**Deep Learning**

**Graded Assignment 3:**

**Multi-Modal Weather Data**

**Group 08**

Bhawna Dixit (02014095)

Thomas Kok (02013791)

Emiel Maes (01308305)

**Table of Contents**

[**Problem description**](#_7b990umyo8op) **2**

[**Data exploration and pre-processing**](#_ltzqejq03qs6) **3**

[Data exploration](#_gcfhb69gnca4) 3

[Data pre-processing](#_6f6ucsjecdfd) 7

[**Last timestep interpolation (simplest baseline)**](#_sqqg4cp0yo0o) **8**

[**Dense model (features of the last timestep)**](#_62o2a9imz01d) **10**

[**Dense model (pollution with time window)**](#_u30ey7e9jd3k) **16**

[**Convolution based model**](#_w7ynw7cypezk) **19**

[Unregularized model](#_grwjv5sgl9qk) 19

[Regularized model](#_scb1fddkfrdy) 35

[Performance on test set](#_yqkxemwhu4gd) 49

[**Recurrent based model**](#_5hdgbj68xkts) **50**

[Unregularized model](#_iro378aqiaeh) 50

[Regularized model](#_coqfcc53exey) 67

[Performance on test set](#_ar3ppm48u9rb) 72

[**Hybrid Deep Learning**](#_y314v0hmz9w3) **74**

[**Conclusion**](#_4twnimjdq8dm) **77**

[**Appendix**](#_akew4dn739q9) **78**

[Distribution of the features](#_5cmiihiolo3o) 78

# Problem description

The goal of this assignment is to predict pollution in the city of Beijing as a Citizen Warning System based on data from past pollution and weather. The dataset consists of 5 years of weather data ranging from 2010 up to 2014. These time series are multivariate which means that multiple variables are available for each time step. These variables relate to wind speed, temperature, snow, pressure, dew, rain etc. Pollution prediction in itself is challenging due to dynamic and non-linear features such as fluctuating weather conditions which is itself influenced by other variables.

In the assignment, several different model architectures will be compared with one another. In order to create these models, we first need to understand what data we are dealing with. Therefore, the first step is to do some data exploration and data pre-processing where necessary. During this data pre-processing, some additional features are created which will be discussed in the corresponding section. In total, 23 features will be available for which we have hourly measurements from 2010 up to 2014. The dataset is split into 26.280 training samples (the year 2010, 2011 and 2012), 8760 validation samples (the year 2013) and 8760 test samples (the year 2014). As we are interested in predicting the pollution value, we do not want to predict the value of the next hour but the pollution value 3 hours in the future.

Once the data pre-processing has been done, it is then possible to come up with some baseline models. Here, three different baseline models will be looked into:

* Simplest baseline model which is based on the last timestep interpolation (section 3)
* Dense model with features that are available during the last time step (section 4)
* Dense model with a pollution time window (section 5)

The results of these baseline models will then serve as a benchmark to compare them with the performance of the subsequent more complex models. The more complex models that will be looked into here relate to the Convolutional Neural Networks (section 6) and the Gated Recurrent Unit network architectures (section 7). The Convolutional Neural Networks typically make use of convolution layers for local trend feature detection. These are well known for images in which they use Conv2D layers for detecting certain features while in time series, it is possible to use Conv1D layers which can detect trends in 1 dimensional data. The Gated Recurrent Unit architecture makes use of a temporal representation learning of the features in which the network has some sort of memory and is able to remember certain special events which have an impact on the label that is being predicted.

In the final step, a Hybrid Deep Learning method is explored which is based on the idea of combination of various deep neural network architectures. The aim is to explore the performance of the combination of the previously tuned CNN and GRU on the pollution prediction problem.

# Data exploration and pre-processing

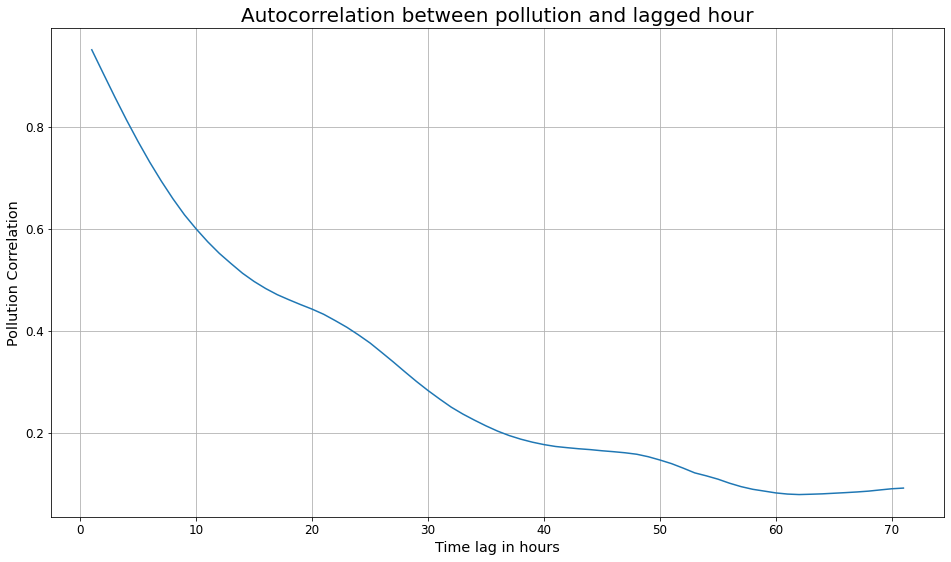
## Data exploration

The following variables are available in the provided dataset and relate to weather data of 5 years in Beijing.

* Year
* Month
* Day
* Hour
* Pollution (the variable which we want to predict later on)
* Dew
* Temperature
* Pressure
* Wind direction
* Wind speed
* Snow
* Rain

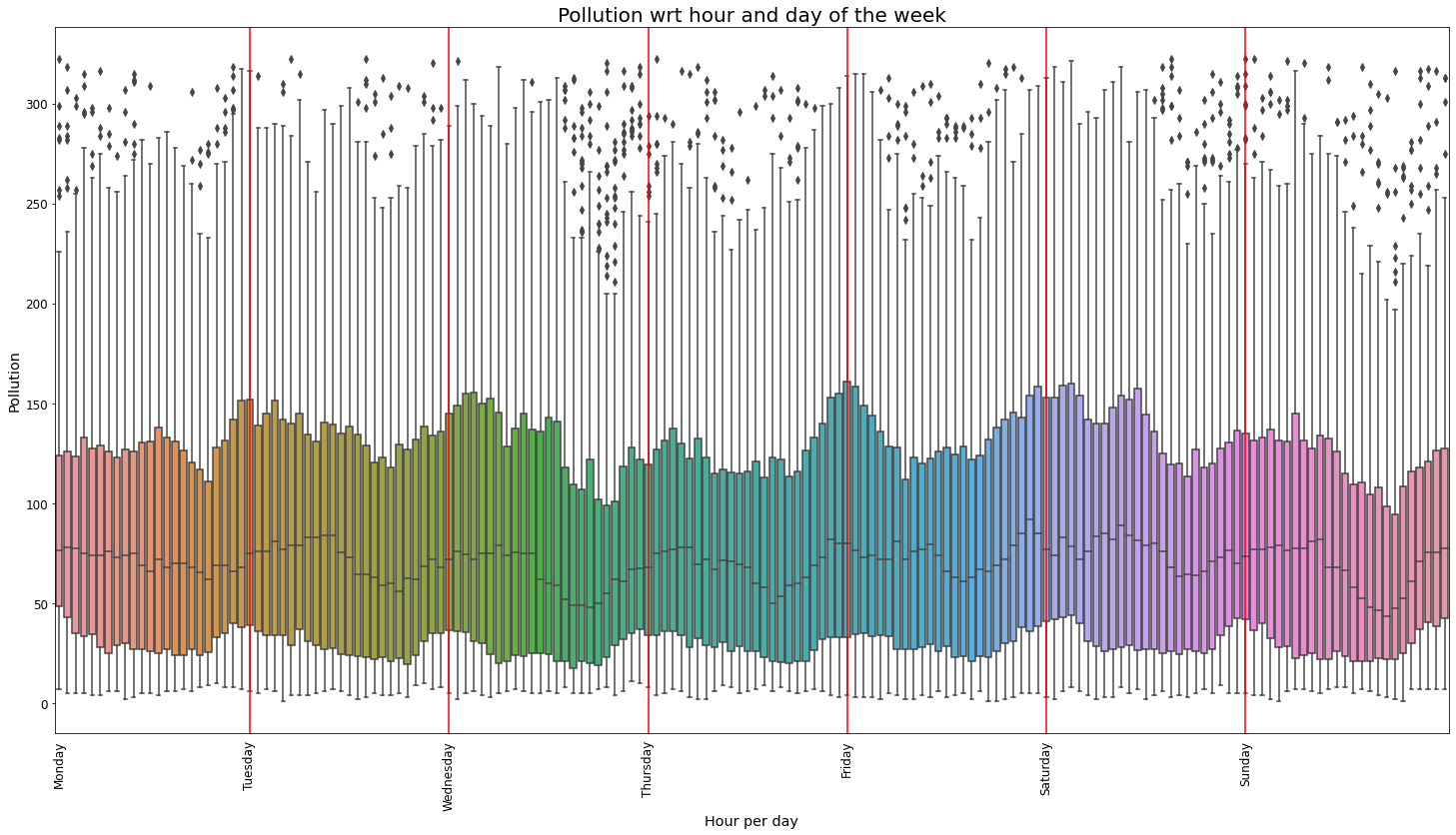
When creating a model, it is of utmost importance to understand what the data looks like and therefore, some initial data exploration steps were done. Some of the things that were investigated relate to the autocorrelation between pollution values (Figure 1), influence of time and day of the week on the pollution level (Figure 2), the distribution of the label (Figure 3) and the distribution of the features.

As can be seen from the autocorrelation plot, the lagged pollution value will be an important variable when trying to predict the future pollution value. The pollution value at time step t seems to be somewhat correlated up to the pollution value at time step t -60 hours and then fluctuates around the correlation value of 0.2. This could already indicate that when choosing a certain window size, maybe it will not be really of interest to go further than 60 hours back in time. However, from this, we cannot yet see if there are any seasonal or weekly patterns so in the end, it could still be of interest to enlarge the windows size.



