Digital snow melt - Automated forecasting from snow parameters

Florian Donhauser, Fabienne Greier, Md. Forhad Hossain,

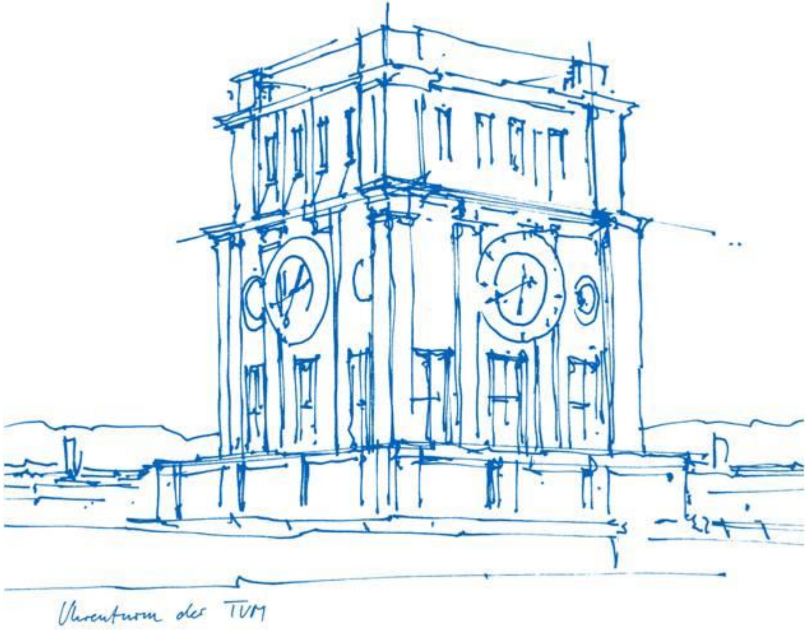
Robin Mittas, Wudamu

Technical University of Munich & ThinkOutside

Munich Data Science Institute (MDSI)

TUM Data Innovation Lab (TUM-DI-LAB)

Munich, 25th of February 2022



Team members

Robin,
Mathematics in
Data Sciene

Forhad,
Data Engineering
and Analytics

Florian,
Informatics

Fabienne,
Robotics, Cognition,
Intelligence

Wudamu,
Electrical and
Computer
Engineering


THINK OUTSIDE

Overview

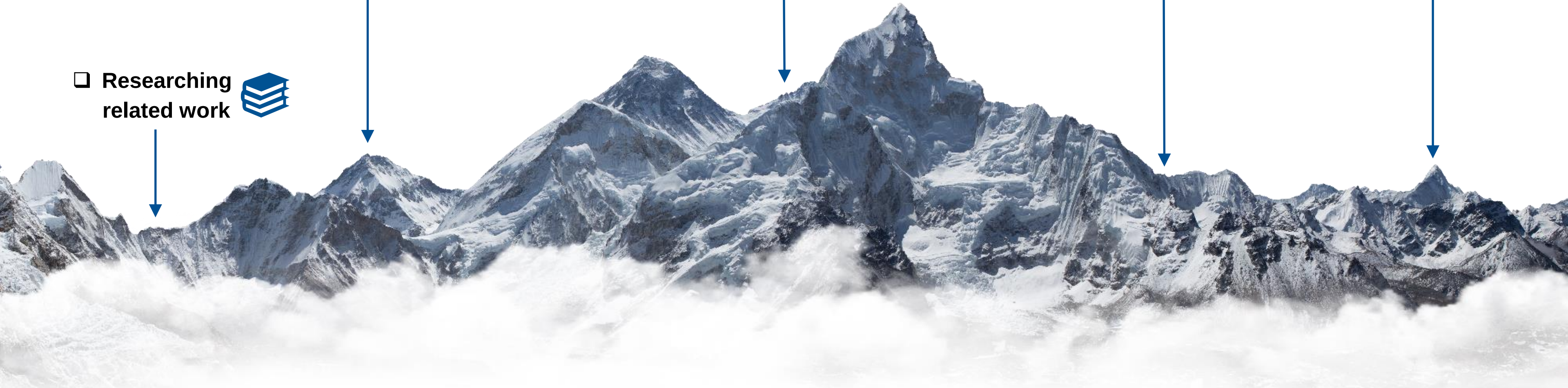
- ❑ **Researching related work**

- ❑ **Developing the *DataLoader***
  - ⇨ Filling in the missing values in the downloaded data
  - ⇨ Time resolution of the data

