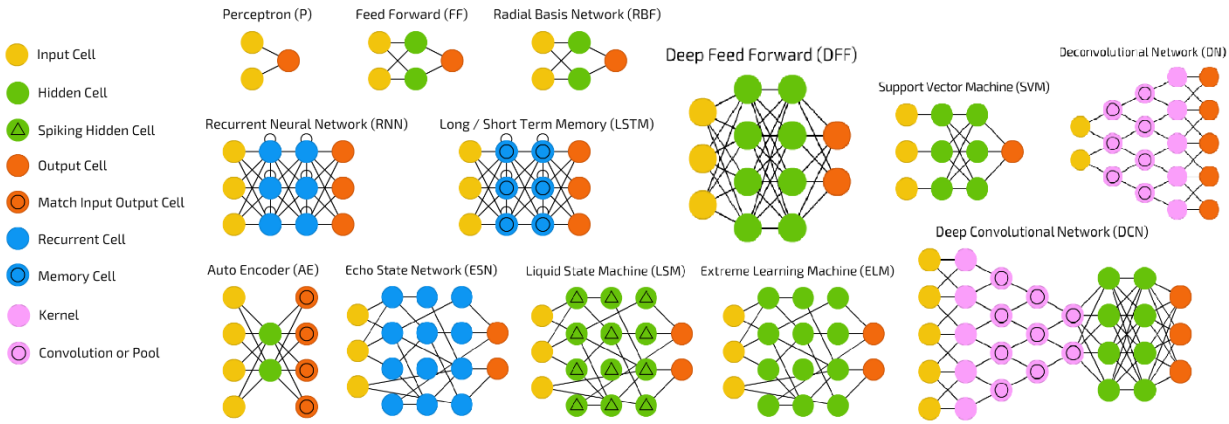


Figure 2.2: Neural Networks architecture types [34]

More specifically, these simple processing units are called neurons. The network can contain any number of neurons in any number of layers. In a multi-layer network, a neuron is a mathematical function which serves the purpose of collecting and classifying information in accord with a specific architecture. In this regard, the network bears a strong resemblance to statistical methods such as curve fitting and regression analysis [35]. Furthermore, an artificial neuron is characterised by its architecture, its mathematical function (Activation Function) and its training parameters (synaptic weights). The architecture refers to the structure and the type of connection between the neurons. These topologies and the variety of architectures highlight the flexibility and adaptability of the algorithm to the different purposes it may serve. Some architecture types are shown in Figure 2.2, and each of them blends better together with particular scenarios. Given the architecture, neurons process the provided signals, and then they transmit them to other topologically connected neurons. These connection signals are real numbers, and the output of each neuron is computed by a nonlinear function on its inputs' sum. This nonlinear function, or just this weighted sum, is called the Activation Function (AF), and it might as well be a well-known function such as unit step, linear, sigmoid or hyperbolic tangent. Different layers of neurons

may perform different transformations on their inputs due to the Activation Function that was used. Neurons typically have a weight that adjusts as learning moves forward. The synaptic weight increases or decreases the strength of the signal at its connection. Signals gradually travel from the first layer (input layer) to the last layer (output layer), possibly after traversing the layers multiple times. The input layer collects the input patterns. The output layer has output signals to which input patterns may map. During the learning phase, the network learns and improves itself by adjusting these weights in order to be able to predict the correct, or closest, result for the input data. In this phase, the hidden layers fine-tune the input weights until the margin of error is minimal. They infer dominant features in the input data that have predictive power regarding the outputs.

Neural networks can adapt to input changes, so the network generates the best possible result without needing to redesign the output parameters. This adaption is achieved by the optimisation method. The most common and efficient one is Backpropagation (BP) [36]. This method lets the neuron weights modify themselves, should the solution they find is not the expected one. This way, they compensate for each error found during learning. Technically, BP calculates the gradient (the derivative) of the cost (loss) function (usually Mean Squared Error – MSE and Mean Absolute Error – MAE) associated with a given state concerning the weights. The objective is to minimise the loss, or simply eliminate the error. The weight updates can be done utilising algorithms as Stochastic Gradient Descent (SGD) or other methods, such as Extreme Learning Machines (ELM). To sum up with all the tunable parameters described, the universal name used for them is Hyperparameters. A constant parameter whose value is defined before the learning process begins. Their values are derived through learning. Examples of hyperparameters include the number of hidden layers, the number of nodes in each layer, the choice of AF, the number of iterations of backpropagation applied and the number of the input features [37].

NNs have found applications in many disciplines because of their ability to reproduce and model nonlinear processes [38]. Nowadays, due to the improved computing power and volume of data available, NNs are becoming more popular. They are extremely precise and powerful, especially with the volume of data we have available now. The neural networks can distinguish subtle nonlinear interdependencies and patterns that other methods cannot. Despite the specific algorithm, the well-prepared input data are the ones that determine their level of success. They are applicable in classification, regression, clustering, regression analysis, data mining and data processing problem categories. Furthermore, they are capable of solving multi-class or multi-output problems. Application areas include pattern recognition, image classification, speech and handwriting recognition, motion detection, target tracking, product maintenance, time series analysis, stock market forecasting, weather prediction, fraud detection and risk assessment.

Regarding their advantages, they are high-performing, and if the model, loss function and learning algorithm are selected appropriately, the result is robust [39]. Additionally, they are able to handle large amounts of data, but they also require many data and might take more time to train, more than a typical algorithm does. They solve problems humans may not be able to conceptualise, but on the other side, it's hard to understand why NNs make the conclusions that they do since it's hard to trace back the numeric values these models produce. Numerous trade-offs exist between learning algorithms. Almost any algorithm works well with the right hyperparameters for training on a specific data set. However, tuning and selecting an algorithm for training on unseen data requires substantial experience and experimentation. In some cases, they might prove themselves expensive due to the amount of computational power required. While in some other cases, they face the possibility of overfitting. This

problem arises in over-specified systems when the network capacity significantly exceeds the needed parameters. Two approaches address overfitting. The first is to apply cross-validation and similar techniques to check for the presence of overfitting and to select hyperparameters to minimise the generalisation error. The second one is to use some form of regularisation on the data. Generalisation on unseen data is a vital aspect of Machine Learning algorithms in general.

2.2.3 Naive Bayes

Naive Bayes is a classification method based on Bayes' Theorem with an assumption of independence among individual features. Simply put, a Naive Bayes (NB) classifier assumes that the existence of a particular feature in a class is unrelated to the presence of any other feature [40]. It is a theorem that works with conditional probabilities. Conditional probability is the probability of a hypothesis *h* that will happen and is affected by a new evidence *e* that has been brought to light. It can give the probability of an event using its prior knowledge.

$$P(h|e) = \frac{P(e|h)\,P(h)}{P(e)} \tag{2.1}$$

Bayes' Theorem is shown in Equation (2.1). More specifically [41], the *P(h|e)* is the probability (the so-called likelihood) of hypothesis *h* occurring, given that the evidence *e* happened. This information is called the posterior probability. *P(e|h)* is the probability of data *e* given that hypothesis *h* was true. Additionally, *P(h)* is the probability of hypothesis *h* being true (regardless of the data), called the prior probability of *h*. Lastly, *P(e)* is the probability of the evidence or data (regardless of the hypothesis). First, the model creates a frequency table (similar to prior probability) of all the classes and then creates a likelihood table, the posterior probability. After calculating the posterior probability for several different hypotheses, it selects the hypothesis with the highest probability. This class, or output value in general, is the maximum probable hypothesis and may formally be called the Maximum A Posteriori (MAP) hypothesis. In Figure 2.3, a classification diagram example visualises the way the algorithm chooses the most dominant category.

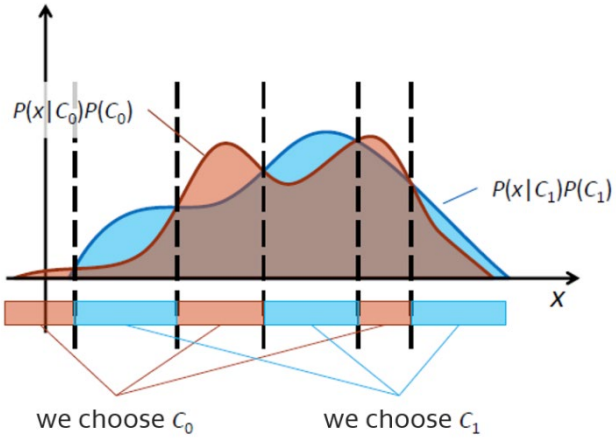


Figure 2.3: Naive Bayes classification diagram example [42]

Naive Bayes got this name because the calculation of the probabilities for each hypothesis is simplified to make their calculation traceable [43]. Instead of attempting to calculate the values of each attribute, they are assumed to be conditionally independent and calculated as $P(e1|h) \times P(e2|h)$ and so on. This is a strong assumption that is most unlikely in real-life cases, i.e. that the attributes do not interact, but it is a common cause that might hinder the performance of the classifier. Regardless, the approach functions remarkably well on data where this assumption is false. Naive Bayesian models are fast and easy to implement and particularly useful for small and medium-sized data sets, even for large amounts of data. Together with simplicity, Naive Bayes is known to outpace even highly sophisticated classification methods.

Naive Bayes is a machine learning algorithm for two-class and, even, multi-class classification problems. Variants of Naive Bayes are frequently used to enhance baseline performance depending on the features used, dataset and model variant [44]. The most conventional version of Naive Bayes is the Gaussian NB, which works better for continuous types of data. The fundamental assumption of Gaussian NB is that the attributes follow a normal (Gaussian) distribution. Another version is the Multinomial NB, used for discrete counts and data that is multinomial distributed. It is widely used in text and document classification in Natural Language Processing (NLP) problems. Lastly, another commonly-used variant of Naive Bayes is Bernoulli NB, which is useful when the features are binary-valued as in 'bag of words' text classification models.

Naive Bayes algorithm is used in various critical and noncritical domains, as seen in the related works, such as diagnosis of diseases, sentiment analysis, real-time and multi-class prediction, spam filtering, text classification and recommendation systems. Concerning the algorithm's advantages, it is a simple, highly extensible algorithm and very fast in response times even compared to other models like logistic regression. In the case of categorical input variables, it performs better, compared to numerical variables. For numerical variables, Normal distribution is taken for granted. Some shortcomings of the algorithm [45] are when a categorical variable, in the testing set, was not observed in the training set then the model is unable to make a prediction. This issue is known as 'Zero Frequency' and smoothing techniques, such as the Laplace estimation, have to be used. A significant disadvantage is also that the requirement of independent variables. In the real world, it is nearly impossible to have a set of entirely independent features. Last but not least, Naive Bayes is not directly applicable to regression problems. Data transformations have to take place beforehand, with methods such as linear or logistic regression, in order to be able to use it afterwards.

2.2.4 k-NN

The k-Nearest Neighbours (k-NN) algorithm is a straightforward, easily implemented supervised Machine Learning algorithm, which can be utilised in both classification and regression scenarios. It is an instance-based learning algorithm, which does not require any parameters [46]. It surmises that similar things exist in close proximity. In other words, similar things tend to be near each other. k-NN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics calculating the distance between points on a graph. In both problem types, the input consists of the k closest training samples in the feature environment. The result relies on whether the

problem is classification or regression. In classification, the output is a class label. An object is classified by a plurality vote of its neighbours, with the object being set to the class most common among its k nearest neighbours. The value of k is a positive integer and is typically small. When k is equal to 1, then the object is simply allocated to the class of that single nearest neighbour. In regression problems, the property value for the object is considered as the output. This value is the mean of the values of k Nearest Neighbours. The neighbours are taken from a set of data for which the object property value (for k-NN regression) or the class (for k-NN classification) is known. This data can be regarded as the algorithm's training set, yet without requiring an explicit training step.

There are many ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, a commonly used and familiar distance metric for continuous variables is (the straight-line) Euclidean distance [47]. For discrete variables, like for text classification, a different metric can be applied, such as the overlap metric (or Hamming distance). Typically, the classification accuracy of k-NN can be greatly enhanced if the distance metric is learned with specialised algorithms such as Neighbourhood components analysis or Large Margin Nearest Neighbor. Since this algorithm relies on distance, normalising the training data can improve its accuracy dramatically. The training samples are vectors in a multidimensional feature environment, each with an output property value or a class label. The training phase of the algorithm is only composed of the storing of the feature vectors and outputs of the training samples.

The k is a user-defined constant, and an unknown vector (a query or test point) is classified by assigning the label which is most dominant among the k training samples nearest to that query point. The optimal choice of k depends upon the data. Generally, higher values of k diminish the effect of the noise on the classification, but make less distinct the boundaries between classes. Various heuristic techniques can select a good k. One common way of choosing the empirically optimal k in this setting is by the bootstrap method [48]. Another way is to run the k-NN algorithm numerous times with different values of k and choose the k that reduces the number of errors we encounter while maintaining its ability to effectively make predictions when it's given unseen data. As the value of k decreases to 1, the predictions become less stable, while, as it increases, the predictions become more stable due to averaging, and therefore more inclined to make more precise predictions (up to a certain extent). Eventually, an increasing number of errors is witnessed, and at this point, the value of k increased more than it should. In binary problems, it is better to choose k to be an odd number as this prevents tied votes.

A specificity of the k-NN algorithm is that it is influenced heavily by the local structure of the data. A drawback of the essential "majority voting" classification happens when the class distribution is skewed. That is, samples of a more frequent class tend to dominate the prediction of the new example because they tend to be mutual among the k Nearest Neighbours due to their large number [49]. One way to surpass this problem is to weigh the classification, with regards to the distance from the test point to each of its k nearest neighbours. The class (or value) of each of the k nearest points is multiplied by a weight (1/d) proportional to the inverse of the distance d from that point to the other. A different method of overcoming skew is to abstract the data representation. For instance, in a Self-Organising Map (SOM), each node is a representative (centre) of a cluster of similar samples, regardless of their density in the original training input. k-NN can then be deployed to the SOM. Furthermore, the accuracy of the k-NN algorithm can be harshly degraded by the presence of irrelevant or noisy features, or if the feature scales are not consistent with their importance. An incredibly popular approach is the use of evolutionary

algorithms to optimize feature scaling. Another typical one is to scale features by the shared information of the training data with the training classes.

In addition to all that, for high-dimensional data (more than 10), dimension reduction is typically performed before applying the k-NN algorithm in order to avoid the effects of the curse of dimensionality [50]. The curse of dimensionality means that Euclidean distance is pointless in high dimensions because all vectors are almost equidistant to the search query vector (multiple points lying more or less on a circle with the query point at the centre; the distance from the query to all data points in the search space is almost the same). Feature extraction and dimensionality reduction can be applied at once as a pre-processing step by Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), or Canonical Correlation Analysis (CCA) techniques, and then perform k-NN clustering on feature vectors in reduced-dimension space. This process can also be found as low-dimensional embedding in machine learning. Data reduction is one of the most critical problems when working with massive data sets. Generally, only a handful of data points are needed for accurate classification.

Coming into conclusion, one of the most attractive features of the K-NN algorithm is that it is quite simple to understand and easy to implement. Additionally, there is no need to build a model, train, tune several hyperparameters, or make any additional assumptions to be productive. With minimal training time, it can be an effective tool for instantaneous analysis of some datasets. The algorithm is flexible and can be used for search, regression and classification. Some of its popular applications are anomaly detection, recommender systems (an application of k-NN search), text mining, economic and climate forecasting, data compression and genetics [51]. Although relatively straightforward along with other classic data mining methods, it also works quite well in comparison to more complex and more recent approaches, according to a large scale experimental analysis. Moreover, k-NN functions effortlessly with multi-class datasets, while on the contrary, other algorithms are hardcoded for binary use cases only. The algorithm has some strong consistency results. A confusion matrix is often used as a tool to validate the accuracy of k-NN classification. More powerful statistical methods can also be applied, such as the likelihood-ratio test.

Some of the algorithm's disadvantages are that it gets significantly slower as the volume of the data samples increases. This fact makes it an impractical choice in environments where predictions need to be rapid. In this regard, various improvements to the k-NN speed are possible by using proximity graphs [52]. In addition, one of its apparent drawbacks is that the computational cost is demanding because of calculating the distance between the data points for all the training samples. This phenomenon might make it impractical in industry settings. Last but not least, quite evident as described above is also that it always has to determine the value of k, which may be too complicated in many cases and impossible in others.

2.2.5 K-Means

One of the most popular topics under the realm of unsupervised learning in Machine Learning is k-Means clustering. The majority of unsupervised learning-based applications employ the subfield called clustering. Clustering is the method of grouping data samples together into clusters based on a particular feature that they share without being trained by the response variable, which is the purpose of

unsupervised learning in the first place. The algorithm aims to divide n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centres or cluster centroids), serving as a representative of the cluster and while keeping the centroids as small as possible [53]. In simple words, the objective of k-Means is to group similar data points and discover underlying patterns. To achieve this objective, K-means searches for a fixed number (k) of clusters in the dataset. A cluster denotes a collection of data observations accumulated together because of certain similarities. The number k refers to the number of centroids that the algorithm yields and applies in the dataset. The term centroid stands for the imaginary or real location representing the centre of the cluster. In the end, this whole process of grouping belongs to the training phase of the algorithm. The result of training would be a model that takes an unseen data sample as input and yields the cluster that the new data point belongs to, according to the training that the model went through.

The training process of the input data starts with a group of randomly selected centroids, which are used as the initial points for every cluster, and then performs iterative (repetitive) calculations to adjust and optimise the positions of the centroids. A simple demonstration of the process is presented in Figure 2.4. It halts creating and optimising clusters when the centroids have stabilised, and there is not any change in their values since the clustering has been successful. As it might be easily suspected, the value of the hyperparameter k is of great importance [54]. In most cases, it is quite tough to forecast its value, and such a task requires much experimentation. Either way, it is a common practice to execute the algorithm on the same dataset repeatedly until a most common clustering formation, and solution, are established. Moreover, the algorithm minimises within-cluster variances using squared Euclidean distances, and not regular Euclidean distances. The latter would be a more complicated solution since the mean optimises squared errors, whereas only the geometric median minimises Euclidean distances. For such a case, better Euclidean solutions can be found using k-Medians and k-Medoids.

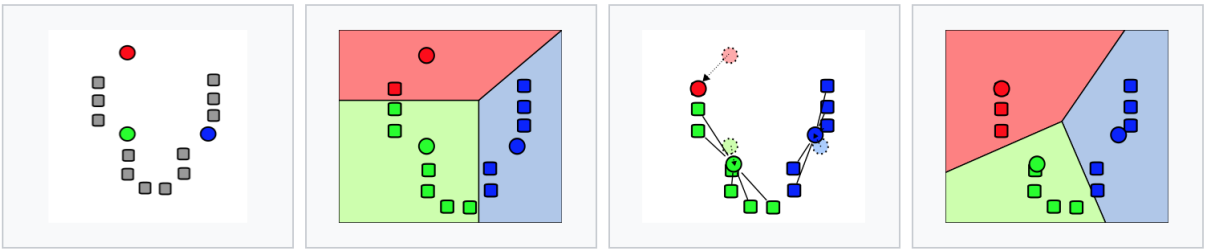


Figure 2.4: k-Means algorithm process demonstration [55]

The algorithm has a loose connection with the k-NN classifier that is often confused with k-means due to the name. A 1-Nearest Neighbour classifier applied to the cluster centres, that have been obtained by k-means, can classify new data into the existing clusters. This new method is known as the Nearest Centroid classifier or Rocchio algorithm [56]. Nevertheless, k-Means tends to obtain data points under the same class as identical as possible, and the data points in a separate class as dissimilar as possible. Bearing this perspective in mind, k-Means clustering is the most frequently practised, simple and straightforward clustering method to classify a dataset into a number of k classes.

Supervised learning is where existing data is already labelled, and the behaviour that we want to recognise from new datasets is already known. Conversely, unsupervised learning doesn't exhibit labelled datasets, and algorithms are there to explore and discover patterns and relationships in the data.

It is a known fact that the data and information are usually obscured by noise and redundancy, and they are complicated and mismanaged, so putting them into groups with similar features is the decisive action to bring some insights. The real-world conditions barely reveal where these types of algorithms can be applied to [57]. That is where k-Means comes into place and proves itself very suitable for exploratory data analysis and data mining. In the task of understanding data entirely and getting inferences from all data types (despite the data's form being images, text content or numeric), k-Means works flexibly. It is widely applied in content promotions, recommendation systems, Call Detail Record (CDR) analysis or even fraud detection scenarios. Furthermore, k-Means often finds application in pre-processing, and analysis tasks, as the data are effectively transformed and prepared to be used in a classification or regression ML problem, such as with a Neural Network or SVM model. It can also serve as the model of lossy images compression technique as k-Means clusters pixels of an image trying to decrease its total size.