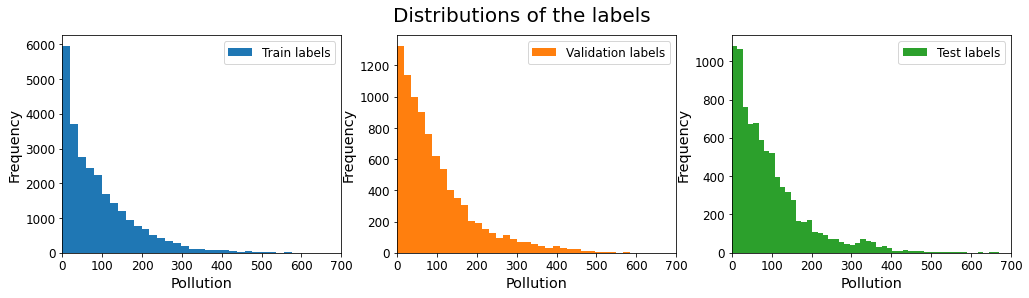
*Figure 1: Autocorrelation between pollution and lagged hour*

Figure 2 shows the pollution with respect to the hour and the day of the week. The y-axis contains the pollution value while the x-axis shows a certain hour in a day. To make the graph a bit easier to interpret, vertical red lines denote the start of a certain day which corresponds to the hour 00:00 of that day. In general, during the first 10 hours of the day, the pollution values seem to be high. Then a typical drop occurs for about 8 hours and then another increase happens during the last 6 hours of the day. This pattern seems to be followed by each day, except for Mondays. Therefore, we think it could be of interest to one-hot encode the days of the week and not just split them in weekend vs weekdays. Next to that, it could also be of interest to include a variable which keeps track of the seasonal pattern within a day. The features that are created for this will be discussed during the data pre-processing step.

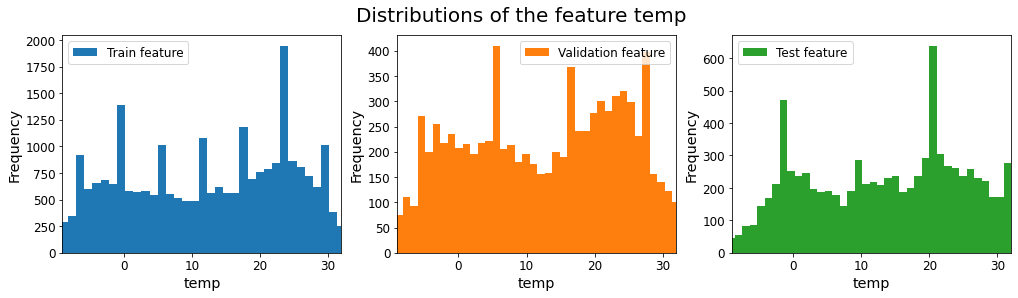


*Figure 2: Pollution with respect to the hour and the day of the week*

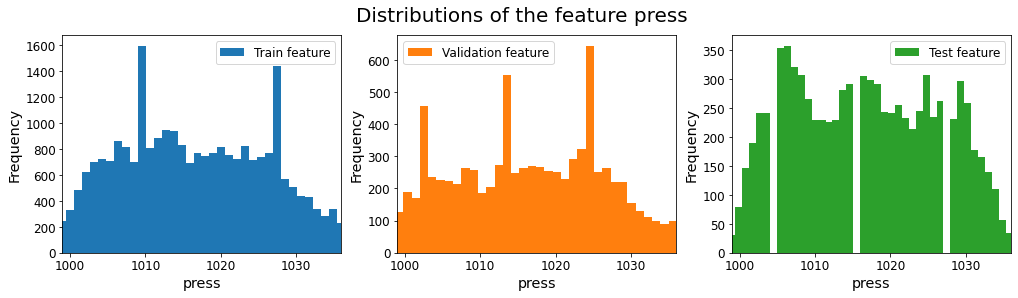
In order to be able to predict the pollution label, one needs to first check if it is actually possible and make sure that the distribution of the features and labels is the same (identically distributed) which is one of the key fundamental assumptions when applying machine learning. Figure 3 shows the distributions of the train, validation and test labels. It looks like they follow a somewhat similar distribution. However, it looks like the validation and test data have a higher density in the pollution range 20 to 100 which could make it more difficult to predict these labels. The distribution of all the features are included in the appendix. In this section, we include only the distributions of the 2 features which seem to be somewhat different in each of the sets (the other feature distributions are included in the appendix). As can be seen from figure 4, it looks like the temperature feature follows a slightly different distribution in the validation set compared to the train and test set. Another interesting finding relates to the pressure feature (Figure 5) which seems to have higher values in the test set compared to the pressure values in the train and validation set. Next to that, it also has certain gaps in the test data. These are some important observations which can influence the performance of the model later on.



*Figure 3: Distribution of the pollution labels*



*Figure 4: Distribution of the temperature feature*

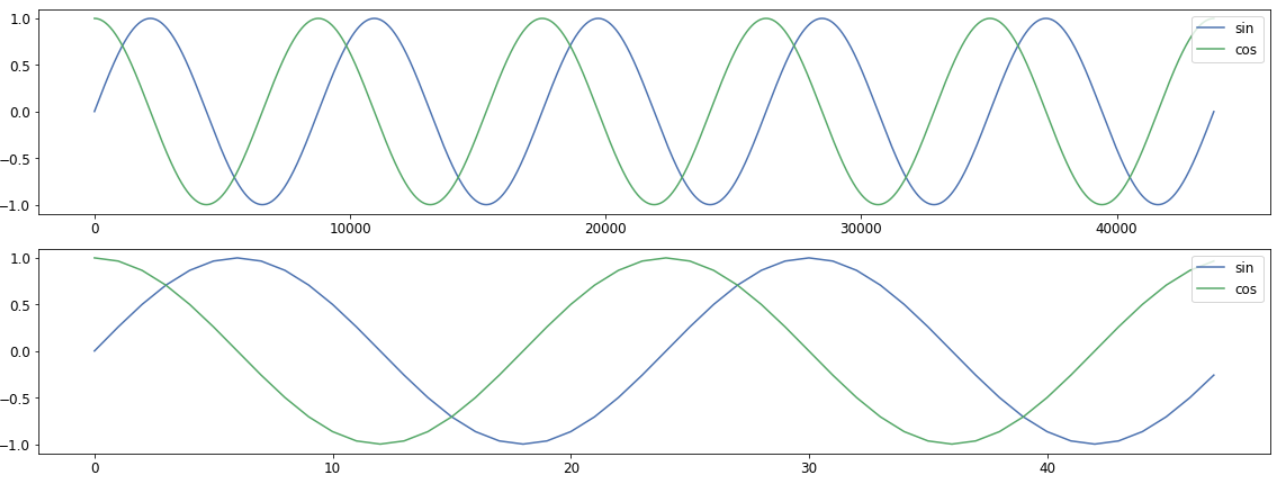


*Figure 5: Distribution of the pressure feature*

## Data pre-processing

As can be seen from the data, quite some features are discrete (year, month, day and hour), while others are categorical (wind directions). In order to work with them, some data preprocessing is needed. This pre-processing is based on domain knowledge and is meant to convert the features into one-hot encoded features or continuous variables. The following data-preprocessing steps were done:

* Translating the day-of-the-year into periodic features by transforming this discrete feature into sines and cosines which can then represent a trend which is basically a linear combination of a sine, cosine and a constant value. A visualization on how this sin and cos relate to the day of the year is given in the upper part of figure 6.
* Translating the hour of the day into periodic features by transforming this discrete feature into sines and cosines. This extra feature is added because of our initial data exploration which showed that this feature could be of importance for understanding the fluctuations in the pollution level. A visualization on how this sin and cos relate to the hour of the day is given in the lower part of figure 6. Here, one can see that the sin and cos are repeated every 24 hours and thus can reflect a certain hourly pattern.
* Converting the day of the week into weekday or weekend which could reflect working days vs non-working days. However, this feature will probably be less of importance as already could be seen from the exploratory data analysis.
* Creating one hot encoded variables for each day of the week. As could be seen from the exploratory data analysis, the day of the week could be a factor which determines the pollution level.
* Converting the wind direction features into one-hot encoded variables which are multiplied with their corresponding wind speed



*Figure 6: Sin and cos features which reflect certain trends in the data*

# Last timestep interpolation (simplest baseline)

The simplest baseline is to simply interpolate the pollution levels of the last known timestep. Since we are predicting 3 hours in advance, we take the most recent information we can use for the prediction and simply use that value.

Simply based on what we know about this prediction, we can expect that it will never be too far off: its error is simply how much the pollution has changed in the last 3 hours. Even though the data is somewhat volatile, within the time of just 3 hours it is usually at least somewhat similar.

The results on the validation and test set are as follows:

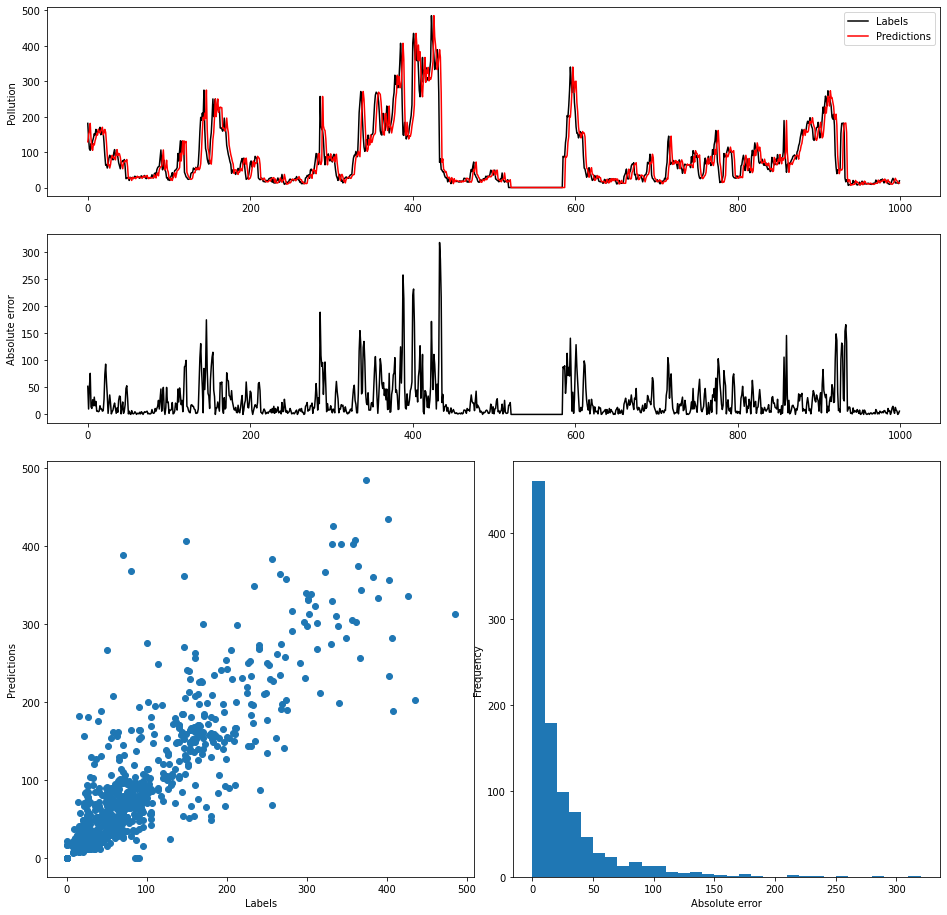
"Current pollution" baseline:

train mae = 26.28013

validation mae = 28.706179

test mae = 26.04659

These values provide us a baseline that we should always aim to beat, as well as some information about the datasets. The validation set seems to be more volatile in the short term and perhaps somewhat harder to predict, than the train and test set.



