Abstract

This thesis is about developing a high quality Air Pollution Prediction model for giving precise forecasts of air quality based on previous data of air pollution levels and historical weather conditions and weather forecasts. The thesis was given by Telenor in cooperation with NTNU. The Air Pollution training data was provided by NILU and the weather data was provided by Yr.

Preface

First of all, I would like to thank Studentersamfundet in Trondheim for three amazing years filled with joy, laughter and a lot of activities all year long. I am really thankfull for discovering a second home here in Trondheim filled with fantastic people.

Second, I am grateful for having a master thesis in machine learning, because I believe Artificial Intelligence and machine learning is the future and I would like to participate in making the world a better place using this technology.

A huge thanks goes to my friends and family for supporting me with joy, understanding and love. It would be an understatement to say it would be hard to achieve this without them.

Table of Contents

List of Tables

List of Figures

Abbreviations

AQI	=	Air Quality Index
NILU	=	Norwegian Institute for Air Research
datapoint	=	A row in a dataset
feature	=	A column value of datapoint
NaN	=	Not a number
CPU	=	Central Processing Unit
GPU	=	Graphics Processing Unit
$\hat{y}$	=	the predicted output
y	=	the true output
IoT	=	Internet of things
API	=	Application Programming Interface
Epoch	=	One iteration of training all datapoints on a model

Chapter 1

Introduction

1.1 Background

Trondheim kommune have been planning a project in which they want to record air quality of the traffic through equipment mounted on vehicles driving around the city area. This means recording dynamic data, resulting in greater coverage of air pollutants over larger areas of land. This will in turn open for the possibility of providing high quality Air Pollution Predictions.

Today, there exists a national online AQI forecasting platform (Luftkvalitet i Norge, 2019a). It is a cooperation between the government agencies Miljødirektoratet, Statens vegvesen, Vegdirektoratet, Meteorologisk institutt, Folkehelseinstituttet and Helsedirektoratet (NILU, 2019b).

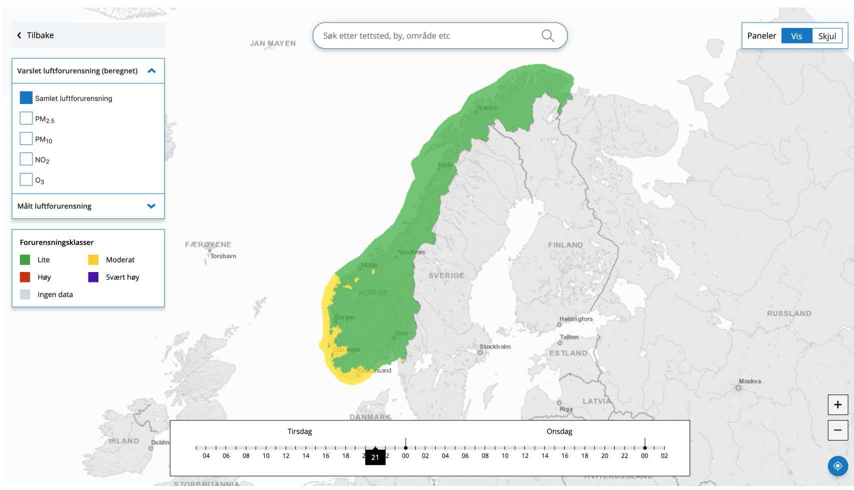

Figure 1.1: Luftkvalitet i NorgeLuftkvalitet i Norge (2019b)

The platform presents the current air quality levels of several cities and municipalities as well as the forecast for the next day. It includes the option of searching for several places in Norway and also has the option of navigating through a map. This map has a color coded transparent overlay for viewing the air pollution levels on top of the map. Under the map, there is an option to view and select the hourly predictions from 02.00 the current day and 48 hours forward in time.

1.2 Motivation

The thesis was given by Telenor with the goal of getting high accuracy prediction of air quality. The utilization of moving weather stations opens the possibility of increasing the coverage of the air pollution retrieval over large areas. The point being that moving weather stations are more optimal for giving a better picture of the air quality level, compared to a weather station fixed to a single position. The fixed weather stations will in turn yield more reliable data because of its relatively constant surroundings and situation.

A dynamic vehicle will be exposed to a large amount of uncertainty and unreliability. This is however a small price to pay compared to the value of great land coverage. Especially areas where setting up a static weather station is hard or impossible. A good prediction model and smart processing of the AQI data could be utilized for reducing the unwanted unreliability.

1.3 Problem Description

The vision of this project is to create an Air Pollution Prediction model that predicts a forecast of air pollution with great accuracy,. This is the first step in a large process, which ultimately should end up with a high quality online prediction platform, which covers all of Norway and only requires available and easy accessible data. The data should come from mobile weather stations mounted on vehicles and stationary weather stations fixed at a position.

This vision is something to aim towards, in the process of reaching the goal of this project. By working towards this vision, it propagates the thesis to reach the goal of this project, which is to create a model that is able to give high quality predictions based on the available data from stationary weather stations through the NILU API and the weather data available through Yr.

1.4 Contribution

This project aim to simplify the process of making good air pollution predictions all over Norway. This is done by using data from as few sources as possible so that its reliability is easy to maintain. By making it easier to give forecasts for air pollution it will also become more available for more people. And with the growing interest for IoT, smart gadgets capable of doing air quality meassurements, will perhaps be a common thing for people to own. An application of this project would be to use this data, combined with weather data, to predict the local air quality.

1.5 Research Question

Based on the problem description and the motivation for this project we have the following research questions:
1. What kind of machine learning model will give the best air pollution predictions?
2. What kind of factors leads to an accurate prediction model?
3. How can the results be validated and verified?

1.6 Report structure

This report consists of 9 chapters and 3 appendices. The chapters is divided into sections, and the appendencies consists of 3 different types of information. Appendix A contains plots from the results in chapter 7, appendix B consist of tables from the results in chapter 7, and appendix C is the relevant code used in this project.

Chapter 1 is the introduction of the thesis. It describes the background and motivation behind this thesis. Next chapter 2 is giving an overview of the problem and the theory that is being used as a base for this thesis. The third chapter 3 is about the datasets and presenting the data used in the project. Chapter 4 presents the preparation of the data this project is going to use to predict air polution data. The next chapter 5 describes and defines the prediction model that is going to be used to predict air quality. Chapter 6 describes the process of implementing the model. Next chapter 7 presents the results of the prediction model and shows tables and plots of the findings. In chapter 8 I discuss some of the findings I find interesting. And finaly, in chapter 9, I conclude the thesis and the project.

Chapter 2

Overview

2.1 Machine Learning Methods

This project will be using machine learning to predict air pollution data. This is achieved by using regression techniques through an artificial neural network. An artificial neural network is a network of nodes, designed for training and predicting values. An artificial neural network is inspired by on of natures greatest inventions, the brain. Our brain is one huge neural network, and an artificial neural network is mimicking the architecture and connections of neurons in the brain.

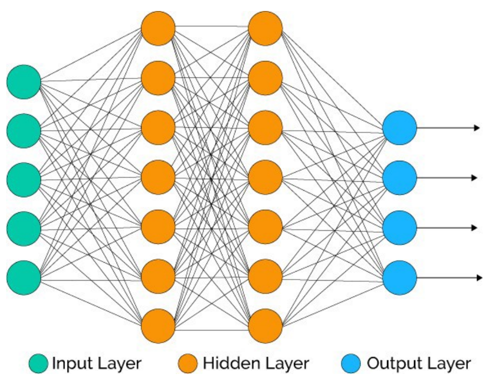

Figure 2.1: An example of an Artificail Neural Network Bhatia

The artificial neural network is trained by feeding data through the network via the input layer to the output layer and correct the error. This is done by adjusting the weights so the output will get more and more accurate by each training iteration Dormehl (2019).

2.2 Theory

2.2.1 Air Pollution

The air pollution regarded in this project is $PM_{2.5}$, PM_{10}, NO_2 and NO. These pollutants are the most common and interesting data to be predicted. These pollutants are categorized as either gasses or air particles. The pollutants effects the health of the human body, as well as the environment, ecosystems and vegetation (Luftkvalitet i Norge, 2019c).

The most common sources of pollution in Norway are the traffic on the roads. This is because the vehicles generates particulates and gasses through the exhaust of the combustion chamber, and also generate fine particulates through the wearing of e.g. the tires, the brakes and the asphalt on the ground.

Another important source of pollution in Norway is wood burning and combustion. The smoke generated from this process carries several types of air particles, including ultra fine particulates like $PM_{2.5}$.

Also, other sources of air pollution in Norway are industry, harbours and ships. This is regarded as likely contributors to the local air pollution. Ships often keep the engine going idle while being moored. This generates large amounts of pollution. The weather also plays its part in distributing the pollution, i.e. the wind carrying and spreading the pollution over greater distances.

Particulates

Particulates are often generated through combustion from e.g. traffic or industry, and carried by the whirling wind produced in e.g. traffic or from the nature itself.

The particulate matter $PM_{2.5}$ describes dust particles or particulates, with a diameter less than 2.5 μm. These fine grains can get into the lungs through the air we breathe.

Likewise, the particulate matter PM_{10} describe particulates with a diameter lower than 10 μm. These particles can often get to the upper airways.

Gasses

NO and NO_2, often called NO_x are gasses that are generated by high temperature combustion processes. These gasses is most often generated in traffic. When NO is in the presence of ozone, it converts to NO_2.

2.2.2 Formulae

Mean

$$\mu = \sum_{i=1}^{N} \frac{x_i}{N} \tag{2.1}$$

Standard Deviation

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \sigma)^2}{N-1}} \tag{2.2}$$

Normalization

$$z = \frac{x - \mu}{\sigma} \tag{2.3}$$

Prediction Accuracy

The prediction accuracy calculates the aggregate of the performance over all the classifications.

$$AUC = 1 - \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{\sum_{i=1}^{N} y_i} \tag{2.4}$$

Root Mean Square Error

Root mean square error is the standard deviation of the prediction error (Stephanie, 2016).

$$RMSE = \sqrt{\sum_{i=1}^{N} \frac{(\hat{y}_i - y_i)^2}{N}} \tag{2.5}$$

2.2.3 Equations

Sigmoid

$$S(x) = \frac{1}{1 - e^{-x}} \tag{2.6}$$

Sigmoid prime

$$S'(x) = S(x)(1 - S(x)) \tag{2.7}$$

ReLU

Rectified Linear Unit

$$f(x) = max(0, x) \tag{2.8}$$

ReLU prime

Rectified Linear Unit

$$f'(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases} \tag{2.9}$$

Chapter 3

Datasets

3.1 Introduction

One of the most important factors of a great model is quality data. The data should be true to the nature of what it represent. It doesn't matter if there is a large amount of data if the data is unreliable and unrepresentable. Therefore, it is crucial to retrieve the data needed for the prediction model from sources known for their reliability and quality.

Another important factor, as previously mentioned, is the amount of data. In the same way of thinking of the quality of the data, it is not sufficient to have small amounts of data even though it is of great quality. The broader the dataset is, the more datapoints is available, and therefore giving a better position for finding the optimal regression model.

To give it a human perspective, think of having a small amount of data as to not having i.e. visual sensibility, while having great hearing functions. Thus, the world is not represented by its full informational potential. However, it is fully possible to give accurate predictions of the surroundings.

In essence, for setting the best base for high quality predictions, large amounts of data of as high quality as possible is desired.

3.2 Air Quality Data

3.2.1 Tromsø

Initially the only dataset available for the project originated from Hansjordnesbukta Weather Station. This weather station provides AQIs consisting of PM_{10}, $PM_{2.5}$, NO, NO_2. These AQIs set the base line and constitute the prediction labels, the features to be predicted, for the project. The weather station is governed by NILU (2019b), the Norwegian Institute for Air Research, and is regarded as a reliable source of information and data. The project is heavily based on the data from this weather station, as the models were solely trained and fitted with data from Hansjordnesbukta.

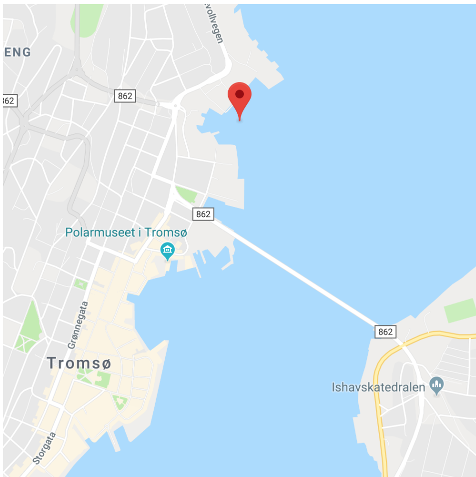

Figure 3.1: Hansjordnesbukta - (Google, 2019)

3.2.2 Other Areas

In the beginning of this project there was no easy way to retrieve AQI data other than through a password protected web interface from NILU. It was slow and often crashed when trying to download large quantities of data. However, when it did work, there was only need to download the data once. Another factor was the fact that only one area, Tromsø, was available for acquiring AQI data.

Now, NILU has provided an API (NILU, 2019a) for requesting historical AQI data from any of its stations here in Norway. That is why AQI data from other areas, e.g. Oslo or Bergen, was not used for training the air pollution prediction model. However, it was used for cross validation and test data, and is worth looking into in the future for fine tuning the precision of the model.

3.3 Weather Data

Yr is the joint online weather service from the Norwegian Meteorological Institute (met.no) and the Norwegian Broadcasting Corporation (NRK). Yr provides a unique offer of free weather data from all over the world (Yr, 2019b). The data retrieved from the Yr API included humidity, precipitation, temperature, wind and wind direction. All of the retrieved data included timestamps.

This data is the largest contributor to the training data and it is extremely important that the data is accurate. Fortunately, the data from Yr is regarded as reliable and precise. Also, the use of the weather API corresponds well with the NILU API, making it a natural choice for giving quick and accurate data for a possible future online prediction platform.

3.4 Traffic Data

A feature worth looking into was the traffic data from Vegvesenet. More precisely, the interesting data was the rate of traffic from a given road. The idea was to get the amount of traffic passing by within a given radius from the position of a given weather station at a given time. Unfortunately, the data that was available was not in a continuous form, but rather as a single integer value. It was only available, in some cases not, as the average daily traffic throughout a year. In many cases the traffic data was updated too many years in the past to be considered qualified for describing the traffic of the current year. In some cases the data was last updated e.g. the year 2005 in Oslo.

The traffic data was not used in this project, as it was too unreliable. However, it may be useful for future training of an improved version of the prediction model as feature indicating how much traffic a given area could expect.