One of its limitations is that the output is highly influenced by the original data, such as the number of clusters and the initial values of the centroids [57]. In some cases, the algorithm could get utterly wrong if the starting positions of the centroids were wrong. A right initialisation approach would be to choose the initial positions from the dataset itself, setting one of the data points as the mean value of its surroundings. These issues highlight the importance of human involvement in some applications since there is not a ground truth error estimation (unsupervised learning). Usually, another measure for the hyperparameters is required. While in some applications, a person looking at the plot and determining the k value is adequate. Additionally, on some occasions, clusters show complex spatial graphs and views; therefore executing clustering is not a decent choice. Rescaling should also be very cautious as to whether it should be applied since normalisation or standardisation of data points can change the output entirely. Finally, one of the obvious strong points of k-Means is the fact that it gives an average value of the attribute over the cluster, not just the cluster itself. This information could be beneficial in segmentation-related image processing tasks.

2.3 Implementation tools for Machine Learning

In a similar manner to the previous section, we will study extensively the most popular tools, libraries, and frameworks available in Machine Learning and Deep Learning scenarios. The features, advantages, and disadvantages of each one will be presented in order to highlight and determine which of them is the most suitable in this particular use case. It is a necessary step that needs to be taken as every specific application is unique and requires tools that effectively deliver its tasks. Additionally, an objective that was set for this project involves a tool that is considered state-of-the-art, in a way that it could easily be seen being used as part of an application on a production scale.

2.3.1 Scikit-learn

Scikit-learn (also referred to as Sklearn) was initially developed by David Cournapeau as a Google Summer of Code (GSoC) project in 2007. Not long after, Matthieu Brucher got involved in the project, incorporating it into his thesis. INRIA (Institut national de recherche en Informatique et en Automatique – in French) got involved later and the first public release was made on February the 1st 2010 [58]. The

library is envisioned to have a level of solidity and support, required for use in production systems. This means there is a heavy focus on concerns like code quality, ease of use, performance, documentation and collaboration. It is a free open-source software library for Machine Learning and is quite popular on GitHub and commercial uses. The project has paid sponsorship from Tinyclues, Google, INRIA and the Python Software Foundation, and its active contributors are now counted more than thirty. Various organisations like Evernote, Spotify, JP Morgan, Aweber, Booking.com and several others are using Sklearn.

There are several reasons why it is now widely considered to be the most popular Python library for Machine Learning projects, especially in production systems. Scikit-learn harnesses Python's rich environment to offer state-of-the-art implementations of various popular ML algorithms, whilst preserving a simple and straightforward interface. This answers the growing need for statistical data analysis by non-specialists in the software and web industries, as well as in fields outside computer science, such as economics, business management, biology, or physics [59]. Rather than providing as many features as possible, the project's focus is to provide solid implementations and ensure the code's quality by unit tests and static analysis tools. Furthermore, its bare-bone design and API, as well as the access to huge amounts of documentation and tutorials, preserve precision with regards to the algorithms employed. More than three hundred (300) pages of user guides include narrative documentation, class references, tutorials, installation instructions and real-world applications. All this variety of resources is more than enough to acquire the necessary knowledge. It is highly supported by its community and the development of the library is strict, which means that it is an extremely solid tool. There is a clear, consistent code style which ensures that the code is easy to understand and reproducible, and lowers the barrier to entry for coding Machine Learning models. Its functionality can be enriched to suit a series of use cases since it is extensively supported by third-party tools [60]. In addition, it is licensed under a permissive simplified BSD license and is extensively distributed under many free software Linux distributions, encouraging academic and commercial use.

The library offers an extensive variety of Machine Learning algorithms, both unsupervised and supervised, utilising a reliable, task-oriented interface, thus providing easy methods comparison for a specified application. It can easily be integrated into applications outside the traditional range of statistical data analysis since it relies on the scientific Python ecosystem. By using a high-level language, the algorithms can be utilised as building blocks for use-case-specific approaches. Additionally, a selection of efficient models and tools for Machine Learning and statistical modelling are available for utilisation. Scikit-learn focuses on modelling the data instead of loading, manipulating, and summarising them. Those models include almost all the popular supervised learning algorithms for classification or regression problems, and all the popular unsupervised learning algorithms, from clustering, factor analysis, Principal Component Analysis (PCA) to unsupervised neural networks [61]. A very thorough and helpful flowchart about algorithm selection based on the problem requirements is demonstrated in Figure 2.5. Moreover, other important features are cross-validation, parameter tuning and model optimisation. Dimensionality reduction, feature extraction and selection, and in general its built-in pre-processing and data mining tools, are some of its main advantages. Last but not least, the library offers pipelining options, for chaining together all the steps in an ML workflow, and ensemble methods, for combining the predictions of multiple supervised models. Sklearn's simplicity derives from the fact that it is fairly easy to pick up, and by learning how to use it, a good grasp of the key steps in a typical ML

workflow is gained. What makes it so straightforward to use is that, regardless of the model or algorithm being applied, the code structure for the model training and testing is the same.

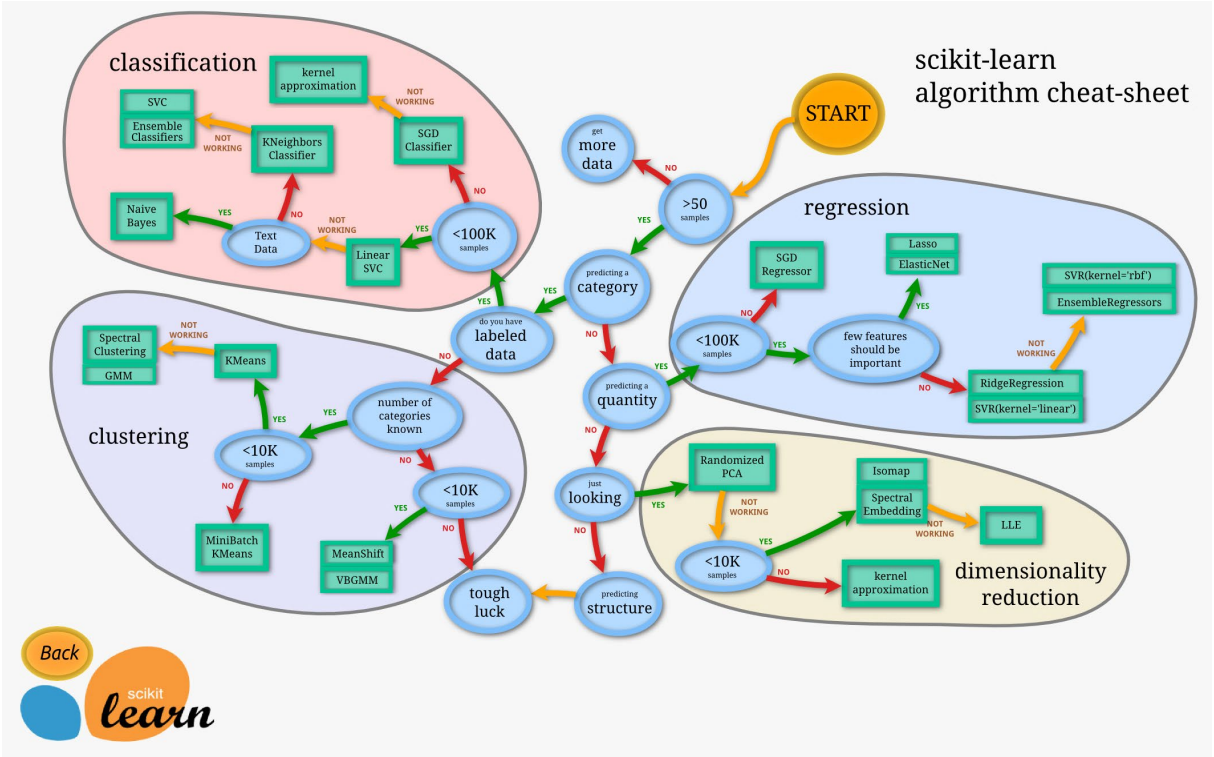


Figure 2.5: Scikit-Learn algorithm selection flowchart [62]

Regarding the underlying technologies used, while Scikit-learn focuses on ease of use, and is mostly written in a high-level language, maximising the computational efficiency has been taken care of as well [63]. The library is built upon some well-structured tools such as NumPy (Numerical Python extensions), which is the base data structure used for the data and model parameters and provides basic arithmetic operations. Input data are presented as NumPy n-dimensional arrays, achieving seamless integration with other scientific Python libraries. Another technology that couldn't be missing is SciPy (Scientific Python), which is a fundamental library for scientific computing and offers algorithms for linear algebra, sparse matrix representation and basic or advanced statistical functions. NumPy and SciPy are usually considered to be the same technology stack and this stack also includes tools like Matplotlib (comprehensive 2D/3D plotting), pandas (data structures and analysis) and SymPy (symbolic mathematics). Despite being a high-level library, Scikit-Learn manages to remain computationally efficient by incorporating C libraries such as LAPACK, LibSVM, LibLinear and Cython. Cython is a language used for merging C in Python. It facilitates achieving the performance of compiled languages with high-level operations and Python-like syntax. It is also used to combine compiled libraries and manages to omit the boilerplate code of C/Python extensions. Overall, Scikit-learn is a powerful tool that effectively delivers demanding tasks by combining performance, documentation, flexibility of use and quality of code, even in industry-level scenarios.

2.3.2 Theano

Theano is an open-source library released under the BSD license and was primarily developed by the MILA (Montreal Institute for Learning Algorithms) group at the University of Montreal in Canada in 2007. In September 2017, MILA stated that major development would cease after the 1.0.0 release due to competing offerings by strong industrial players [64]. When, in May 2018, the PyMC development team officially took up control of Theano maintenance once MILA stepped down. Many experts in the area of Machine Learning and Deep Learning still depend on Theano as there is a relatively active community with more than 50 contributors. It is a compiler for mathematical expressions in Python that combines the convenience of NumPy's syntax with the speed of optimised native machine language for efficient CPU and GPU computation. The user forms mathematical expressions in a high-level description that mimics the syntax of NumPy. This expression syntax is symbolic, which can be discouraging to beginners used to normal software development. Theano's expression types cover much of the same functionality as NumPy and include some of what can be found in SciPy.

Large Neural Network algorithms used in Deep Learning require specific types of computation and Theano is explicitly designed to handle these forms [65]. It was one of the first libraries of its kind and is considered an industry standard for Deep Learning research and development. Before performing the computation, the library automatically optimizes the choice of expressions, translates them into C++ (or CUDA for GPU use) and compiles them into dynamically loaded Python modules. It can power large-scale computationally intensive investigations quite efficiently. Common Machine Learning algorithms implemented with Theano are 1.6 to 7.5 times quicker than competitive alternatives when compiled for the CPU and from 6.5 to 44 times faster when compiled for the GPU [66]. Theano achieves good performance by minimising the use of temporary variables, minimising pressure on fast memory caches, making full use of BLAS subroutines, and generating fast C code that is specialised to sizes and constants in the expression graph. One of Theano's greatest strengths is its ability to generate custom-made CUDA kernels, which can not only significantly outperform CPU implementations but alternative GPU implementations as well. Furthermore, it supports parallel execution which is a significant asset.

The inaccessible interface and unhelpful error messages overshadow the library's impressive computing performance. It also requires a computational graph to be defined by the user and then be executed, which can be very complex and confusing. To this end, Theano is primarily used in combination with more user-friendly wrappers, such as Lasagne, Blocks, and Keras. These high-level frameworks strive for fast model testing and prototyping. They offer data structures and behaviours in Python explicitly designed to generate Deep Learning models rapidly and dependably, whilst making sure that fast and efficient models are created and executed by Theano as backend. Since Theano's API is low-level, these frameworks get above the need to be quite familiar with the algorithms that, in other frameworks, are hidden away behind the scenes. Theano is a go-to library if the user has substantial academic Machine Learning expertise, is looking for very fine-grained control of his models, or wants to implement a novel or unusual model [67]. In general, Theano trades ease-of-use for flexibility. In addition, the library does not provide any helpful operators or monitoring and debugging tools, which might make its standalone use even tougher. Nevertheless, as Theano is considered an industry standard for Machine and Deep Learning research and development, it was originally designed to implement state-of-the-art Machine Learning algorithms. However, considering that it probably won't be used directly, its numerous use

cases expand as it is used as the foundation for other libraries. These use cases can be digit and image recognition, object localisation, and even chatbots [68].

2.3.3 PyTorch

PyTorch is a free and open-source Machine Learning library, under the modified BSD license, developed and maintained by Facebook's AI Research lab (FAIR) since September 2016. Although its Python interface is more polished, popular, and the primary focus of development, it also has a C++ interface. PyTorch is based on Torch (Torch7), which is an open-source project for Deep Learning written in C and generally used via the Lua language interface [69]. Torch is a predecessor project to PyTorch and is no longer actively supported and developed. PyTorch recently replaced most of the low-level code reused from the Torch project. In addition, as Facebook started operating on both PyTorch and Caffe2 (Convolutional Architecture for Fast Feature Embedding), a very well-known Deep Learning library as well, the company realised that the models defined by the two were mutually incompatible. This initiated the Open Neural Network Exchange (ONNX) project to address this issue and, eventually, they ended up merging Caffe2 into PyTorch as its backend in March 2018.

PyTorch emphasises in flexibility and allows Deep Learning models to be expressed in idiomatic Python. This handiness and approachability found early adopters in the research community, and soon after its initial release, it has evolved into a highly esteemed Machine and Deep Learning tool across a vast series of applications. The library managed to become a popular framework among developers and researchers. Several pieces of software are built on top of PyTorch, including Tesla Autopilot, Uber's Pyro, HuggingFace's Transformers, PyTorch Lightning, and Catalyst [70]. The library provides an excellent introduction to Deep Learning. It has turned out to be abundantly qualified for use in professional contexts for real-world high-profile works.

PyTorch is a mathematical library at its core, which is able to execute, on graph-based models, efficient computation and automatic differentiation. Achieving this directly is challenging, although its modern API provides classes and idioms that allow the easy development of a huge collection of Machine Learning models. To this regard, PyTorch’s clear syntax, streamlined modern API and easy debugging make it an excellent choice for introducing Deep Learning. Furthermore, PyTorch provides a core data structure, the tensor, which is a multidimensional array that shares many similarities with NumPy arrays but offers the ability to execute on GPUs and provide strong acceleration on numerical computations. Around that foundation, PyTorch comes with features to perform accelerated mathematical operations on dedicated hardware in general, which makes it convenient to design Neural Network architectures and train them on individual machines or parallel computing resources. These techniques often yield speedups in the range of 50 times over doing the same calculation on a CPU. Unlike other libraries like Theano and TensorFlow, where an entire computational graph has to be defined first before the model can be executed, PyTorch can define the graph dynamically [71].

Regarding performance, the library is written in Python, but also in C++ and CUDA, which is a language from NVIDIA that can be compiled to run with massive parallelism on GPUs. This capability delivers a consistent strategy for deploying models in production. Moving computations from the CPU to the GPU only requires a couple more function calls. Moreover, PyTorch provides facilities, natively by

tensors, that support numerical optimisation on generic mathematical expressions through its autograd engine under the hood [72], which Deep Learning uses for training. By having tensors and the autograd-enabled tensor standard library, PyTorch can be utilised for simulation, optimisation, rendering, modelling and physics. Nevertheless, PyTorch is most importantly a Deep Learning library, and as such it provides all the building blocks needed to build Neural Networks and train them. A standard setup that loads data, trains a model and then deploys that model to production is depicted in Figure 2.6. Before the training data even reaches the model, a bit of data processing is needed. Firstly, the user needs to physically obtain the data, usually from some sort of storage as the data source. Then he needs to convert each sample from the data into something that PyTorch can handle, which is tensors. As data storages are often slow, specifically due to access latency, the user wants to parallelise data loading and PyTorch provides several utilities to achieve that. In the training loop, the model runs the required calculations on the CPU or GPU (single or multiple, locally or not), and once it has the data, the computation can start immediately. At the end of it, the user is rewarded with a trained model whose parameters have been optimised on his task. In the end, the deployment part of the process may involve putting the model on a production server, exporting it to load it to a cloud engine, integrate it with a larger application, or run it on a phone. PyTorch offers a method for compiling models beforehand through TorchScript [73]. PyTorch can serialise a model using TorchScript and transform it into a set of instructions which can be called independently from Python. It functions as a virtual machine with a limited instruction set, specific to tensor operations. This allows to export the model, either as TorchScript to be used with the PyTorch runtime or in a standardised ONNX format, and serve it based on the particular use case.

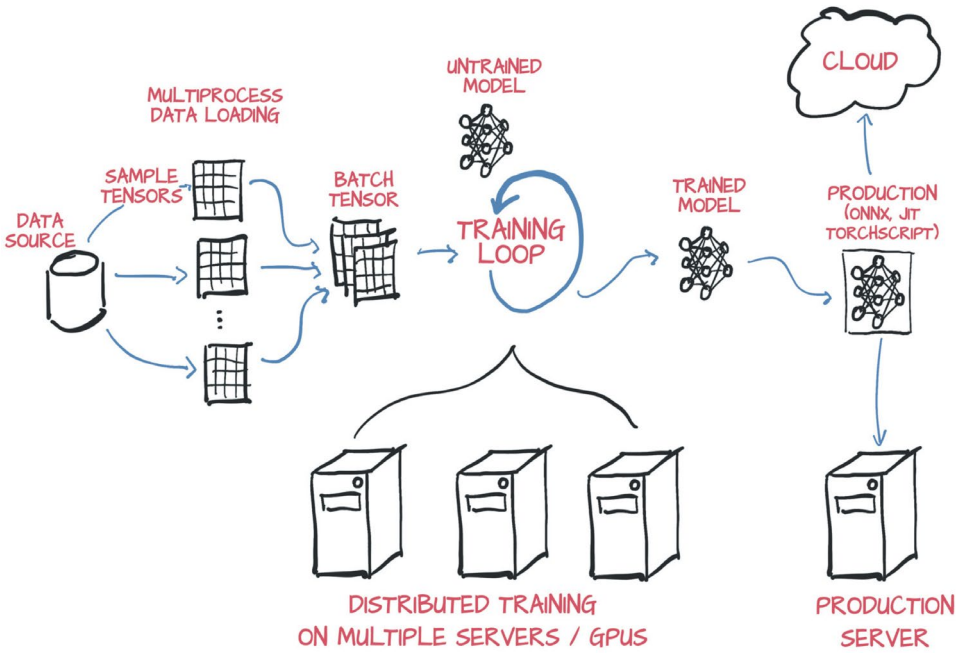


Figure 2.6: Basic structure of a PyTorch project [73]

All things considered, PyTorch is easy to recommend because of its simplicity. Many researchers and students are easily able to learn, utilise, debug and eventually extend it. The library has encroached into the teaching and research communities, due to its ease of use, and has subsequently gained momentum, as graduates and researchers train students and move to industry. It has also paved the way in terms of production solutions. PyTorch consists of multi-GPU support [72], custom data loaders and simplified

pre-processors which is a considerable benefit. It is used primarily for applications such as handwriting recognition, forecast time sequences, style transfer, computer vision and natural language processing. The PyTorch API is simple and flexible, making it a favourite in the development of new Deep Learning models and applications. The extensive use has led to many extensions for specific applications (such as text, computer vision, and audio data), and may pre-trained models that can be used directly. Sometimes, the flexibility of PyTorch comes at the cost of ease of use, especially for beginners, as compared to simpler interfaces like Scikit-learn or Keras. The choice to use PyTorch over them gives up some ease of use, a marginally sharper learning curve, and more code for more versatility.