- ❑ **Model architecture**
  - ⇨ Improving the code quality, flexibility and readability
  - ⇨ Creating building blocks to remove code duplicates
  - ⇨ Creating config files
  - ⇨ Combat overfitting

- ❑ **Automatic hyperparameter tuning**

- ❑ **Testing**
  - ⇨ Yearly forecasts
  - ⇨ Monthly forecasts
  - ⇨ Weekly forecasts
  - ⇨ Sum over all stations



Related work

Long-term Reservoir Inflow Forecasts: Enhanced Water Supply and Inflow Volume Accuracy Using Deep Learning (Herbert, Z. et al.)

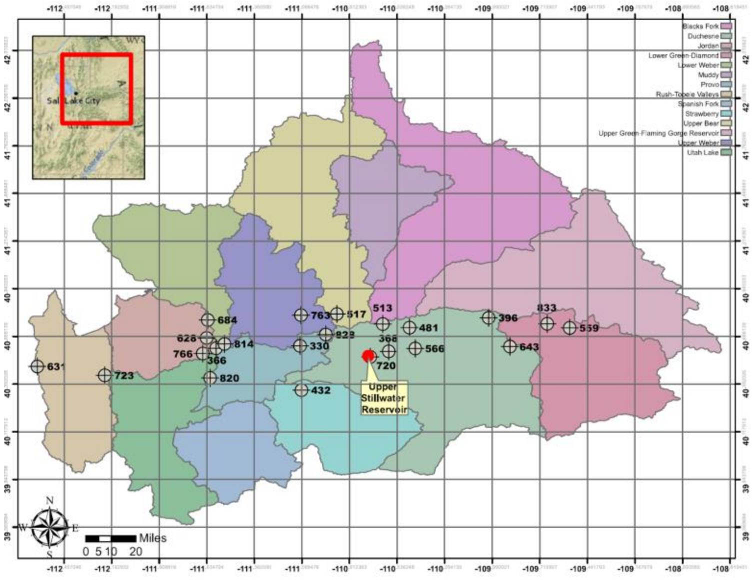


Fig. 1. Study site location: Upper Stillwater Reservoir, Utah.