Chapter 4

Data preparation

4.1 Overview

This chapter will present the process of preparing the data from the AQI dataset 3.2.1 and the weather dataset 3.3 for training and fitting the model.

As previously discussed in section 3.2.1, the AQI data used for the fitting and training of the model is from Hansjordnesbukta in Tromsø. The weather data is from Yr and the time data is extracted from both the AQI and the Weather data. The data is represented by hourly measurements in the range from January 1st 2010 to December 31st 2018. This gives 71,248 datapoints, which is not far from the expected value of 78,864 datapoints. This indicates some gaps in the dataset and is not regarded as an issue. The gaps will simply vanish after the data completely processed.

Yr	**NILU**
Timestamp	Timestamp
Humidity	$PM_{2.5}$
Precipitation	PM_{10}
Temperature	NO
Wind	NO_2
Wind direction	

Table 4.1: Features

4.2 Merging

The first step was to combine the data from the different datasets together. As will be discussed in Chapter 5, the model needs certain data from the dataset. As presented in table 4.1, each dataset has a timestamp asosciated with each datapoint. The timestamp

Timestamp			Yr Weather Data					NILU AQI Data			
Month	Day	Hour	humidity	precipitation	temperature	wind	wind direction	PM_{10}	$PM_{2.5}$	NO_2	NO

Table 4.2: Merged Dataset

functions as an unique id for each datapoint. With this in mind, the data from each dataset can be merged into a new dataset by matching the timestamps as the index.

The features in the merged dataset 4.2 are humidity, precipitation, temperature, wind, wind direction PM_{10}, $PM_{2.5}$, NO and NO_2. In addition to these features, the model requires the following features: Month, Day of the month and hour of the day. These features are extracted from the timetamp index.

4.3 Invalid measurements

Now a merged and complete dataset is created and it needs to be cleaned. First, the datapoints, that are either invalid or whose values makes no sense, needs to be processed. The data that are invalid will be assumed to have a value of NaN. These datapoints will eventually be removed and are for now regarded as important to keep. This is because the data eventually will be sequenced. Therefore, for the simplicity of the process, the NaN data needs to remain in the dataset.

The datapoints that make no sense are when e.g. a percentage value is negative when the value really should be between 0 and 100, i.e. a humidity value. Another value type that makes no sense are negative AQI values. An air particle 2.2.1 or a gass 2.2.1 cannot have a negative value. In fact, the temperature feature is the only feature that can have a negative value. Therefore, these datapoints will be assumed as NaN and will remain in the dataset, just like the invalid data.

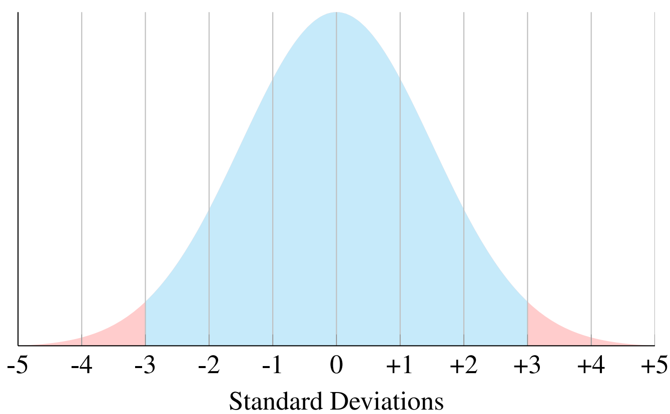

Figure 4.1: Normal Distribution (Sebastiano, 2017)

4.4 Outliers

Next, the outliers needs to be invalidated. An outlier is a datapoint that lies far away from the other datapoints. In this project, an outlier is classified as a datapoint that lies more that three standard deviations 2.2.2 away from the mean value 2.2.2 in the normal distribution curve. This creates a range that says which data is valid. All the data that are less than the mean minus three standard deviations and all the data that are more than the mean value plus three standard deviations will be invalidated 4.1.

4.5 Label Encoding

All of the data is continuous numerical values, except the wind direction data, which is discrete string values. The wind direction data is thus required to be encoded to numerical values, as the model only works with numbers. Label encoding means representing the label, the strings, as a unique number for each unique string. There are 16 unique wind directions in the data set. The wind direction labels are mapped to a value ranging from 1 to 16.

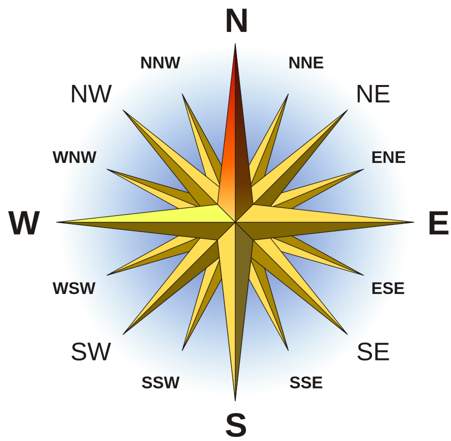

Figure 4.2: 16 Point Compass Rose - (I, Andrew pmk, 2007)

4.6 Normalization

Every value in the dataset is numerical. However, it is not normalized. Each column in the dataset are in different scales e.g. humidity which ranges from 0 to 100 percent and wind direction which ranges from 1 to 16. The model needs to be trained on data which is balanced and at the same scale. This is because the model minimizes its loss by calculating the gradient descent and move towards the local minimum. If the values are not on the same scale, the gradient will oscilate greatly and then take long time in locating the local minimum value. Ideally, all the features should have values ranging around -1 to around +1, the mean of each feature should be equal 0 and the standard deviation for each feature should be 1. This is achieved by normalizing the dataset, feature by feature, by using the normalization equation 2.2.2.

4.7 Splitting

The architecture of the model requires the data to be split in certain groups and shapes, which will be presented and discussed in section 5.1. Currently, the dataset is normalized and its shape is unchanged. From this point, the data is split up into five seperate datasets.

Time	Historical Weather	Weather Forecast	Other Pollutants	Prediction AQI
Month	Precipitation	Precipitation	Historical AQI 2	AQI 1
Day	Temperature	Temperature	Historical AQI 3	
Hour	Wind	Wind	Historical AQI 4	
	Wind Direction	Wind Direction		
	Humididity			

Table 4.3: The new sub datasets derived from the original dataset.

The historical weather dataset is going to represent the past weather values based on how many hours the model is set to look back. The features included are humidity, precipitation, temperature, wind strength and wind direction.

The weather forecast dataset is the future weather that is most likely to occur in the future. In this case, the dataset already contains a weather forecast, or rather the actual weather that occured some given hours from a given time. The features in this dataset are precipitation, temperature, wind strength and wind direction. The reason for excluding humidity as a feature is because Yr does not predict the humidity in its weather forecast. Since this model is heavily reliant on the Yr data, it seemed logical to adapt their structure onto this model. This is a proactive decision made to make it possible to create a tool for the model in the future 9.2.

In the next dataset, we find the other pollutants. This dataset consists of the labels, the AQIs, that are not to be predicted. The model is only capable of predicting one label at the time, but the other labels can still be useful. The air pollutants, the AQIs, is most likely to be influenced by each other, i.e. the AQI label NO should be influencing the NO_2, because, when in presence of ozone, NO converts to NO_2 2.2.1.

The time dataset is representing the meta data from each datapoint. This data should reflect what season, what time of the week and what time of the day the AQI data was recorded. Certainly, it is intuitive to think that a recorded AQI value will be greater at rush hour mid day on a Tuesday, compared to a value recorded at a late Sunday evening. Also, seasons seem to have an impact on the air pollution. In the winter, it is often more articles about the thick layer of air pollution hovering in the air near traffic, probably caused by increased heating. However, this would also be reflected by the temperature and the precipitation, indicating the typical weather of each season.

Last, the Prediction AQI dataset contains the prediction feature, the AQI to be predicted. This dataset is made up by the previous recorded AQI data over a given range of time in the past from a given time. The datapoints indicate a pattern of behaviour for the AQI over time. This dataset would fit nicely into a regression model.

4.8 Sequencing

The new datasets still includes the invalidated NaN values. This is done to retain the hourly ratio between each datapoint. When sequencing the data, the NaN values will be regarded as valid until the last step, which is to remove any datapoints containing NaN values in any of its columns.

Date	month	day	hour	humidity	precipitation	temperature	wind	wind_from	NO	NO2	PM10	PM2.5
2010-11-30 08:00:00+01:00	11	30	8	69	0.0	-10.8	2.9	northeast	241.7	125.7	18.3	12.1
2010-11-30 09:00:00+01:00	11	30	9	73	0.0	-11.2	1.4	north	237.6	119.3	20.1	12.2
2010-11-30 10:00:00+01:00	11	30	10	74	0.0	-11.3	0.6	west	NaN	NaN	24.8	15.7
2010-11-30 11:00:00+01:00	11	30	11	69	0.0	-10.0	1.0	south-southeast	NaN	NaN	29.0	17.9
2010-11-30 12:00:00+01:00	11	30	12	62	0.0	-8.6	2.5	north-northeast	287.7	146.1	27.2	14.7
2010-11-30 13:00:00+01:00	11	30	13	61	0.0	-8.2	2.3	north-northeast	258.7	135.1	22.6	10.6
2010-11-30 14:00:00+01:00	11	30	14	61	0.0	-8.2	3.1	north-northeast	261.2	124.5	19.0	10.2
2010-11-30 15:00:00+01:00	11	30	15	64	0.0	-8.6	3.1	north-northeast	151.9	92.3	9.7	5.5
2010-11-30 16:00:00+01:00	11	30	16	66	0.0	-9.6	2.7	northeast	94.4	79.1	9.1	6.0
2010-11-30 17:00:00+01:00	11	30	17	67	0.0	-9.9	1.4	northeast	122.4	91.8	12.8	8.2
2010-11-30 18:00:00+01:00	11	30	18	73	0.0	-11.4	2.7	northeast	110.9	86.1	15.4	10.4
2010-11-30 19:00:00+01:00	11	30	19	74	0.0	-12.1	3.1	north-northeast	61.3	68.3	16.3	13.1
2010-11-30 20:00:00+01:00	11	30	20	75	0.0	-12.5	1.0	northeast	91.2	78.1	21.7	15.0
2010-11-30 21:00:00+01:00	11	30	21	82	0.0	-13.1	0.3	north	74.5	76.0	23.9	17.7
2010-11-30 22:00:00+01:00	11	30	22	80	0.0	-13.4	0.9	east-northeast	70.0	72.9	29.8	22.8
2010-11-30 23:00:00+01:00	11	30	23	78	0.0	-13.7	1.1	northeast	57.0	65.6	45.3	35.4

Figure 4.3: A view of the data with the shape of the "window"

Date	month	day	hour	humidity	precipitation	temperature	wind	wind_from	NO	NO2	PM10	PM2.5
2010-11-30 08:00:00+01:00	11	30	8	69	0.0	-10.8	2.9	northeast	241.7	125.7	18.3	12.1
2010-11-30 09:00:00+01:00	11	30	9	73	0.0	-11.2	1.4	north	237.6	119.3	20.1	12.2
2010-11-30 10:00:00+01:00	11	30	10	74	0.0	-11.3	0.6	west	NaN	NaN	24.8	15.7
2010-11-30 11:00:00+01:00	11	30	11	69	0.0	-10.0	1.0	south-southeast	NaN	NaN	29.0	17.9
2010-11-30 12:00:00+01:00	11	30	12	62	0.0	-8.6	2.5	north-northeast	287.7	146.1	27.2	14.7
2010-11-30 13:00:00+01:00	11	30	13	61	0.0	-8.2	2.3	north-northeast	258.7	135.1	22.6	10.6
2010-11-30 14:00:00+01:00	11	30	14	61	0.0	-8.2	3.1	north-northeast	261.2	124.5	19.0	10.2
2010-11-30 15:00:00+01:00	11	30	15	64	0.0	-8.6	3.1	north-northeast	151.9	92.3	9.7	5.5
2010-11-30 16:00:00+01:00	11	30	16	66	0.0	-9.6	2.7	northeast	94.4	79.1	9.1	6.0
2010-11-30 17:00:00+01:00	11	30	17	67	0.0	-9.9	1.4	northeast	122.4	91.8	12.8	8.2
2010-11-30 18:00:00+01:00	11	30	18	73	0.0	-11.4	2.7	northeast	110.9	86.1	15.4	10.4
2010-11-30 19:00:00+01:00	11	30	19	74	0.0	-12.1	3.1	north-northeast	61.3	68.3	16.3	13.1
2010-11-30 20:00:00+01:00	11	30	20	75	0.0	-12.5	1.0	northeast	91.2	78.1	21.7	15.0
2010-11-30 21:00:00+01:00	11	30	21	82	0.0	-13.1	0.3	north	74.5	76.0	23.9	17.7
2010-11-30 22:00:00+01:00	11	30	22	80	0.0	-13.4	0.9	east-northeast	70.0	72.9	29.8	22.8
2010-11-30 23:00:00+01:00	11	30	23	78	0.0	-13.7	1.1	northeast	57.0	65.6	45.3	35.4

Figure 4.4: Moving the "window" to the next index

The five new datasets is currently two-dimensional, i.e. shape of (datapoints, features). In this step, the datasets are converted to be three-dimensional with the shape of (datapoints, hours to look back, features) or (datapoints, hours to predict, features), depending on what type of data it is and what it will be used for in the model.

The process of converting the current flat datasets can be visualized as moving a sliding window over the dataset and keep the data if the all of the values is not NaN as seen in figure 4.3 and figure 4.4. Here the process starts at the fifth datapoint because there are some NaN values in one of the columns. The green area indicates the area that has to be valid in order to be included in the new data set. The smaller boxes in the window are each a sequence. A datapoint in the new and final dataset.

In this project, the data used to train the prediction model was based on a value of six hours to look back, and a range from 1 to 3 hours, plus 12, 24 and 48 hours to predict. After sequencing the data for each prediction hour value, the datasets is combined into a new dataset. From this point the data has reached its final shape. This results in six datasets, one for each prediction hour. In this project, there will be trained models for each AQI feature. Finally, this sums up to the total of 24 datasets.

4.9 Padding

The Weather prediction data needs to be padded to fit the input shape of the model, or vice versa depending on what shape is larger. If axis 1 of the datasets is not eqaul, they will not concatenate, i.e. the shape (datapoints, 3, 4) for the weather forecast data will not concatenate with the shape (datapoints, 6, 1) for the historical AQI data when it is given as an input to the model. The padding value is zero, and will be appended to the data. The values in axis 0 and 2 is irrelevant for the concatenation. The data with the lowest value in axis 1 will be padded to match the value of axis 1 to the data with the largest value in axis 1.