2.3.4 TensorFlow

TensorFlow is an end-to-end open-source platform for Machine Learning, which is written in Python, C++ and CUDA. It has a flexible and complete ecosystem of tools, libraries, and community resources. This enables users to push the state-of-the-art in Machine Learning and developers to develop and deploy ML-powered applications with ease. At first, TensorFlow was developed by engineers and researchers working on the Google Brain team within Google's Machine Intelligence Research organisation to conduct Deep Neural Networks and Machine Learning research [74]. Google's initial objective was to improve its internal services, such as Gmail, Google search engine and Google Photos. In 2011, the team built DistBelief, a Machine Learning system based on Deep Learning Neural Networks, which soon met rapid growth in both research and commercial applications. After refactoring, simplifying, and generalising its codebase into a faster and more robust application-grade library in late 2015, TensorFlow was first made public. The first stable version appeared in February 2017 under the Apache Open Source license 2.0. The framework provides stable Python and C++ APIs, as well as non-guaranteed backward compatible API for other languages such as Go, Java, JavaScript and Swift [75]. TensorFlow is one of the most popular open-source Deep Learning and Machine Learning frameworks available today.

It can operate at a large scale and in heterogeneous environments [76]. TensorFlow uses unified dataflow graphs to represent computation, shared state, and the operations that transform that state during the training phase. All the computations in the dataflow graphs are done by connecting tensors together, similar to PyTorch. It maps the nodes of a dataflow graph distributed across many machines in a data centre, and within a machine across multiple computational devices, including multi-core CPUs, general-purpose GPUs, and Google's custom-designed ASICs (Application-Specific Integrated Circuits, hardware chips) known as Tensor Processing Units (TPUs). A TPU is a programmable AI accelerator arranged to offer high throughput of low-precision arithmetic (8-bit) and focused on operating or controlling models instead of training them. Third-generation TPUs are able to deliver up to 420 teraflops of performance and 128 GB of High Bandwidth Memory (HBM). Google is running TPUs inside its data centres for more than 4 years. TensorFlow gives the opportunity to write a program in Python and then compile and run it on either CPU, GPU or TPU, so the user doesn't have to write at a C++ or CUDA level to run on GPUs. This whole architecture gives flexibility to application developers and enables them to experiment with novel optimisations and training algorithms.

As TensorFlow grew, its popularity among research papers was diminishing in favour of PyTorch. The team announced a release of a new major version of the framework in September 2019. TensorFlow 2.0

introduced many changes including removal of old and redundant APIs, cross-compatibility, consistency, and unification between trained models on different versions of TensorFlow, and significant improvements to the performance on GPUs. The most significant change was introducing the eager execution mode [77], which changed the automatic differentiation scheme from the static dataflow computational graph to the define-by-run scheme already popular in PyTorch. Eager execution is a programming environment that assesses operations instantaneously, without building graphs, and the operations return explicit values rather than constructing a computational graph to run later. This makes it simple to get started with TensorFlow and debug models, whilst decreasing boilerplate code. Eager execution is enabled by default in TensorFlow version 2.0.

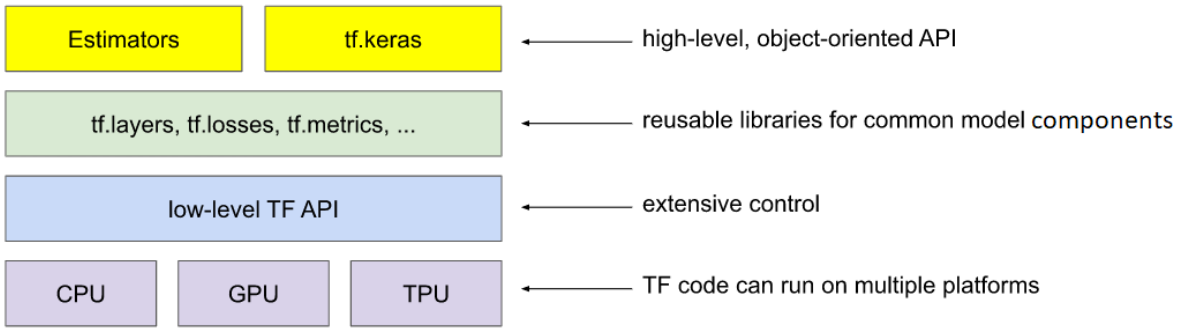


Figure 2.7: TensorFlow toolkit hierarchy [78]

Moreover, the framework is easy for experts, and beginners, as it offers a different level of abstraction and flexibility for researchers, data scientists and developers by providing several APIs as demonstrated in Figure 2.7. These APIs are arranged hierarchically, with the high-level APIs built on top of the low-level ones [78]. At the low level, there is the subclassing API which provides a define-by-run interface for advanced research. By utilising it, the user can create a class for his model, write the forward and backward pass and create custom layers, activations, and training loops. While it assists researchers and experts in exploring new Machine Learning algorithms without limitations and offers extensive control over the whole process, this option is very hard to pick up by the general public. To this regard, there are specific reusable abstract methods and libraries, which are highly optimised for model components. These methods facilitate in building, training, and evaluating a model with ease and promptness. On the top of the hierarchy, there are the high-level and object-oriented APIs which include the estimator API and the sequential and functional APIs from the Keras wrapper. The estimator API offers the opportunity to build (train and predict) and deploy production-ready models [79] and is well optimised and easy to use. TensorFlow uses an estimator for a high-level representation of a complete model. Estimators have been designed for asynchronous training and easy scaling. Pre-made estimators allow the user to try out various distinct model architectures only through minimal code adjustments. In the same manner as Theano, TensorFlow supports the user-friendly Keras wrapper, which even got integrated directly into the framework's core codebase as one of its toolkits in TensorFlow 2.0. Keras' sequential API is the simplest and most approachable method to utilise as a beginner. It offers the ability to create a sequential class and then, from input to output, append layers to the model in a linear manner one by one. In general, it provides all the necessary tools to build and deploy Neural Network models by plugging together building blocks. The Keras functional API is a method for creating models that are more versatile than the sequential API [80]. It can handle models with shared layers, non-linear topology and even multiple outputs or inputs. Its main concept is that a Deep Learning model is oftentimes a directed acyclic graph

of layers. Therefore, the functional API is a way to build graphs of layers. A graph of layers is an innate mental image for a Deep Learning model, and this API is a method of creating models that closely reflects this. However, when building models that are not easily expressible as directed acyclic graphs of layers, model subclassing offers better flexibility to the user. Nevertheless, TensorFlow is so well designed that all models in the Keras API can interact with each other, whether they're sequential models, functional models, or subclassed models that are written from scratch, and can be used interchangeably.

TensorFlow is built to be accessible to everyone. It supports a wide range of applications, with an emphasis on training and testing on Deep Neural Networks. The framework also incorporates different APIs to build at scale Deep Learning architectures, like CNNs and RNNs. Additionally, TensorFlow enables developers to get quickly and easily started with Deep Learning in the Cloud. The framework has become a popular choice for Deep Learning application development and research, mainly in areas such as natural language understanding, computer vision and speech translation. The industry in turn provides extensive support in every use case. TensorFlow comes with a full suite of visualisation tools [81] (TensorBoard) that make it easy to understand, debug, and optimise applications. With support for a plethora of styles ranging from audio and images to graphs and histograms, massive Deep Neural Networks can be trained with simplicity and speed. It additionally enables utilities for valuable data pipelining. They are comprised of built-in modules for serialisation, visualisation and required inspection of diverse modules. Finally, TensorFlow is meant to be applied and deployed at scale. While it is used locally on desktops (CPUs or GPUs, on Windows, macOS or Linux) and the Cloud as a web service, it is also able to run on mobile devices, like iOS and Android, and IoT applications. TensorFlow Lite, used in mobile and embedded devices, features a reduced code footprint and mathematical tools to facilitate smaller model sizes. It is also suitable for cases where network access is overpriced and unreliable. Furthermore, the framework has a robust pipeline to production, an extensive industry-wide community, and massive mindshare. TensorFlow Extended (TFX) [82] is an end-to-end platform for deploying and managing production Machine Learning pipelines. TensorFlow also attracts the largest popularity on GitHub compared to other Machine Learning libraries, having a large and extremely active community of users who regularly contribute code and resolve issues. The framework also offers access to extensive documentation, crash courses and tutorials that can help accelerate development. Various well-known companies from a wide variety of industries use TensorFlow to solve their biggest Machine Learning problems, such as Airbus, Airbnb, Arm, Intel, Twitter, IBM, and PayPal.

2.3.5 Keras

Keras, as mentioned already, is an open-source Deep Learning library for Artificial Neural Networks written in Python. The project was started by Francois Chollet, a Google engineer, in 2015 and it quickly became a popular framework for developers, becoming one of the most popular libraries in the field [80]. It was originally developed as part of the research effort of the project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). The library up until version 2.3.0 offers support for multiple backends including TensorFlow, Theano, Microsoft Cognitive Toolkit (CNTK), R, and PlaidML. Between 2015 and 2019, developing Deep Learning models using mathematical libraries like PyTorch, Theano and TensorFlow was complicated, due to the requirement of several dozens, if not

more, lines of code to complete simple tasks. The main focus of these libraries was on speed and flexibility, not ease of understanding and use.

Keras' popularity came from the clean and simple API, with which, in just a few lines of code, a standard Deep Learning model can be defined, fit, and evaluated. A secondary reason Keras drew so much attention was that it allowed the use of anyone among the range of the popular Deep Learning libraries as the backend. This allowed the power of these libraries to be combined with a very clean and simple interface. In 2019, Google released a new TensorFlow version 2.0 which integrated the Keras API directly and even promoted this interface as the base and standard for development on the platform. In Keras 2.4.0, the multi-backend support has been discontinued to refocus exclusively on the TensorFlow implementation of Keras [83]. TensorFlow 2.0 has entirely consumed the implementation, maintenance, and support of Keras and, in the Keras 2.4.0 release, all APIs in the standalone Keras package are being redirected to point to the "tf.keras" API ("tf" is short for TensorFlow). This change has several benefits for TensorFlow users, such as support for TPU training, distribution, eager execution, and generally superior integration between high-level concepts, like models and layers, and low-level TensorFlow. This way it is also much better maintained as there is a huge active community supporting the projects on GitHub and Google itself.

The library is designed to enable fast experimentation and it focuses on being user-friendly, modular, and extensible. Today, Keras is the high-level API of TensorFlow 2.0 and a very accessible, exceptionally productive interface for solving ML problems, focused on modern Deep Learning. It provides essential abstractions and building blocks for developing and implementing Machine Learning solutions quite fast. These Neural Network building blocks include optimisers, layers, activation functions and tools to enable work with images and text data. Keras runs flawlessly on GPUs and CPUs, supports RNN and CNN networks, or their combinations, and empowers quick as well as straightforward prototyping. It is consistent and can minimise the number of user actions required for common use-cases, while it provides clear feedback on user errors [84]. This ease of use does not come at the cost of reduced flexibility, because Keras integrates deeply with low-level TensorFlow functionality and can be customised in any pieces of code that are needed. Just like TensorFlow, it can be easily deployed across a great range of platforms, such as servers, browsers, and mobile and embedded devices.

Keras has been strongly adopted across both the research community and the industry. Along with TensorFlow 2.0, Keras has been adopted more than any other Deep Learning solution. Some of the most famous and daily-used applications utilise features built with Keras, such as Netflix, Uber, Yelp, Instacart, Zocdoc, Square, and many others. It is particularly popular among start-up companies, in which Deep Learning is chosen as the core of their products. Keras and TensorFlow are also a favourite among researchers, coming in the first place in terms of mentions in scientific papers in Google Scholar [84]. Keras has also met adoption by researchers at big scientific organisations, like NASA and CERN. Last but not least, Keras is built for the real world, where a successful model begins with data collection and ends with production deployment. To this end, Keras is at the centre of a wide ecosystem of projects that together cover every step of a Machine Learning workflow, including rapid model prototyping (AutoKeras), scalable model training on Google Cloud Platform (TF Cloud), hyperparameter tuning (Keras Tuner), model deployment on mobile, and embedded, (TF Lite) or in the browser (TF.js), and inference model quantisation (TF Model Optimization Toolkit).

2.4 Conclusion

In this chapter, a detailed study has been carried out on the theoretical and technical background of the available Machine Learning algorithms, libraries and frameworks. This study was performed in order to understand the way these algorithms function and the capabilities of the available libraries and frameworks. Their advantages, disadvantages, and key features, have been presented and helped get a good sense of which cases they could be applied to and might be more suitable in. As these details and decisions are very critical to the final performance of the experiment, this study was necessary before continuing with the implementation phase which follows. At this point, the most suitable candidates seem to be the Neural Networks and the Support Vector Machines algorithms, based on the idea we have on how the design of the use-case is going to be. Regarding the libraries and frameworks, Keras with TensorFlow and PyTorch seem to be the most flexible ones and they possibly fit better in the real-time and real-world deployment scenario that we are dealing with in this application. In the next chapters, there will be given more specific answers to these questions, as the whole experiment process is being analysed.

Chapter 3 Experiment

3.1 Introduction

This chapter is dedicated to a thorough technical report of the experiment process. The first section will be related to the data mining and data analysis techniques that were applied to the original raw data and how these data have been adjusted and transformed to suit the topic and design of the Smart Home use case. Documentation about the way the dataset has been designed and formed into this shape is also discussed. Thereinafter, there will be an insight into the general strategy that was followed in terms of planning how the whole workflow will function as an integrated system. A brief draft of this design will be provided. Additionally, the implementation of the training of the Machine Learning model is described, along with any further associated adjustments. Last but not least, this end-to-end experiment is concluded by the model deployment on a Docker container on a real-life server. The core technical procedure of the conducted experiment is presented in this chapter, while a more detailed discussion on its concluding results is delivered in the next and final chapter of the document.

3.2 Data analysis and preparation

In this section, the data are going to be analysed to clarify how and why they have been transformed in such a way in order to reflect the requirements of the thesis' topic. Based on the description of the project, a brief study will be performed in order to explain what information is needed to define the Indoor Environmental Quality (IEQ). In addition, after analysing the required material and formulating the shape the data has to be suited in, we are going to describe the technical methodology which was followed in order to acquire, store and construct the dataset for the model implementation and training. This section is dedicated to the pre-processing phase of a Machine Learning workflow.

3.2.1 Problem formulation and data modelling

As humanity evolves, more environmental risks arise and more premature deaths or diseases appear due to the exposure to urban outdoor air pollution and indoor air pollution. Based on statistics provided by WHO (World Health Organisation) [85], some interesting marks are shown below. Both climate and health are in great peril due to air pollution. Ambient (outdoor) air pollution accounts for an estimated 4.2 million deaths per year due to stroke, heart disease, lung cancer and chronic respiratory diseases. Household air pollution causes 3.8 million people a year to die prematurely from illness due to the inefficient use of solid fuels and kerosene for cooking. Based on WHO air quality limits, only 9% of the global population lives in places within acceptable levels. The combined effects of household and ambient air pollution result in approximately 7 million premature deaths each year. To address this, policies and investments supporting cleaner transport, energy-efficient housing, power generation, industry and better waste management can effectively reduce important sources of ambient air pollution. However, nowadays, people spend a considerable amount of their time in indoor environments and

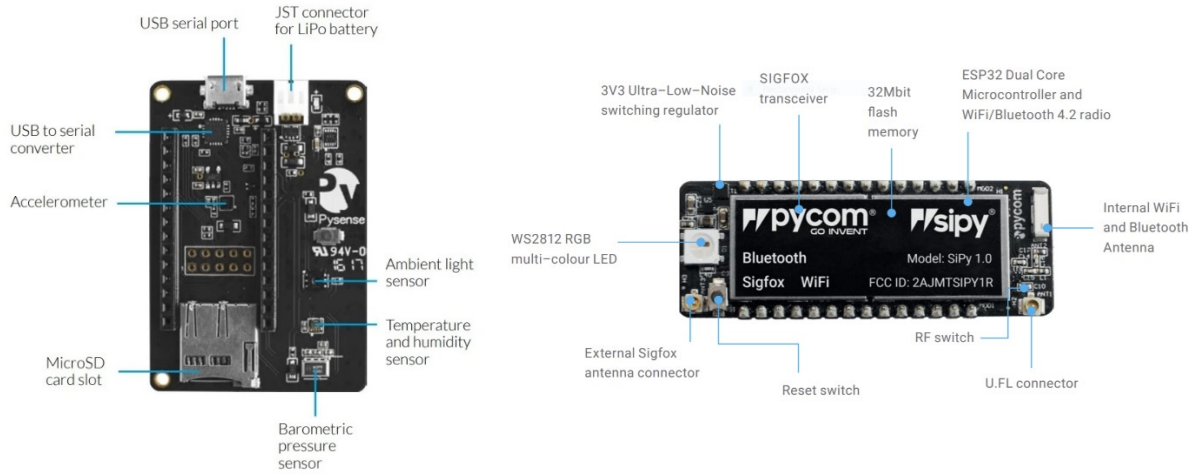
problems of indoor air pollution are increasingly documented as significant risk factors for human health, requiring different management compared to those used for ambient air pollution.

To this end, WHO and its indoor Air Quality Guidelines (WHO AQGs) [86] has reviewed the scientific evidence on health effects resulting from dampness (humidity), microbial growth and contamination of indoor environments, for both private households and public workspaces. Numerical guidelines for specific biological factors could not be identified because of the complex nature of the exposure and related uncertainties. Therefore, they created a set of recommendations addressing several defined indicators of health risk in indoor spaces, such as moisture and the presence of mould in buildings, created by inadequate ventilation. Excessive moisture leads to the development of microbes, such as mould, fungi and bacteria, which produce spores and Volatile Organic Compounds (VOCs) into the air. Furthermore, it worsens the chemical or biological condition of furniture and materials, which also creates indoor air pollution. Humidity is a strong and consistent indicator of allergies, asthma, respiratory symptoms (such as cough and wheeze) and problems of the immune system. These guidelines aim to raise awareness and act as a recommendation on how to reduce health issues related to indoor exposure to biological risk factors and on how to manage indoor air quality in general. Nevertheless, they do not provide specific instructions for succeeding in those goals. The definition of those methods is left to the judgement of each authority and they are more abstract than somebody would assume.

Additionally, there are some guidelines on indoor air pollutants, such as benzene, CO, formaldehyde, naphthalene, NO_2, trichloroethylene, tetrachloroethylene, Particulate Matter (PM), polycyclic aromatic hydrocarbons, radon, and combustion products. These substances are often found indoors in concentrations high enough to raise concern. The main sources of NO_2 and CO in the indoor space are gas cookers and poorly maintained heating appliances, respectively. Formaldehyde sources include water-based paints, vehicle exhaust, clipboard, cigarette smoke and insulation materials. Sources of VOCs include building materials, cleaning products, solvents, glues, furniture, and tobacco smoke. These pollutants have a wide range of effects on health depending on reasons such as their concentration, amount of ventilation, or size of the enclosed environment. Green Guard system evaluates plentiful products for their air quality impact [87]. No matter which of them is to blame in each scenario, the main measure to tackle the problem is controlling and minimising the source of emission within the air, which is a very optimistic solution to this matter. Thus, another good approach is to control their spreading by ensuring sufficiently ventilated spaces or using low-emission materials and appropriate devices. Attempts to guarantee energy efficiency and sustainability in buildings and homes should ensure enhanced health, comfort, and productivity of the occupants. Energy-efficient measures in a building, such as increasing the ventilation rates and ratios of filtered air in the HVAC system, can have a very positive impact on indoor air quality. While moisture management is also very important to avoid mould issues in any type of building. All these problems and symptoms lead to the so-called ‘Sick Building Syndrome’ that disturbs the everyday life of the occupants [88].

Having explicitly defined the elements that affect Indoor Air Quality, we should provide information about the sensory IoT equipment available to us in order to describe what kind of measurements we are able to acquire and how we can model them based on the associated facts. Our equipment is shown in Figure 3.1 and Figure 3.2 and is consisted of Pycom’s Pysense 2.0 X and SiPy. Pysense is a five-sensor shield which is responsible for the collection of the data. The sensors it contains are ambient light sensor, accelerometer, barometric pressure, humidity, and temperature sensor [89]. Among other things, it has

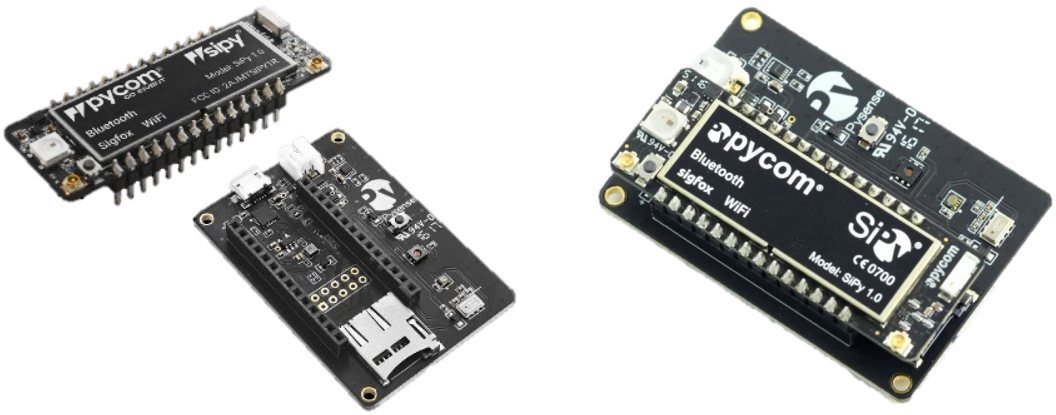
a USB port for wired connections or power supply, a LiPo battery charger and a MicroSD card slot, while consuming very little power.