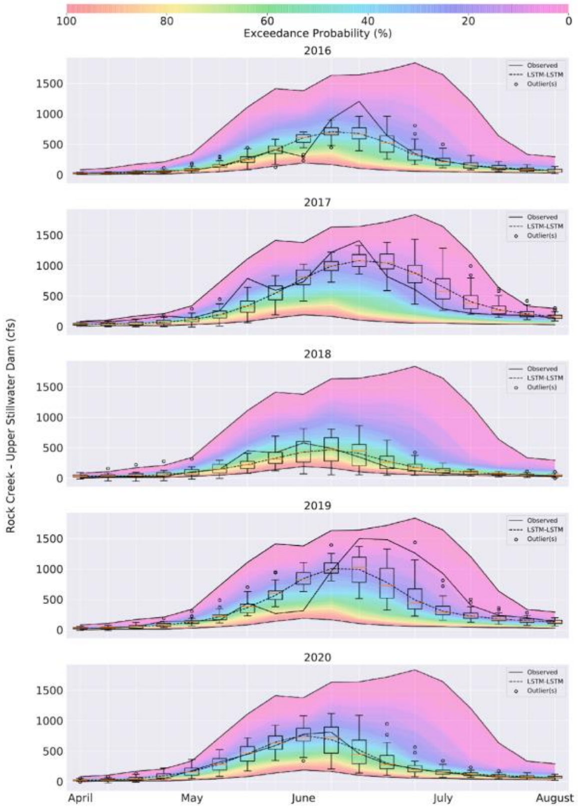
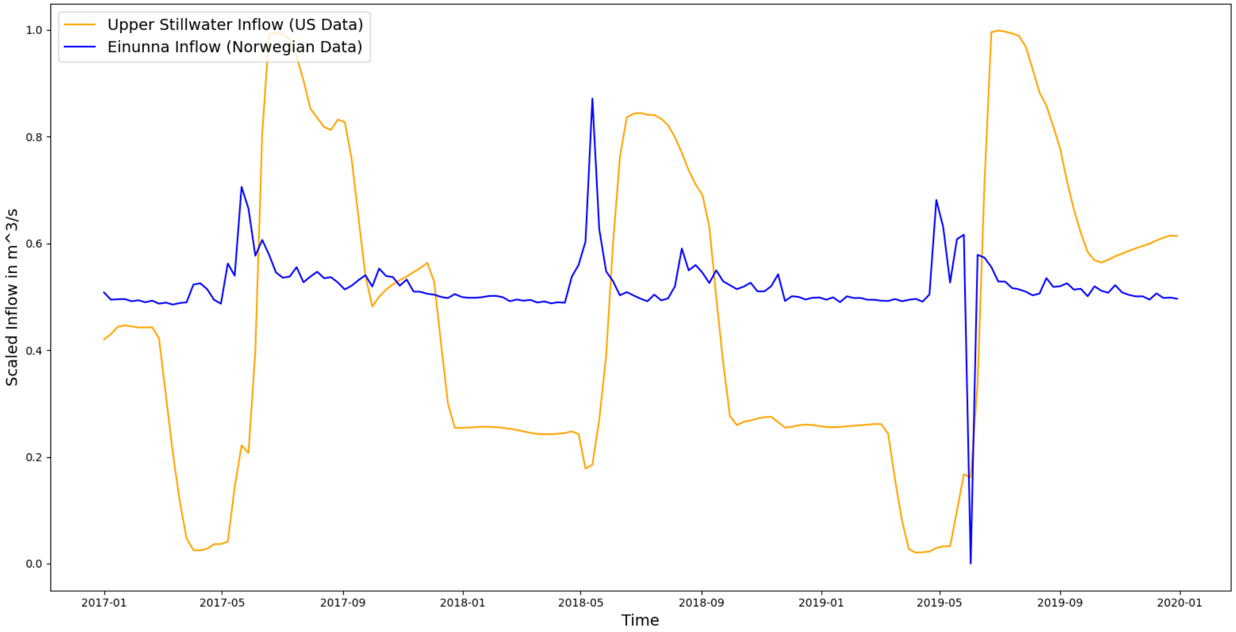


Fig. 7. Multi-step reservoir inflow forecasts for the 2016–2020 hold-out periods.

Data preprocessing



DataLoader

1. Data source
2. Data acquisition
3. Data cleaning

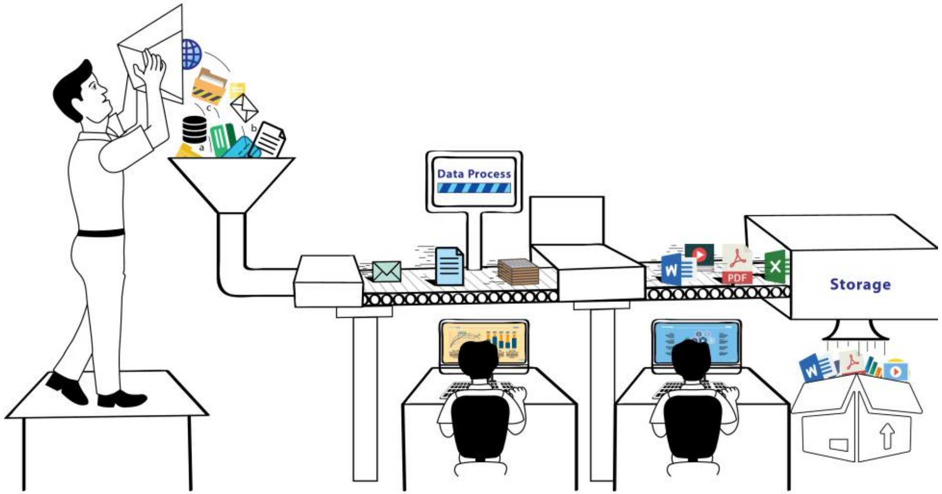


Image source: https://www.wninfotech.com/data-processing

Data source

First data source:

Name:

- Norwegian Water Resources and Energy Directorate (NVE)

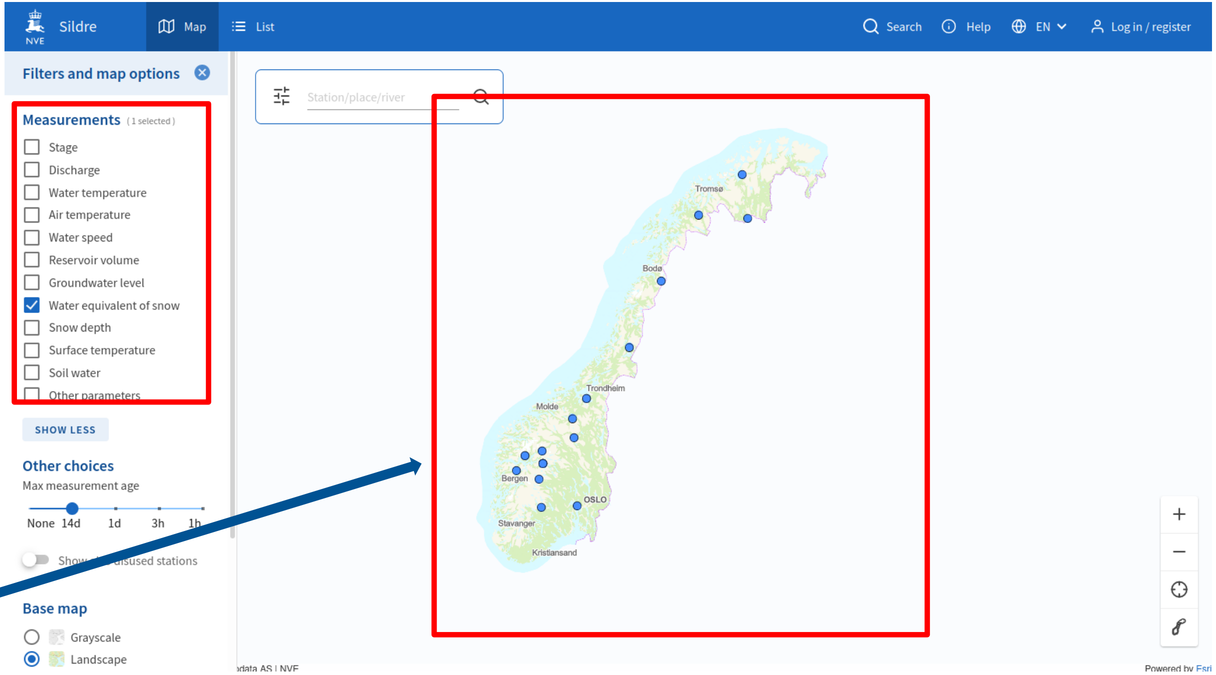
Measurements:

- Water equivalent of snow
- Snow depth
- Air temperature
- Snow depth
- ...

What we used:

- Download the measurements of the stations which include water equivalent of snow
- Includes 15 stations

NVE Data



source: https://sildre.nve.no/map

Data source

Second data source:

Name:

- Inflow-glomma

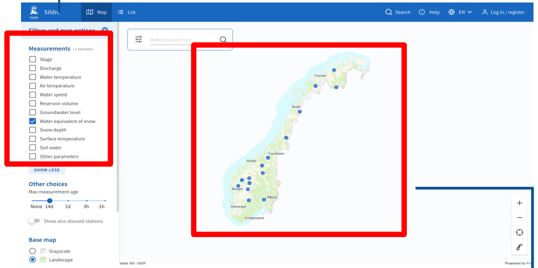
Measurements:

- Inflow data of the station

What we used:

- All 18 stations

Data acquisition



NVE Data

Measurements/ Features

Name of the stations

Start time:
- 01/12/2001

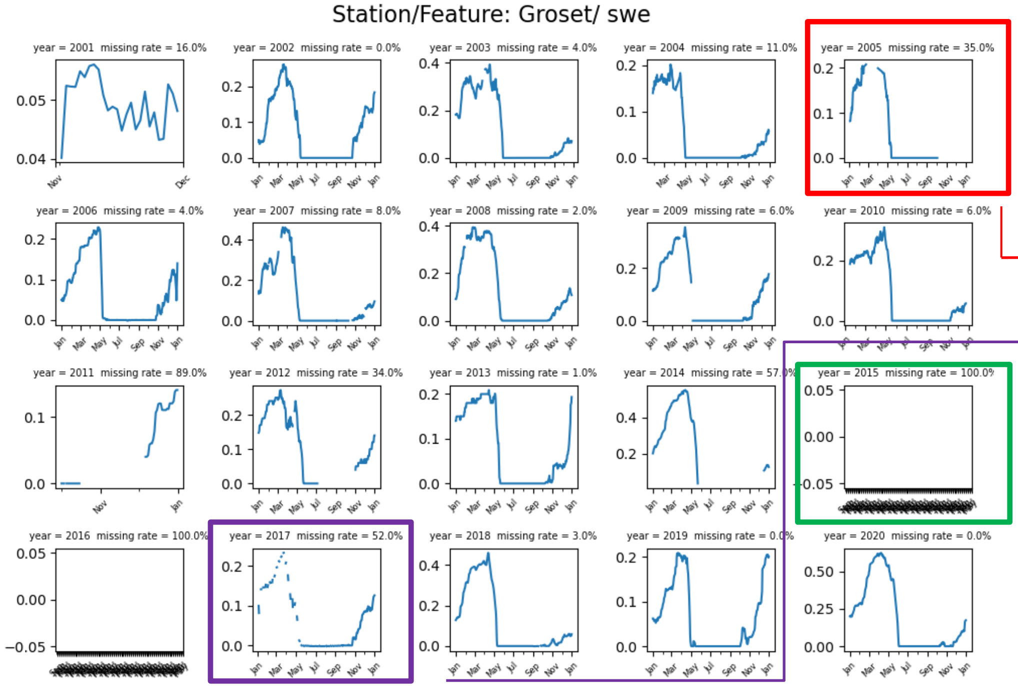
End time:
- 30/11/2021

config file

download_data.py

source: https://sildre.nve.no/map
source: https://pixabay.com/vectors/box-data-download-icon-save-1292866/