Chapter 5

Prediction Model

5.1 Overview

The prediction model is a fully connected neural network, with five distributed sub networks, inspired by Yi et al. (2018). It is designed to capture feature data from different domains, i.e. weather data, air pollutants and time, and make predictions of the future air quality data.

This model will not take location data or AQI data from other nearby stations as input, making the model solely dependent on data from a single weather station, which in this project is Hansjordnesbukta. The reason behind this is because at the time the project started, this was the only available data. However, further into the project, more data became available and was used as validation data for the model.

The idea of a distributed network comes from the the fact that the different domains influences the air quality data, each in its own ways. To capture this, the model several different sub networks that each outputs a prediction, which is combined to form a final air quality prediction.

5.2 Architecture

5.2.1 Overview

The model is built up by a combination of five different sub networks and one output layer. Each sub network needs an input, and the inputs vary in shape, however, axis 0 is required to be equal for all the inputs. This is how many datapoints the model is going to fit or predict. All of the networks fully connected layers, except the last, is activated with a Sigmoid function 2.2.3.

The fully connected layers is then appended with a dropout layer with value 0.2. These dropout layers will set 20% percent of its input values to be equal to zero. This is done to regularize the model and will help reduce overfitting.

The prediction outputs from each sub net are combined through a linear merge into a fully connected layer before flattening the tensors into a final output with the shape (N, hours to predict) 5.2.

Each sub network has the same architecture, except from the input layer. Also, each subnet share a common input dataset, which is the Historical AQI dataset. This is the most feature that is going to be predicted, and is combined with the other datasets in the sub networks to give distributed predictions.

The holistic subnet, shares input with all the other subnets, as this subnet is designed for capturing the whole influence of all the data. In short, there are four subnets designed for capturing the isolated influence of time, other AQIs values, historical weather values or weather forecast values, on the prediction AQI. The last sub net is designed for capturing all the combined influences on the prediction AQI in the same system. The five outputs from these subnets are combined for giving a distributed prediction.

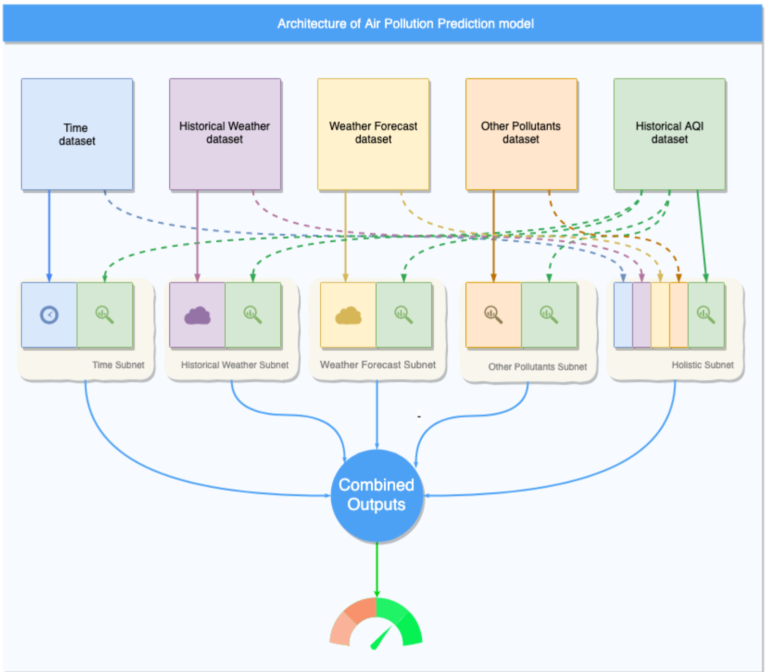

Figure 5.1: An overview of the mMdel Architecture

5.2.2 Sub Networks

Time Subnet

The Time Subnet gets its inputs from the Time Dataset and the Historical AQI dataset. The features are prediction AQI, Month, Day of Month and Hour. This subnet is designed for capturing the influence of time on the AQI to be predicted, by inputing the past values of the AQI.

The idea of this subnet comes from the fact that seasons and time of day have an indirect influence on air pollution. I.e. in the winter, it usually gets much colder than in the summer in Norway, which leads to more wood fire and heating. Also, the time of day tells when there are more activity in the traffic, i.e. the morning rush to work or the long lines of traffic on the way home from work.

Historical Weather Subnet

The Historical Weather subnet get its inputs from the Historical Weather Dataset and the Historical AQI dataset. Its input features are Precipitation, Temperature, Wind, Wind direction, Humidity and the prediction AQI. This subnet is designed for capturing the influence of the historical weather on the historical AQI. This subnet outputs the prediction on the AQI based on the previous weather conditions and the prediction AQI values, in a series some hours long.

Weather Forecast Subnet

The Weather Forecast subnet get its inputs from the Weather Forecast Dataset and the Historical AQI dataset. Its input features are Precipitation, Temperature, Wind, Wind direction and the prediction AQI. This subnet is designed for capturing the influence of the future weather on the historical AQI.

Like the previous subnet, this subnet combines weather data and air pollution data. However, the weather data is not a precise value. It comes with uncertainty as it is a prediction of the future value of the weather. Although, in training, the weather forecast data is actually the future recorded weather data and not a real prediction.

The shape of the data is equal to the shape of the prediction shape in axis 1, i.e. the hours to predict. If the inputs to this subnet is unequal in axis 1, it will be required to be padded 4.9.

Other Pollutants Subnet

This subnet gets all of the AQI types as input. The AQIs data has the shape of (N, hours to look back, 4) where there are four AQI labels. The reason for this subnet is to capture the influences the other AQI has on the prediction AQI. There seem to be a correlation between the rise in the other AQIs and the rise in the prediction AQI, i.e. NO converting to NO_2 2.2.1.

Holistic Subnet

Our final subnet is designed to capture all the influence of each type of data combined. The subnet has input from all the datasets which includes all the features in table 4.3.

5.2.3 Data Features

The data used for fitting this model comes from one single weather station, as previously mentioned. The model has been validated on data from other weather stations as well as the validation data from this single weather station 7.

Training data

The training data for this model came from the 80% first datapoints from the 5 datasets. All the data is historical, except the weather forecast. The prediction AQI is split in two where the training part is the values in the range from hours to look back to current hour from the past. The target data is extracted from the Prediction AQI dataset in the range from next hour to hours to predict. After the split, the training data is shuffled and fit into the model.

Validation data

The validation data comes from the last 20% datapoints of the 5 datasets. It has the same shape of the training data other than axis 0.

5.2.4 Activation functions

In this model, all of the activation functions in the network layers are a sigmoid function 2.2.3. Even thouugh the ReLU 2.2.3 is much more effective and faster than a sigmoid activation(Wikipedia contributors, 2019), after a series of tests, the sigmoid function seems to give the most accurate predictions. In the beginning, the ReLU model learns quickly, but stagnates the learning after a short amount of epochs. The sigmoid improves more in the long run.

5.2.5 Optimization Algorithm

The optimization algorithm for this model is an Adam optimizer (Jason Brownlee , 2017), which is a highly effective and popular algorithm for optimizing the training of a model.

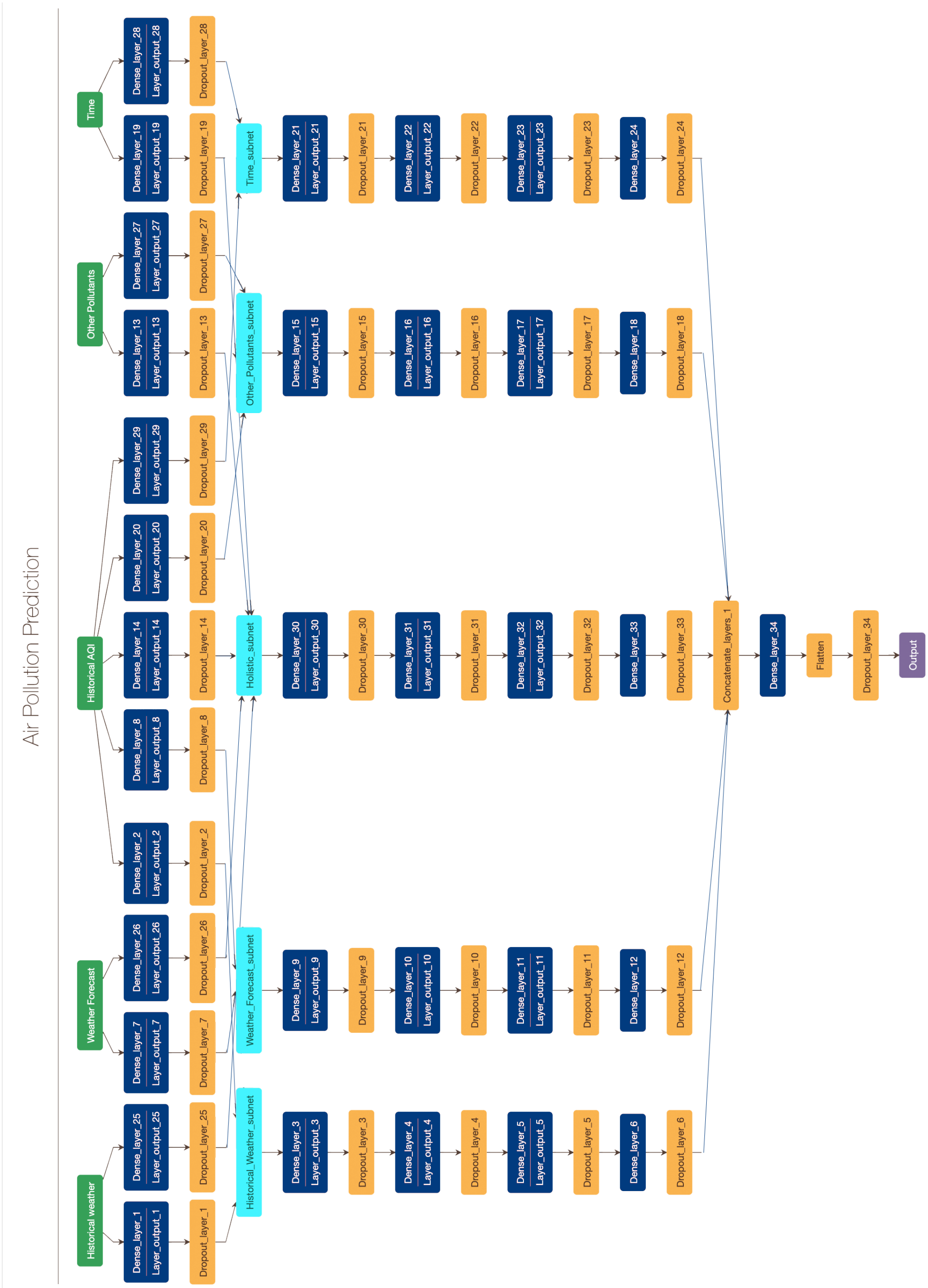

Figure 5.2: A detailed view of the Model Architecture Ethereon

Chapter 6

Implementation

6.1 Environment

In this project, the work was made on my local computer. Then, the project was moved to a server hosted by Telenor. In the end, the project was moved to a Google Cloud server as the access to the Telenor server was no longer available.

6.1.1 Python

All of the code for retrieving the data, processing the data, defining the model and plotting the result was made in Pyhon 3.5.3 (Rossum, 1995) on a Jupyter Notebook (Kluyver et al., 2016) server. Python is an amazing and easy programming language syntax wise. It is highly documented and has access to many good quality libraries, such as Numpy (Oliphant, 2006–) and Pandas (McKinney, 2010) which is great for managing, analysing and processing data.

The main reason for choosing Python as the programming language was the simplicity of the language, the effectiveness and the large community behind it. There are always an answer to a problem that was encountered. Also, the ease of use with large datasets and the easy use of libraries was a huge factor in choosing the language.

Some of the libraries that was attractive was Keras (Chollet et al., 2015), with Tensorflow (Abadi et al., 2015) as backend, Numpy and Pandas. These libraries was essential for this project to be finnished. Also, Python is a very nice language for retrieving data from APIs on the internett, i.e. Yr for weather data and Meteorologisk Institutt air pollution data.

6.1.2 Tensorflow

Tensorflow is a symbolic math library, with support for Python, and is most known for its use in machine learning, i.e. making neural networks. It is developed and researched by Google, by their research team in Google Brain.

Tensorflow has a broad community and there are lots of documentation on the internett. However, it is known for beeing hard and complicated to use. A lot of valuable time and efforts would be spent in learning Tensorflow good enough to make a great prediction model.

6.1.3 Keras

In this project, Keras was used to define the Air Quality Prediction model. Keras is an API built on top of Tensorflow, which is used in this project, and other machine learning libraries. It makes defining models and training them, as well as giving predictions, easy. With this, a lot of time was saved on using this high-level API. Making prototypes and testing them was extremely easy and fast. Ideal for research.

6.2 Google Cloud Computing

The coding and execution of code was done on a Jupyter Notebook server on a Google Cloud Virtual Machine instance. This made it possible to run at faster speeds as the local home computer was too slow and loud to keep running for several days in a row.

Also the service was easy to use regarding customization and tweaking of the system preferences. To upgrade CPU power and memory was as easy as adjusting some settings. The service also recommended new settings when it noticed the VM instance was running slow.

The use of Google Cloud Computing (Krishnan and Gonzalez, 2015) was possible due to the free trial with included credits free to use. Unfortunately, GPU support was not enabled for trial users. Therefore, it was not possible to utilize the power of GPU during the computation and fitting of the air quality prediction models.

6.3 Code

In this project, a lot of code was written. Much of the code was for the retrieval and the processing of the data used in this project. The rest of the code was for defining the model and plotting the results. All the relevant code for retrieving and processing data, and fitting the model with the data is in appendix C.

6.3.1 Retrieving the data

For retrieving the data, one API and one script I provided by Telenor was used. The script was for retrieving historical weather data from Yr.no and the API was used for retrieving historical air pollution data from Met.no in the python script II. The Yr script covered, in general, all of Norway. The Met.no API covered many cities in Norway, but not the entire country.

The weather data from Yr was retrieved by iterating through each date between a given start date and a given end date and extracting the wanted weather data to a dataframe. The dataframe was then stored in a file to prevent the need to repeat the process. Historical data will always be the same any time it is downloaded, so it only needs to be downloaded once. The same process was done for the air pollution data, as also this was historical data.

6.3.2 Processing the data

The processing of data went smooth due to the ease of use of Numpy and Pandas. The data processing is partially in the python script III in functions load_seq_data and clean_data, and in the function fix_data in the python script V.