(a) Pycom Pysense 2.0 X *(b) Pycom SiPy*

Figure 3.1: IoT sensory equipment [90]

SiPy is a triple-network development board, which is programmable using MicroPython and runs on an Espressif ESP32 chipset [91]. Some of its capabilities include Bluetooth Low Energy (BLE), Sigfox and WiFi wireless connectivity, powerful dual CPU and ultra-low power consumption compared to other microcontrollers. This unit is custom-programmed and responsible for the wireless connection to the Internet and the processing and handling of the Pysense sensors and their data. Subsequently, the formulas that were deducted from the analysed facts about Indoor Air Quality, considering the available equipment, are displayed in Table 3.1 resulting in five input values and three outputs. As it should have already been inferred from the problem description, our experiment will be formed as a supervised ML model. Thus, these five variables will constitute the model's input and the three output values will comprise the target values that the model will be trained on and learn from.



(a) SiPy and Pysense side by side *(b) Pysense and SiPy connected*

Figure 3.2: IoT sensory equipment together [90]

Table 3.1: Data modelling

Input	Formula		Output
RH	-	$\xrightarrow{equals}$	RH
OAP, VAP, IAP	$OAP - (VAP \cdot 0.02 + IAP \cdot 0.98)$	$\xrightarrow{equals}$	AP Difference
IAP, T	$\frac{IAP}{Rspec \cdot T}$	$\xrightarrow{equals}$	AD

Where RH: indoor Relative Humidity (%), T: indoor Temperature (°C)
OAP: Outdoor Air Pressure, VAP: Ventilation Air Pressure, IAP: Indoor Air Pressure (Pa)
AP Difference: Air Pressure Difference (Pa), AD: Air Density (kg/m^3)
R specific: specific gas constant for dry air, 287.058 $(J/(kg \cdot K))$

In Table 3.1, we show the way our data has been formulated in order to follow the Indoor Air Quality guidelines. Regarding Relative Humidity (RH), studies have proved that RH directly affects Indoor Air Quality [92]. Specifically, low humidity (RH lower than 30%) causes respiratory problems, such as lung and throat irritation, which are further worsened by the heightened amount of dust particles in the air. High levels of humidity (RH higher than 70%) contain enough moisture for mould colonies and dust mites to grow, which lead to allergies and illnesses. Furthermore, as we already mentioned, proper ventilation is a key measure to improve air quality [93], because it allows for sufficient air circulation reducing the presence of undesirable particles, bacteria, dust mites and VOCs. One of the outcomes of air renewal is maintaining a negligible difference between indoor and outdoor air pressure (±10 Pa) [94]. Higher pressure differences can lead to air leakage through openings, cracks and joints in the building envelope, either through air infiltration (when the outdoor pressure is higher than the indoor pressure) or air exfiltration (when the outdoor pressure is lower than the indoor pressure) [95]. This phenomenon may cause various negative effects on the building itself, such as corrosion, condensation, rotting wood and brick spalling. Moreover, it allows impurities from structures and surroundings (for instance, radon from the beneath soil) to be sucked into the indoor air. All these effects result in the difficulty of controlling indoor temperature and humidity, which in turn further deteriorates the quality of the indoor air [96]. It should be noted that after some empirical calculations and testing on the VAP impact on the overall air pressure difference, we reduced its percentage on the formula from 20 per cent to only 2 per cent as abnormal output values have been noticed on the initial assumption [93]. These percentages should be adjusted based on the environmental and spatial conditions according to each application scenario before use. Regarding air density, it is inversely proportional to temperature and relative humidity, and proportional to the concentration of air pollutants (such as radon, PM and VOCs) [92], [97], [98]. Thus, air density can be an indicator of indoor air pollution, but it should be correlated with the aforementioned factors before any pertinent conclusions can be drawn [88]. Detailed benchmarks on these output parameters can be found in Table 3.2. Last but not least, it should be underlined that these benchmark values of the outputs are calibrated and adjusted to the United Kingdom standards and weather conditions. To this end, they should not be used to characterise Indoor Air Quality in dissimilar environments, as it would draw false results. Any differentiation to the space surroundings can bring unrepresentative outcomes, thus, at least a brief, study and research should be conducted beforehand with the intention of fine-tuning these standards.

Table 3.2: Output Benchmarks

Relative Humidity (%)		**Air Pressure Difference** (Pa)		**Air Density** (kg/m^3)	
Range	**Characterisation**	**Range**	**Characterisation**	**Range**	**Characterisation**
0 – 25	Bad, Too dry	< - 40	Bad	< 1.115	Bad
25 – 35	Okay, Dry	-40 – -10	Okay	1.115 – 1.135	Okay
35 – 45	Ideal	-10 – 10	Ideal	1.135 – 1.155	Ideal
45 – 60	Okay, Humid	10 – 40	Okay	1.155 – 1.175	Okay
60 – 100	Bad, Too humid	> 40	Bad	> 1.175	Bad

3.2.2 Data preparation and pre-processing

This section is dedicated to a thorough description of how the data have been stored and collected, and how the dataset for the ML model has been constructed. In addition to that information, a brief analysis of the nature of the data is being conducted before continuing with the model implementation. Initially, regarding the IoT sensor installation, it should be specified that the experiment has been executed in an, approximately 100 m^2, indoor space of the HPN office as demonstrated in Figure 3.3. By studying the data in the previous section, we settled on indoor, outdoor and ventilation input values. Therefore, one sensor was mounted and connected to a window outside of the building, gauging the Outdoor Air Pressure (OAP) and serving as SmartAir1. The required indoor metrics refer to Relative Humidity (RH), Temperature (T) and Indoor Air Pressure (IAP), hence we installed three sensors inside as this is the most vital aspect, SmartAir3, SmartAir4 and SmartAir5. Subsequently, we deployed two sensors on the ventilation system of the office, which measure the Ventilation Air Pressure (VAP) and they are the services SmartAir6 and SmartAir7.

Moreover, since we could maintain a continuous power supply easily through Pysense's USB port and a fast and reliable 5G Internet connection, we chose to utilise SiPy's WiFi network option. Regarding SiPy's configuration and programming, we used the Atom plugin, called Pymakr [99], provided by the Pycom ecosystem. Pymakr enables the communication with any Pycom development board, such as SiPy, and lets uploading MicroPython code with the purpose of configuring it. In our case, all of our six sensors had already been pre-configured to send real-time measurement updates to the pre-existing FIWARE [100] IoT platform every 10 seconds. The only adjustment that was made from our side was to provide the WiFi credentials to the lab's 5G network. All the SiPy development boards are triggering updates for all of their sensor types on the Pysense, except the accelerometer. The retrieval of their real-time measurements can be done through the IoT platform using a REST API via HTTP requests and specifying the required sensor-board (the Pysense - SiPy combination) service. An example of how a request to the IoT platform looks like for the outdoor SmartAir1 service is presented in Table 3.3. It is a direct HTTP GET request that only by the 'Fiware-Service' and 'Fiware-ServicePath' parameters, we receive a JSON response with the real-time sensor measurements in the last 10 seconds interval.

Figure 3.3: Sensor deployment floor plan

Table 3.3: FIWARE IoT platform request example

Request:

GET *http://137.222.204.81:1026/v2/entities*
Headers:
Fiware-Service: SmartAir1
Fiware-ServicePath: /SmartAir1

JSON Response:


```
[{
    "id": "smartair-outdoors",
    "type": "Outside-Monitoring",
    "Humidity": {
        "type": "float",
        "value": "53.13162",
        "metadata": {}
    },
    "Light in Lux": {
        "type": "int",
        "value": ["40", "53"],
        "metadata": {}
    },
    "Pressure in Pa": {
        "type": "float",
        "value": "100868.7",
        "metadata": {}
    },
```

```
    "Temperature in  deg C": {
      "type": "float",
      "value": "29.75",
      "metadata": {}
    }
}]
```


These measurements are overwritten on every new update (10 seconds), thus we needed a way to store all this information and save it for the construction of the dataset, which will be designed and used for the training of the Machine Learning model. To this end, we deployed two Python scripts running as services in our Virtual Machine (VM) on the server, Demeter and Hermes, and three MongoDB databases for storing the sensor-boards' measurements and their information. MongoDB is a NoSQL document-oriented database, which stores JSON-like documents grouped into collections. In this project, this kind of database is fairly convenient since we have to deal with HTTP responses in JSON format. Our databases' scheme is quite simple, hence we don't provide any visualisation, and these databases are called 'Service', 'Data' and 'Status'. The first one is responsible for keeping the FIWARE credentials for all of the sensor-board services in a different collection for each of them, where we provide the FIWARE service ID and path. Subsequently, the second one is the most critical as it reserves all the real-time data we retrieve from the sensor-boards over time in separate collections for each service. Finally, in the same manner as the other ones, the 'Status' database keeps the functional status of each sensor-board service, namely, it checks whether or not the specified sensor is currently active and sending new data or has stopped operating and updating its measurements in the 'Data' database.

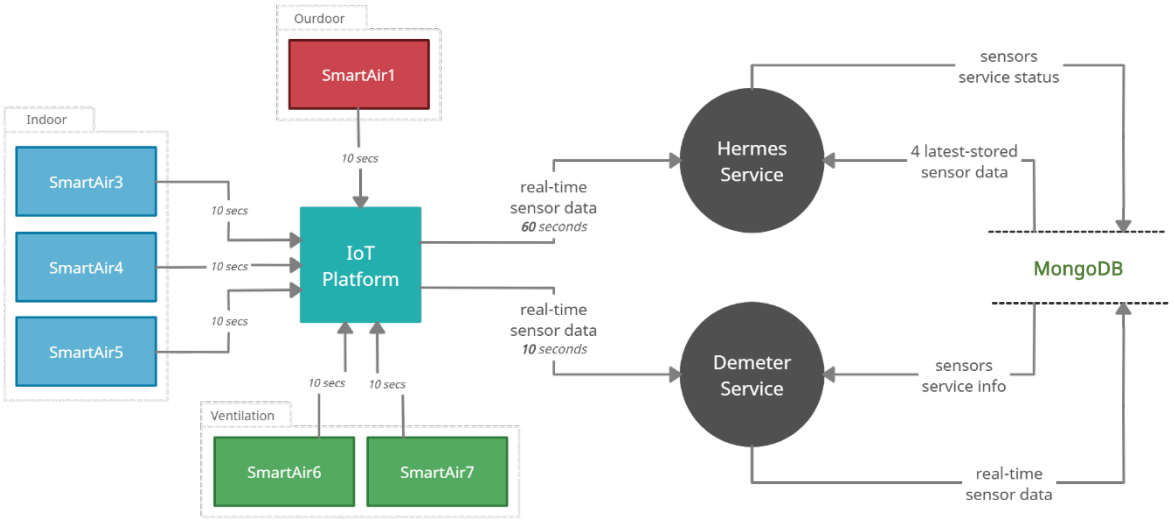


Figure 3.4: Demeter and Hermes Data Flow Diagram

Regarding the two Python scripts, Demeter and Hermes, their purpose is to store and monitor the sensor-board services, respectively. In Figure 3.4, we demonstrate their data flow diagram so as to conceive the way they function with no trouble. As we already described, the IoT platform does not save a real-time sensor data history, thus we have implemented Demeter, a script that sends HTTP requests to the platform every 10 seconds retrieving the last known real-time sensor data based on the sensor-board service information (service ID and path) from the 'Service' database. While, finally, it combines this service information, the timestamp and the sensor measurements in one Python dictionary (JSON-like

document) and stores it in the MongoDB ‘Data’ database. Demeter is set to operate in the VM as a ‘systemd’ service unit found in the Linux distributions, as in our case Ubuntu. The unit files are configuration ini-styled files controlled by the ‘systemd’ init system (a service and system management suite). So, we deployed a service that executes the Demeter script (when the VM boots), and restarts it automatically after three seconds, in case it stops executing. That way Demeter functions as a reliable, and uninterrupted, service and stores the real-time sensor measurements into the database.

In conjunction with Demeter, Hermes is functioning supplementarily to ensure the continuous operation of the sensors. After some days of the sensor-board deployment in the space, we noticed that there was some inconsistency regarding the activity of the sensors. Some of the boards were inactive not updating their measurements, for a shorter or longer period, without any visible or perceptible reason, requiring constant monitoring. This issue was not very frequent yet unpredictable, hence it undoubtedly required a workaround to prevent further problems. To this end, we developed a Python script, Hermes, that continuously monitors the functional status of the sensor-boards, by getting the four latest-stored sensor data from the ‘Data’ database comparing them with the real-time measurements every 60 seconds. If at least one of the stored values is different from the real-time ones, then we know that the corresponding service is active and measuring as intended. This status is being saved in the ‘Status’ database and, at any moment, we are able to check it and act accordingly. Just as Demeter, Hermes is also set to operate in the VM as a ‘systemd’ service unit and, in both cases, we implemented a logging system that records every change of state of the two scripts for troubleshooting and monitoring reasons to certify their well-tempered execution.

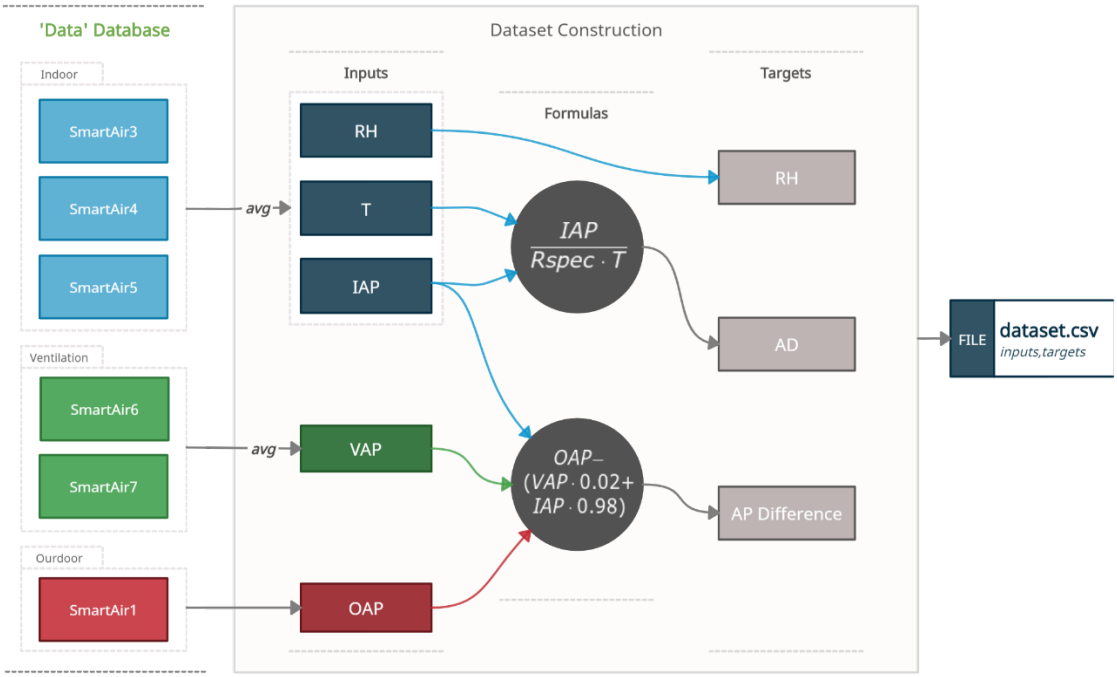


Figure 3.5: Dataset Construction Diagram

Having clarified what and how is being stored in MongoDB, we are well aware of the available sensory data and their structure. In this regard, we are able to describe the way we constructed the CSV file that constitutes our dataset. To begin with, as we thoroughly described our problem formulation and the modelling of our data, we made apparent that our experiment belongs to the supervised learning category

of ML models. Therefore, the model is going to be trained being provided with the target values that we expect it to predict. Furthermore, since we underlined that the benchmark ranges of the output values are calibrated to the United Kingdom standards and weather conditions, we have chosen not to include them in the ML model's scope. These benchmarks are susceptible to change based on the environmental and spatial conditions, while the structure of the concluded formulas should mostly remain the same in other scenarios. This choice leads to having decimal values as target outputs forming our model as a regression problem.

In an attempt to simplify the dataset construction procedure, a diagram is being presented in Figure 3.5. Initially, we retrieved all the sensor measurement records from the 'Data' database since the date we verified that all the sensors were active, while we had already implemented Hermes to monitor them. All the following operations on the data have been applied for each entry separately one by one. On the indoor services, SmartAir3, SmartAir4 and SmartAir5, we synchronised their data based on their timestamp having a tolerance of one second on their time difference and stored their average measurements of RH, T and IAP. With these values, we calculated the target AD based on the defined formula and, along with the RH, we stored them as the two target outputs. Afterwards, we worked with the ventilation services, SmartAir6 and SmartAir7, by synchronising their data in the same manner as above and by storing their average measurement of VAP. Finally, regarding the outdoor service, SmartAir1, we stored its measurement of OAP and calculated the AP Difference by combining the retrieved data, and kept this result as the third target output. The final step involved saving and merging these values in a two-dimensional array that was exported as a CSV file. A small, table form, sample of the dataset can be inspected in Table 3.4. We can notice that there are five input values RH, T, IAP, OAP, VAP and three target outputs for our model, target RH, target AP Difference and target AD. Some of the numbers have been rounded for presentation purposes.

Table 3.4: Dataset sample records

RH (%)	**T** (ºC)	**IAP** (Pa)	**OAP** (Pa)	**VAP** (Pa)	**Target RH** (%)	**Target AP Diff.** (Pa)	**Target AD** (kg/m^3)
19.9463	36.2708	102020.23	101989.3	100781.7	19.9463	-6.162666	1.1485954
22.7412	35.9791	100384.53	100387.5	100656.9	22.7412	-2.479666	1.1312462
27.4816	35.125	100640.33	100616.3	101091.1	27.4816	-33.048666	1.1372713
28.5471	35.5625	100820.17	100827.3	100873.3	28.5471	6.071666	1.1376889
30.0069	36.0208	100348.5	100367	100646.6	30.0069	12.538	1.130687

This process concluded in approximately 100 thousand (99,968) entries, which translates into almost two weeks of measurement records, during the summer. Moreover, it should also be clarified that during the periods that any of the sensor-boards was inactive, the IoT platform was responding with the last known measurements and not blank values. That fact results in having some duplicate service data in our database, but we intentionally preserved them since they act as a larger variety of combinations between the service measurements and they are limited in quantity. That case is different from having completely duplicate entries from all the sensor-boards, this scenario would require removing them from our dataset as they would not contribute to better training of our model. Finally, before closing this section, we would like to perform a brief analysis of our dataset values while still preparing them. This analysis is necessary for an attempt to identify possible tendencies or key facts about our data structure,

which will assist us during the training phase of the ML model as we might need to go back and perform some adjustments or pre-processing techniques on the entries.

Table 3.5: Dataset input values analysis

Value	Location	Mean ± SD	Median	Min	Max
Relative Humidity (%)	Indoor	24.8086 ± 2.69	24.7045	19.04603	30.99366
Temperature (ºC)	Indoor	35.5041 ± 0.52	35.5	33.77083	37.3125
Air Pressure (Pa)	Indoor	100,613.2576 ± 629	100,473.2	98,859.163	102,158.167
	Outdoor	100,602.6338 ± 622	100,472.5	98,849.25	102,139.5
	Ventilation	100,938.2578 ± 324	100,864.35	100,215.6	102,228.75

In Table 3.5, we present some basic statistics that give us a better understanding of the dataset. As we can easily observe, relative humidity fluctuates mainly around 24.8 %, and temperature around 35.5 ºC, with low standard deviation, compared to the total value range. Additionally, air pressure at the three different locations is fairly similar, with indoor and outdoor having approximately the same average value and standard deviation, and ventilation air pressure being relatively close to them but slightly off-centred as the mean value is 300 Pascal higher and the standard deviation is even narrower with half the range. This analysis helped us better comprehend the nature of our parameters and might prove to be a valuable insight. Initially, we didn't apply any pre-processing techniques, such as feature scaling, as our data didn't seem to have any abnormalities, but this possibility remains open before completing the training of our ML model. Finally, a correlation matrix between the input and output values has been calculated but didn't provide any worth mentioning conclusions on how the data are correlated, except for the relations we have already been aware of from the problem formulation study.