*Figure 7: Comprehensive results of the last known value*

*interpolation method on the validation dataset.*

Finally, it should be noted that even if this prediction is not that bad, it is in the end always useless in a practical setting.

# Dense model (features of the last timestep)

We will consider first perhaps the simplest actual model for our problem. This model is based on the assumption that any temporal information in the data is not relevant, and in fact everything to expect can be based on just the current state and not the history. This is most likely not actually the case, but an assumption as such allows us to give a nice benchmark model like this.

For this, we first consider all the features of the last timestep, and will next try to see if we can reduce the size of the set of features used by removing uninformative features.

To find the best model architecture, we used the two-step approach as in previous assignments: first, overfit the network, then add dropout layers and other regularization techniques. As this is simply a baseline model, we will not go in too much detail about every configuration tried, but the final model is as follows (with the learning rate set to 0.01, ridge bias to 1e-3 and batch size to 128):

Model: "sequential\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dropout\_4 (Dropout) (None, 23) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_4 (Dense) (None, 64) 1536

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_5 (Dropout) (None, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_5 (Dense) (None, 64) 4160

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_6 (Dropout) (None, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_6 (Dense) (None, 32) 2080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_7 (Dropout) (None, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_7 (Dense) (None, 1) 33

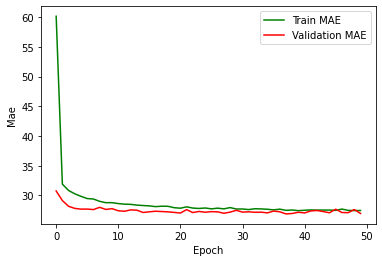
=================================================================

Total params: 7,809

Trainable params: 7,809

Non-trainable params: 0

The model finds its optimal validation performance very quickly, with a validation mean absolute error of 26.9595, noticeably lower than the last pollution baseline of around 28.7.



*Figure 8: Dense model with features of the last time step*

The more comprehensive results show nicely the performance of the model. The main improvement over the last model seems to be the ability to predict extreme outliers better, as there seem to be less of them (although still some).

Next, we consider the specific features that are used as input to see if any of them do actually impact the performance. We will start with the entire feature set, as just assumed. Step by step, we might consider potential features to exclude.

There are 23 features in total, but we will not remove all of them, and some will be combined to be removed all at once. The pollution feature will not be removed, and the following groups of features will be removed in groups:

* Dew
* Temperature
* Pressure
* Wind speed
* Snow, rain
* Seasons\_cos, Season\_sin
* Hour\_cos, Hour\_sin
* Weekdays, is\_Monday, is\_Tuesday, .., is\_Sunday
* NE\_strength, NW\_strength, NS\_strength, cv\_strength

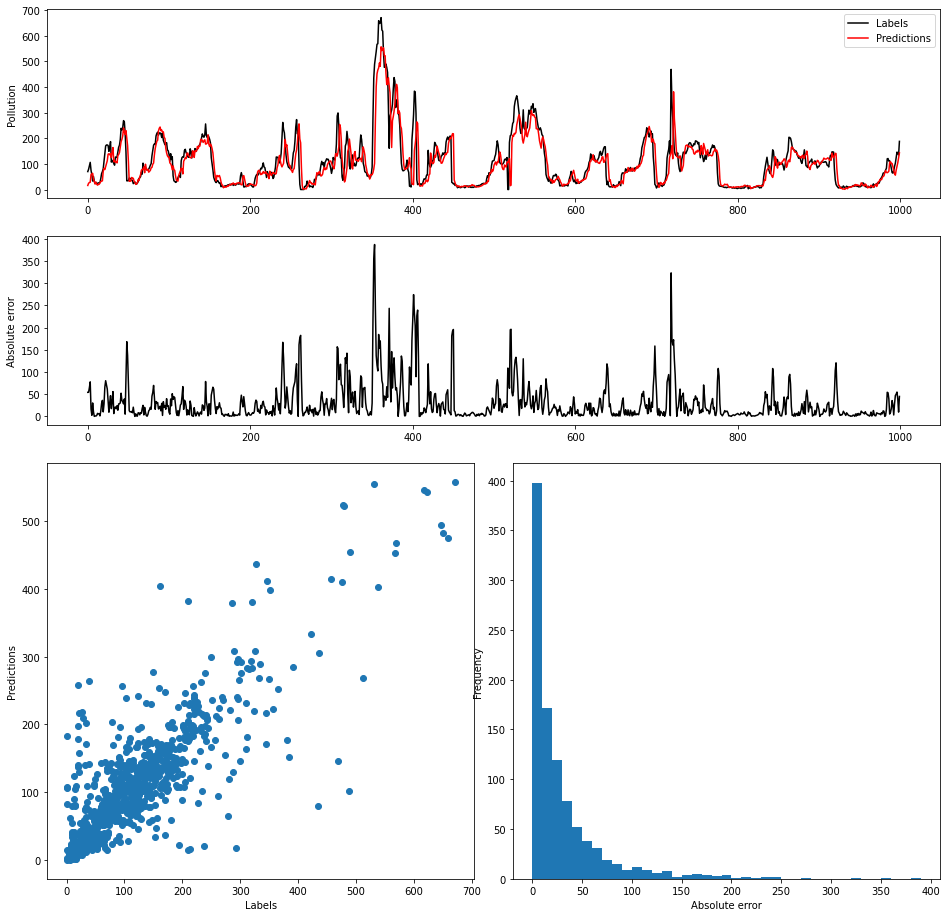
Table 1 shows the learning curves and validation mean absolute errors for each removed feature (set):

*Table 1: Performance overview when removing certain feature set*

| **Removed feature set** | **Learning curve** | **Validation mae** |
| --- | --- | --- |
| Dew |  | 27.405445 |
| Temperature |  | 27.27985 |
| Pressure |  | *26.818771* |
| Wind speed |  | *26.865334* |
| Snow and rain |  | 27.182186 |
| Seasons\_cos / sin |  | 27.855312 |
| Hour\_cos / sin |  | 28.173111 |
| Days |  | 27.516392 |
| \_strength |  | 27.90984 |

While we do not want to draw definite conclusions just from these results, as they are somewhat fluctuating and the differences are not enormously large, they are still interesting. There is a notable difference between the results of several removed feature subsets, with some features seeming to be more essential than others. We can keep this in mind in the later step where we will be tuning more extensive convolutional and recurrent networks, in potential dropping of features. Most features actually seem useful, with the mean absolute error increasing when removed. The most notable feature sets in increasing error seem to be the sinoide features, and the strength related features. On the other hand, the pressure and wind speed features are the only ones that actually cause a decrease in the error when removed. Perhaps these two features are not particularly useful and provide unnecessary or conflicting information.

The final results when re-training on the full training data set (with all features) and evaluating the performance on the test dataset shows a nice improvement on the last pollution baseline. The mean absolute error is decreased to 24.899323 which is a significant improvement. The comprehensive results can be seen in figure 9.

**

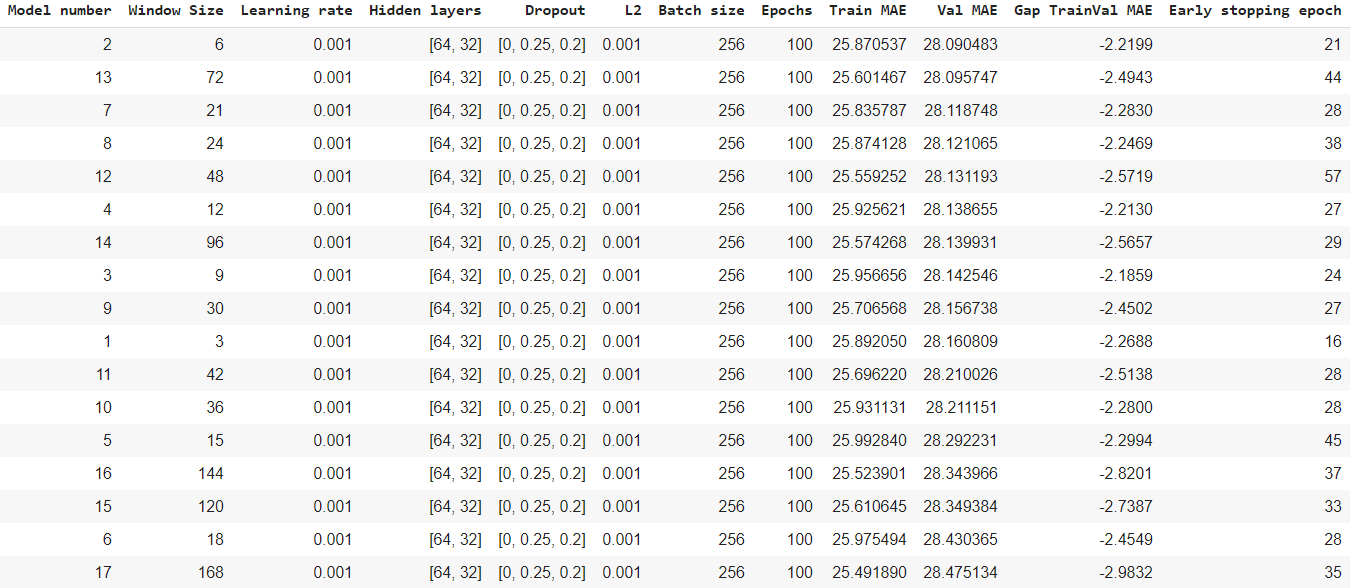
*Figure 9: Performance on the test dataset.*