6.3.3 Defining model

The model is defined as the function dense_model in the python script IV. It takes training input and training output, as well as the activation function, as input. With keras, it is very easy to prototype and make models quickly.

Chapter 7

Results

7.1 Feature Importance

The feature importance was calculated by first estimating the original prediction error, which is done by calculating the root mean square error 2.2.2. Then, for each feature of the model the following calculations was performed:

1. Recreate the feature matrix with random values.
2. Estimate the prediction error of this random matrix.
3. Calculate the feature importance by dividing the this prediction error with the original prediction error.

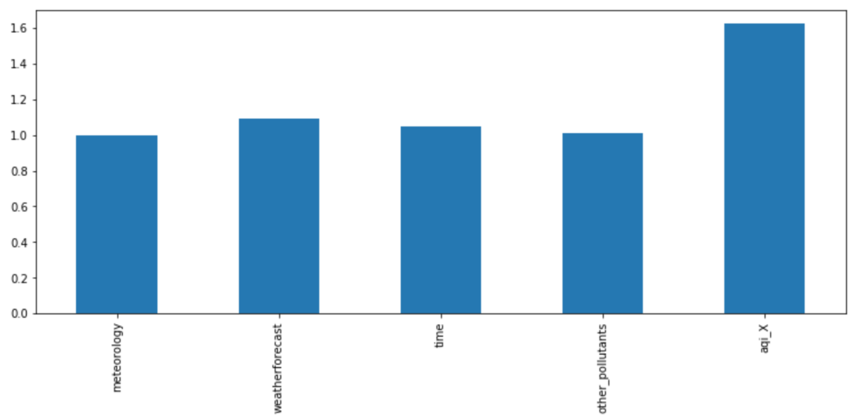

Figure 7.1: Feature importance

After fitting all of the models the most important features 7.1 was the historical AQI data. This is somewhat expected as the dataset is an input for all the sub networks in the prediction model.

7.2 Scoring

In this chapter, the label to be inspected is $PM_{2.5}$, which reduces the amount of models to be inspected to six. The results from the other AQIs is presented in the Appendix A.

We define the scoring metrics for the model in section 2.2.2. The accuracy metric AUC and the error metric RMSE. The RMSE tells us how many units the prediction is from the true value, and the accuracy AUC is the sum of performance on all the output features.

Now, all of the 24 models have been fitted with data from Tromsø. The results of the validating the Tromsø data is presented in table 7.1 and plotted in figure 7.2 and 7.3.

PM2.5	1	2	3	4	5	6	...	44	45	46	47	48
1	0.8190						...					
2	0.8191	0.7877					...					
3	0.8200	0.7914	0.7730				...					
12	0.8209	0.7985	0.7853	0.7778	0.7711	0.7695	...					
24	0.8134	0.7938	0.7825	0.7734	0.7675	0.7637	...					
48	0.7987	0.7730	0.7575	0.7492	0.7439	0.7419	...	0.7289	0.7275	0.7284	0.7288	0.7241

Table 7.1: Accuracy for models 1h, 2h, 3h, 12h, 24h and 48h on Hansjordnesbukta data for AQI $PM_{2.5}$

The results with 82% accuracy are indeed very impressive. This is a strong indicator that our model works as it was meant to. Also, look at how little the accuracy falls as the hours to predict gets larger. This indicates that the model has learned well and that our architecture is working well.

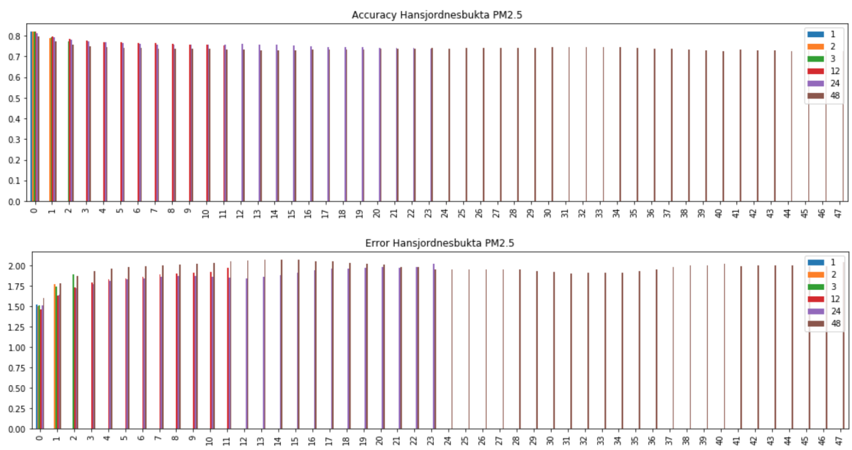

Figure 7.2: Accuracy and error on data from Hansjordnesbukta, Tromsø

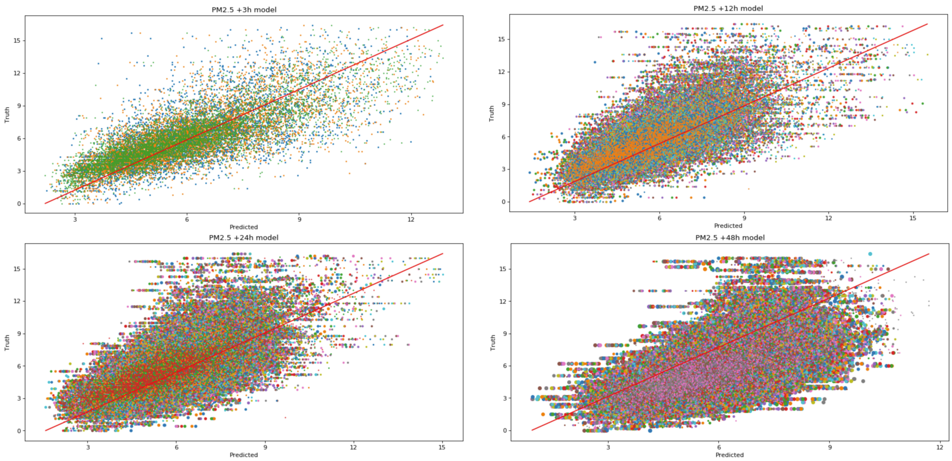

Figure 7.3: Scatter plot of selected models on data from Hansjordnesbukta, Tromsø for AQI $PM_{2.5}$

To look at how general the model is a validation on the data from Oslo was made to give an indication on how accurate it predict on data from other locations. We will use all the $PM_{2.5}$ AQI data from all of the weather stations in Oslo combined. It is interesting to see how the model performs on this data. Keep in mind that this model is only trained on weather data from Tromsø. We can see from A.2 that the

Using all of the AQI data from Oslo on the models to predict, the resulting accuracy of the models are very close to each other A.2. From the plot A.3 and A.4 it is clear that the spread is wide. However, that would be expected as the predictions become more and more inaccurate when predicting further into the future.

	0	**1**	**2**	**3**	**4**	...	**43**	**44**	**45**	**46**	**47**
1h	0.785266					...					
2h	0.783809	0.716501				...					
3h	0.787370	0.720358	0.671145			...					
12h	0.748812	0.698694	0.668034	0.648348	0.637171	...					
24h	**0.793599**	**0.752586**	**0.725421**	**0.706648**	**0.694303**	...					
48h	0.790442	0.747041	0.718389	0.698438	0.683190	...	**0.622065**	**0.62631**	**0.626092**	**0.624261**	**0.624574**

Table 7.2: Accuracy for models 1h, 2h, 3h, 12h, 24h and 48h on all $PM_{2.5}$ data.

The longest models are almost as good, or sometimes better than the short models. This is a nice result, in that there is only need for one model per AQI label. That is, one model with 24 or 48 hours to predict, give just as good, or sometimes better predictions as the short ones, and the long models output can be sliced to fit the shape of a desired output.

If there one day is need for e.g. a 1 to 24 hour prediction, or a 12 to 16 hour prediction, all this can be achieved by slicing the output array. This is a very interesting result, with regards to a possible future web platform project.

7.3 Comparison

With regard to the paper from Yi et al., this projects results is not comparable to the result of their paper as this projects model is based on data from one weather station without the influence of other weather stations. Also the results from the paper is based on data from another location and situation than here in Norway.

The application of this model could be i.e. on the mobile weather stations in Trondheim that won't need other weather stations to be able to give air pollution predictions. Unfortunately, this project cannot be compared with the Beijing data predictions, as it was not trained in that climate. The comparison would not be a correct or natural comparison.

Chapter 8

Discussion

8.1 Model

8.1.1 Model type

This project ended up with choosing a fully connected neural network. Initially, the idea was to implement a recurrent neural network. However, this resulted in slow learning and poor results. It was not ideal to use much time on training bad networks instead of prototyping and testing quicker networks. This was a problem because of the limitations of hardware at hand. If the project was built on an environment with a CUDA GPU, the computation time would drastically decrease and more time to develop prototypes would increase.

First, the idea of a Long Short Term Memory (LSTM) network came to life because of the data used in this project was in the forma of a time series sequence. Unfortunately, it was no success in prototyping and testing. The same goes for a GRU network. To save the project some time and problem solving, a more familiar model type was used. Also, the fully connected type showed more promising results in the testing phase of the project.

8.1.2 Model Structure

The structure of the model is not in a perfect state and the could always be made improvements and further experimentation on the model structure. There could be more layers in the subnets, or in the concatenation junction. Since the project got hands on a whole lot more data this late in the project, there was little time to consider this data to be used in training. Regardless, the amount of data now available does not match the size of the model. On the other hand, the system the project is developed on, does not work ideally with large models and many parameters.

One thing that would be interesting to look further into would be the use of recurrent neural networks, in some or all of the sub networks. Now that all this data is available, there is enough data to cover the amount of parameters it would have required. Ideally one wants to have about half the number of parameters as one has datapoints. Also, the fact that we now know that there is only need for 4 models, i.e. one model for each AQI where hours to look back is equal to six or more and hours to predict is equal to 48.

Overall, the model is close to great. However, it could still be tweaked and adjusted to get closer to perfection, not saying it is anywhere near perfection. Still, it does provide good predictions.

8.1.3 Features

A thought throughout the project was regarding the features, and which to choose as prediction AQIs. An idea would be to find a way to predict all AQIs, e.g. NOx or SO3, at the same time to further reduce the unique models to fit, down to one single model, i.e. one model able to predict any label based on six or more look back hours and 48 hours to predict.

Other features considered was getting live and historical traffic data. This would possibly help the model in capturing the influence on the air quality, based on the amount of traffic near a location at any time of day. This was attempted through Vegvesenet, but it proved to be unreliable and low quality and non continous. It could be interesting to use data from the ship traffic at harbours or public transport traffic.

One feature that was activelly looked after and pursued was data from events given a time and position. E.g. if there was a marked downtown, one would think that the amount of people at that place and time would increase, and therefore the traffic leading to the place and time would increase. Also other special days or hollidays could be recorded and used as an extra feature in the Time dataset.

8.1.4 Outputs

One thought of the project is whether it is necessary with this many models to predict the future air quality, or maybe there is only need for one model taking any AQI as input through an embedding and predicting up to 48 hours of air quality.

8.1.5 Limitations

An obvious limitation for the practical use of the model is the requirement of features. The model needs many features to be able to give a prediction. This limits the amount of weather stations the model will work on, for now, as it only works with four AQIs, no more or less. This could be solved with an embedding solution or zero filled vectors instead of a given AQI when that AQI is unavailable.

8.2 Work environment

The equipment was varying in reliability, power and accessibility. Therefore, I ended up using google cloud computing developing the model and for training the data. Unfortunately, i was unable to access any GPUs and only got to compute on some CPUs. Also, movig the project from server to server costed the project some time, as internet connection and data transfer were slow.

8.3 Work distribution

About 55pct of the project time was spent on processing and formatting the data. This was a long project with a lot of trials and fails. Also, the equipment was not always on the helpful side resulting in time being spent in fixing unrelevant errors.
Around 25pct of the time was spent on finding a model that gave promising results. A lot of trial and error and experimentation was performed to find the model that seemed to work well with the data at at hand. When the data also changed from time to time, the model needed update too. Therefore, there was a symbiotic process in processing the data and prototyping models.
10 pct of the time was spent on retrieving and merging the data from APIs and scripts. The script i received from Telenor was not retrieving every feature that was needed, so some time was needed to alter the script.
Finally, the last 10 pct was spent on writing this thesis and plotting data to present the findings in this project.

Chapter 9

Conclusion

9.1 Experience

In this project, I have learned to conduct research and to explore the process of finding my own answers to real life challenges. It has been a challenging, but interesting task and I have learned a lot about machine learning, deep neural networks and naturally, air pollution.

9.2 Future work

9.2.1 Features to include

In the future, I reckon that the model will be able to accept any AQI feature and also be able to predict any AQI feature. Maybe there is an idea looking into spatial transformation, capturing the influence from other weather stations. However, I like the fact that the current model is not reliant on other weather stations to work.

9.2.2 Training data

The model should be trained on other than only Tromsø data. Now, with access to more data from all over Norway, it would be wise to fit the model with data other than just Tromsø. Also, more data is not bad and always welcome to a machine learning problem.

9.2.3 Web Platform

This model can be utilized on a web platform to give live air pollution forecast on the active stations in Norway from 1 hour to 48 hours in the future, making it interactive and open for integration with mobile weather stations for live updating of air quality data and forecasting. The platform could have a map with a graphic overlay indicating the air pollution level and have a slider to view the forecast by up to 48 hours.

9.3 Bottom line

Finally, this project has resulted in a success. A model able to predict air pollution with 79 percent accuracy for the next hour and 62 percent accuracy in predicting air pollution in 48 hours is quite impressive. If this project could eventually be used to make the world a better place, I would have succeeded.

Bibliography

Jason Brownlee , 2017. Gentle introduction to the adam optimization algorithm for deep learning.
URL https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
URL http://tensorflow.org/

Bhatia, R., 2018. When not to use neural networks.
URL https://medium.com/datadriveninvestor/when-not-to-use-neural-networks-89fb5062242

Chollet, F., et al., 2015. Keras. https://keras.io.

Dormehl, L., 2019. What is an artificial neural network?
URL https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/

Ethereon, 2018. Netscope.
URL https://github.com/ethereon/netscope

Google, 2019. Hansjordnesbukta - google map location.
URL https://www.google.no/maps/place/Hansjordnesbukta/@69.6566867,18.952272,14z/data=!3m1!4b1!4m5!3m4!1s0x45c4c4512d0f8f1f:0xa9f04460ae7d3ec0!8m2!3d69.6566879!4d18.9697815

I, Andrew pmk, 2007. 16 point compass rose. Created 16 June 2007.
URL https://en.wikipedia.org/wiki/Points_of_the_compass#/media/File:Compass_Rose_English_North.svg

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., 2016. Jupyter notebooks – a publishing format for reproducible computational workflows.