3.3 Model Implementation

In this section, we focus on the thorough report of the Machine Learning model implementation, as in its training and testing phase. Any parameter tuning, pre-processing adjustment and model architecture experimentation will also be reported and discussed. Finally, the testing results will be presented before moving on to the last section of the experiment process, the model deployment on the Docker container.

To begin with, as already discussed a detailed study has been carried out on the theoretical and technical background of the available Machine Learning algorithms and frameworks. This study and some further research and experimentation on simple and explanatory examples have assisted in getting familiar with them and deciding which tools are closer to our needs, skills and knowledge. All these key points contributed to determining for each of them, to which cases they can be applied and are more suitable. Furthermore, taking into consideration the problem formulation and the conducted data mining, this experiment applies to the category of multi-output regression Machine Learning problems. This is

derived from the fact that our ML model has to predict three numerical values and the multi-output characteristic complicates and restricts our implementation options.

As already stated, the most suitable candidates seemed to be Neural Networks and Support Vector Machines, and based on the design of our use-case the most straightforward solution is Neural Networks since they simply need to have three neurons on the output layer to achieve a multi-output result. Support Vector Machines don't have this capability directly unless the problem splits into smaller ones. Additionally, regarding the libraries and frameworks, Keras with TensorFlow backend seems to be the most flexible and beginner-friendly option. It also appears to fit better in the real-time and real-world deployment scenarios like the one we are dealing with in this application since it provides relatively simple methods of extracting the trained ML model and deploying it. This key feature was another point our application required and we had to explicitly search for. Moreover, we are going to utilise some useful data manipulation and mining functions from Scikit-learn, given the fact that it offers a huge variety of popular and simple methods specialised in these needs. To sum up, our experiment is a multi-output regression problem and our implementation comprises of the supervised learning Neural Networks algorithm and the TensorFlow Keras and Scikit-learn tools.

Before going into more detail about parameter tuning during the training of the ML model, we should clarify that every different configuration version of the model has been documented and stored into CSV files using a logging system provided by TensorFlow and Keras, regarding the training history and testing performance. This information was stored along with plots, created using Matplotlib, that show the training performance of the model and useful insights on the prediction results during the testing phase. It should also be noted that all the performance metrics that will be presented have been acquired after 10-fold cross-validation applied on all the different versions of the model to avoid overfitting and underfitting. Great attention has been given to saving and storing any information related to the performance of the model on every parameter adjustment, even between folds and training epochs. This policy has been invoked for later reference on determining which model architecture and configuration has been the most performance efficient. Furthermore, regarding the most suitable performance metrics on regression problems that were used for specifying which model was performing better, brief research was conducted and Mean Absolute Error (MAE) and Mean Squared Error (MSE) has been selected for this task. Particularly, MSE has been found to be more representative of the model's performance since it points out, highlights and draws more attention to the difference between the model prediction and the target value, whereas MAE might be negligible on the same occasion. To this end, in many cases, only MSE is being provided as a performance metric.

To begin with the actual implementation of the Neural Network, Keras' sequential API is being utilised for the model's training and testing. As described, it allows developing the architecture of the model by adding its layers one by one. It provides a great variety of layer types, for simple to more complex problems, such as speech and image recognition. In this application, it does not seem to require any unconventional type of layer since the problem is expected to be fairly straightforward, thus we are going to be using only Dense layers. Fully connected feedforward layers, just like the ones used in ordinary traditional Neural Networks, where each neuron from the previous layer is connected to all neurons of the next one. Regarding the ML procedure that was followed, at first, the CSV dataset file is being read, shuffled and the entries are split into two arrays, the input attributes and the target outputs. As already deliberately described and discussed in the previous section, there are five input values, RH,

T, IAP, OAP and VAP, and three target outputs for our model, target RH, target AP Difference and target AD. Consequently, the execution of the 10-fold cross-validation loop starts, utilising Scikit-learn's KFold method, and, as it is easily implied, we are using 90% of the total amount of data for training and the rest 10% for testing on unknown data. This results in a training set of 89,970 instances and a testing set of 9,997 instances. In each loop, the function that creates the Sequential model is being called and, then, this model is fitted and trained on the training samples using 20% of them for validation purposes, thus we have 71,976 known entries and 17,994 unknown. The validation set is being used so as to have a better understanding of how well the model performs on unseen data during training and in between its weight adjustments. This is a common technique aiming to avoid overfitting or underfitting on the known training set. Following the end of the training period, the model is being evaluated by being provided with the testing data and making its predictions on samples it has not seen before. All this process and the performance of the model is being stored in CSV files and plots. Last but not least, the model is also being exported and saved for the server deployment, but more details on that matter will be discussed in the next section.

Starting up with the actual ML model training, we are going to discuss the tuning of the hyperparameters and how we managed to adjust and improve the final model's performance. Many parameters affect directly the model's success, some of them are the number of training epochs and the batch size (if any), which defines the number of samples per gradient update. More significantly, the Neural Network's architecture, shape, and size are also very critical parameters. As described earlier, its architecture refers to the number of layers and neurons, the type of Activation Function (AF) in each layer and the optimiser, which is the function that the model uses to minimise loss. All these parameters need to be fine-tuned until the margin of error is minimal. Hyperparameter tuning during training is usually a trial-and-error process and there is not any proven and well-established methodology. To this end, the very first attempts showed poor performance and instability. This performance also seemed to have reached a margin that was not possible to be surpassed using any adjustment. The most stable early attempt is presented in Figure 3.6 and will constitute the initial comparison reference concerning the model's performance. This model consists of five layers, excluding the input layer which inevitably has five neurons. Particularly, there are four hidden layers of 64, 256, 32 and 256 neurons respectively. Their Activation Functions are all ReLU (Rectified Linear Unit), which is a function that returns the input value if it is greater than zero or zero in any other case. The output layer has invariably three neurons and a Linear AF, which is necessary for a multi-output Neural Network and cannot be modified. The optimiser that is being used is the Adam algorithm, which is a stochastic gradient descent method, with a learning rate of 0.005. The number of epochs that were found to be sufficient is 250. Additionally, it is observed without difficulty that there is a problem with the convergence of the model. This is an example of underfitting and the model has some trouble adjusting to the dataset. The final training and validation losses are 385.1684 MSE and 11.3588 MAE, while the testing performance is 36.6458 MSE and 13.5045 MAE. These results are very abnormal and the 10-fold cross-validation was not applied, as this architecture is certainly unreliable. This is also a confirmation that MAE can be misleading in some cases such as this one, where it would have drawn false conclusions by presenting a close-to-zero loss.

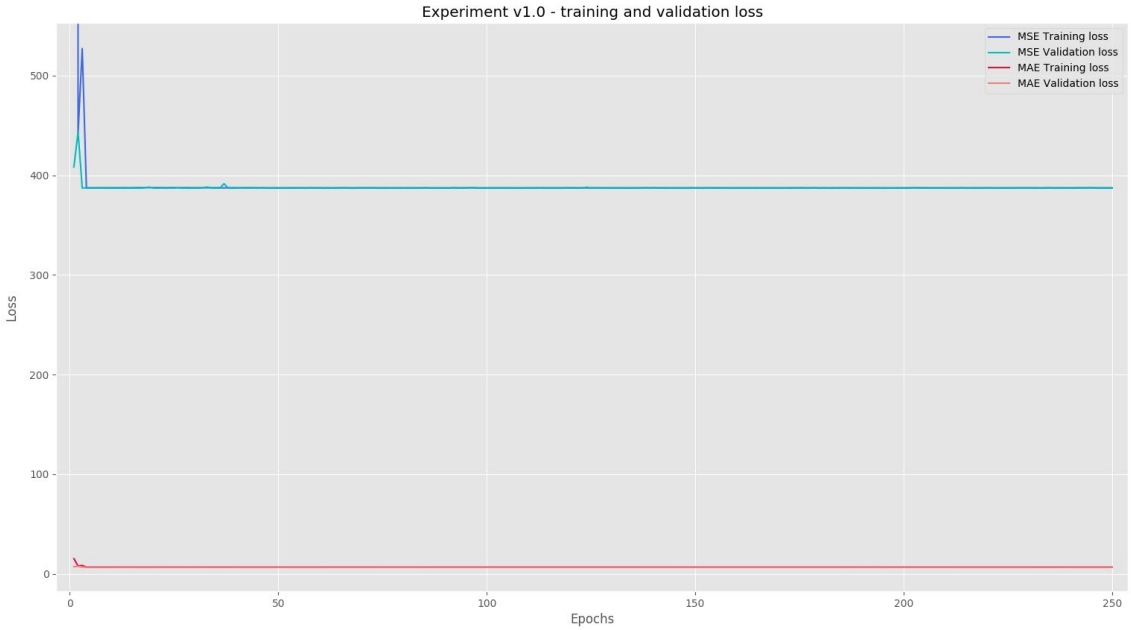


Figure 3.6: Model Experiment v1.0 - Training Loss

This behaviour is a sign that the model either needs more epochs, more complex architecture or some pre-processing technique applied to the input data before being fed into the model. Since the first two options had been already tried with no luck, the next step is to choose and implement a pre-processing method with the intention of assisting the model to fit and converge. There was an inkling since the data preparation that feature scaling might be necessary, seeing that our features have a large scale difference in their values, like for instance RH and IAP. Those differences cause the model to make the false assumption that higher ranging numbers are of greater importance, thus playing a more decisive role during training. A very common approach to this problem is scaling all the input features into the same value range and this technique can facilitate faster convergence of the model. The most popular methods are standardisation and normalisation, and both of them are being tested out. On the one hand, standardisation (StandardScaler) typically rescales the data so as to have a mean of 0 and a standard deviation of 1 (unit variance). This scaler might not be very suitable in cases where the data is not normally distributed. On the other hand, normalisation (MinMaxScaler) rescales the values into a range, in our case a range of [0,1], but might be sensitive to outliers. To this end, both of them are being applied to the same model as before and, then, they are being compared. Initially, the scaler is fitted into the training input data and then it transforms both the training and the testing inputs. In Figure 3.7 (model v2.0, standardisation) and Figure 3.8 (model v3.0, normalisation), we can see their training and validation performance per epoch. Specifically, when standardisation was applied the training loss was 0.0941 MSE and 0.0838 MAE, and the testing was 16.7439 MSE and 0.0965 MAE. In the case of normalisation, during training, the loss was 0.3187 MSE and 0.1600 MAE, while during testing it was 21.6195 MSE and 0.3135 MAE. Each case showed a vast improvement over training and testing data, but the testing performance on unknown input is still not adequate and acceptable.

Figure 3.7: Model Experiment v2.0 (Standardisation) - Training Loss

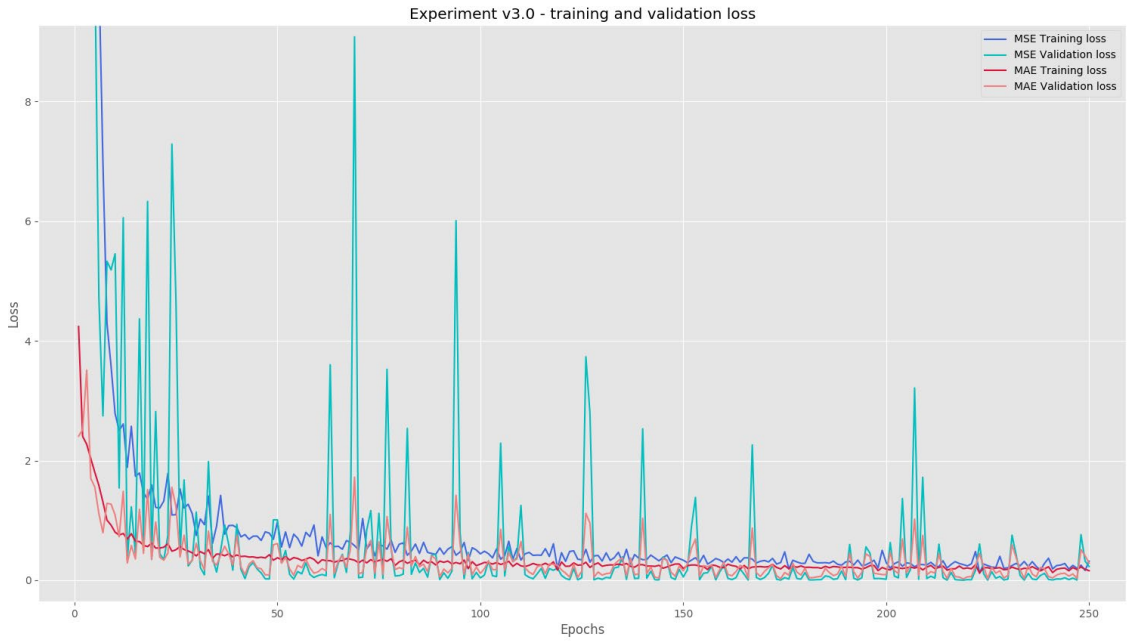


Figure 3.8: Model Experiment v3.0 (Normalisation) - Training Loss

Continuing with the experimentation and the parameter tuning, the next version of the model involves the previous one, version 3.0 with normalisation, but has a simpler shape. It consists of three layers, excluding the input layer. More particularly, there are two hidden layers of 64 and 32 neurons respectively. The output layer, the AFs and the optimiser remain the same. We can see its performance during training in Figure 3.9, with 0.0939 MSE and 0.0512 MAE. The test evaluation metrics are 0.1260 MSE and 0.1059 MAE. The performance of the model has increased significantly, despite its smaller size, but the experimentation will continue in search of an even lower error rate.

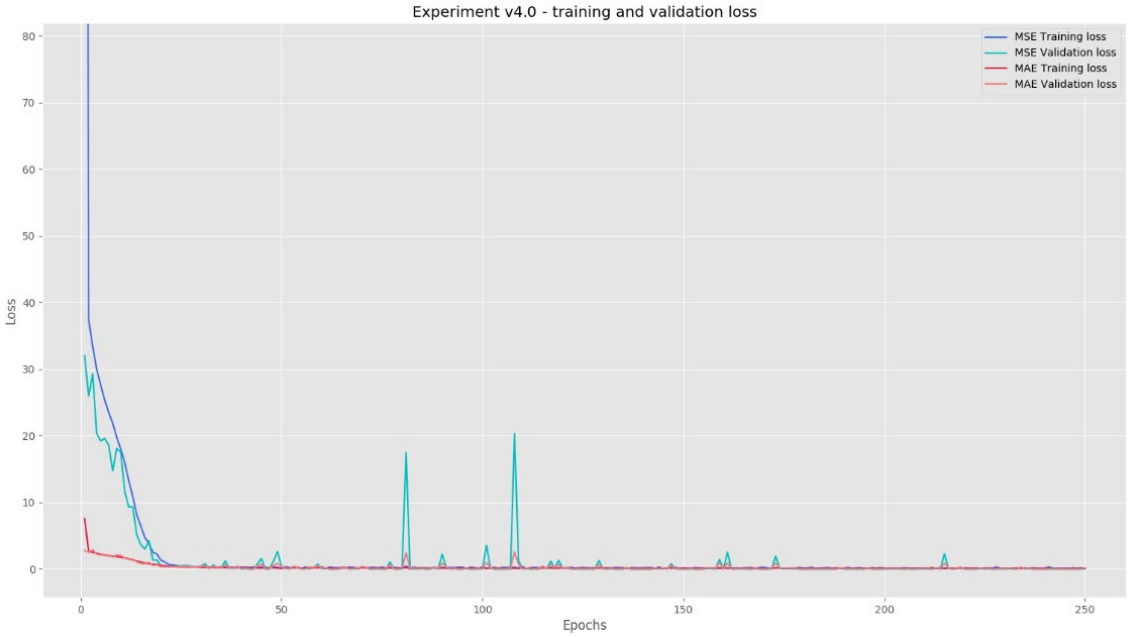


Figure 3.9: Model Experiment v4.0 - Training Loss

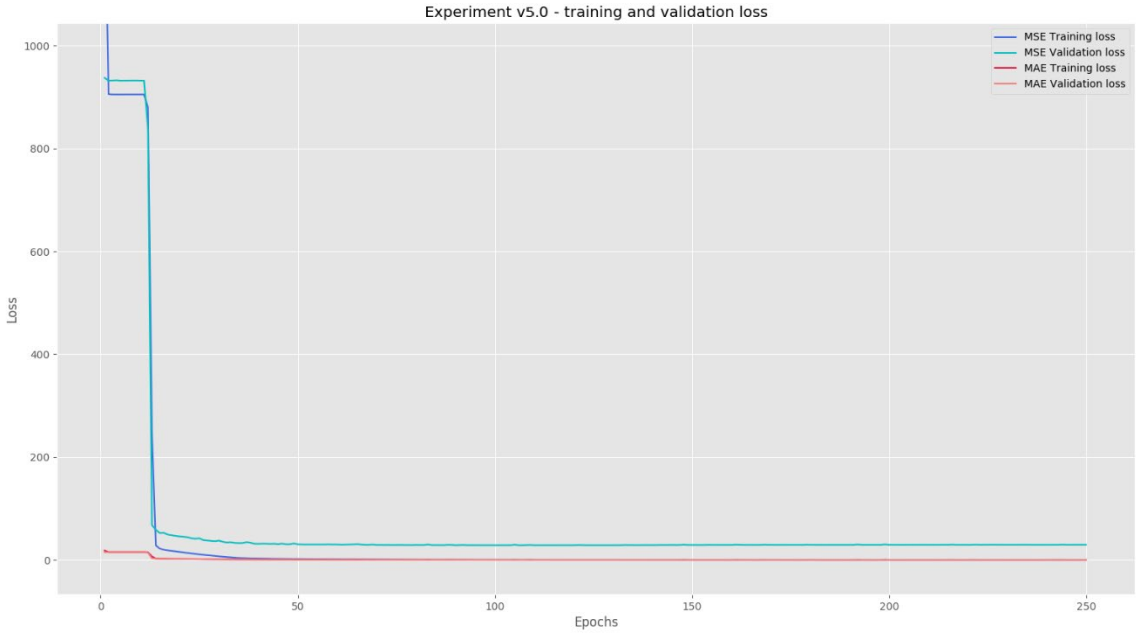


Figure 3.10: Model Experiment v5.0 - Training Loss

In the same manner as before, and only by changing the AFs of the hidden layers to sigmoid, we have the fifth version of the model. In this example, the possibility of achieving a lower loss with a different AF is being examined. Figure 3.10 shows the model's performance during training, which is 0.3154 MSE and 0.1606 MAE. The prediction loss during testing is 1.4309 MSE and 0.2615 MAE. We notice a poorer performance in this scenario, while additionally there is a high validation error, at 29.3751 MSE, that does not improve over the epochs. It appears that the sigmoid AF is not suitable for our use case.

Since the last experiment v5.0 with the sigmoid AF was not successful, we roll back to model version 4.0, which has a smaller structure and normalisation scaling. Here, we study the effect of applying standardisation instead of normalisation on our data. In Figure 3.11, an improved representation of the training loss is shown, where we display more concretely only the MSE and demonstrate more successfully the average error of the 10-fold cross-validation per epoch. Furthermore, a 95% Confidence Interval (CI) overlay is being used, aiming to measure the degree of certainty or uncertainty on the loss value. It essentially draws an area in which there is a 95% probability that our loss will fall inside. The narrower the margin, the more certain we are about the value the error will have, in 95% of the cases. In this model version, we have 2.6204 MSE during training and 2.7608 MSE during testing. Even though the CI is fairly narrow and stable, standardisation (model v6.0) did not manage to exceed normalisation's performance (model v4.0), having the same model configuration, and the error was significantly higher.