# Dense model (pollution with time window)

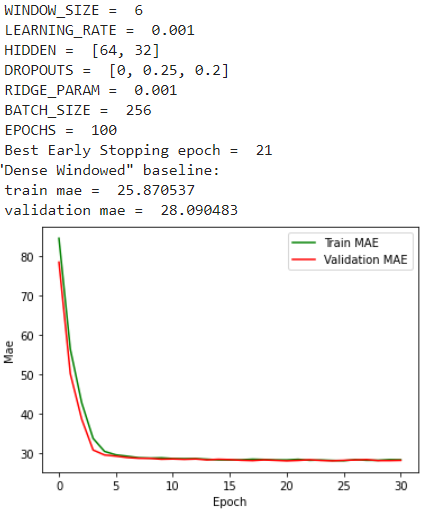
Different window sizes were tried out with an optimal model architecture which was found by doing a hyperparameter sweep. When doing the hyperparameter sweep, several batch sizes (128, 256), learning rates (0.01, 0.001, 0.0005), hidden layers (2, 3, 4), number of neurons per hidden layer (16, 32, 64, 128), L2 regularization (1e-3, 1e-4) and dropout layers (0.2, 0.25, 0.3, 0.4) were tried out. As it is not the focus of the assignment to discuss this hyperparameter tuning process, we decided to not discuss this much in detail but similar steps were taken as during the first two graded assignments (first, overfitting the training data and then adding regularization to close the gap between the train and validation MAE). During this iteration process, we focused on making sure to have a stable training and validation MAE. This means that there are no heavy fluctuations in both the training and validation errors. When high differences in the validation errors would be observed, this would be an indication that a lower-variance model would be preferred which is not the case in our situation!

Having found a good model architecture, the next step was to investigate in different window sizes and see how important the historical values of the pollution are. The window sizes which were investigated are the following ones: [3, 6, 9, 12, 15, 18, 21, 24, 30, 36, 42, 48, 72, 96, 120, 144, 168]. The performances are shown in Table 2. In general, it does not really look like the window size is super important as the validation MAE for most of the window sizes seem to be somewhat similar and are between 28.09 and 28.47.

*Table 2: Overview performance of the optimal model with different window sizes*

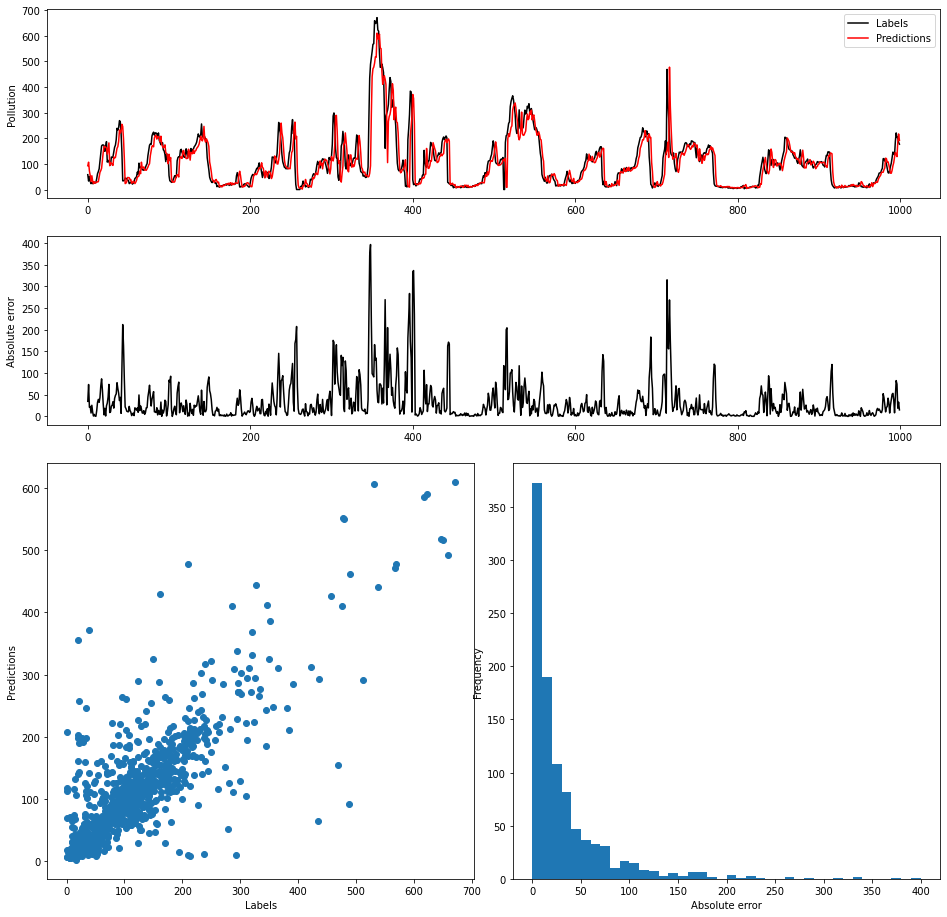


As in our situation the window size of 6 gave the best validation performance (MAE of 28.0904), along with a smooth validation curve (Figure 10), we decided to choose this window size and re-train the model on the full training data set and evaluate the performance on the test set. The best epoch was reached after 21 epochs, so we chose this as the number of epochs to train on the full training data set. Training on this full training set gives us an MAE of 26.50 on the training data along with an MAE of 25.51 on the test set. This result is shown in Figure 11.



*Figure 10: Model performance with a window size of 21*

*and the corresponding model architecture*

**

*Figure 11 : Performance of best window based model on the test set*

# Convolution based model

## Unregularized model

The initial starting architecture was based on a blog by Machine Learning Mastery[[1]](#footnote-0) where they apply time series classification with a CNN1D model. As we are dealing with regression, we had to change a few things to their initial model architecture so the one we used to start looks as follows:

* Block 1: 3x3 Conv1D 64 - Relu - 3x3 Conv2D 64 - Relu - Maxpool 2x2
* Flatten layer
* Dense layer with 64 neurons, Relu activation
* Dense layer with 1 neurons, Linear activation

Learning rate of 0.001

Number of epochs: 30

Batch size: 32

CNN window size of 6 was used as this was supposed to be the best window size during the previous step. However, as the features did not really show that one was explicitly better than the other, we opted to include all of them.

*Table 3: Tuning process for the unregularized CNN model*

| **Step** | **Model architecture or hyperparameter which is being tuned** | **Optimal architecture at the end of this step** |
| --- | --- | --- |
| 1 | Starting model which is the  initial architecture (as defined above) which is based on the blog from Machine Learning Mastery | The **initial model architecture** that was described above. This model gives the following performance which already looks quite good for an initial model. As it looks like the number of epochs is too low in order to obtain convergence on the training data, we decide to increase it to 100. |
| 2 | Check performance when increasing the number of epochs | **Same architecture as the one in step 1, but now with 100 epochs**  As can be seen, the performance of the model is still not converging but we however, decide to continue with 100 epochs and make the model more powerful such that it might be able to converge faster. |
| 3 | Defining the number of blocks | Adding an additional block with the following structure 3x3 Conv1D 128- Relu - 3x3 Conv1D 128- Relu - Maxpool 2x2 resulted in a more powerful model as the train MAE managed to reach to 9.21 after 100 epochs. When adding 2 more blocks (similar to the one added above) but with respectively 128 channels and 256 channels, the model would not be constructed due to a shape mismatch. This leaves that we will continue with the 2 blocked CNN. **The optimal number of blocks to add was 1 block with 128 channels.** |
| 4 | Defining the number of layers per block | For this step, we try to see if the number of Conv1D/ReLu layers in each block is appropriate. The number of such layers was 2 in step 3, we tried 1 and 3 here (with the same neurons as the other layers in the corresponding block, thus 64 in the first block and 128 in the second block). When using 1 layer in each block, a train MAE of 14.54 was obtained. However, when using 3 layers in each block, a train MAE of 9.23 was obtained. Both performances are visualized below.  **None of the configurations really improved upon the 2 layer-block**. This means that the optimal architecture after this step is still the same as the one obtained from the previous step and we decide to continue with the 2 layer block. |
| 5 | Number of channels per block | At the moment, we have two blocks with corresponding number of channels in each block: 64 and 128. We will keep the non-decreasing (because this makes most sense with CNNs conceptually) number of channels, but try several different configurations (different number of channels within a block). The architectures which are tried out relate to the following ones:   * Block 1 with 32 channels, Block 2 with 64 channels: train MAE 14.54 * Block 1 with 16 channels, Block 2 with 32 channels: train MAE 19.51   Both models do actually not improve the training performance but from the last architecture with **16 channels in the first block and 32 channels in the second block**, we do somewhat see a **smoother training curve** and therefore, we conclude to **continue with this model**. Next to that, it should also be mentioned that in this case with time series models, when the model is too powerful, one can still try to regularize it but in practice, it is quite difficult to remove overfitting in this case. This is another reason on why to continue with the simpler model. |
| 6 | Window size | In order to see if the window size has a big impact on the model architecture, the 3 window sizes that gave the best performance in our previous dense, window based model are explored. The window sizes that are explored are thus 6, 21 and 72. This basically tells us how much history to use.   * Window size of 6: train MAE 19.51 * Window size of 21: train MAE 16.86 * Window size of 72: train MAE 10.54   It seems that the larger window sizes do not really converge much faster than the one of 6. Therefore, **we conclude to just continue with a window size of 6.** |
| 7 | Filter size | Very small filters combine information in short time windows. This gives you a pattern kind of hierarchy. Very large filters for example as large as the time window would basically just result in a dense neural network. Very large filters also result in much more weights which makes the model again more complex which is not preferred in this use case. Kernel sizes of 2, 3, and 4 are used in each block.   * Kernel size of 2: train MAE 20.44 * Kernel size of 3: train MAE 19.51 (see previous step 6) * Kernel size of 4: train MAE 18.88       At the moment, we are still feeding each feature as a separate channel into the CNN model. However, maybe it could be useful to start from a different set of features by first mixing the original features and then getting useful information out of these. This can be done by setting the **filter size of the first** **Conv1D layer to 1x1**. This combination is tried out **with a filter size of 3 in the other Conv1D layers.** The result is somewhat similar to the other model architectures (train MAE of 20.24 see figure below) but we decide however to **continue with this architecture** (along with the kernel sizes of 3 in the other conv1D layers) as this seems to give a somewhat very steep, followed by a flatter training curve. |
| 8 | Pooling type | Max pooling is better able to extract the most important features like peaks while average pooling extracts more smooth features. Therefore, we expect that the MaxPooling1D will result in higher performance. However, we also looked at replacing the current MaxPooling1D layers with AveragePooling1D layers. Next to that, the performance was also evaluated when replacing the last MaxPooling1D by a GlobalMaxPooling1D.   * AveragePooling1D : train MAE 20.97 * GlobalMaxPooling1D: train MAE 20.38   However, none of these seemed to give any improvement over the current best model and therefore, **we decided to not change anything to the optimal model architecture so far**. |
| 9 | Number/neurons of dense layers | Here, we tried some other architectures (both wider and deeper) in the dense layers. Some of the architectures we tried, look as follows and can be seen below with their performances:   * 64->32->1: train MAE 19.67 * 128->1: train MAE 19.80 * Flatten Layer -> 1: train MAE 21.31 * 16->1: train MAE 21.40 * 4->1: train MAE 21.09             None of them seemed to really improve the existing performance. Therefore, we chose to **continue with the existing model architecture as this gave the best trade-off between a steep training curve and computations that need to be done.** |
| 10 | Batch normalization (before activation function) | Another adjustment to the model architecture relates to including a batch normalization layer before the activation function. When it was applied, not much difference could be detected. Therefore, **we decided to not include the batch normalization layers**. |
| 10 | Initial learning rate and batch size | According to our training curves, it looks like we already have a good initial learning rate and batch size, but some other configurations were tried as well to double check on how they would affect the performance. The initial learning rate was changed between 0.01 and 0.0001. Also a bigger and smaller batch size of respectively 64 and 16 were tried out.  Setting a too low learning rate combined with a too low batch size results in high fluctuations. Therefore, we decided to continue with the learning rate of 0.0001 and a batch size of 32 as this gave both a flat training and validation curve.   * Learning rate 0.01 and batch size 16: train MAE 22.74 * Learning rate 0.0001 and batch size 32: train MAE 23.30 * Learning rate 0.0001 and batch size 64: train MAE 22.96 * Learning rate 0.001 and batch size 64: train MAE 20.86 * Learning rate 0.001 and batch size 32: train MAE 21.02 |