Krishnan, S. T., Gonzalez, J. U., 2015. Building your next big thing with google cloud platform: A guide for developers and enterprise architects.

Luftkvalitet i Norge, 2019a. Aqi forecast.
URL https://luftkvalitet.miljostatus.no/

Luftkvalitet i Norge, 2019b. Aqi forecast explained.
URL https://luftkvalitet.miljostatus.no/artikkel/581

Luftkvalitet i Norge, 2019c. Lokal luftforurensning.
URL https://luftkvalitet.miljostatus.no/artikkel/170

McKinney, W., 2010. Data structures for statistical computing in python.

met.no, 2019. The norwegian meteorological institute.
URL http://met.no/

NILU, 2019a. Nilu – api.
URL https://api.nilu.no/docs/

NILU, 2019b. Nilu – norwegian institute for air research.
URL https://www.nilu.no/en/about-nilu/

Oliphant, T., 2006–. NumPy: A guide to NumPy. USA: Trelgol Publishing, [Online; accessed <today>].
URL http://www.numpy.org/

Rossum, G., 1995. Python reference manual.

Sebastiano, 2017. Normal distribution graph. Created 9 Feb 2017.
URL https://tex.stackexchange.com/a/352969

Stephanie, 2016. RMSE online.
URL https://www.statisticshowto.datasciencecentral.com/rmse/

Vegvesenet, 2019. Api vegvesenet.
URL https://www.vegvesen.no/nvdb/apidokumentasjon/#/

Wikipedia contributors, 2019. Rectifier (neural networks) — Wikipedia, the free encyclopedia. [Online; accessed 10 May 2019].
URL https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

Yi, X., Zhang, J., Wang, Z., Li, T., Zheng, Y., 2018. Deep distributed fusion network for air quality prediction.
URL http://doi.acm.org/10.1145/3219819.3219822

Yr, 2019a. Facts about yr.
URL https://hjelp.yr.no/hc/en-us/articles/206557169-Facts-about-Yr

Yr, 2019b. Free weather data service from yr.
URL https://hjelp.yr.no/hc/en-us/articles/360001940793-Free-weather-data-service-from-Yr

Appendix A

Graph Appendix

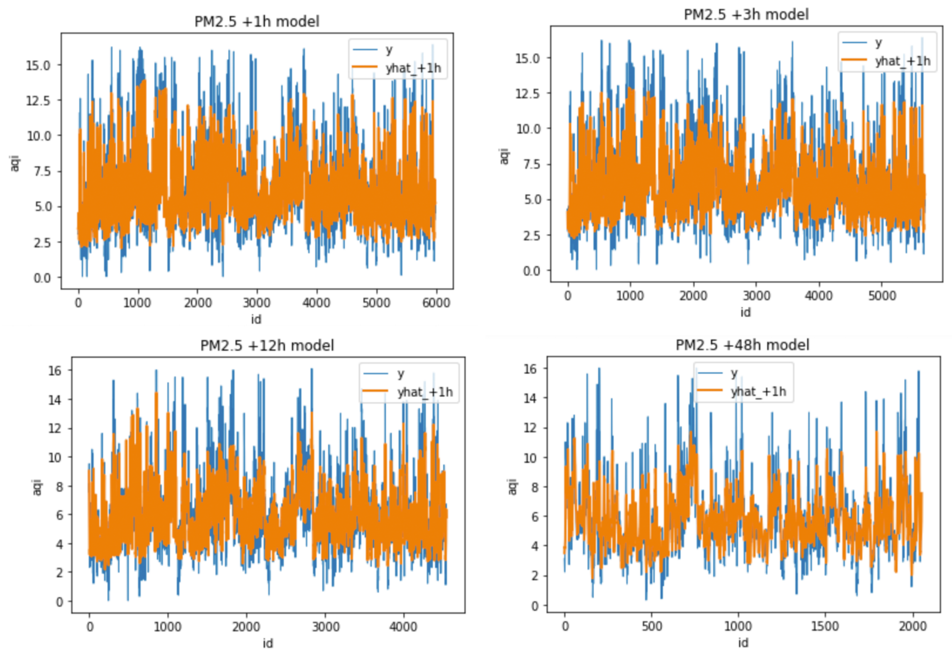

Figure A.1: Plot of selected models on data from Hansjordnesbukta, Tromsø for AQI $PM_{2.5}$

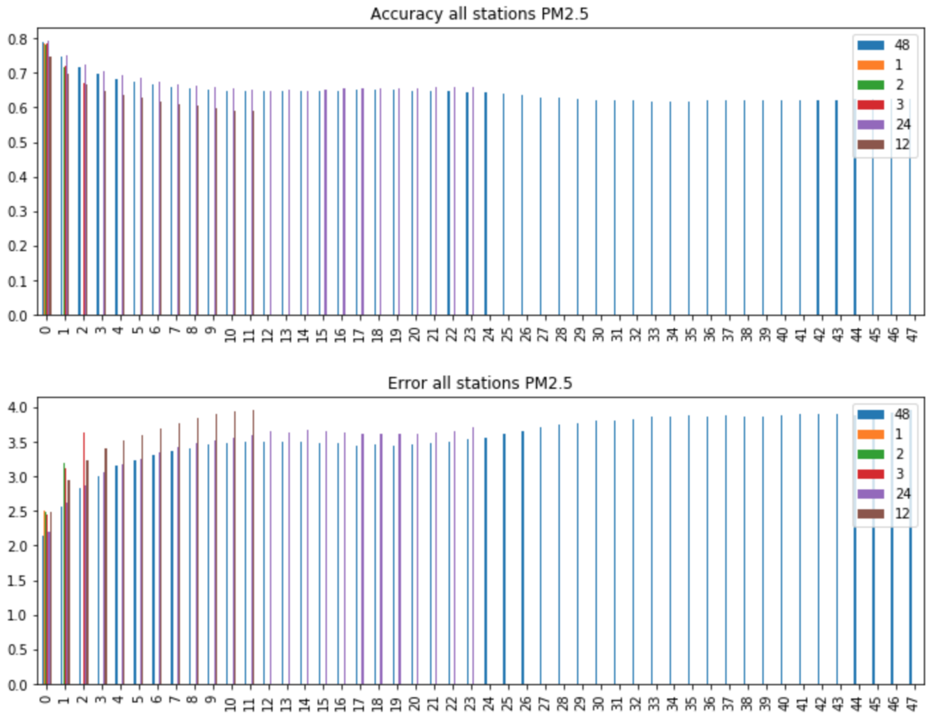

Figure A.2: Metrics of the models on all data for AQI $PM_{2.5}$

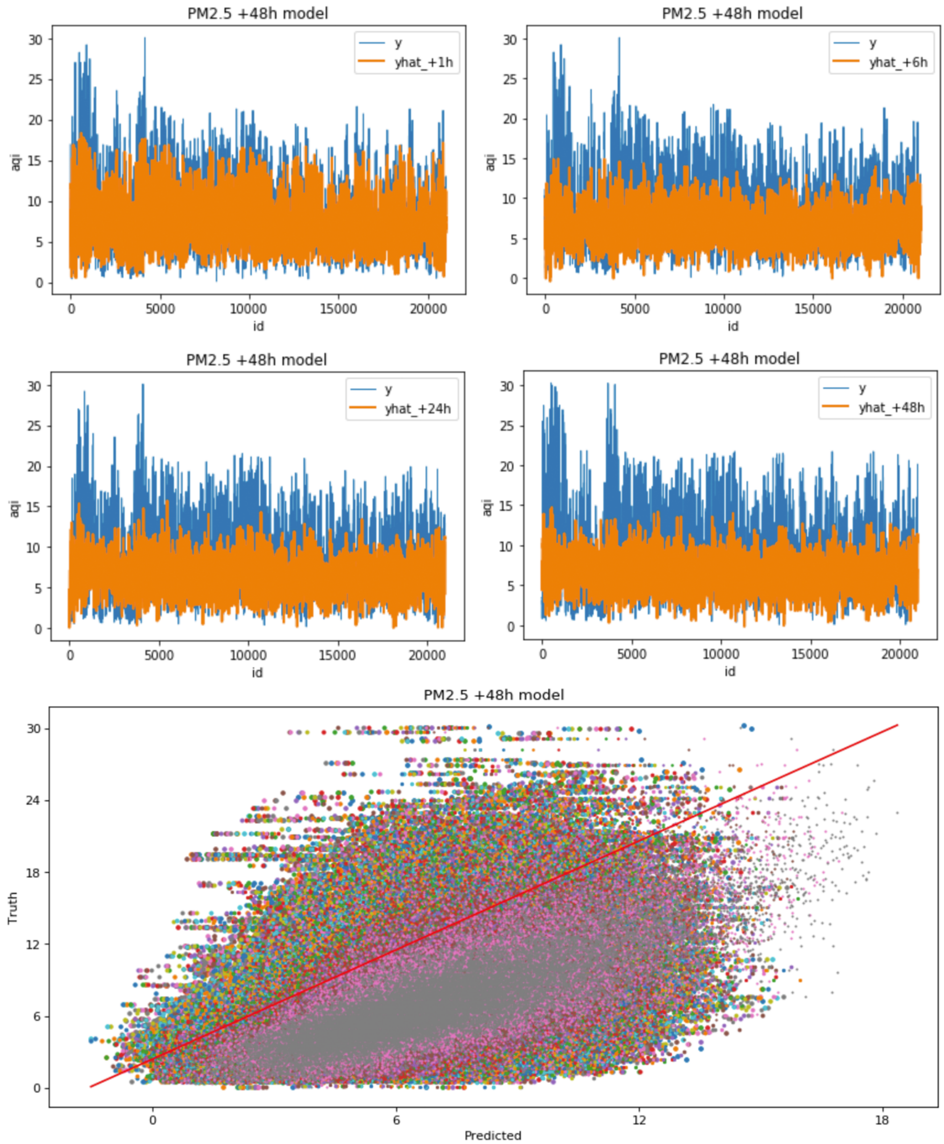

Figure A.3: Plots and scatter plot of the model 48h on all data for AQI $PM_{2.5}$