Downloading the original data from NVE



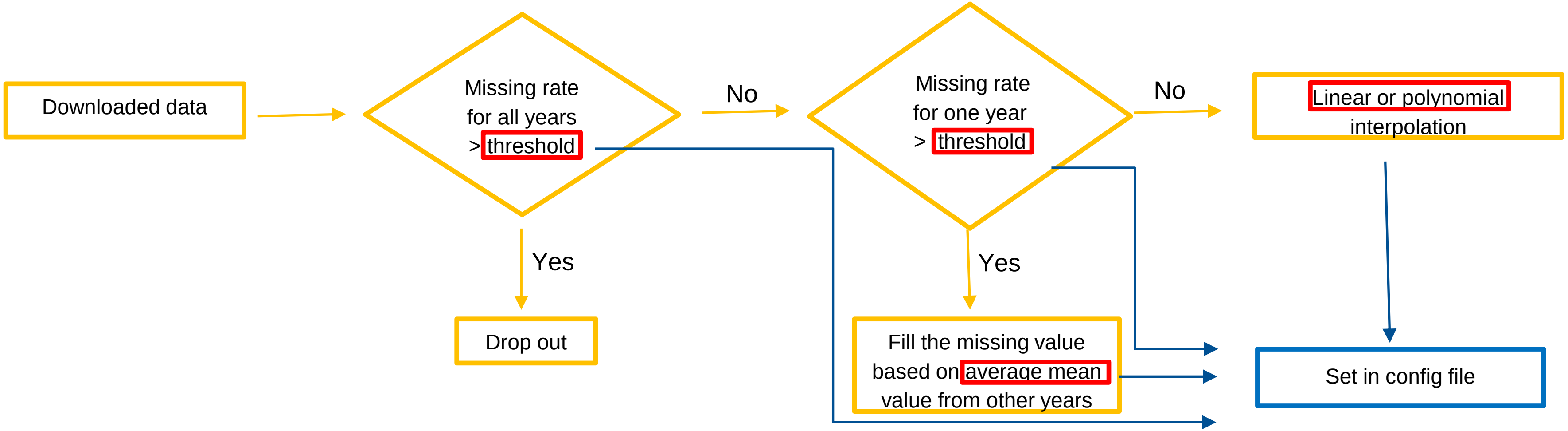
Example of downloaded data

Missing a lot of data:

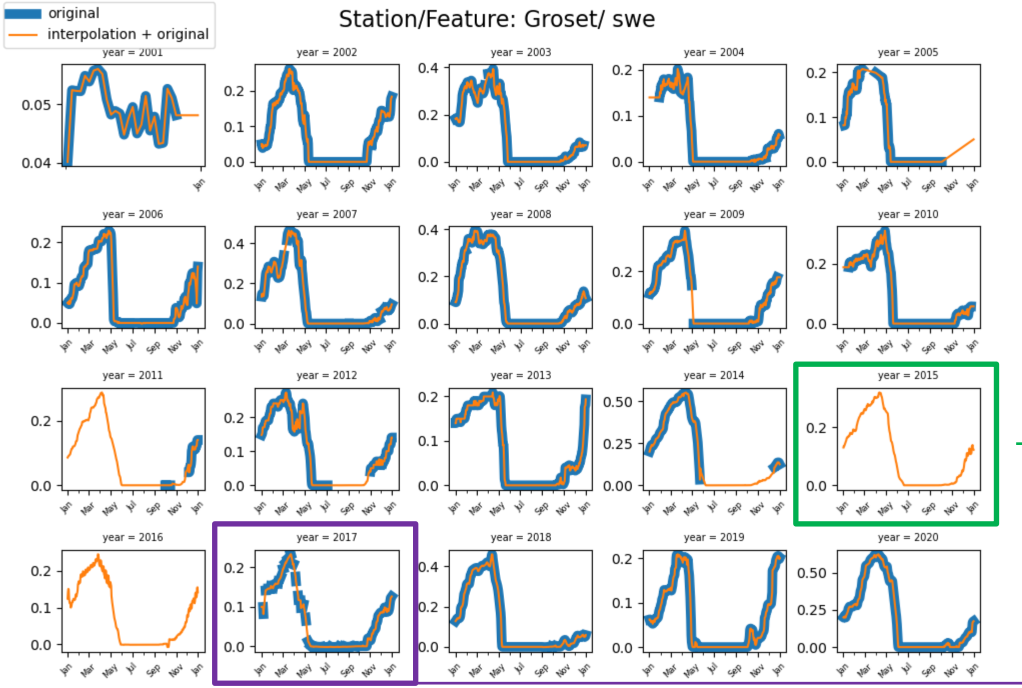
For example:

- Missing 35.0% data in 2005,
- Missing 100% data in 2015,
- Missing 52.2% data in 2017,
- …