The final model architecture for the unregularized CNN model looks as follows and its performance is visualized below.

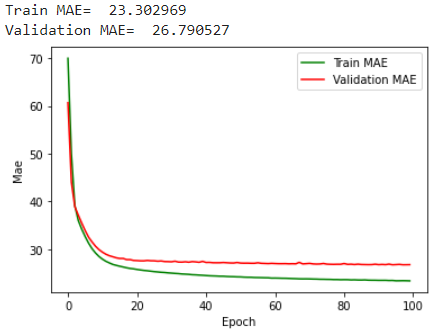
* Block 1: 1x1 Conv1D 16-Relu - 3x3 Conv1D 16- Relu - Maxpool 2x2
* Block 2: 3x3 Conv1D 32-Relu - 3x3 Conv1D 32- Relu - Maxpool 2x2
* Flatten layer
* Dense layer with 64 neurons, Relu activation
* Dense layer with 1 neuron, Linear activation

Learning rate of 0.0001

Number of epochs: 100

Batch size: 32

Window size: 6



*Figure 12: Train and validation performance*

*for the unregularized CNN model*

## Regularized model

The next step is now to fine tune the unregularized model by adding different regularization techniques. The following tables describe the tuning process in which we look at the best configurations for dropout layers, L1 and L2 regularization and maxnorm.

*Table 4: Tuning process for the dropout configuration of the CNN model*

| **Step** | **Dropout configuration** | **Train/validation curves** |
| --- | --- | --- |
| 1 | 3 dropout layers were applied in which two layers were applied at convolutional blocks and the last dropout was applied at dense layer in a sequentially increasing manner (with batchnorm model).  0.1 (Conv1D)  0.2 (Conv1D)  0.3 (Dense 64) |  |
| 2 | Dropout of 0.1 was applied after each conv1D block and at the first dense layer. |  |
| 3 | The dropout of 0.2 was applied at Dense 64 layer. |  |
| 4 | The dropout of 0.1 was applied after the Dense 64 layer. |  |
| 5 | The dropout of 0.3 was applied at Dense 64 layer. |  |
| 6 | The dropout was applied to the Conv1D blocks (p=0.1, p=0.2) and Dense layer (p=0.2). |  |
| 7 | The dropout was applied to the last Conv1D block (p=0.1) and Dense layer (p=0.2). | Train MAE = 23.401354  Validation MAE = 27.60673 |

From the dropout tuning, the fourth model (with a dropout of 0.1 after the dense layer) was selected for further tuning with L1 and L2 regularization.

*Table 5: Tuning process for the L1 and L2 configurations of the CNN model*

| **Model** | **L1 and L2** | **Performance of the corresponding model** |
| --- | --- | --- |
| 1 | Set the L2 in the Conv1D layers to  L2 = 1e-3 |  |
| 2 | Set the L2 in the Conv1D layers to  L2 = 1e-2 |  |
| 3 | Set the L2 in the Conv1D layers to  L2 = 1e-1 |  |
| 4 | Set the L2 in the Conv1D layers to  1e-1 and L2 in the dense layer to 1e-3 | As can be seen from steps 1,2 and 3, the best performance was obtained by setting the L2 to 1e-1 for the Conv1D layers. Now, we will fine tune the L2 for the dense layers. |
| 5 | Set the L2 in the Conv1D layers to  1e-1 and L2 in the dense layer to 1e-2 |  |
| 6 | Set the L2 in the Conv1D layers to  1e-1 and L2 in the dense layer to 1e-1 |  |
| 7 | L2 Conv1D: 1e-1  L2 Dense: 0  L1 Conv1D: 1e-3  L2 Dense: 0 | The steps 4, 5 and 6 showed that by using L2 in the dense layers, the performance is not being improved and therefore we continue with an L2 of 1e-1 in the Conv1D layer and no L2 in the dense layer. In the next steps, we will fine tune the L1 for the Conv1D layer and then also for the dense layer. |
| 8 | L2 Conv1D: 1e-1  L2 Dense: 0  L1 Conv1D: 1e-2  L2 Dense: 0 |  |
| 9 | L2 Conv1D: 1e-1  L2 Dense: 0  L1 Conv1D: 1e-3  L2 Dense: 1e-3 |  |
| 10 | L2 Conv1D: 1e-1  L2 Dense: 0  L1 Conv1D: 1e-3  L2 Dense: 1e-2 |  |
| 11 | L2 Conv1D: 1e-1  L2 Dense: 0  L1 Conv1D: 1e-3  L2 Dense: 1e-4 |  |

The best regularized model so far was obtained in step 7 after 113 epochs and has the following characteristics:

* A dropout layer of 0.1 after the dense layer
* L2 Conv1D: 1e-1
* L2 Dense: 0
* L1 Conv1D: 1e-3
* L2 Dense: 0

The final step is now to see if the performance can even be further improved by adding max norm to the Conv1D or dense layers. This is discussed below in Table 6.

*Table 6: Tuning process for the Max norm configurations of the CNN model*

| **Model** | **Maxnorm** | **Performance of the corresponding model** |
| --- | --- | --- |
| 1 | Maxnorm in the Conv1D layers set to 3 |  |
| 2 | Maxnorm in the Conv1D layers set to 2 |  |
| 3 | Maxnorm in the Conv1D layers set to 1 |  |
| 4 | Maxnorm in the dense layer set to 3 |  |
| 5 | Maxnorm in the dense layer set to 2 |  |
| 6 | Maxnorm in the dense layer set to 1 |  |

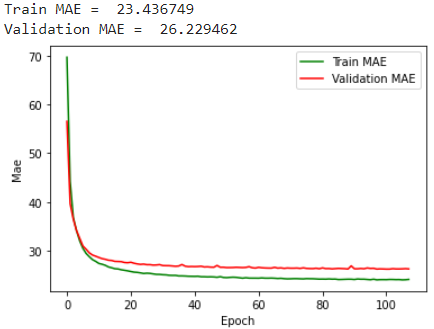
The final model architecture for the regularized CNN model was obtained during step 5 in Table 6 and looks as follows. Its performance is visualized below in figure 13.

* Block 1: 1x1 Conv1D 16-Relu - 3x3 Conv1D 16- Relu - Maxpool 2x2 (with L1 = 1e-3 and L2 = 1e-1 in the Conv1D layers)
* Block 2: 3x3 Conv1D 32-Relu - 3x3 Conv1D 32- Relu - Maxpool 2x2 (with L1 = 1e-3 and L2 = 1e-1 in the Conv1D layers)
* Flatten layer
* Dense layer with 64 neurons (with Maxnorm of 2), Relu activation
* Dropout layer of 0.1
* Dense layer with 1 neuron, Linear activation