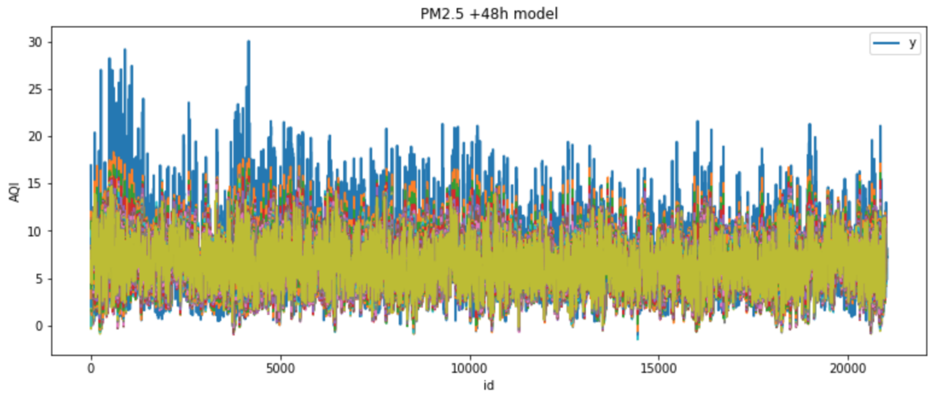

Figure A.4: Plot of model 48h for AQI $PM_{2.5}$

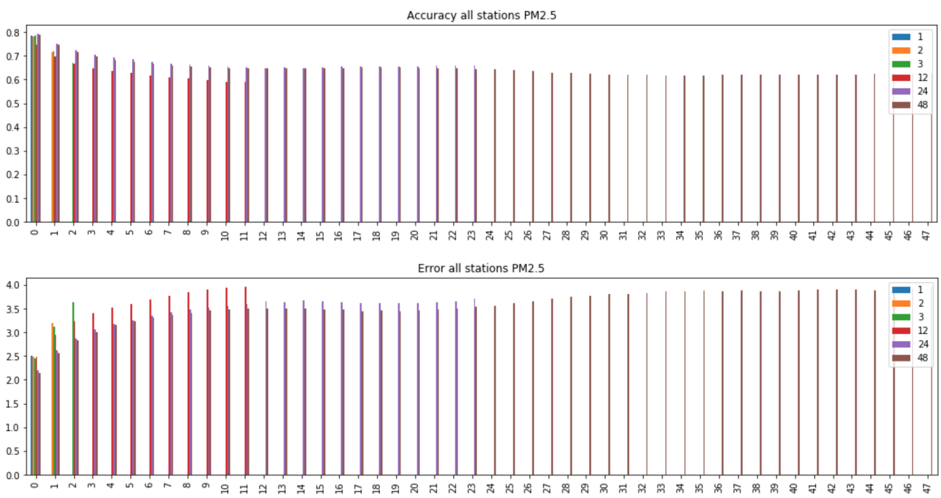

Figure A.5: RMSE and ACC of model 48h for AQI $PM_{2.5}$

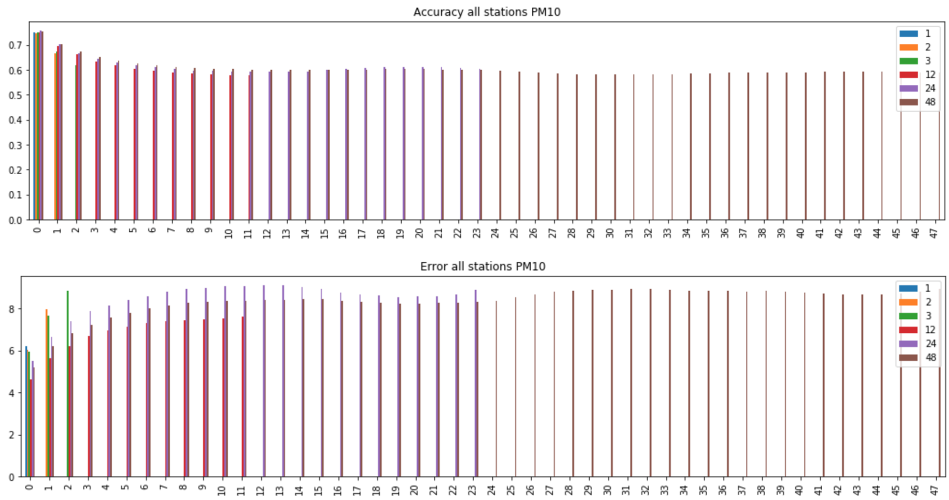

Figure A.6: RMSE and ACC of model 48h for AQI PM_{10}

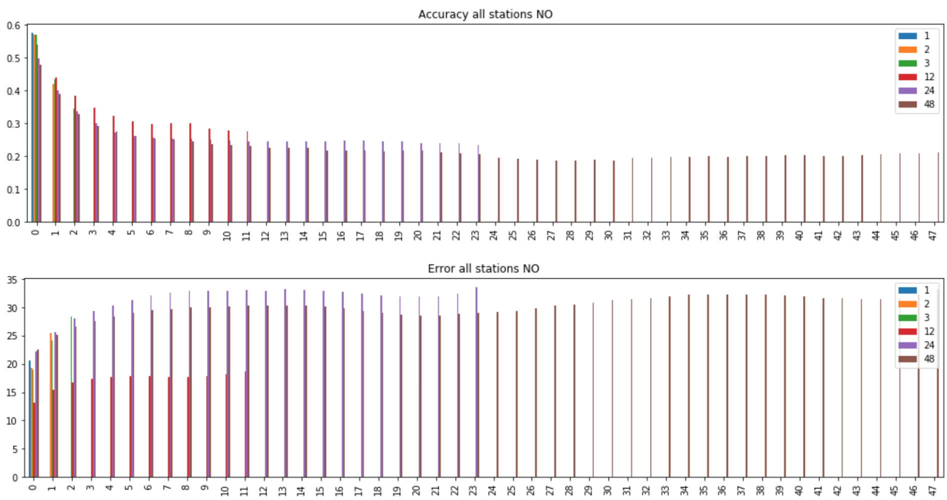

Figure A.7: RMSE and ACC of model 48h for AQI NO

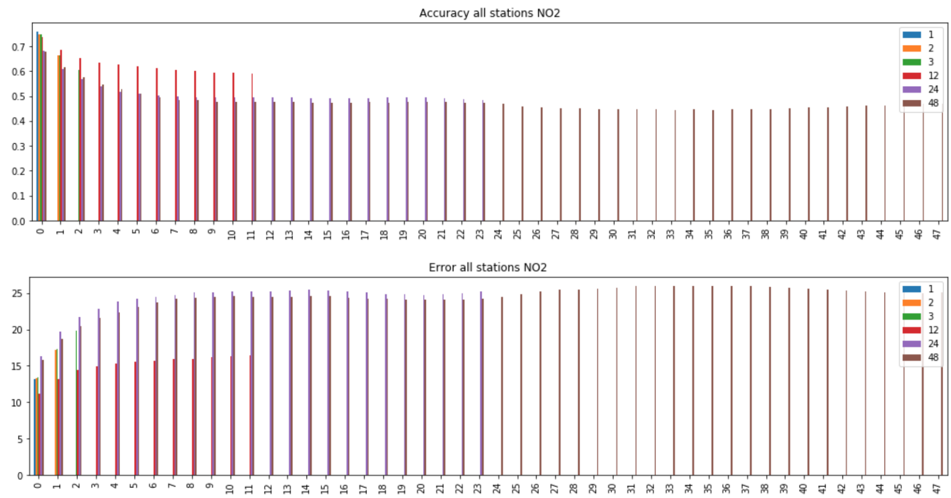

Figure A.8: RMSE and ACC of model 48h for AQI NO_2

Appendix B

Tables

PM2.5	**1**	**2**	**3**	**12**	**24**	**48**
1	0.7852662256591139	0.7838088807360142	0.7873698396019656	0.7488123549182938	0.7935994686827478	0.7904420153889806
2		0.7165006452307204	0.7203577647209534	0.6986944915664517	0.7525857734063601	0.7470413519228847
3			0.6711446273791519	0.6680335281623846	0.7254207591669948	0.7183887003330798
4				0.6483475206649489	0.7066477487551577	0.6984382874371384
5				0.6371707197128536	0.6943031031732718	0.6831900308844505
6				0.6289896041210288	0.6852288819144043	0.67334440045215
7				0.6190720176438496	0.6755759159414143	0.6658649989746805
8				0.6113069802333688	0.6679591541084291	0.6582917202747602
9				0.6039928750299212	0.6625778055353179	0.65492740947493
10				0.5970997468619954	0.6583590461757121	0.6505266226161399
11				0.5914412889024194	0.6550711913914917	0.6487493847987403
12				0.5918401125983286	0.6520915022790402	0.6483473545369682
13					0.6495346658801513	0.6472656278863094
14					0.6511920842034926	0.6476898552347681
15					0.6486050873588498	0.6484561798289177
16					0.6521554618657721	0.6491206687748331
17					0.6538595200909099	0.6486779492417991
18					0.6541987882480212	0.6511018575837453
19					0.6557309966114162	0.6500388920676483
20					0.6555558971981467	0.6521662329346967
21					0.6573772497567643	0.6490754694238585
22					0.6587667364900126	0.6486990417780046
23					0.6575558796737978	0.6472194159478046
24					0.6579982209302186	0.6444884634369835
25						0.6440918387490142
26						0.6389242187906885
27						0.6352198360645236
28						0.630345898916301
29						0.6269196502724153
30						0.6244438738090706
31						0.6218170664420171
32						0.6228336403292098
33						0.6215394112799564
34						0.6187048912226006
35						0.6182121661230691
36						0.6185085435526101
37						0.6208373257294613
38						0.6200186109931398
39						0.6219010784877783
40						0.6227943840423942
41						0.6209250950037233
42						0.6205363815647822
43						0.6211140344495647
44						0.6220653904512081
45						0.626310101632409
46						0.6260918954025176
47						0.6242614863210969
48						0.6245740515613883

Table B.1: All Oslo PM2.5 AQI model accuracy

PM10	1	2	3	12	24	48
1	0.7492129234940805	0.7454189363710371	0.7494945843630636	0.7514415505943493	0.7570683646295238	0.7525606632284199
2		0.664854049415365	0.6728830171912104	0.695225599008817	0.7033498641461469	0.7037146760434785
3			0.6198267516801582	0.6603544784569404	0.667113729770852	0.6740583298788052
4				0.6325782277993177	0.6436864086158853	0.6524090439690259
5				0.6167464110955715	0.6301067825895534	0.6370441629677509
6				0.6051742714874735	0.6195325658912434	0.6261851848895845
7				0.5944500099172503	0.6113251085268259	0.6176660842388311
8				0.587922082012798	0.602924309215793	0.609992217273795
9				0.5853511874520083	0.5976440378224352	0.6052583611108836
10				0.581488692574669	0.5956464085919788	0.6031444960079464
11				0.5785437795579249	0.5934855207005623	0.6017316556340357
12				0.5791466059869779	0.5932654262475285	0.6015065156462112
13					0.5924839653900323	0.6004886130810483
14					0.591127029361113	0.6001473480019183
15					0.5941098176482735	0.5992070720319793
16					0.5986679505501122	0.5984971367987324
17					0.6034799681605812	0.6007191816834088
18					0.6070790367048369	0.6015208672266574
19					0.6092094027205561	0.6021469033228417
20					0.6117918447558524	0.6028989211120654
21					0.6112108703981726	0.6020458397205571
22					0.6105519770037702	0.6012008572269341
23					0.6087334053751314	0.6004138613662767
24					0.6040686917360423	0.599193959460828
25						0.5976278859474617
26						0.5932800379315342
27						0.5886561134041957
28						0.5835680499654194
29						0.5802794362089323
30						0.5803166203748154
31						0.5798019560596308
32						0.5803852146171625
33						0.5811570711174908
34						0.5826587698094878
35						0.5848774813102192
36						0.5865411660360871
37						0.5871307280167969
38						0.5888951016103079
39						0.5887207182075243
40						0.5890169032232191
41						0.5898543980573442
42						0.5910394788452269
43						0.5911863042219268
44						0.5916691588588212
45						0.591112464653628
46						0.5902133164792522
47						0.588920148242011
48						0.5881138616973255

Table B.2: All Oslo PM10 AQI model accuracy

NO2	1	2	3	12	24	48
1	0.7587670002540479	0.7467929709066725	0.749319870766693	0.7366080685999072	0.6800550136039588	0.6773031587861558
2		0.6624845943342079	0.6630229454707328	0.6866341121623246	0.6092433022160848	0.6142712153800093
3			0.6043230472191736	0.6529262359068784	0.5667238758976978	0.5745651095471295
4				0.6357683331080868	0.5394825680285258	0.5476575422760909
5				0.626644590068363	0.5185039037657637	0.5271698106051421
6				0.6185274728714366	0.5109634727577654	0.5094870866510839
7				0.611331430855584	0.5033375444941858	0.49556642713289045
8				0.6048350280956174	0.49842748243393786	0.4854184002127325
9				0.6009565860356499	0.4934106209970942	0.48242316837707955
10				0.5934813412950578	0.49514507153036813	0.47779831788641747
11				0.5930079478390728	0.493964587466862	0.4754204879301307
12				0.5895923920322093	0.49323964301289824	0.4761847520165784
13					0.493724471052042	0.4759336995050769
14					0.4930090729314005	0.47467536216790207
15					0.49095001955511175	0.4730617546923649
16					0.4908538143329245	0.4726742683180968
17					0.49216964470710667	0.47300254273019904
18					0.49271814740176023	0.4757000209440726
19					0.49534388133178997	0.47443490258863485
20					0.4943871307057127	0.4751974813086819
21					0.49433726166824266	0.47546105163642916
22					0.48939415081952453	0.4767173078151261
23					0.48585698237054464	0.4746191735411214
24					0.48529999308659766	0.47230739247923403
25						0.4676698137713542
26						0.459284721656699
27						0.452955904442505
28						0.4500156336372202
29						0.44978380018721575
30						0.4485145631494417
31						0.44793289441547113
32						0.4457945441887148
33						0.4455179352199583
34						0.4445550779107217
35						0.44599693720314293
36						0.4453223222394693
37						0.44577464501759667
38						0.4463744078800943
39						0.4481594472652174
40						0.4500481869590708
41						0.4538808716578806
42						0.4555028506380844
43						0.45949456508403586
44						0.4613754451056007
45						0.4635514551021763
46						0.4652853165463904
47						0.4677990491905415
48						0.46980169141398376

Table B.3: All Oslo NO2 AQI model accuracy

NO	1	2	3	12	24	48
1	0.5757082532341252	0.5707539967094992	0.5692640036119818	0.5397060601727628	0.4973127378575639	0.47919866046167137
2		0.4212778160613657	0.435142871216483	0.4388195676069482	0.3995044403369483	0.3886183687222218
3			0.34354375350719546	0.3838190432168517	0.3361072003142358	0.3291004281602332
4				0.3470034170921559	0.2997558955465489	0.2930604229809177
5				0.32146837837802034	0.2738515118256708	0.2740444642644301
6				0.3063169112247873	0.26139111118494385	0.2622013318223465
7				0.2976597892400822	0.25608534919561954	0.2518829312526356
8				0.3015255602752186	0.2522637150248588	0.24896131088923745
9				0.30140046600753534	0.253190989852602	0.24436111467800448
10				0.2847823271524146	0.24996937693025856	0.23553962245515092
11				0.2772387188795915	0.2479597187419671	0.23346510690069489
12				0.2763504500890833	0.24445415040547924	0.22972911738397916
13					0.24503437730128308	0.22574850552489112
14					0.24427456153041982	0.22509321893904688
15					0.2447313032545133	0.2241211269041452
16					0.2440808379074706	0.21804969776015182
17					0.24682028680873558	0.21728158978844692
18					0.24660435838804262	0.2174113298443826
19					0.24584014849847036	0.2155756342632894
20					0.24441366231852213	0.21573842711896252
21					0.2391141302274985	0.2161572994450347
22					0.24011395554219828	0.21173621481619265
23					0.2401621962780447	0.20920933995269508
24					0.23319822523704203	0.20566616430567553
25						0.19511832213089364
26						0.19156324324135776
27						0.18982003650782298
28						0.1862049318217488
29						0.18634829204459524
30						0.18875575004538858
31						0.18672100578674256
32						0.19435606914011072
33						0.19466192213537614
34						0.19823825998111477
35						0.19791451852130504
36						0.19923586259141968
37						0.19653430369767932
38						0.20001440643988067
39						0.20048807846121974
40						0.20270561922829655
41						0.20210485844494142
42						0.19897308645276146
43						0.20074750867639712
44						0.2028178978434504
45						0.20456922531593846
46						0.20760295925602257
47						0.2074812045555421
48						0.21059965212378062

Table B.4: All Oslo NO AQI model accuracy

PM2.5	1	2	3	12	24	48
1	2.5139530117943987	2.478650699966351	2.444116494346637	2.489361115390605	2.1984751522631685	2.1362513405229118
2		3.187735338639318	3.1272140389389884	2.94617414631982	2.6128908047871224	2.5578981911622702
3			3.6325104086331454	3.224485876890353	2.877964474534852	2.8217042553376848
4				3.4092155081354885	3.0530524526548186	3.0038585514709424
5				3.5137433310078774	3.167625540556119	3.148591031932766
6				3.6029015280875565	3.2529369120827036	3.2363102406533377
7				3.693437194173146	3.341157770938495	3.3049059575336033
8				3.767633222528121	3.4197823791262376	3.3691588376338877
9				3.8388283129123573	3.480286609234758	3.4098735049398026
10				3.9014635575137144	3.5284959657212434	3.4557745808587805
11				3.949143836957768	3.565121503323458	3.4777731859008165
12				3.9600864172372505	3.6044085408742013	3.4930894987454573
13					3.647314539058517	3.5049822183452544
14					3.6351463124785344	3.5005840878358843
15					3.6733202578967448	3.4912101907639808
16					3.660387780745519	3.4840244079113294
17					3.6378050117347516	3.4810005201459897
18					3.6246671506289494	3.451956076208123
19					3.6206656338614125	3.452854676166612
20					3.622154804467665	3.4357445803187816
21					3.61329367356708	3.463572810182482
22					3.628407692932544	3.473508195034346
23					3.6589636874374047	3.492935618111791
24					3.703936238128872	3.533349359568214
25						3.5592920606999288
26						3.6183101935081585
27						3.6522763308601025
28						3.711204797861913
29						3.7439200644271455
30						3.772536319829894
31						3.8041982321144756
32						3.807965756675359
33						3.8301740564750495
34						3.857854501660689
35						3.8699064283519817
36						3.8777562303457045
37						3.8664656262202715
38						3.8838971555709088
39						3.870827228972408
40						3.8720406323207417
41						3.881486479944837
42						3.896652068519922
43						3.8977154103264207
44						3.8998514385075174
45						3.874142479029236
46						3.8908626038757554
47						3.920731009718751
48						3.959310990991841