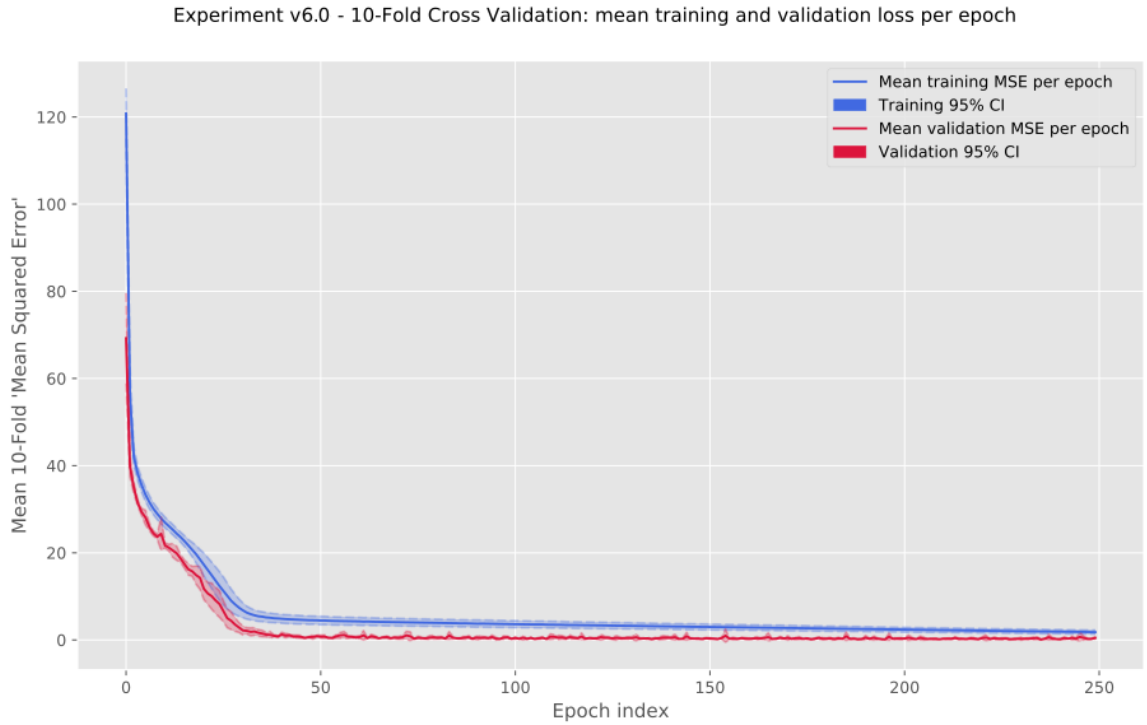


Figure 3.11: Model Experiment v6.0 - Training Loss

As final experimentation, and in order to ensure that the model's complexity is only as necessary, we increase the number of neurons a little so as to confirm that the performance does not improve with a more complex structure. In that way, we will make sure that the architecture of the model is neither smaller nor bigger than it should be, successfully achieving the lowest error rate possible. To that end, in versions 7.0 and 8.0, we keep the number of hidden layers to two but extend the number of neurons to 128 and 64 respectively. The rest of the parameters is preserved as-is. In model v7.0, normalisation is applied and its performance is demonstrated in Figure 3.12. The model converged at 0.0374 training MSE and reached a 0.0429 MSE at testing. This appears to be the best performing model so far, while its loss' CI is also very low and narrow. Finally, in model v8.0, we apply standardisation to the same architecture, and we can observe its loss per epoch in Figure 3.13. This is a clear example of overfitting, as validation loss (18.4454 MSE) is much higher than training loss (0.3561 MSE). Therefore, testing evaluation metrics are also very poor as expected, with 12.3298 average testing MSE on unseen data.

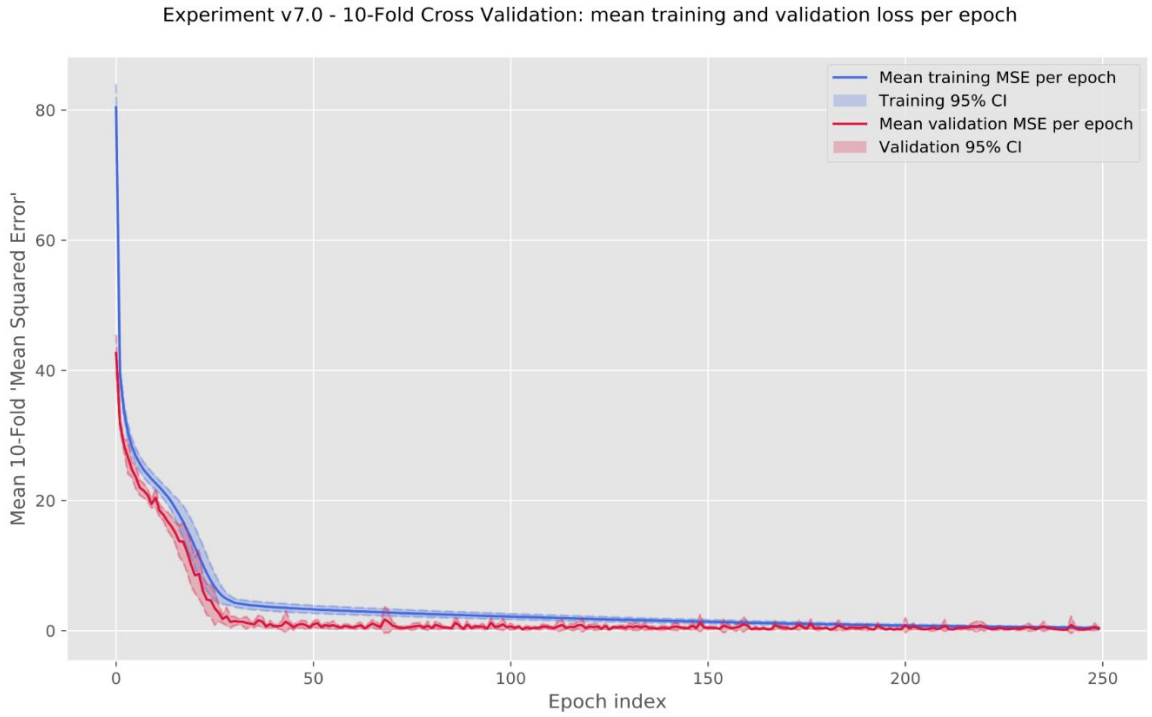


Figure 3.12: Model Experiment v7.0 - Training Loss

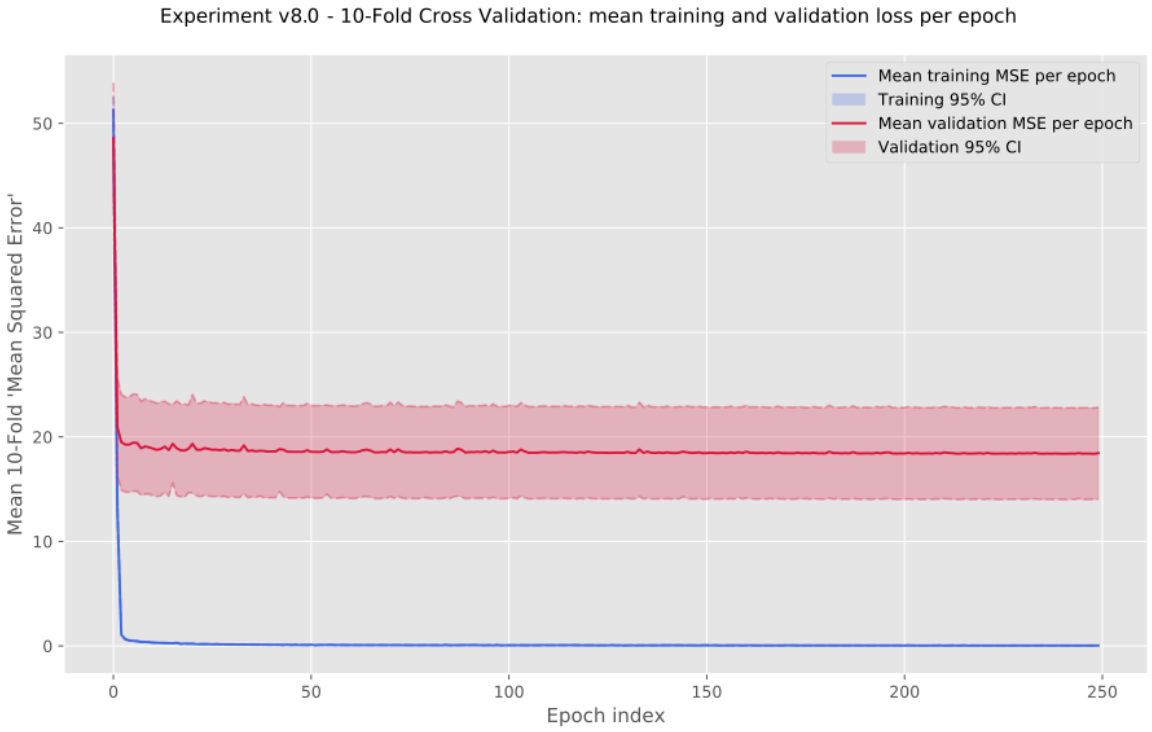


Figure 3.13: Model Experiment v8.0 - Training Loss

To this point, the training experimentation and parameter tuning of our ML Neural Network are concluded. Consequently, we present a comparison table of the implemented model experiments, concerning their training and testing performance metrics, along with the corresponding configuration of each model version. The parameters that remained the same throughout the training process are not mentioned in Table 3.6. These are the input layer, which has five neurons, the output layer, which has

three neurons and a Linear AF, and the optimiser function, which is the Adam algorithm. The Adam optimiser is one of the most popular options in Neural Networks since it applies stochastic gradient descent and is usually the most suitable, effective and endorsed algorithm in the majority of the cases. The training epochs that were used were 250, after some tuning experimentation in the early stages. Based on Table 3.6, it becomes apparent that the best model version, as in its performance, is v7.0, with 0.0374 training MSE and 0.0429 testing MSE. Naturally, the testing loss is the most vital metric, as it determines the model's ability to predict accurately the required values in the production scale. Upon experimenting with more simple and complex scenarios, we found the most optimal combination of the parameters that achieves the lowest error rate. The second worth-mentioning case is version 4.0, which has slightly higher loss values, of 0.0939 training MSE and 0.1260 testing MSE. Additionally, we notice that the best performing scenarios are the ones with normalisation scaling applied, despite the fact that, in the first pre-processing attempt, the standardisation case was more efficient.

Table 3.6: Model Experiments Comparison

Model Version	**Pre-processing & Characteristics**	**Hidden Layers**	**Neurons**	**AF**	**Training MSE**	**Testing MSE**
v1.0	no scaling	4	64, 256, 32, 256	ReLU	385.1684	36.6458
v2.0	standardisation	4	64, 256, 32, 256	ReLU	0.0941	16.7439
v3.0	normalisation	4	64, 256, 32, 256	ReLU	0.3187	21.6195
v4.0	normalisation & more simple	2	64, 32	ReLU	0.0939	0.1260
v5.0	normalisation & sigmoid AF	2	64, 32	Sigmoid	0.3154	1.4309
v6.0	standardisation & more simple	2	64, 32	ReLU	2.6204	2.7608
v7.0	normalisation & more complex	2	128, 64	ReLU	0.0374	0.0429
v8.0	standardisation & more complex	2	128, 64	ReLU	0.3561	12.3298

In a similar manner, we demonstrate the testing MSE of the model in a comparison histogram, aiming to visualise the loss and assist in the model's evaluation. Figure 3.14 compares the performance of the eight model configurations and points out their differences. The error that was used, as it is reasonable, was the testing MSE. The model's performance on data it has not seen before is its most important characteristic since in production it won't know the real-time instances. The histogram verifies our observations from Table 3.6, that model version 7.0 is the best choice with the lowest loss value, at 0.0429 testing MSE.

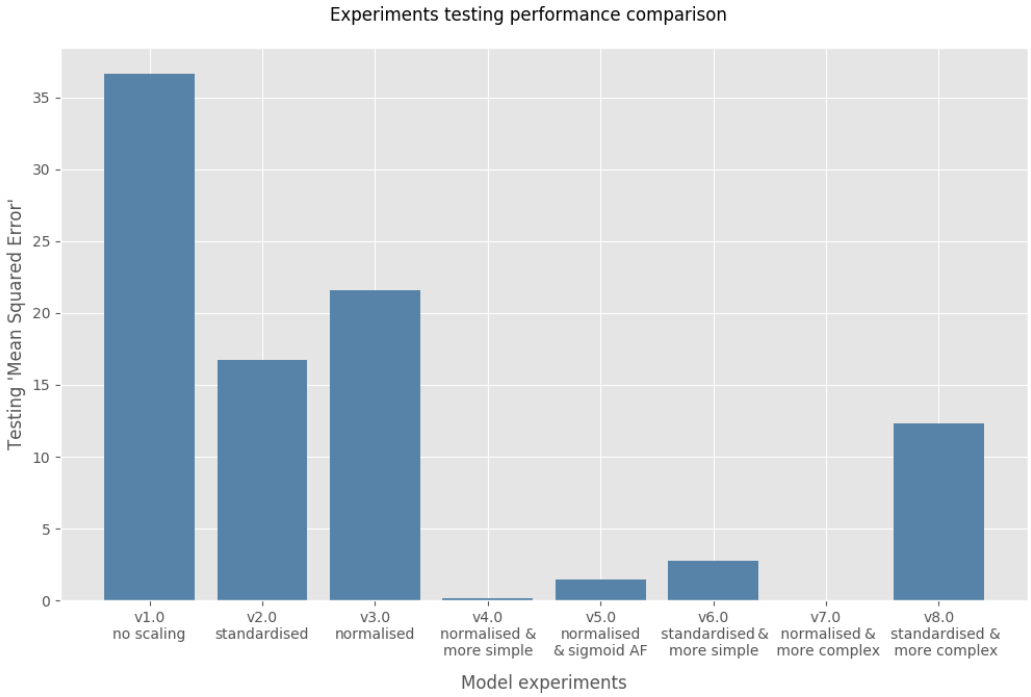


Figure 3.14: Model Experiments Testing Performance Comparison Histogram

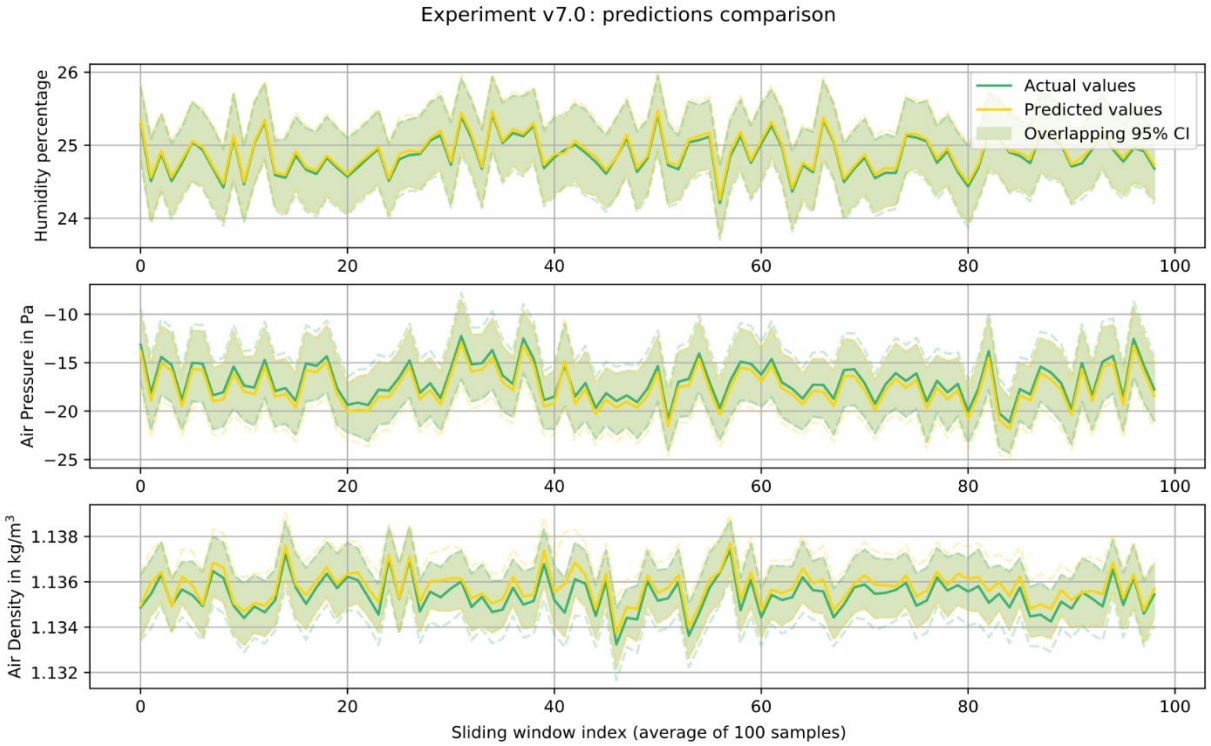


Figure 3.15: Model Experiment v7.0 - Testing predictions comparison

In an effort to visualise the testing loss of the model, given that the number itself is not very intuitive and intelligible, we provide a comparison plot of the predictions during testing in Figure 3.15. It is self-explanatory now why we choose model experiment v7.0 for the real-time air quality evaluation on the Docker container. Thus, using this model, we present a comparison between the actual/real values of the samples (green line) and the predicted values (yellow line) that the model delivered during testing.

We are using a sliding window, which shows the average value of every 100 samples, and for that reason, we provide the overlapping 95% CI to demonstrate their shared value range. There are three different plots for each output value, the Relative Humidity percentage, the Air Pressure Difference (Pa) and the Air Density (kg/m^3). We notice that the model can consistently predict the RH and AP Difference, as shown from the proximity of the actual (green) and predicted (yellow) lines. At the same time, AD predictions tend to be slightly more inaccurate, possibly due to the small scale of its values. Overall, the model is very precise to its predictions and the low testing MSE (0.0429 MSE) confirms that. By concluding this section, we deliver a final visualisation figure that demonstrates the model's (v7.0) architecture. Figure 3.16 shows the final Neural Network's structure that will be used for the Docker deployment and the real-time air quality evaluations.

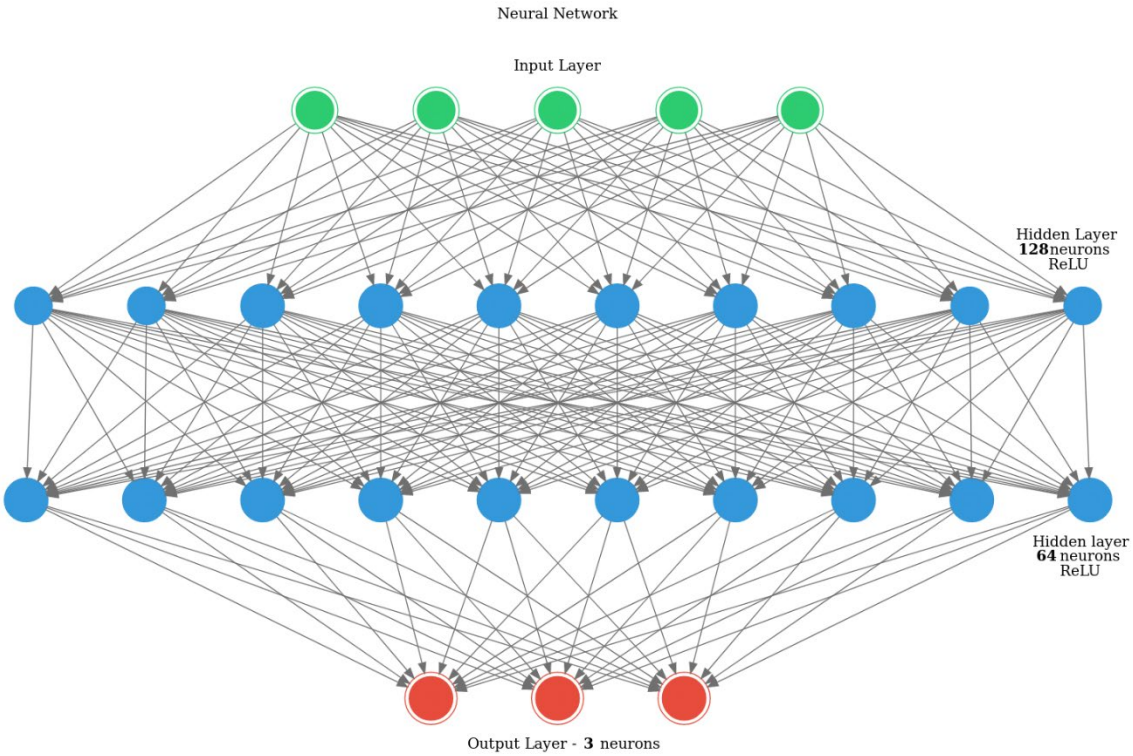


Figure 3.16: Final Model Architecture (model v7.0)

3.4 Model Deployment

This section is dedicated to the process that was followed for the deployment of the model to the Docker container, the production phase. The objective is to serve the trained model in real-time and present its air quality evaluation predictions into an IoT dashboard. The process involves exporting and saving the trained Neural Network, deploying it into our Virtual Machine, serving it with real-time sensor measurements and plotting its responses.