Learning rate of 0.0001

Batch size: 32

Window size: 6



*Figure 13: Train and validation performance*

*for the regularized CNN model*

## 

## 

## 

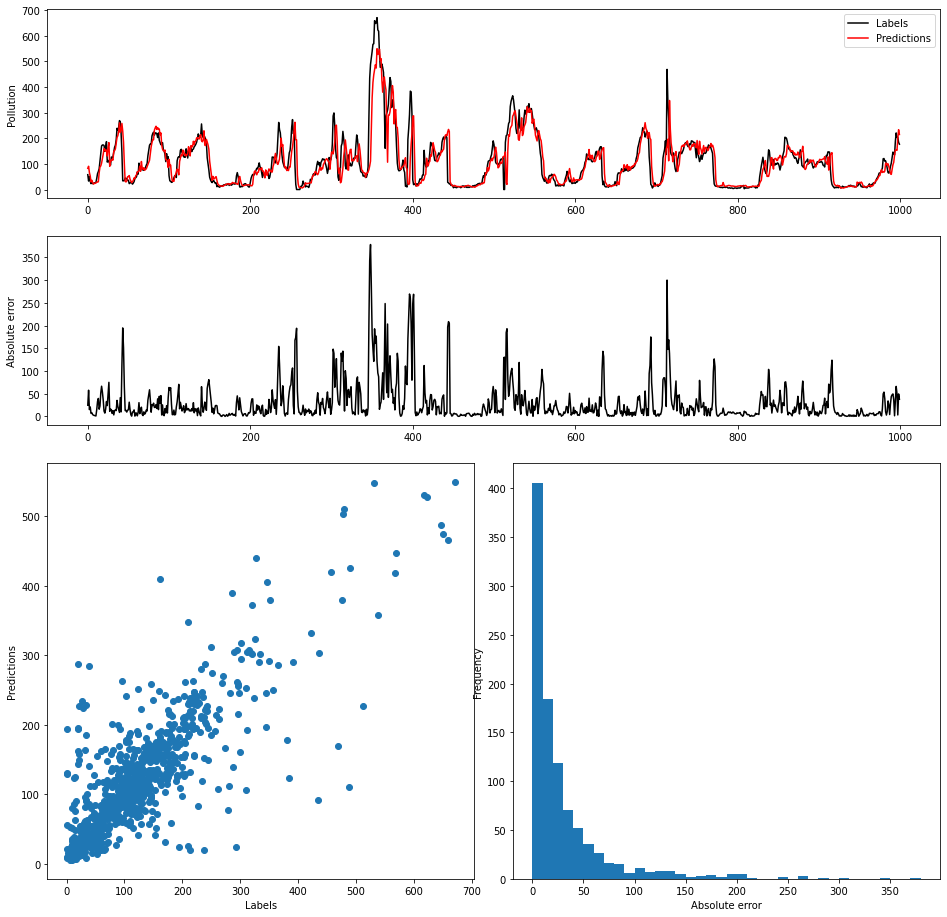
## 

## 

## 

## Performance on test set

Once we have found the best regularized CNN model, the next step is now to re-train the model on the full training data set and evaluate the performance on the test set. Our best model was reached at epoch 101. Therefore, we chose this as the number of epochs to train on the full training data set. Training on this full training set gives us an MAE of 24.2761 on the training data together with an MAE of 23.6714 on the test set. The performance on the test set is shown in figure 14 which shows that the model is quite good at predicting the future pollution and is even able to anticipate peaks.



*Figure 14: Final CNN model performance on the test set.*

# Recurrent based model

## Unregularized model

We based our initial model design on several sources (blogs and papers),[[2]](#footnote-1)[[3]](#footnote-2) and decided to start simple. As noted in the assignment, a simpler GRU network is preferred for this problem. As such, we start with one GRU layer where we will first tune the number of neurons, followed by one Dense layer, where again we will tune later to see if additional Dense layers can improve our network.

The first parameter we tune is the number of GRU neurons, with the rest of the network looking as follows:

* GRU layer with \_ neurons
* Dense layer with 1 neuron, linear/no activation layer
* Window size = 6
* No dropout layers, no regularization
* Batch size = 128
* Learning rate = 0.001
* Adam optimizer

*Table 7: Tuning process for the unregularized GRUmodel*

| **Step** | **Model architecture or hyperparameter which is being tuned** | **Optimal architecture at the end of this step** |
| --- | --- | --- |
| 1 | Number of neurons in GRU layer | We tried several numbers of neurons for this layer, ranging from very simple (4) to more complex (64) using powers of 2. In a general sense, increasing the number of neurons seemed to cause 2 effects: 1) decreasing the number of epochs needed to train the model, and 2) increasing the overfitting of the model.  We selected 16 as the optimal number of neurons for the GRU layer. Because this is a regression problem, and we are using the GRU model which is harder to regularize, we are not aiming to overfit as much as possible. Instead, we are aiming to find a stable training process, which 16 neurons provides the best.  4:  train mae = 24.838013  validation mae = 28.446884    8:  train mae = 23.532244  validation mae = 27.385284    16:  train mae = 21.896729  validation mae = 27.792086  32:  train mae = 19.06591  validation mae = 29.606064    64:  train mae = 12.415097  validation mae = 34.40753 |
| 2 | Dense layer configuration | We used a single dense layer with a single neuron as a basis, but we can surely find improvement. We will consider adding layers previous to it one-by-one. In the next step, we will also consider the used activation function, and use relu for now.  First thing we can note from looking at the learning curves, is that adding dense layers (even just a single additional one) allows the network to train much faster. Adding more layers allows even faster training.  For the results, intuitively we expected that the number of neurons in the first dense layer should not be more than the number of neurons in the GRU layer, but this did not turn out to be the case. The best configuration seemed to be 4 layers, with 32, 16, 8, and 4 neurons. However, 3 dense layers with 32, 16 and 8 neurons provided similar results, and this also was the simplest network that removed the early plateau from the curves. As such, we went with this configuration of 3 dense layers.  Adding 1 dense layer with 10 neurons:  train mae = 24.09318  validation mae = 27.228727    Adding 2 dense layers with 16 and 8 neurons:  train mae = 23.874304  validation mae = 26.857088    Adding 3 dense layers with 16, 8, and 4 neurons:  train mae = 23.395298  validation mae = 26.561802    Adding 3 dense layers with 16, 16, 8 neurons:  train mae = 23.946524  validation mae = 26.96116    Adding 3 dense layers with 32, 16, 8 neurons: train mae = 23.394793  validation mae = 26.480997    Adding 4 dense layers with 32, 16, 8, and 4 neurons:  train mae = 23.343103  validation mae = 26.293797    Adding 4 dense layers with 16, 16, 8, and 8 neurons:  train mae = 23.533367  validation mae = 26.68131    Adding 5 dense layers with 64, 32, 16, 8, and 4 neurons:  train mae = 23.456972  validation mae = 26.566074 |
| 3 | Dense layer activation function | We used the relu activation function for the dense layers for the previous step (so excluding the final dense layer, which will stick with a linear/no activation function). We will consider several activation functions for this step.  For all the four different activation functions we considered, there was no improvement over relu. As such, we will stick with relu.  With the elu activation function:  train mae = 23.43642  validation mae = 26.402134    Sigmoid:  train mae = 65.095  validation mae = 67.84425    Tanh:  train mae = 65.09499  validation mae = 67.841965    Selu:  train mae = 23.674873  validation mae = 26.443207 |
| 4 | Window size / sequence length | While we kept a consistent window size (or sequence length) so far, we will experiment with several values to see if there is an improvement of performance.  We can notice from the performance metrics and the training and validation curves at different window sizes, that the most important part is that the window is simply that it is large enough to catch the necessary information, but scaling it up beyond that does not seem to add much needed information. The best curves seem to be around a window size of 6 to 8, with a larger window size being slightly but not significantly worse. This probably means that most information is available in the last 6 to 8 hours, with information beyond that not being very relevant. For this reason, we will stick with a window of 6 hours.  Window size = 4:  train mae = 23.755701  validation mae = 26.636549    Window\_size = 8:  train mae = 23.464355  validation mae = 26.387852    Window\_size = 12:  train mae = 23.383375  validation mae = 26.739138    Window\_size = 18:  train mae = 23.55114  validation mae = 26.689995    Window\_size = 24:  train mae = 23.226513  validation mae = 26.422073    Window\_size = 48:  train mae = 23.439747  validation mae = 26.625498 |
| 5 | Initial batch size and learning rate | Finally, for the unregularized model we will experiment with the optimal batch size and learning rate. The aim is to try to stabilize the training and validation curves, with batch size and learning rate values that make sense.  We will initially try increasing and decreasing the batch size and the learning rate, to see which direction might have improvements.  It became obvious quickly that increasing the learning rate is not a good idea: it cause generally higher errors as well as a volatile training and validation curve. Dropping it all the way to 0.0001 seemed to not work that well either, although more stable curves usually had slightly higher errors. For the batch size, interestingly, 128 seemed already a quite nice value. Increasing it to 256 or decreasing it to 32 gave clearly worse results. However, a batch size of 64 gave somewhat better results. To compensate for the less stable training, we updated the learning rate to 0.005. This gave the best results combined.  Learning rate = 0.01, batch size = 128:  train mae = 22.273584  validation mae = 27.936316    Learning rate = 0.01, batch size = 256:  train mae = 22.65271  validation mae = 27.614883    Learning rate = 0.001 (no change), batch size = 64:  train mae = 23.073267  validation mae = 26.325333    Learning rate = 0.001 (no change), batch size = 32:  train mae = 22.588894  validation mae = 26.885506    Learning rate = 0.001 (no change), batch size = 256:  train mae = 23.897186  validation mae = 26.776894    Learning rate = 0.0001, batch size = 128:  train mae = 24.49181  validation mae = 26.80342    Learning rate = 0.0001, batch size = 64:  train mae = 24.157148  validation mae = 26.682024    Learning rate = 0.0005, batch size = 64:  train mae = 23.735086  validation mae = 26.484436 |

The final model architecture for the unregularized RNN GRU model looks as follows and its performance is visualized below.

* GRU layer: 16 hidden neurons
* Dense layer with 32 neurons, Relu activation
* Dense layer with 16 neurons, Relu activation
* Dense layer with 8 neurons, Relu activation
* Dense layer with 1 neuron, Linear activation

Learning rate of 0.0005

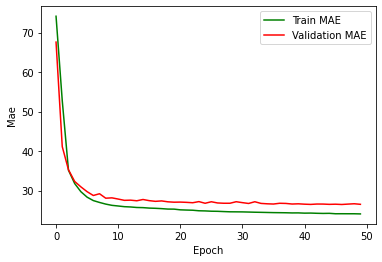
Number of epochs: 50

Batch size: 64

Window size: 6

train mae = 23.976143

validation mae = 26.541172



*Figure 15: Train and validation performance*

*for the unregularized RNN model*

## Regularized model

The next step, as it always is, is regularizing the model.The steps are discussed in table 8.