Data cleaning for NVE data



Example of the cleaned NVE data

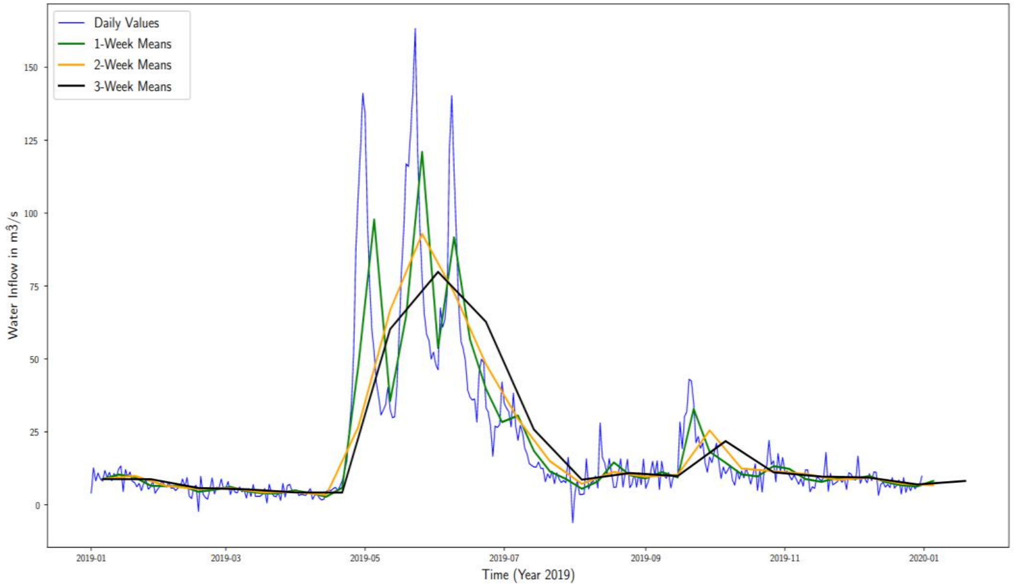


Fill Null values according to average data from other years

Linear interpolation

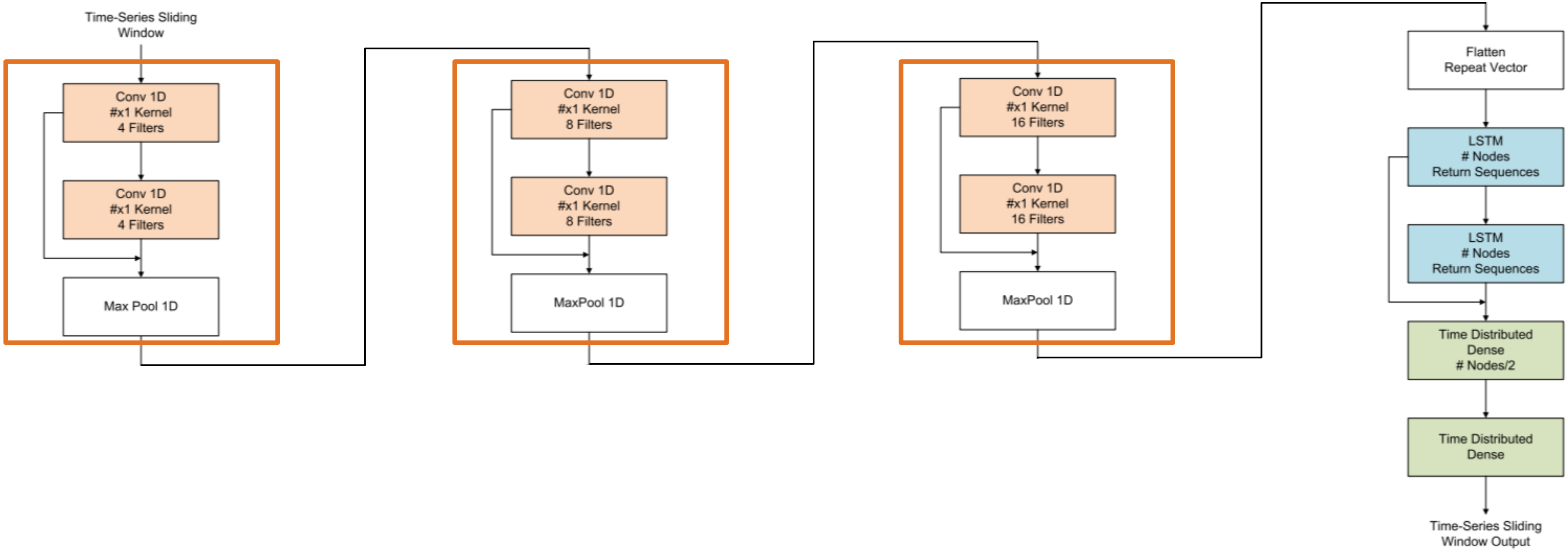
Water inflow data

Water inflow data is provided by "Think Outside " and it is an Excel file which contains 18 stations over 40 years



Inflow values of Norwegian reservoir lake "Aursunden" with different resampling methods

Existing architecture by Herbert et al.