Table B.5: All Oslo PM2.5 AQI model rmse

PM10	1	2	3	12	24	48
1	6.191650423665176	6.049952736569319	5.953690405280387	4.618357375227446	5.488048021134793	5.198090777056822
2		7.9475560212414775	7.640527263645733	5.628445251758559	6.630603582613667	6.206704851187601
3			8.866756956449477	6.218484844216862	7.381635156443408	6.8073288056184245
4				6.710183638440727	7.890344367425959	7.2314212592325315
5				6.94844808030564	8.163849312297566	7.562350148129964
6				7.1261378592184315	8.41703987184162	7.795065197384583
7				7.309505443454492	8.600164481259482	7.990127865192293
8				7.385519104469912	8.80273832212562	8.164800862188994
9				7.431329851754262	8.942925808540968	8.269712650368891
10				7.495091348722321	8.999979612600509	8.3289445758874
11				7.546339272622614	9.047136551718813	8.363117716311095
12				7.616775874865285	9.072540474873236	8.37514079885369
13					9.092218260601662	8.401088926775897
14					9.11332485556976	8.420830158194239
15					9.032434703661743	8.450132584643265
16					8.923660534751907	8.462886807824844
17					8.767098434081658	8.359163751376828
18					8.67730275256859	8.327782289413829
19					8.616271890113016	8.29060682438621
20					8.560730121173762	8.234928731346917
21					8.577199708767017	8.23583944312438
22					8.602402693739903	8.258152908473367
23					8.67642499925393	8.255809879554711
24					8.895469180271562	8.323559485937242
25						8.381792895333291
26						8.521530135008176
27						8.655475019388042
28						8.786032294389868
29						8.866660526895602
30						8.887754955758721
31						8.898989540720649
32						8.918723217380727
33						8.914259018054114
34						8.901651019030048
35						8.8532760382837
36						8.83581096667907
37						8.842847691385787
38						8.814532254799916
39						8.839394928720234
40						8.80410684438403
41						8.762213169010648
42						8.70976955342013
43						8.668055551587774
44						8.66430575623849
45						8.685007577740034
46						8.72437388870263
47						8.816605811055842
48						8.957245113572842

Table B.6: All Oslo PM10 AQI model rmse

NO2	**1**	**2**	**3**	**12**	**24**	**48**
1	13.191815739770192	13.246491156783152	13.42976880027114	11.1919826370782	16.275192640316085	15.753082692454923
2		17.208388414744718	17.30783762250318	13.171541478679934	19.67188913373553	18.69223045715057
3			19.781402214529685	14.396478139458877	21.645981444224535	20.42508804881987
4				14.971931454829797	22.886205720452455	21.564879718642572
5				15.243281418945466	23.87357117184529	22.355886218538625
6				15.511510146860065	24.201958915717306	23.0639069775623
7				15.722006501619775	24.50118466577419	23.673833982391542
8				15.882635058796376	24.769062918931592	24.190600248182488
9				15.967890858177816	25.06513390620669	24.335266467230326
10				16.241189961780794	25.04546137976557	24.446683822044864
11				16.282698751725693	25.152362012359884	24.52142438360912
12				16.467554657523475	25.234748968150512	24.468084531068243
13					25.21835872725522	24.46789831351671
14					25.329365173242763	24.490455903998257
15					25.448822515745615	24.55311207174214
16					25.38754574006256	24.540035443479837
17					25.244256005102397	24.347124933468105
18					25.109297327436902	24.16665954830203
19					24.815713189227687	24.196218432309074
20					24.826492415675865	24.083536153013938
21					24.721977700002352	24.057368358459062
22					24.870625452346324	24.030675221828236
23					25.012085781066563	24.113535278562722
24					25.161966365565732	24.253372618173685
25						24.482414211958062
26						24.862192050168893
27						25.178047253318784
28						25.42862123726995
29						25.507949771002252
30						25.62935665500577
31						25.74174924855552
32						25.953108993093853
33						25.967776671746385
34						25.96124940597461
35						25.919655449940727
36						25.925441979538768
37						25.930103474649254
38						25.915914485311617
39						25.842608298484404
40						25.754431556847617
41						25.55537439414575
42						25.467880837066986
43						25.334319922896782
44						25.22071307845378
45						25.10458258924333
46						25.03179597988543
47						24.990800034556962
48						25.088263577279633

Table B.7: All Oslo NO2 AQI model rmse

NO	1	2	3	12	24	48
1	20.655393905192536	19.298727338428332	18.99347905594914	13.085777202603268	22.152182972970085	22.543671273982937
2		25.392032051135786	24.236747827373247	15.46817094986018	25.66396737354039	25.063658653475763
3			28.386974170091765	16.677873329391495	28.034878337054497	26.657372913339273
4				17.31866268049552	29.364754134089537	27.62497023415702
5				17.65359525480392	30.385036876846993	28.42896233861056
6				17.781873435600254	31.21765546808076	29.00538881962254
7				17.859928626751337	32.049166165022235	29.47263106155742
8				17.692182514236848	32.576581977978115	29.731042363350326
9				17.658177581938883	32.91334567429618	30.005710248988674
10				17.88685947476869	32.93523207971006	30.076161500797884
11				18.10561018328713	32.916468465228625	30.2090658222665
12				18.683636371443463	33.06311575158279	30.295566710083172
13					32.97659988017982	30.392783141717
14					33.193014582060925	30.313867790516774
15					33.025888721339584	30.269328207697065
16					32.87164764642953	30.187013765691567
17					32.68410725895447	29.758883635252644
18					32.4049948918254	29.422052423397147
19					32.07076369824646	28.993155771129697
20					31.983622862503474	28.70803906575781
21					31.988194923650035	28.568626336144472
22					31.988172524682994	28.59397212073471
23					32.41829753033004	28.833410307844584
24					33.572855095976	29.1070158040041
25						29.24800076223714
26						29.414890305029523
27						29.90131051197158
28						30.260168546541326
29						30.529969912457222
30						30.87213818555672
31						31.256788992842736
32						31.521720109885653
33						31.6240630317442
34						31.967900473005145
35						32.207366220044534
36						32.29653351482978
37						32.26654777858492
38						32.34556607358811
39						32.3465802200042
40						32.18169118345302
41						31.912369075523205
42						31.541604673794616
43						31.562650558393223
44						31.456682698238705
45						31.44584503865449
46						31.646202075131065
47						32.20910855680164
48						33.18311516330645

Table B.8: All Oslo NO AQI model rmse

Appendix C

Code Appendix

I Weather data script

```
#! /usr/bin/env python

# weather_data.py

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import pandas as pd
from pandas import DataFrame
import itertools
from six.moves import urllib

import datetime
import re
import numpy as np
import json

URL_FORMAT = 'https://www.yr.no/place/{location}/almanakk.html?dato={date}'

REGEXES = dict(
    time = re.compile(r'<th scope="row">.*<strong>(.*)</strong></th>'),
    temperature = re.compile(r'<td class="temperature .*">(.*) *C</td>'),
    rain = re.compile(r'<td>(.*) mm</td>'),
    humidity = re.compile(r'<td>(.*) %</td>'),
    wind = re.compile(r'<img src=".*" height=".*" width=".*" alt="(.*)" class="wind".*/>')
```

```
)

def format_string(string):
    return str(string).replace('ae','%C3%A6').replace('oe','%C3%B8').replace('aa','%C3%A5')

def get_url(location, date):
    return URL_FORMAT.format(location=location, date=date)

def get_datetime(date):
    return datetime.strptime(date, '%Y-%m-%dT%H:%M')

def get_measurements(date, lines):
    messages = []
    message = None

    active = False
    for line in lines:
        line = line.decode()
        line = line.strip()

        time_match = REGEXES['time'].search(line)
        if time_match is not None:
            message = dict()

            when = get_datetime('{date}T{time}'.format(date=date, time=time_match.group(1)))
            message['timestamp'] = when.strftime('%d.%m.%Y %T')
            message['temperature']=[]
            continue

        temperature_match = REGEXES['temperature'].search(line)
        if message is not None and temperature_match is not None:
            message['temperature'].append(float(temperature_match.group(1)))
            continue

        rain_match = REGEXES['rain'].search(line)
        if message is not None and rain_match is not None:
            message['precipitation'] = float(rain_match.group(1))
            continue

        wind_match = REGEXES['wind'].search(line)
        if message is not None and wind_match is not None:
            st = wind_match.group(1).split(' ')
            if (len(st)>4):
```

```
                message['wind'] = st[-4]
                message['wind_from'] = st[-1]
            continue

        humidity_match = REGEXES['humidity'].search(line)
        if message is not None and humidity_match is not None:
            message['humidity'] = int(humidity_match.group(1))
            message['temp_max'] = None
            message['temp_min'] = None
            if len(message['temperature']) > 2:
                message['temp_max'] = message['temperature'][1]
                message['temp_min'] = message['temperature'][2]
            message['temperature'] = message['temperature'][0]

            messages.append(message)
            message = None
            continue

    return messages

def load_weather_data(date_from, date_to, location='Norge/Oslo/Oslo/Oslo'):
    loc = format_string(location)
    date_from_d = datetime.strptime(date_from, '%Y-%m-%d')
    date_to_d   = datetime.strptime(date_to, '%Y-%m-%d')
    date_range  = pd.date_range(date_from_d, date_to_d)

    data = []
    for i, d in enumerate(date_range):
        date = d.strftime('%Y-%m-%d')
        with urllib.request.urlopen(get_url(location=loc, date=date)) as url:
            lines = url.readlines()
            data.append(get_measurements(date, lines))
    return data
```

II AQI script

```
#! /usr/bin/env python

# AQI_data.py

from urllib.request import urlopen
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
```

```
def load_aqi(lat, lon, date, radius=3.0):
    date_string = date.strftime("%Y-%m-%d")
    aqi_url = 'https://api.nilu.no/obs/historical/{}%2000:00/{}%2023:59/{}/{}/{}'.format(date_string, date_string, lat, lon, radius)
    return df_url(aqi_url)

def df_url(url):
    file = urlopen(url)
    data = file.read()
    file.close()
    return pd.read_json(data)

def get_aqi_area(datefrom, dateto, area='oslo'):
    stations_url = 'https://api.nilu.no/lookup/stations?area={}'.format(area)
    area_dict = {}

    df = df_url(stations_url)[['id','station','components','firstMeasurment','lastMeasurment','latitude','longitude']]
    for w in df.id:
        d = df[df.id == w]
        lat = d.latitude.values[0]
        lon = d.longitude.values[0]

        date_range = pd.date_range(datefrom, dateto)
        if lat != 0 and lon != 0:
            for i, d in enumerate(date_range):

                aqi_df = load_aqi(lat, lon, d)
                for index, row in aqi_df.iterrows():
                    if row.station not in list(area_dict.keys()):
                        area_dict[row.station] = dict()

                    if 'lat' not in list(area_dict[row.station].keys()):
                        area_dict[row.station]['lat'] = row.latitude

                    if 'lon' not in list(area_dict[row.station].keys()):
                        area_dict[row.station]['lon'] = row.longitude


                    if 'aqi' not in list(area_dict[row.station].keys()):
                        area_dict[row.station]['aqi'] = pd.DataFrame(columns=[row['component']])

```

```
                    for val in row['values']:
                        if row.component not in area_dict[row.station]['aqi'].columns:
                            area_dict[row.station]['aqi'][row.component] = np.NaN
                        t_str = val['toTime']
                        k = t_str.rfind(":")
                        t_str = t_str[:k] + t_str[k+1:]
                        ix = datetime.strptime(t_str, "%Y-%d-%mT%H:%M:%S%z")
                        if val['qualityControlled']:
                            area_dict[row.station]['aqi'].at[ix,row.component]= float(val['value'])

    return area_dict
```

III Data script

```
#!/usr/bin/env python

# data.py


import pandas as pd
import numpy as np
import sys
import os
import pathlib
from math import sqrt
import math
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Input

import sys
from datetime import datetime, timedelta
```

```python


def load_seq_data(station, area, predict_label, hours_to_predict, hours_to_lookback, dataframe):

    yr_labels = ["humidity", "precipitation", "temp_max", "temp_min", "temperature", "wind", "wind_from"]
    time_label = ['month', 'day', 'hour']
    feature_labels = [c for c in dataframe if c in ['PM10', 'PM2.5', 'NO', 'NO2']]

    hist_weather_label = ['humidity', 'precipitation', 'temperature', 'wind', 'wind_from']
    weather_prediction_label = ['precipitation', 'temperature', 'wind', 'wind_from']

    AQI_dict = {}
    # get data for all AQI's
    for l in feature_labels:
        AQI = l
        other_pollutants = feature_labels[:feature_labels.index(AQI)] + feature_labels[feature_labels.index(AQI)+1:]
        AQI_dict[AQI]=other_pollutants

    df = clean_data(dataframe=dataframe, feature_label=predict_label, dropnan=False)

    for col in df:
        if col != 'wind_from':
            df[col] = df[col].astype(float)

    aqi = series_to_sequence(df[[predict_label]], n_out=hours_to_predict, n_in=hours_to_lookback, dropnan=False).values
    aqi_X = aqi[:,:hours_to_lookback].reshape(-1, hours_to_lookback,1)[hours_to_lookback:]
    aqi_y = aqi[:, hours_to_lookback:][hours_to_lookback:]

    other_pollutants = series_to_sequence(df[AQI_dict[predict_label]], n_out=0, n_in=hours_to_lookback, dropnan=False).values[hours_to_lookback:].reshape(-1, hours_to_lookback,len(AQI_dict[predict_label]))
    time = series_to_sequence(df[time_label], n_out=0, n_in=hours_to_lookback, dropnan=False).values[hours_to_lookback:].reshape(-1, hours_to_lookback,len(time_label))