*Table 8: Tuning process for the regularized GRU model*

| **Step** | **Model architecture or hyperparameter which is being tuned** | **Optimal architecture at the end of this step** |
| --- | --- | --- |
| 1 | L2 for GRU layer | We tried all values ranging from 1, 0.1, to 1e-6. This was applied using the kernel\_regularizer parameter. This led to the following results (using early stopping, and best model saving based on val mae, thus eliminating the influence of volatility). Some values that seem promising after the first run, are run again to check for variability.  L2 = 1: 26.71719  L2 = 0.1: 26.46452 | 26.46620  L2 = 0.01: 26.50187  L2 = 1e-3: 26.52519  L2 = 1e-4: 26.35475 | 26.52501  L2 = 1e-5: 26.50488  L2 = 1e-6: 26.54312  L2 = 0: 26.32615 | 26.30978    The best run, using L2 = 0.  As the two best runs are both with L2 = 0, we use L2 = 0 and continue to test different methods of regularization which might have more improvement. |
| 2 | L1 for GRU layer | We tried all values ranging from 0.1, 0.01 to 1e-6. This was applied using the kernel\_regularizer parameter. As L2 is set to be 0, we are tuning L1 independently here. Some values that seem promising after the first run, are run again to check for variability.  L1 = 1: 67.76680  L1 = 0.1: 26.60646  L1 = 0.01: 26.27202 | 26.47727 | 26.38502 L1 = 1e-3: 26.33908 | 26.39541 | 26.38366  L1 = 1e-4: 26.44783 L1 = 1e-5: 26.43598  L1 = 1e-6: 26.53365  L1 = 0: 26.50999 (| 26.32615 | 26.30978. from the previous step)  So as you can see from these results, they can vary quite a bit between runs. However, the L1 = 1e-3 gave solid results, each time below 26.4 (the only value to get this result each time). In addition, we took a look at the learning curve, and L1 = 1e-3 gave stable curves:    L1 = 1e-3 for the last run.  For these reasons, we will set L1 to 1e-3 for the GRU layer. |
| 2 | Dropout for GRU layer | We will follow the same process as in the previous step, now changing the dropout value for the GRU layer.  0: 26.33908 | 26.39541 | 26.38366 (from previous step)  0.1: 28.55243  0.2: 30.99368  It is clear from these two runs that adding the regular dropout to the GRU layer has a very negative effect on the validation error, and we will thus not explore more values. |
| 3 | Recurrent dropout for GRU layer | We will follow the same process as in the previous step, now changing the recurrent dropout value for the GRU layer.  0: 26.33908 | 26.39541 | 26.38366 (from previous step)  0.1: 26.71979 | 26.56285  0.2: 26.65747  Adding recurrent dropout seems detrimental for the performance of the model, and thus we will not add it. |
| 4 | L2 for the Dense layers | We will follow the same process as in all previous steps, adding L2 regularization to the three relu-based Dense layers.  1: 67.86629  0.1: 27.07224  0.01: 26.47387  1e-3: 26.44959  1e-4: 26.42619  1e-5: 26.29210 | 26.41454 | 26.45012  1e-6: 26.73855 | 26.46428  0: 26.33908 | 26.39541 | 26.38366 (from previous step)  Even though L2 = 1e-5 had one very good run, the other two runs for it had relatively worse results. Due to this lower consistency among the runs, we will not use this value and instead stick with L2 = 0 here. |
| 5 | L1 for the Dense layers | We will follow the same process as in all previous steps, adding L1 regularization to the three relu-based Dense layers.  0.1: 26.79183  0.01: 26.82597  1e-3: 26.39207  1e-4: 26.54860 1e-5: 26.30836 | 26.31084 | 26.35242  1e-6: 26.53206  0: 26.33908 | 26.39541 | 26.38366 (from previous step)  Setting L1 at 1e-5 for the dense layers seem to provide a very low error value without much variability: all three runs are quite similar in result and seem better than the previous result. We will keep L1 at 1e-5 for these layers.    The final run for 1e-5. |
| 6 | Dropout for Dense layers | As a final step, we will add dropout layers after the three dense layers, to see if this helps. We will follow the same process as in the previous steps.  Dropout rate = 0: 26.30836 | 26.31084 | 26.35242 (from previous step)  0.1: 26.55470 | 26.63802  0.2: 26.88104  Adding dropout after these dense layers clearly does not improve the network, and only increases the error. In addition, it also causes the validation curve to be a lot more unstable, so we will not use dropout here. |

The final regularized model is as follows:

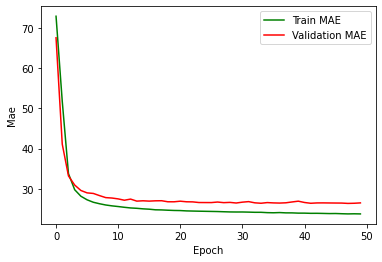
* GRU layer: 16 hidden neurons, L1 = 1e-3
* Dense layer with 32 neurons, Relu activation, L1 = 1e-5
* Dense layer with 16 neurons, Relu activation, L1 = 1e-5
* Dense layer with 8 neurons, Relu activation, L1 = 1e-5
* Dense layer with 1 neuron, Linear activation

Learning rate of 0.0005

Number of epochs: 50

Batch size: 64

Window size: 6



*Figure 16: Final regularized GRU model*

## Performance on test set

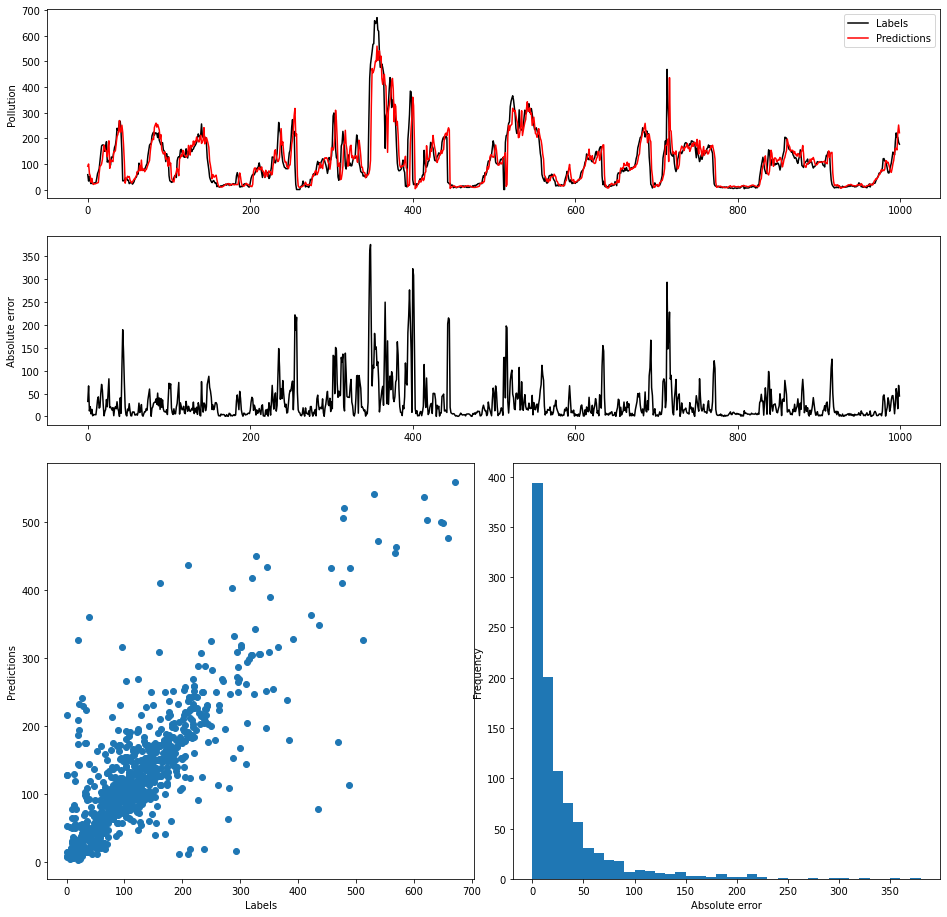
The final performance was found by training on the entire training set, so including the validation data, for 49 epochs (which gave the best results for the validation error). This resulted in the following test set performance:

"RNN-GRU" model:

train all mae = 24.402163

test mae = 23.515162

Which is impressive, as it is the best MAE on the test set of all models we have seen so far, and a significant improvement over the baselines.



*Figure 17: Results on the test set for the regularized GRU model.*

The comprehensive results show that there is not much “lagging” behind anymore. The network truly aims to predict what will happen next. The downside of this is that sometimes the network under- or overestimates the expected pollution. The results in general seem good though, with no major outliers.

# Hybrid Deep Learning

Based on the optimal architectures that were found for the CNN and GRU model, we want to see if we can combine their architectures and obtain an even better model. Here, we will not perform a really big deepdive but will see if there is some potential to merge both architectures and first start with some Conv1D blocks, followed by a GRU layer and some dense layers.

Simply as a proof of concept, we designed this network by using the knowledge we found in our previous two networks, taking the convolutional structure from the CNN and the recurrent structure from our GRU-based network.

We will not go into detail on the tuning of this (as this is simply to explore the option of combining these networks), but we ended up with the following network:

* Block 1: 1x1 Conv1D 16-Relu - 3x3 Conv1D 16- Relu - Maxpool 2x2 (with L1 = 1e-3 and L2 = 1e-1 in the Conv1D layers)
* Block 2: 3x3 Conv1D 32-Relu - 3x3 Conv1D 32- Relu - Maxpool 2x2 (with L1 = 1e-3 and L2 = 1e-1 in the Conv1D layers)
* GRU layer: 16 hidden neurons, L1 = 1e-3
* Dense layer with 32 neurons, Relu activation, L1 = 1e-5
* Dense layer with 16 neurons, Relu activation, L1 = 1e-5
* Dense layer with 8 neurons, Relu activation, L1 = 1e-5
* Dense layer with 1 neuron, Linear activation

Learning rate of 0.001

Number of epochs: 50

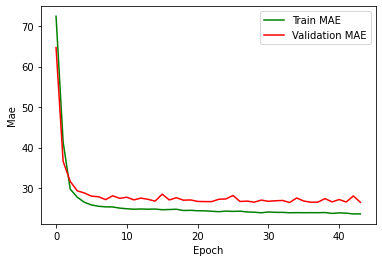
Batch size: 128

Window size: 6

The final results were quite good, but did not show an improvement over simply using the recurrent network.