Going into more detail, the first step was to export the trained model into a format that is specifically supported by TensorFlow Serving. TensorFlow Serving is a flexible, high-performance serving system for Machine Learning models, designed for production environments. This format is called 'SavedModel', which is a standalone serialisation format for TensorFlow models, and, by using it, we

can export and save any Keras Sequential model into a single file. This file includes the trained Neural Network's architecture, its adjusted weight values, its training configuration (for instance, its loss function), and its optimiser and its state. 'SavedModel' provides a language-neutral format to save Machine Learning models that are easily recoverable and self-contained so as to facilitate distribution, and enables higher-level tools to deploy TensorFlow models into production. TensorFlow Serving provides out of the box integration with TensorFlow models. The only required parameters are the model itself, in our case the trained model v7.0, and the path to be exported at. The folder that is created by the export contains a TensorFlow checkpoint, containing the model weights, a 'SavedModel' prototype, containing the underlying TensorFlow graph, and the model's architecture configuration.

Similarly, any pre-processing technique applied to our input data has to be employed during the production phase as well. In our scenario, normalisation scaling is being performed before feeding our instances into the Neural Network. This scaler is being fitted into the training inputs and then transforms the training and testing data. Its parameters are being adjusted based on the form of the inputs used for training. The state of the scaler, after fitting into the training data, has to be preserved for later use in the production phase, since this state determines how the unknown data has to be transformed before real-time inference. This state is not available during production, thus we are using the Python library 'Joblib', which enables the persistence of an arbitrary Python object into one file. With 'Joblib', we export and store the fitted scaler, which later has to be re-loaded in order to be applied to the real-time data. These operations are based on the Python pickle serialisation model, which is responsible for serialising and de-serialising a Python object structure by converting it into a byte stream and, naturally, the inverse process. Likewise, the only required parameters are the fitted normalisation scaler and the path to be exported at.

Having stored and exported the trained Neural Network and the scaler, the model deployment process can be initialised. Our working environment is an Ubuntu Virtual Machine in one of the servers of the HPN office. Originally, Docker has to be installed into the system and then we have to create our own serving Docker image with our model built-in. For that, we need an open port on our host to serve on, our 'SavedModel' to serve and a name for our model that our client will refer to. To begin with, we pull down a minimal Docker image with TensorFlow Serving preinstalled and run a new container using it. Then, we copy our exported 'SavedModel' model from our local machine to the predefined directory that the container stores its models. We also have to create a new image from the container's changes by committing it. Finally, we can serve our Neural Network by creating a new container from the latest configured image, by specifying the container's port (port 8501) to the host, our copied 'SavedModel' and its name ('air_quality_model'). This deployed TensorFlow Serving container points to our model and loads it for serving on startup by opening the REST API port (8501). We are able to interact with our model with the 'Predict' RESTful API it offers, by sending a POST request with the real-time retrieved sensor data. In the request, the input data must be in valid JSON format and be keyed as "instances". An example of a POST request to the deployed model can be found in Table 3.7. The input data should have been scaled first, but we are presenting them unmodified for readability purposes.

Table 3.7: Deployed model POST request example

POST Request:

POST *http://localhost:8501/v1/models/air_quality_model:predict*
Headers:
Content-Type: application/json
Data:

```
{
   "instances": [ [
      22.10033,          1. Relative Humidity
      36.41666,          2. Temperature
      101974.8333,       3. Indoor Air Pressure
      101943.5,          4. Outdoor Air Pressure
      102055.85          5. Ventilation Air Pressure
   ] ]
}
```

Model's JSON Response:

```
{
   "predictions": [ [
      22.08507,          1. Relative Humidity
      -33.12566,         2. Air Pressure Difference
      1.14763            3. Air Density
   ] ]
}
```

Up to this point, we have deployed our model to the Docker container and we are able to receive real-time air quality predictions using a POST request, with an average response time of 6 milliseconds. Nevertheless, we do not have continuous air quality updates running as a service that is available at any time. To this end, we have created a Python script that operates in the VM as a 'systemd' service unit, just like Demeter and Hermes. In Figure 3.17, we demonstrate the way our script handles and implements the real-time air quality service. This script starts with retrieving real-time sensor measurements from the IoT platform every 60 seconds. Afterwards, it stores the value of the outdoor sensor (OAP) and calculates the mean values of the indoor sensors (mean RH, T and IAP) and the ventilation sensors (mean VAP). These input data are being scaled before being sent to the deployed model, by loading the exported and stored normalisation scaler that we applied during training. The five inputs are being transformed into a valid JSON format and we send a POST request to the Docker container as we described above. The model responds with its JSON formatted air quality predictions, which are being posted back to the IoT platform as a service. The 'Fiware-Service' and 'Fiware-ServicePath' parameters are set to the 'mlmodel' identifier. That way we can have access to the latest model's predictions at any time without providing any input, just by requesting the service's data from the IoT platform as presented in Table 3.8. This ability is deliberately planned and is going to be very beneficial in the next and final step of the experiment process.

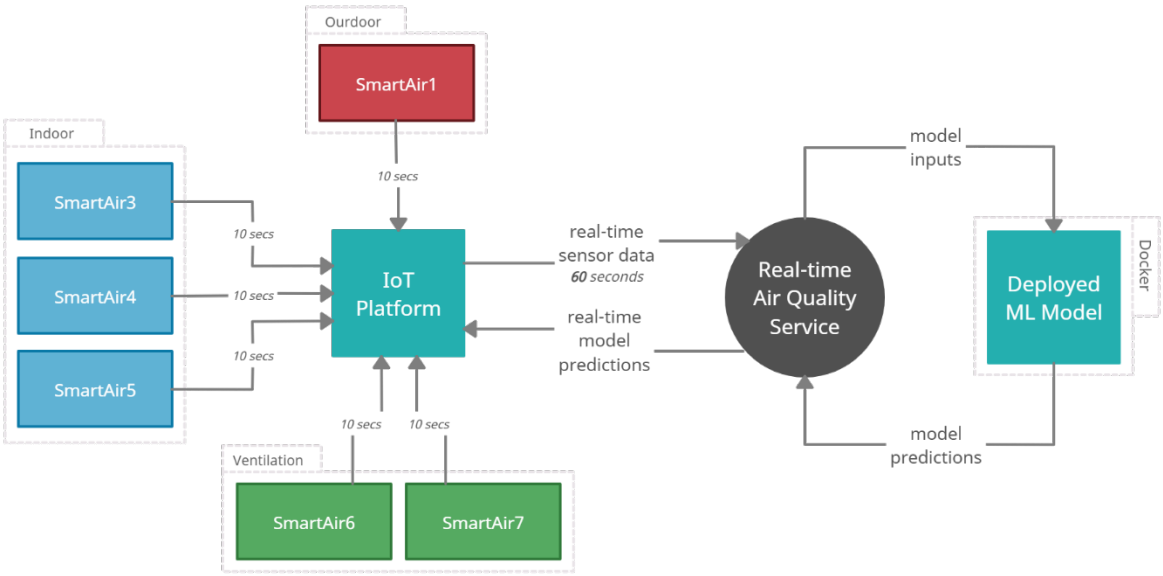


Figure 3.17: Real-time Air Quality Service Data Flow Diagram

Table 3.8: IoT platform model prediction request example

Request:

GET *http://137.222.204.81:1026/v2/entities*
Headers:
Fiware-Service: mlmodel
Fiware-ServicePath: /mlmodel

JSON Response:


```
[{
    "id": "ModelPrediction",
    "type": "type",
    "AirDensity": {
        "type": "float",
        "value": "1.14763",
        "metadata": {}
    },
    "AirPressureDif": {
        "type": "float",
        "value": "-33.12566",
        "metadata": {}
    },
    "Humidity": {
        "type": "float",
        "value": "22.08507",
        "metadata": {}
    }
}]
```


The technical implementation of this dissertation is concluded in this final phase, where we demonstrate and visualise the real-time evaluations of the air quality in the office. As a closure to the experiment, we

have chosen to provide the end-user with a convenient means of updating on the current air quality condition, on an ongoing basis at all times. In this regard, we have utilised the open-source real-time web-based IoT dashboard builder 'Freeboard', which gave us the opportunity to use various widgets and design a dashboard based on our needs. Freeboard supports several data sources, one of which is the Orion Context Broker [101], [102] that is developed as part of the FIWARE platform. Orion is an implementation of the NGSI REST API that allows the management of context information, including updates and queries as described and applied in Table 3.3. Using this Freeboard's method for retrieving real-time data from our FIWARE IoT platform, we display the latest air quality evaluation predictions from our model. At this point, it should be clear why we needed to embed our model predictions in the IoT platform, considering that Freeboard is only able to query a data source and does not provide any way of sending input information as required for our Docker container inference.

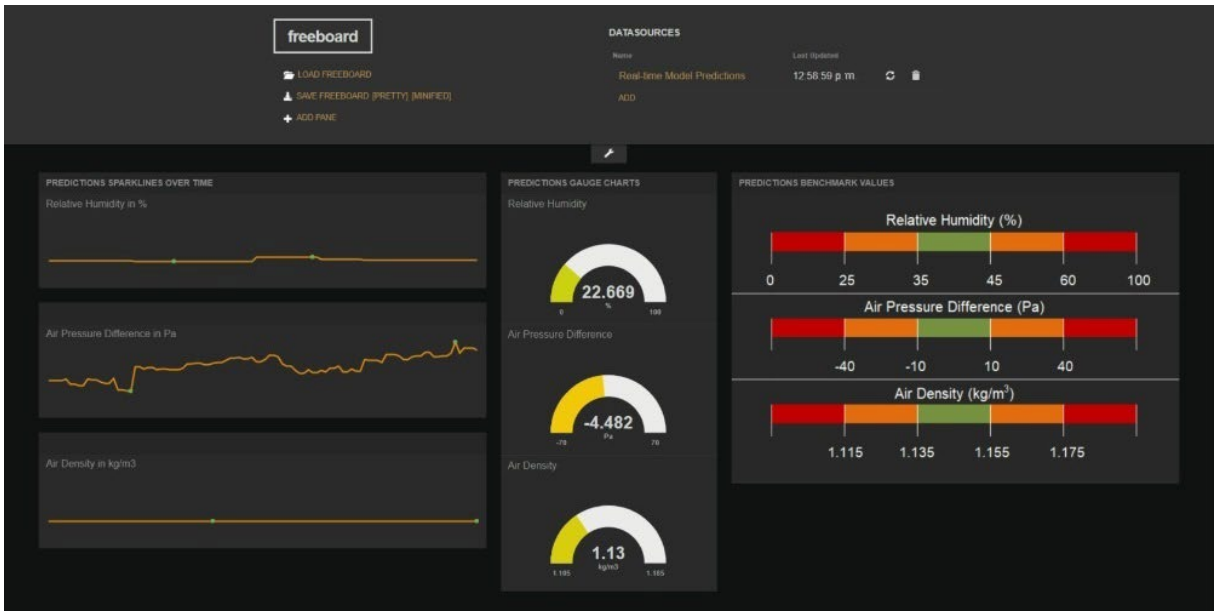


Figure 3.18: Real-time model predictions in Freeboard IoT dashboard

In Figure 3.18, a snapshot of our configured IoT dashboard in Freeboard is presented. There, Gauge charts from the real-time model predictions and graphs from the air quality evaluations over time are being populated and displayed, updating every 60 seconds, along with their corresponding benchmark values. After some brief testing on the model's performance during the production phase, it appears that it is quite capable of carrying through its task. By this means, we offer a cross-platform end application with sustained access to the current state of the air quality in the HPN office. This implementation settles the fully-featured end-to-end experiment that we set to accomplish in this project.

3.5 Conclusion

At this point, the thorough technical report of the experiment process is concluded. Initially, the problem that we were going to resolve has been extensively shaped and formulated based on the conducted study and the available equipment and knowledge. The data has been modelled and designed according to the nature of the Indoor Environmental Quality problem. In compliance with these crucial aspects of the project, the original raw data has been analysed, prepared and transformed to suit the topic of the

building automation use case. Additionally, we documented the way the dataset was constructed into its final shape, and the resources and services that were used for this implementation. Thereinafter, the ML model training experimentation and hyperparameter tuning procedures have been thoroughly described, along with any pre-processing methods that have been tested out. That phase resulted in a two hidden-layered Neural Network with normalisation scaling and impressive testing performance of 0.0429 MSE. Last but not least, this trained Keras ML model was deployed into a Docker container on a real-life server, utilising TFX and TensorFlow Serving capabilities. In the end, during the production phase, we provide real-time air quality evaluation predictions, which are visualised and presented into a Freeboard IoT dashboard, functioning as an end application and service with continuous access. To sum up, this chapter constituted the complete technical procedure of the conducted experiment and contributed to the comprehension of its entire workflow as an integrated system.

Chapter 4 Conclusion and Further Improvements

In this chapter, we finalise and conclude the conducted study and experiment. After extensive research on the related works and fields, we managed to implement an end-to-end application which combines our two points of interest. On the one hand, Internet of Things grows in popularity daily and aids people in gaining total control over their lives, by helping them live smarter and more qualitatively. On the other hand, Machine Learning is responsible for the vast majority of the AI advancements and breakthroughs that everyone knows about and takes advantage of their benefits. By aiming to join their forces and highlight their great potential, we chose to develop an integrated system that utilises IoT equipment and applies ML to a real-world environment with state-of-the-art tools. Although Machine Learning is not part of the original concept of Smart Homes and the Internet of Things, it can get the most value out of these deployments through analysing IoT data, extracting hidden information, and automating procedures. The proposed idea was to utilise Machine Learning in order to evaluate the Indoor Air Quality in real-time by using Internet of Things wireless sensor equipment. Nowadays, the quality of the air we breathe even inside our homes or working environments is getting more polluted day by day. Many factors can affect its quality and cause a deterioration in the health of the building occupants. In this dissertation, we undertook the real-time monitoring and evaluation of the air quality in a closed environment, the HPN office in the University of Bristol, implemented as a Smart Home (or Smart Building) use case.

Based on WHO records, statistics and guidelines, indoor air pollution is influenced by a different set of elements and characteristics than outdoor. After our study on these characteristics, we settled on three different factors that outline the quality of the air inside a building. These are Relative Humidity (RH), which directly affects Indoor Air Quality, Air Pressure Difference (AP Difference), which is influenced and maintained by sufficient air renewal between indoor and outdoor, and Air Density (AD), which is proportional to the concentration of air pollutants and highly affected by temperature and relative humidity. For estimating those components and based on the available sensory IoT equipment, we had to deploy the sensors indoors, outdoors and inside the ventilation system, as it affects the rotation of air pressure. Thus, we deployed six sensors, three inside, one outside and two in the ventilation, intended for acquiring the measurements of RH, Temperature, and Indoor, Outdoor and Ventilation Air Pressure. Therefore, this study resulted in 5 input values and 3 output values for the ML implementation. After explicitly formulating our problem and modelling our data, we developed two services, Demeter and Hermes, for retrieving and storing those sensor measurements in our database every 10 seconds, and monitoring the functional status of the sensors every 60 seconds, respectively. When those records were sufficient, we constructed our dataset according to the format we defined from data modelling and stored it into a CSV file ready for the ML model. Thereinafter, we conducted the implementation of the model training and the parameter tuning until a minimum margin of error was reached. Since our problem belonged to the multi-output regression category, we chose to use a Neural Network for that use case. Keras with TensorFlow backend has been utilised for the Sequential model implementation and, after several attempts, we finalised the model's architecture and performance. The final Neural Network, that was selected for deployment, had normalisation scaling applied and an architecture of two hidden layers of 128 and 64 neurons with ReLU Activation Function and Adam optimiser. The performance we achieved during training (with 250 epochs) and testing was 0.0374 and 0.0429 MSE, respectively. Consequently, having completed the ML implementation, the chosen model had been exported using TensorFlow Serving, along with the applied scaler, and had been deployed into a Docker container

which serves our trained Neural Network for real-time inference in the production phase using REST API POST requests. Finally, aiming to provide a real-time Indoor Air Quality service, which will offer the real-time evaluation of the air quality in the office at any given time, we embedded the model predictions into the IoT platform we have been using and visualised them into a web-based IoT dashboard, which presents the predictions in graphs and charts on an ongoing basis, along with the produced benchmark values for each of the three outputs. By this means, we provide a cross-platform end application with continued access to the present state of the air quality in the HPN office. This end application settles the fully-featured end-to-end experiment that we set to accomplish in this project, using state-of-the-art tools and offering a low-priced and affordable solution, which is easily extendable and adaptive in similar use cases and scenarios.

Regarding these scenarios and where our implementation might be applicable and beneficial, the first candidate is naturally a real-world Smart Home use case, as it was originally intended, delivering valuable insights on the real-time Indoor Air Quality state of a building and possibly suggest suitable solutions to tackle any issues. Additionally, this integrated system is able to provide maintenance services to uphold a healthy and safe environment of the surrounding place, on even bigger spaces, such as universities, offices or whole companies, where there is professional staff that undertakes the preservation of the building and its occupants. Last but not least, our project can potentially be embedded into a smart device, for instance in air purifiers or dehumidifiers, with the intention of evaluating the air quality and assisting in its corresponding settings adjustments.

Concerning any further improvements that could have been implemented, a good practice would be to acquire more data if there was enough time since more data are always advantageous for any ML model, as it gets to see a greater variety of instances. On top of that, we could also re-train the same model with these new instances over time and at intervals so as to improve its performance and preserve a minimum error rate throughout the seasons. Another suggestion is that we could have used more suitable and specialised IoT sensors that are able to measure specific pollutants in the air and provide a much more detailed report on the type of source that deteriorates the Indoor Air Quality. However, these sensors are more expensive and this fact would discourage their use in domestic or low-budget scenarios. Moreover, we could have developed our own dashboard or mobile app, using Flutter with the programming language Dart, that deliver natively compiled applications for web, mobile, desktop, and embedded devices from a single codebase. That way we would not have to depend on third-party software for our end application to the user. Going one step further, another idea is that we could have implemented a kind of alarm system when the three output values have exceeded the acceptable ranges, so as to warn the user and prompt him to take the necessary actions. In a similar manner, in a Smart Home scenario, we should have fully automated the HVAC control system in an effort to eliminate any human involvement in its regulation. Regarding any alternative ways of implementation but not necessarily improvements, a different approach to the problem would be the development of an LSTM Neural Network, which is suitable for time-series predictions in the future, and provide a forecasting service. In general, we should have tried different ML models for our implementation, but the multi-output characteristic of our problem complicated this option. Furthermore, we could have attempted to train separately three different models one for each output value and then combine them in order to achieve higher accuracy and have a greater variety of options for ML models. Finally, another idea could be to perform the air quality evaluation for specific areas in the building individually and not calculate their average values, thus allowing the localised control and monitor of particular sections. Taking everything

into consideration, this section concludes the proposed experiment we set to accomplish in this dissertation with notable success.

REFERENCES

[1] Z. Fan and R. Liu, "Investigation of machine learning based network traffic classification," in *2017 International Symposium on Wireless Communication Systems (ISWCS)*, Bologna, 2017, pp.1-6.

[2] S. Zander, T. Nguyen and G. Armitage, "Automated traffic classification and application identification using machine learning," in *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)l*, Sydney, NSW, Australia, 2005, pp. 250-257.

[3] E. Alexandrova and A. Ahmadinia, "Real-Time Intelligent Air Quality Evaluation on a Resource-Constrained Embedded Platform," in *2018 4th IEEE International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS)*, Omaha, pp. 165-170, 2018.