```

```
    meteorology = series_to_sequence(df[hist_weather_label], n_out=0, n_in=hours_to_lookback, dropnan=False).values[hours_to_lookback:].reshape(-1, hours_to_lookback, len(hist_weather_label))
    weatherforecast = series_to_sequence(df[weather_prediction_label], n_out=hours_to_predict, n_in=0, dropnan=False).values[hours_to_lookback:].reshape(-1, hours_to_predict, len(weather_prediction_label))

    del_rows = null_rows([other_pollutants, aqi_X, aqi_y, time, meteorology, weatherforecast])
    del_mask = np.ones(len(aqi_X), dtype=bool)
    del_mask[del_rows] = False

    # Remove rows with NaN
    other_pollutants = other_pollutants[del_mask]
    aqi_X = aqi_X[del_mask]
    time = time[del_mask]
    meteorology = meteorology[del_mask]
    weatherforecast = weatherforecast[del_mask]
    aqi_y = aqi_y[del_mask]

    # Transform the weather direction
    w_shape=weatherforecast[:,:,-1:].shape
    m_shape=meteorology[:,:,-1:].shape

    encoder = LabelEncoder()
    encoder = encoder.fit(['east',
'east-northeast',
'east-southeast',
'north',
'north-northeast',
'north-northwest',
'northeast',
'northwest',
'south',
'south-southeast',
'south-southwest',
'southeast',
'southwest',
'west',
'west-northwest',
'west-southwest'])
    weatherforecast[:,:,-1:] = encoder.transform(weatherforecast[:,:,-1:].flatten()).reshape(w_shape)
    meteorology[:,:,-1:] = encoder.transform(meteorology[:,:,-1:].flatten()).reshape(m_shape)
```

```
    return meteorology, weatherforecast , other_pollutants ,time,aqi_X,aqi_y

def load_data( location ="tromso", del_col =[], force_reload =False):
    filepath = "data/combined_data_{}.json".format( location ).lower()
    df = None
    if os.path. isfile ( filepath ) and not force_reload :
        df = pd.read_json( filepath )
    else:
        print("Loading data from NILU")
        #Load data from nilu
        nilu_df = load_nilu_dataframe ( location =location )
        print("Loading data from Yr.no")
        # load yr data to nilu data using timestamp
        yr_df = load_yr_dataframe( nilu_df .index [0], nilu_df .index[-1], location =
            location )
        print("Done loading")
        print("Processing ... ")
        df = combine_data(nilu_df, yr_df)
        print("Done processing")
        print("Storing data")
        # Store the data
        pathlib .Path( filepath . rsplit ('/',1) [0]) .mkdir(parents=True, exist_ok=
            True)
        df. to_json( filepath )

    predict_labels = ['PM2.5', 'PM10','NO2','NO']

    # remove columns not to be included
    df = df.drop(columns=del_col)

    # Set prediction label at back
    df = df[[c for c in df if c not in predict_labels ] + [c for c in
        predict_labels if c in df]]
    return df

def progressbar (index, total , start_time =None):
    sys. stdout . write ('\r')
    j = (index + 1) / total
    sys. stdout . write ("[%-20s] %d%%" % ('='*int(20*j), 100*j))
    if (index % (round(total/1000)) == 0) and start_time is not None:
        now = datetime.now()
        duration = (now-start_time). total_seconds ()
        expected_duration = ( duration /( j*100))*100
        time_left = expected_duration - duration
```

```
            if(index % (round(total/1000)) == 0):
                sys.stdout.write("\tETA: " + str(timedelta(seconds=time_left)))

    sys.stdout.flush()


# Get NILU data
def load_nilu_dataframe(return_labels=False, location="tromso"):
    nilu_df = pd.read_csv("data/aqi_values_{}.csv".format(location).lower())
    nilu_df['Date'] = pd.to_datetime(nilu_df['Date'])
    nilu_df['month'] = nilu_df['Date'].dt.month
    nilu_df['day'] = nilu_df['Date'].dt.day
    nilu_df['hour'] = nilu_df['Date'].dt.hour
    nilu_df.set_index('Date', inplace=True)

    cols_at_start = ['month', 'day', 'hour']
    if return_labels:
        return [c for c in nilu_df if c not in cols_at_start]
    nilu_df = nilu_df[[c for c in cols_at_start if c in nilu_df] + [c for c in nilu_df if c not in cols_at_start]]

    return nilu_df

# Get Weather data
def load_yr_dataframe(from_time, to_time, location="tromso"):
    #yr_weather_df = pd.read_csv("data/yr_hourly_2008_2018.csv")
    yr_weather_df = pd.read_csv("data/yr_weather_{}.csv".format(location).lower())
    yr_weather_df['timestamp'] = pd.to_datetime(yr_weather_df['timestamp'])
    yr_weather_df = yr_weather_df[yr_weather_df.timestamp >= from_time]
    yr_weather_df = yr_weather_df[yr_weather_df.timestamp <= to_time]
    yr_weather_df.set_index('timestamp', inplace=True)
    yr_weather_df.index.name = 'Date'
    return yr_weather_df

def combine_data(a, b):
    combined_df = a.copy()
    for c in b.columns:
        nan_arr = np.empty(combined_df.shape[0])
        nan_arr.fill(np.nan)
        combined_df = combined_df.assign(x=pd.Series(nan_arr).values)
        combined_df = combined_df.rename(index=str, columns={'x': c})
    total = (b.shape[0])
    counter = 0
    start_time = datetime.now()
```

```
    for i, row in b.iterrows():

        for x in range(0, len(b.columns)):
            combined_df.loc[ (a.index) == i, b.columns[x]] = row[x]

        progressbar(counter, total, start_time)
        counter +=1
    return combined_df



# convert time series to sequence
def series_to_sequence(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg


# root mean squared error (rmse) for regression
def rmse(y_true, y_pred):
    from keras import backend as K
    return K.sqrt(K.mean(K.square(y_pred - y_true)))

def rmse_acc(y_true, y_pred):
    return 1-rmse(y_true,y_pred)

# mean squared error (mse) for regression
```

```
def mse(y_true, y_pred):
    from keras import backend
    return backend.mean(backend.square(y_pred - y_true), axis=-1)

def mse_acc(y_true, y_pred):
    return 1-mse(y_true,y_pred)

# coefficient of determination (R^2) for regression
def r_square(y_true, y_pred):
    from keras import backend as K
    SS_res = K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return (1 - SS_res/(SS_tot + K.epsilon()))

def r_square_loss(y_true, y_pred):
    from keras import backend as K
    SS_res = K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return 1 - ( 1 - SS_res/(SS_tot + K.epsilon()))

def scale_down(data):
    scaler = StandardScaler().fit(flatten(data))
    return scale(data, scaler).astype(float), scaler


def scale_up(data, scaler):
    return scaler.inverse_transform(data)


def null_rows( data_list ):
    rows = []
    for d in data_list :
        for r in pd.isnull(d).any(1).nonzero()[0]:
            rows.append(r)
    return list(set(rows))

def pad(array, reference, offsets):
    # Create an array of zeros with the reference shape
    result = np.zeros(reference.shape)
    # Create a list of slices from offset to offset + shape in each dimension
    insertHere = [slice(offset[dim], offset[dim] + array.shape[dim]) for dim in
        range(a.ndim)]
    # Insert the array in the result at the specified offsets
    result[insertHere] = a
    return result
```

```
def flatten(X):
    flattened_X = np.empty((X.shape[0], X.shape[2]))  # sample x features array.
    for i in range(X.shape[0]):
        flattened_X[i] = X[i, (X.shape[1]-1), :]
    return(flattened_X)

def scale(X, scaler):
    for i in range(X.shape[0]):
        X[i, :, :] = scaler.transform(X[i, :, :])

    return X

def clean_data(dataframe, feature_label, dropnan=False):
    df = dataframe.copy()
    # remove data below 0.0 except temperature .......
    wind_from = df['wind_from']
    df['wind_from'] = 1.1
    df = df.astype(float)
    below_mask = df < 0.0
    below_mask['wind_from'] = False
    below_mask['temperature'] = False
    df['wind_from'] = wind_from
    df[below_mask] = np.NaN
    label_data = df[feature_label]

    #remove 3*standard deviation
    std=label_data.std()*3
    mean = label_data.mean()
    max_thresh = mean+std
    max_mask = label_data>max_thresh
    label_data[max_mask] = np.NaN
    df[feature_label] = label_data

    # fill row with nan if nan in a column
    df.iloc[pd.isnull(df).any(1).to_numpy().nonzero()[0]] = np.nan
    # drop rows with NaN values
    if dropnan:
        df.dropna(inplace=True)
    return df
```

IV Model definition script

```python
#! /usr/bin/env python

# model_bank.py


def dense_model(X_train, y_train , act="relu"):

    def make_submodel(model_inputs, sub_name):
        models_dense = []

        for m_in in model_inputs:
            m_name = m_in.name.split('_')[0]
            models_dense.append(keras.layers.Dropout(0.2)(Dense(int(m_in.shape.
                dims[1]), activation=act)(m_in)))



        output = keras.layers.concatenate(models_dense, name=sub_name)
        timesteps = int(output.shape.dims[1])
        n_features = int(output.shape.dims[2])

        output = Dense(timesteps, activation=act)(output)
        output = keras.layers.Dropout(0.2)(output)

        output = Dense(64, activation=act)(output)
        output = keras.layers.Dropout(0.2)(output)

        output = Dense(32, activation=act)(output)
        output = keras.layers.Dropout(0.2)(output)

        output = Dense(n_features)(output)
        output = keras.layers.Dropout(0.2)(output)

        return output

    # embedd
    hw_input = Input(shape=X_train[0].shape[1:], name="HW_input")
    wf_input = Input(shape=X_train[1].shape[1:], name="WF_input")
    sp_input = Input(shape=X_train[2].shape[1:], name="SP_input")
    mp_input = Input(shape=X_train[3].shape[1:], name="MP_input")
    aqi_input = Input(shape=X_train[4].shape[1:], name="AQI_input")

    hw = make_submodel([hw_input, aqi_input], "HW")
    wf = make_submodel([wf_input, aqi_input], "WF")
    sp = make_submodel([sp_input, aqi_input], "SP")
```

```
    mp = make_submodel([mp_input, aqi_input], "MP")
    hi = make_submodel([hw_input, wf_input, sp_input, mp_input, aqi_input], 'HI')

    # Make submodel take x amount of model inputs and concatenate
    merged = keras.layers.concatenate([hw, wf, sp, mp, hi])
    merged = Dense(5)(merged)

    merged = keras.layers.Flatten()(merged)
    merged = keras.layers.Dropout(0.2)(merged)

    # output = Dense(y_train.shape[-1], kernel_initializer='lecun_normal', activation='linear')(merged)
    output_dim = y_train.shape[-1]
    if y_train.ndim < 2:
        output_dim = 1
    output = Dense(output_dim)(merged)
    return Model(inputs=[hw_input, wf_input, sp_input, mp_input, aqi_input],
                 outputs=output)
```

V Model train script

```
#!/usr/bin/env python

# train_model.py

import os
import keras
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.utils import plot_model
from keras import backend as K, optimizers, Sequential

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler

import data
import model_bank


def fix_data(meteorology, weatherforecast, other_pollutants, time, aqi_X, aqi_y):
    # Scale data
    met_scaled, _ = data.scale_down(meteorology)
    temp_wf_scaled, _ = data.scale_down(weatherforecast)
```

```
    op_scaled, _ = data.scale_down(other_pollutants)
    t_scaled, _ = data.scale_down(time)
    aqiX_scaled, _ = data.scale_down(aqi_X)
    aqiy_scaled, scaler = data.scale_down(aqi_y.reshape(-1, hours_to_predict ,1))
    aqiy_scaled = aqiy_scaled.reshape(-1, hours_to_predict)
    # Padding weather forecast
    dimdiff = np.array(aqi_X.shape) - np.array(temp_wf_scaled.shape)
    wf_scaled = np.pad(temp_wf_scaled,[(0, dimdiff[0]), (0, dimdiff[1]),(0,0)],
        mode='constant')
    # Splitting data train test
    portion = int(len(aqiy_scaled) * 0.8)
    met_train, met_test = met_scaled[:portion],met_scaled[portion:]
    wf_train, wf_test = wf_scaled[:portion],wf_scaled[portion:]
    op_train, op_test = op_scaled[:portion],op_scaled[portion:]
    t_train, t_test = t_scaled[:portion], t_scaled[portion:]
    aqiX_train, aqiX_test = aqiX_scaled[:portion],aqiX_scaled[portion:]
    aqiy_train, aqiy_test = aqiy_scaled[:portion], aqiy_scaled[portion:]

    train_list = ([met_train, wf_train, op_train, t_train, aqiX_train], aqiy_train
        )
    test_list = ([met_test, wf_test, op_test, t_test, aqiX_test], aqiy_test)

    return train_list, test_list, scaler

def fit_model(model, trainX, trainy, testX, testy):


    # callbacks
    tbCallBack = keras.callbacks.TensorBoard(log_dir='logs/{}'.format(model.
        name), histogram_freq=0, write_graph=True, write_images=True)

    # Save the checkpoint in the /output folder
    filepath = 'checkpoints/{}_best.hdf5'.format(model.name)

    # Keep only a single checkpoint, the best over test accuracy.
    checkpoint = ModelCheckpoint(filepath,
                                 monitor=acc_name,
                                 verbose=0,
                                 save_best_only=True,
                                 mode='max')

    #early stopping
    es = EarlyStopping(monitor='val_loss', mode='min', verbose=0, patience=200)
    # fit network

```

```
        history  = model. fit (
            trainX ,
            trainy ,
            epochs=1000,
            #batch_size =64,
            validation_data =(testX ,  testy ) ,
            verbose=0,
            shuffle =True,
            callbacks =[
                checkpoint ,
                tbCallBack ,
                es
            ],
            #callbacks =[checkpoint , tbCallBack]
        )
        return  history



hours_to_lookback = 6
area  = "Tromso"
station  = "Hansjordnesbukta"

opt  = 'adam'
act  = 'sigmoid'
loss_name = 'mse'
acc_name = "val_rmse_acc"
loss  = data . mse

df  = data . load_data ( area ,  del_col =['temp_max','temp_min'])

for  predict_label  in  [ 'PM2.5' 'PM10', 'NO2', 'NO']:
    for  hours_to_predict  in  [1,2,3,12,24,48]:
        model_name='Dense_all_{}_{}−{}'.format(predict_label, hours_to_predict ,
            hours_to_lookback)
        a  = data . load_seq_data ( station ,  area ,  predict_label ,  hours_to_predict ,
            hours_to_lookback, df )
        train_list  ,   test_list  ,  _  =  fix_data (a [0], a [1], a [2], a [3], a [4], a [5])

        train_list_X  =  train_list [0]
        aqiy_train  =  train_list [1]
        test_list_X  =  test_list [0]
        aqiy_test  =  test_list [1]

        model = model_bank.tromso_dense5( train_list_X ,  aqiy_train ,  act=act)
```

```
        model.name = model_name
        model.compile(optimizer=opt, loss=loss, metrics=['accuracy',data.
            mse_acc,data.rmse_acc, data.r_square])
        model_yaml = model.to_yaml()

        with open("models/{}.yaml".format(model_name), "w") as yaml_file:
            yaml_file.write(model_yaml)
        print(model.name)
        fit_model(model, train_list_X, aqiy_train, test_list_X, aqiy_test)
```