Train MAE = 23.549528

Validation MAE = 26.45351

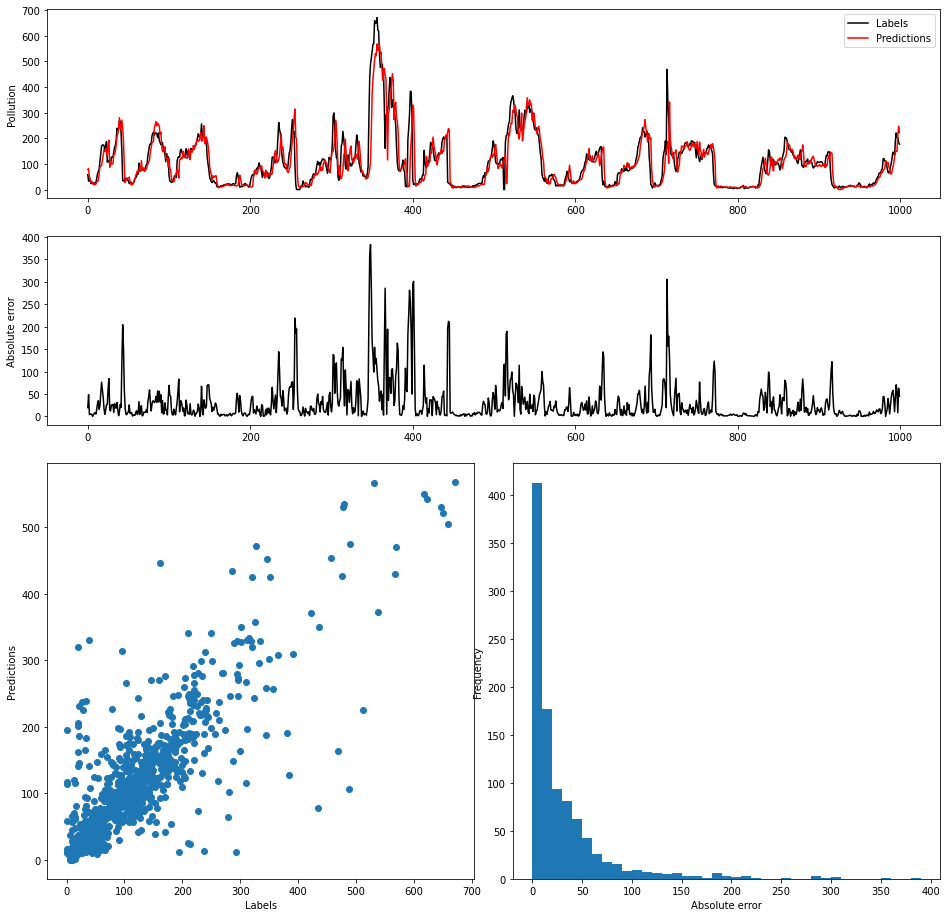


*Figure 18: Error curves for the hybrid model, for the train and validation data.*

We tested this model on the test set, to be able to compare more fully the results to the other methods. The results can be seen in Figure 19. The results bear some similarities to the results of the recurrent network, albeit slightly worse in general.

train all mae = 24.55204

test mae = 23.817049



*Figure 19: Comprehensive results on the test data, for the hybrid model.*

# Conclusion

In order to get a good overview on the differences between the models, Table 19 is provided. In this table, a brief summary is given on the performances of the models that were tried in the different sections. As can be concluded, the best performing models are the CNN and GRU which seem to have somewhat similar performance. Both models are able to outperform the simplest baseline and the standard dense neural networks. This confirms that these CNN and GRU models are quite powerful when working with time series data.

The recurrent neural network is able to perform ever-so-slightly better than the convolutional neural network. We assume this is because simply, the recurrent network is better at dealing with specific sequential data such as this one. It also seems that, based on the solid performance of our baselines and the relatively low window size after tuning, most information is encoded in the most recent timesteps. We feel that a recurrent neural network is better able to combine this than a convolutional network, which by definition must combine all information in the form of its kernels, which means it can not look easily at the last timestep information, nor can it find relations that are larger than its kernel size easily.

This might also be part of the reason that the hybrid model does not perform as well (other part of the reason is less tuning, perhaps). Putting convolution before the recurrent layers maintains advantages from both structures, but also gets these potential disadvantages from both models.

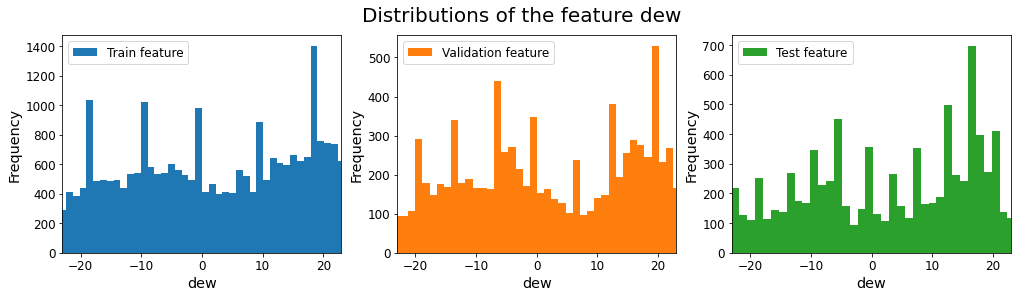
*Table 9: Overview of the best models along with*

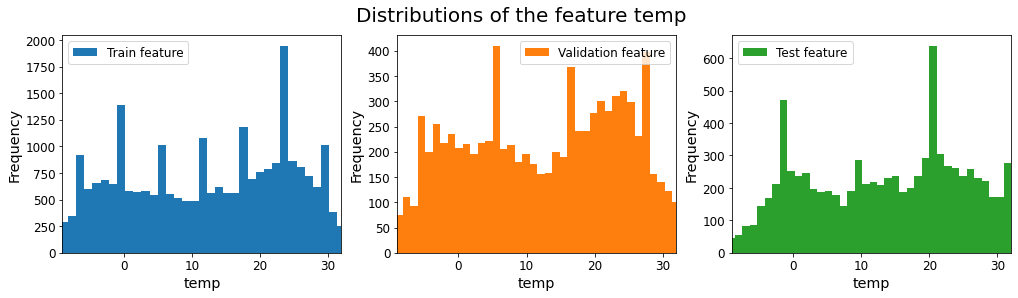
*their performance on the validation and test set*

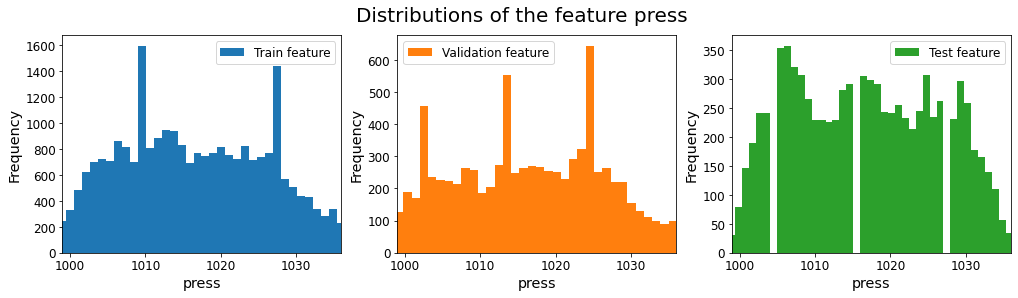
| **Model** | **Validation MAE** | **Test MAE** |
| --- | --- | --- |
| Last timestep interpolation (simplest baseline) | 28.706179 | 26.04659 |
| Dense model (features of the last timestep) | 26.9595 | 24.899323 |
| Dense model (pollution with time window) | 28.0904 | 25.51374 |
| CNN model | 26.2294 | 23.6714 |
| GRU model | 26.3083 | 23.5152 |
| Hybrid CNN-GRU model | 26.4535 | 23.8170 |

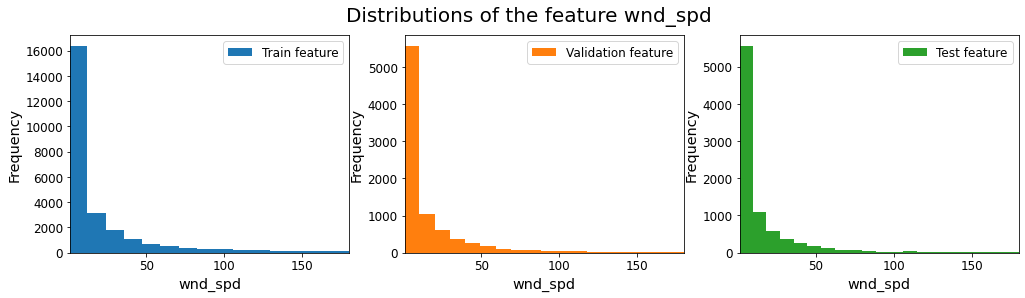
# Appendix

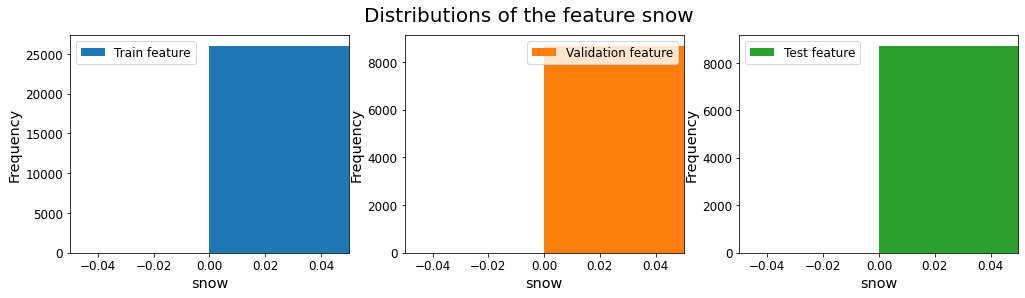
## Distribution of the features

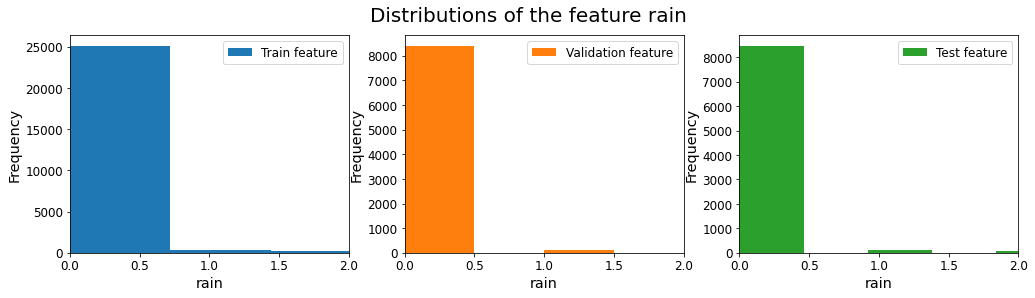












1. <https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/> [↑](#footnote-ref-0)
2. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/ [↑](#footnote-ref-1)
3. https://arxiv.org/pdf/1901.00069.pdf [↑](#footnote-ref-2)