[4] J. Vergara-Reyes, M. C. Martinez-Ordonez, A. Ordonez and O. M. Caicedo Rendon, "IP traffic classification in NFV: A benchmarking of supervised Machine Learning algorithms," in *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*, Cartagena, Colombia, 2017, pp. 1-6.

[5] M. Shafiq, X. Yu, A. A. Laghari, L. Yao, N. K. Karn and F. Abdessamia, "Network Traffic Classification techniques and comparative analysis using Machine Learning algorithms," in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, Chengdu, China, 2016, pp. 2451-2455.

[6] S. Choudhury and A. Bhowal, "Comparative analysis of machine learning algorithms along with classifiers for network intrusion detection," in *2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, Chennai, India, 2015, pp. 89-95.

[7] M. Shafiq, X. Yu, A. A. Laghari, L. Yao, N. Kumar Karn, F. Abdesssamia and Salahuddin, "WeChat Text and Picture Messages Service Flow Traffic Classification Using Machine Learning Technique," in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Sydney, NSW, Australia, 2016, pp. 58-62.

[8] L. Kong, G. Huang, K. Wu, Q. Tang and S. Ye, "Comparison of Internet Traffic Identification on Machine Learning Methods," in *2018 International Conference on Big Data and Artificial Intelligence (BDAI)*, Beijing, China, 2018, pp. 38-41.

[9] L. Kong, G. Huang and K. Wu, "Identification of Abnormal Network Traffic Using Support Vector Machine," in *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Taipei, Taiwan, 2017, pp. 288-292.

[10] K. A. Taher, B. Mohammed Yasin Jisan and M. M. Rahman, "Network Intrusion Detection using Supervised Machine Learning Technique with Feature Selection," in *2019 International Conference on Robotics,Electrical and Signal Processing Techniques (ICREST)*, Dhaka, Bangladesh, 2019, pp. 643-646.

[11] M. Saber, S. Chadli, M. Emharraf and I. Farissi, "Modeling and Implementation Approach to Evaluate the Intrusion Detection System," in *International Conference on Networked Systems*, 2015, pp. 513-517.

[12] A. S. Ashoor and S. Gore, “Importance of Intrusion Detection System (IDS),” *International Journal of Scientific and Engineering Research,* vol. 2, no. 1, pp. 1-4, 2011.

[13] J. Zheng, H. Yu, F. Shen and J. Zhao, “An Online Incremental Learning Support Vector Machine for Large-scale Data,” in *Neural Computing and Applications*, 2013, pp. 1023–1035.

[14] F. Gharibian and A. A. Ghorbani, “Comparative Study of Supervised Machine Learning Techniques for Intrusion Detection,” in *Fifth Annual Conference on Communication Networks and Services Research (CNSR '07)*, Frederlcton, NB, Canada, 2007, pp. 350-358.

[15] J. Schmidhuber, “Deep Learning in Neural Networks: An Overview,” *Neural Networks (CoRR),* vol. 61, pp. 85-117, 2015.

[16] M. I. AlHajri, N. T. Ali and R. M. Shubair, “Classification of Indoor Environments for IoT Applications: A Machine Learning Approach,” *IEEE Antennas and Wireless Propagation Letters,* vol. 17, no. 12, pp. 2164-2168, 10 September 2018.

[17] W. Song, J. Han, J. Xie, Y. Gao and L. Song, “System for Detecting and Forecasting PM2.5 Concentration Levels Using Long Short-Term Memory and LoRa,” in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Atlanta, 2019, pp. 834-841.

[18] V. M. Suresh, R. Sidhu, P. Karkare, A. Patil, Z. Lei and A. Basu, “Powering the IoT through embedded machine learning and LoRa,” in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Singapore, Singapore, 2018, pp. 349-354.

[19] P. R. K. Varma, K. P. Raj and K. S. Raju, “Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms,” in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India, 2017, pp. 294-299.

[20] S. P. Choudhary and D. Vidyarthi, “A Simple Method for Detection of Metamorphic Malware using Dynamic Analysis and Text Mining,” *Procedia Computer Science,* vol. 54, pp. 265-270, 2015.

[21] B. Rajesh, P. Reddy, M. Patil and H. Pareek, “ANDROINSPECTOR: A System for Comprehensive Analysis of Android Applications,” *International Journal of Network Security & Its Applications,* vol. 7, no. 5, 2015.

[22] Y. Zhou and X. Jiang, “Dissecting Android Malware: Characterization and Evolution,” in *2012 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, 2012, pp. 95-109.

[23] R. Seth and R. Kaushal, “Detection of transformed malwares using permission flow graphs,” in *2017 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, Bangalore, India, 2017, pp. 17-21.

[24] P. R. K. Varma, V. V. Kumari and S. S. Kumar, “A Novel Rough Set Attribute Reduction Based on Ant Colony Optimisation,” *International Journal of Intelligent Systems Technologies and Applications,* vol. 14, no. 3/4, pp. 330-353, 2015.

[25] X. Fafoutis, L. Marchegiani, A. Elsts, J. Pope, R. Piechocki and I. Craddock, “Extending the battery lifetime of wearable sensors with embedded machine learning,” in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Singapore, 2018, pp. 269-274.

[26] M. Chen, W. Saad and C. Yin, “Resource Management for Wireless Virtual Reality: Machine Learning Meets Multi-Attribute Utility,” in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Singapore, 2017, pp. 1-7.

[27] A. E. Abbas, "Constructing Multiattribute Utility Functions for Decision Analysis," in *INFORMS TutORials in Operations Research*, Published online: 14 Oct 2014, 2010, pp. 62-98.

[28] M. Chen, M. Mozaffari, W. Saad, C. Yin, M. Debbah and C. S. Hong, "Caching in the Sky: Proactive Deployment of Cache-Enabled Unmanned Aerial Vehicles for Optimized Quality-of-Experience," *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Human-In-The-Loop Mobile Networks,* vol. 35, no. 5, pp. 1046-1061, 2017.

[29] MIT Technology Review Insights, "Machine Learning: The New Proving Ground for Competitive Advantage," MIT Technology Review, 16 March 2017. [Online]. Available: https://www.technologyreview.com/2017/03/16/106260/machine-learning-the-new-proving-ground-for-competitive-advantage/. [Accessed September 2020].

[30] L. Wang, Ed., "Support Vector Machines: Theory and Applications," in *Studies in Fuzziness and Soft Computing*, vol. 177, Berlin, Springer-Verlag Berlin Heidelberg, 2005, p. 431.

[31] T. Evgeniou and M. Pontil, "Support Vector Machines: Theory and Applications," in *Machine Learning and Its Applications, Advanced Lectures*, Chania, 2001, pp. 249-257.

[32] K. Diamantaras, *Μηχανές Διανυσμάτων Υποστήριξης,* Thessaloniki: Alexander Technological Institute of Thessaloniki, 2016, p. 37.

[33] C.-W. Hsu, C.-C. Chang and C.-J. Lin, "A Practical Guide to Support Vector Classification," Department of Computer Science, National Taiwan University, Taiwan, 2003, Updated on 2016.

[34] F. VAN VEEN, "THE NEURAL NETWORK ZOO," 14 September 2016. [Online]. Available: https://www.asimovinstitute.org/neural-network-zoo/. [Accessed September 2019].

[35] A. Zheng and J. Jin, "Using AI to Make Predictions on Stock Market," Stanford University, California, 2017.

[36] V. Zhou, "Machine Learning for Beginners: An Introduction to Neural Networks," Towards Data Science, 5 May 2019. [Online]. Available: https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9. [Accessed August 2019].

[37] M. K. J. Ang, E. Valla, N. S. Neggatu and A. W. Moore, "Network Traffic Classification via Neural Networks," University of Cambridge, Cambridge, 2017.

[38] B. Jayaraman, "Artificial Neural Networks - Theory and Applications, ResearchGate," June 2017. [Online]. Available: https://www.researchgate.net/publication/323425401_Artificial_Neural_Networks_-_Theory_and_Applications. [Accessed June 2019].

[39] M. Schott, "Artificial Neural Networks for Machine Learning," Medium, 26 April 2019. [Online]. Available: https://medium.com/capital-one-tech/artificial-neural-networks-for-machine-learning-79c67d0681e9. [Accessed August 2019].

[40] G. James, D. Witten, T. Hastie and R. Tibshirani, An Introduction to Statistical Learning with Applications in R, 1 ed., New York: Springer-Verlag New York, 2013, p. 426.

[41] R. Dwivedi, "What Is Naive Bayes Algorithm In Machine Learning?," Analytics Steps, 29 April 2020. [Online]. Available: https://www.analyticssteps.com/blogs/what-naive-bayes-algorithm-machine-learning. [Accessed June 2020].

[42] K. Diamantaras, *Στατιστικά Μοντέλα και ο Κανόνας του Bayes,* Thessaloniki: Alexander Technological Institute of Thessaloniki, 2016, p. 20.

[43] J. Brownlee, "Naive Bayes for Machine Learning," Machine Learning Mastery, 11 April 2016. [Online]. Available: https://machinelearningmastery.com/naive-bayes-for-machine-learning/. [Accessed September 2019].

[44] M. Patel, “Naive Bayes - Machine Learning Algorithm,” Medium, 21 December 2019. [Online]. Available: https://medium.com/@meetpatel12121995/naive-bayes-machine-learning-algorithm-aaf57bdc8d87. [Accessed June 2020].

[45] S. Ray, “6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R,” Analytics Vidhya, 11 September 2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/. [Accessed 2019].

[46] K. Zakka, “A Complete Guide to K-Nearest-Neighbors with Applications in Python and R,” 13 July 2016. [Online]. Available: https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/. [Accessed September 2019].

[47] “K-Nearest Neighbor (KNN) Algorithm for Machine Learning,” Java T Point, [Online]. Available: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning. [Accessed 13 September 2019].

[48] P. Hall, B. U. Park and R. J. Samworth, “Choice of neighbor order in nearest-neighbor classification,” *Annals of Statistics,* vol. 36, no. 5, p. 2135–2152, 2008.

[49] D. Coomans and D. L. Massart, “Alternative k-nearest neighbour rules in supervised pattern recognition : Part 1. k-Nearest neighbour classification by using alternative voting rules,” *Analytica Chimica Acta,* vol. 136, pp. 15-27, 1982.

[50] K. Beyer, J. Goldstein, R. Ramakrishnan and U. Shaft, “When is "nearest neighbor" meaningful?,” Database Theory—ICDT'99, Wisconsin-Madison, 1999.

[51] S. Imandoust and M. Bolandraftar, “Application of K-nearest neighbor (KNN) approach for predicting economic events: Theoretical Background,” *Int J Eng Res Appl,* vol. 3, no. 5, pp. 605-610, 2013.

[52] G. T. Toussaint, “Geometric proximity graphs for improving nearest neighbor methods in instance-based learning and data mining,” *International Journal of Computational Geometry & Applications,* vol. 15, no. 2, pp. 101-150, 2005.

[53] M. J. Garbade, “Understanding K-means Clustering in Machine Learning,” Towards Data Science, 13 September 2018. [Online]. Available: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1. [Accessed September 2019].

[54] A. Al-Masri, “How Does k-Means Clustering in Machine Learning Work?,” Towards Data Science, 15 May 2019. [Online]. Available: https://towardsdatascience.com/how-does-k-means-clustering-in-machine-learning-work-fdaaaf5acfa0. [Accessed July 2019].

[55] W. Pace, *Demonstration of the standard algorithm k-Means,* 2007.

[56] M. Inaba, N. Katoh and H. Imai, “Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering,” in *10th ACM Symposium on Computational Geometry*, New York, 1994.

[57] N. Tyagi, “What is K-means Clustering in Machine Learning?,” Analytics Steps, 14 March 2020. [Online]. Available: https://www.analyticssteps.com/blogs/what-k-means-clustering-machine-learning. [Accessed August 2020].

[58] “Scikit Learn - Introduction,” Tutorials Point, [Online]. Available: https://www.tutorialspoint.com/scikit_learn/scikit_learn_introduction.htm. [Accessed June 2019].

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M.

Perrot, E. Duchesnay and G. Louppe, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research,* vol. 12, 2012.

[60] R. Vickery, “A Beginners Guide to Scikit-Learn,” Towards Data Science, September 2020. [Online]. Available: https://towardsdatascience.com/a-beginners-guide-to-scikit-learn-14b7e51d71a4. [Accessed September 2020].

[61] J. Brownlee, “A Gentle Introduction to Scikit-Learn: A Python Machine Learning Library,” Machine Learning Mastery, 16 April 2014. [Online]. Available: https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/. [Accessed August 2019].

[62] Scikit-learn, “Choosing the right estimator,” [Online]. Available: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html. [Accessed 2019].

[63] G. Varoquaux, L. Buitinck, G. Louppe, O. Grisel, F. Pedregosa and A. Mueller, “Scikit-learn: Machine Learning Without Learning the Machinery,” *GetMobile: Mobile Computing and Communications,* vol. 19, no. 1, pp. 29-33, 1 June 2015.

[64] S. Deoras, “Tensorflow Vs. Theano: What Do Researchers Prefer As An Artificial Intelligence Framework,” Analytics India Magazine, December 2017. [Online]. Available: https://analyticsindiamag.com/tensorflow-vs-theano-researchers-prefer-artificial-intelligence-framework/. [Accessed September 2019].

[65] J. Brownlee, “Introduction to the Python Deep Learning Library Theano,” Machine Learning Mastery, May 2016. [Online]. Available: https://machinelearningmastery.com/introduction-python-deep-learning-library-theano/. [Accessed 3 September 2019].

[66] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-farley and Y. Bengio, “Theano: A CPU and GPU math compiler in python,” in *Proceedings of the 9th Python in Science Conference*, 2010.

[67] M. May, “An Overview of Python Deep Learning Frameworks,” KDnuggets, 2017. [Online]. Available: https://www.kdnuggets.com/2017/02/python-deep-learning-frameworks-overview.html. [Accessed August 2019].

[68] “Choosing an Open Source Machine Learning Library: TensorFlow, Theano, Torch, scikit-learn, Caffe,” Altexsoft, 18 August 2017. [Online]. Available: https://www.altexsoft.com/blog/datascience/choosing-an-open-source-machine-learning-framework-tensorflow-theano-torch-scikit-learn-caffe/. [Accessed 2019].

[69] J. Brownlee, “PyTorch Tutorial: How to Develop Deep Learning Models with Python,” Machine Learning Mastery, March 2020. [Online]. Available: https://machinelearningmastery.com/pytorch-tutorial-develop-deep-learning-models/. [Accessed June 2020].

[70] PyTorch, “Ecosystem Tools,” PyTorch, 2020. [Online]. Available: https://pytorch.org/ecosystem/. [Accessed August 2020].

[71] D. Mwiti, “Deep Learning with PyTorch: An Introduction,” Heartbeat, 5 October 2018. [Online]. Available: https://heartbeat.fritz.ai/introduction-to-pytorch-for-deep-learning-5b437cea90ac. [Accessed June 2020].

[72] “GitHub,” 2020. [Online]. Available: https://github.com/pytorch/pytorch/blob/master/README.md. [Accessed July 2020].

[73] E. Stevens, L. Antiga and T. Viehmann, “Introducing deep learning and the PyTorch Library,” in *Deep Learning with PyTorch*, New York, Manning Publications Co., 2020, pp. 3-15.

[74] Guru99, “What is TensorFlow? Introduction, Architecture & Example,” Guru99, [Online]. Available: https://www.guru99.com/what-is-tensorflow.html. [Accessed September 2020].

[75] TensorFlow, “API Documentation,” Google TensorFlow, 2020. [Online]. Available: https://www.tensorflow.org/api_docs. [Accessed 2020].

[76] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zheng, “TensorFlow: A System for Large-Scale Machine Learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’16)*, Savannah, GA, USA, November 2016.

[77] TensorFlow, “Effective TensorFlow 2,” Google TensorFlow, August 2020. [Online]. Available: https://www.tensorflow.org/guide/effective_tf2#eager_execution. [Accessed September 2020].

[78] Google Developers, “Introduction to TensorFlow - Machine Learning Crash Course,” Google, March 2020. [Online]. Available: https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit. [Accessed 2020].

[79] A. Kadimisetty, “TensorFlow - A hands-on approach,” Towards Data Science, July 2018. [Online]. Available: https://towardsdatascience.com/tensorflow-a-hands-on-approach-8614372f021f. [Accessed September 2019].

[80] J. Brownlee, “TensorFlow 2 Tutorial: Get Started in Deep Learning With tf.keras,” Machine Learning Mastery, 19 December 2019. [Online]. Available: https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/. [Accessed March 2020].

[81] AWS Amazon, “TensorFlow on AWS,” AWS Amazon, 2020. [Online]. Available: https://aws.amazon.com/tensorflow/. [Accessed September 2020].

[82] TensorFlow, “TensorFlow For Production,” Google TensorFlow, 2020. [Online]. Available: https://www.tensorflow.org/tfx. [Accessed October 2020].

[83] F. Chollet, “Keras Releases,” Google, GitHub, 2019. [Online]. Available: https://github.com/keras-team/keras/releases. [Accessed September 2020].

[84] Keras, “Why choose Keras?,” Google Keras, 2020. [Online]. Available: https://keras.io/why_keras/. [Accessed 2020].

[85] World Health Organisation, “Air Pollution Guidelines,” World Health Organisation, 2017. [Online]. Available: https://www.who.int/airpollution/guidelines/en/. [Accessed August 2020].

[86] WHO Regional Office for Europe, “Evolution of WHO air quality guidelines: past, present and future,” World Health Organisation (WHO), Copenhagen, 2017.

[87] Green Guard, “SPOT Green Guard,” SPOT, [Online]. Available: https://spot.ul.com/main-app/products/catalog/. [Accessed 2020].

[88] A. Mendes, J. Teixeira and P. Wexler, “Sick Building Syndrome,” in *Encyclopedia of Toxicology (Third Edition)*, Oxford, Academic Press, 2014, pp. 256-260.

[89] Pycom, “Pysense 2.0 X Datasheet,” Pycom, [Online]. Available: https://docs.pycom.io/datasheets/expansionboards/pysense2/. [Accessed July 2019].

[90] Pycom, “Pycom,” Pycom, [Online]. Available: https://pycom.io/webshop. [Accessed 2019].

[91] Pycom, “SiPy Docs,” Pycom, [Online]. Available: https://docs.pycom.io/datasheets/development/sipy/. [Accessed July 2019].

[92] M. A. Mujeebu, "Edited by", Indoor Environmental Quality, InTechOpen, 2019.

[93] J. Son and Y.-S. Son, “A Correlation Analysis of Indoor Environmental Quality and Indoor Air Quality using IoT,” in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, South Korea, 2019, pp. 977-979.

[94] V. Leivo, M. Kiviste, A. Aaltonen, M. Turunen and U. Haverinen-Shaughnessy, “Air Pressure Difference between Indoor and Outdoor or Staircase in Multi-family Buildings with Exhaust Ventilation System in Finland,” *Energy Procedia,* vol. 78, no. 6th International Building Physics Conference, IBPC 2015, pp. 1218-1223, November 2015.

[95] R. Quirouette and B. Arch, *Air pressure and the building envelope,* Canada: Home to Canadians.

[96] J. Lstiburek, “Air Pressure and Building Envelopes,” Building Science Press , Massachusetts , 1999.

[97] J. B. Sullivan, M. E. Peterson, G. R. Krieger, P. A. Talcott and M. D. Van Ert, “Chapter 14 - Indoor Environmental Quality and Health,” in *Small Animal Toxicology (Third Edition)*, Saint Louis, W.B. Saunders, 2013, pp. 139-158.

[98] G. E. Kaufman, G. Griffith, R. E. Miller and M. Fowler, “Chapter 12 - Sustainable Practices for Zoological Veterinary Medicine,” in *Fowler's Zoo and Wild Animal Medicine*, Saint Louis, W.B. Saunders, 2012, pp. 86-97.

[99] Pycom/Open-Source, “Pymakr Atom Package on GitHub,” [Online]. Available: https://github.com/pycom/pymakr-atom. [Accessed July 2019].

[100] FIWARE, “WHAT IS FIWARE?,” [Online]. Available: https://www.fiware.org/about-us/. [Accessed 2019].

[101] FIWARE, “WELCOME TO ORION CONTEXT BROKER.,” [Online]. Available: https://fiware-orion.readthedocs.io/en/master/. [Accessed 2019].

[102] FIWARE, “Orion Context Broker,” [Online]. Available: https://github.com/telefonicaid/fiware-orion. [Accessed 2019].