Existing architecture

Code duplicates in their code

☹

```
model = Conv1D(filter1, kernel_size, padding = 'causal')(visible1)
residual1 = ReLU()(model)
model = Conv1D(filter1, kernel_size, padding = 'causal')(residual1)
model = Add()([residual1, model])
model = ReLU()(model)

model = Conv1D(filter1, kernel_size, padding = 'causal')(model)
residual2 = ReLU()(model)
model = Conv1D(filter1, kernel_size, padding = 'causal')(residual2)
model = Add()([residual2, model])
model = ReLU()(model)
model = MaxPooling1D()(model)

model = Conv1D(filter2, kernel_size, padding = 'causal')(model)
residual3 = ReLU()(model)
model = Conv1D(filter2, kernel_size, padding = 'causal')(residual3)
model = Add()([residual3, model])
model = ReLU()(model)

model = Conv1D(filter2, kernel_size, padding = 'causal')(model)
residual4 = ReLU()(model)
model = Conv1D(filter2, kernel_size, padding = 'causal')(residual4)
model = Add()([residual4, model])
model = ReLU()(model)
model = MaxPooling1D()(model)
```

Existing architecture – further problems

- Hard-coded values, no configuration file:

```
# Declare model input parameters
input_length, output_length, repeats, year = 20, 15, 30, 2019
```

- No real code structure, complete code in one big file
- No main function
- Dependencies without version number, partly outdated libraries
- Insufficient documentation/comments in the code

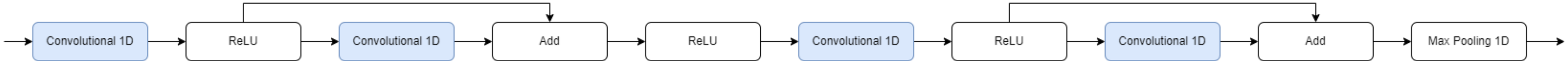
Dependencies

- Python 3
- Climata
- Tensorflow
- Matplotlib
- Sklearn
- Seaborn
- Datetime
- Pandas
- Numpy

Our model architecture

- Remove code duplicates by creating a “block”

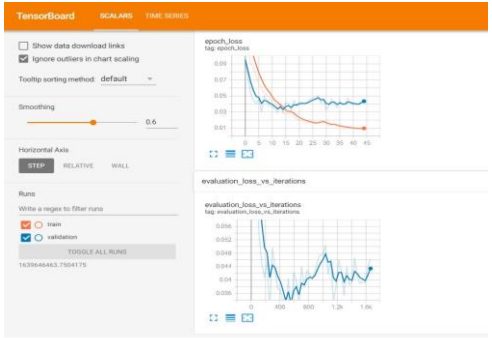
- Structure of each residual CNN block



- Move hyperparameters into config file (config.yaml)

```yaml
n_timesteps: 40        # n_timestamps length of input window, 40 weeks (in case resampling method=1W)
n_outputs: 52          # n_outputs how many weeks we want to predict (in case resampling method=1W)
```

- Add TensorBoard logging

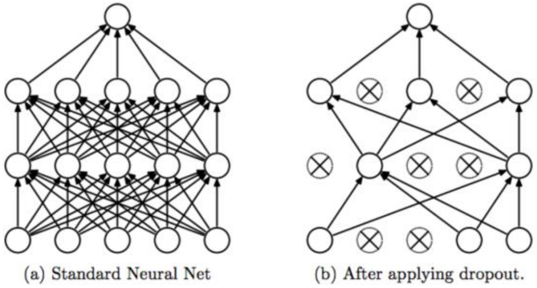


Techniques to combat overfitting

- Dropout
  - For LSTM layers and between dense layers
  - Dropout probability in config file

- Batch Normalization
  - Output of 1D convolutions
  - Can be turned on or off

- L2 Regularization (weight decay)
  - Applied to weights of 1D convolutions and LSTM layers
  - Regularization factor in config file



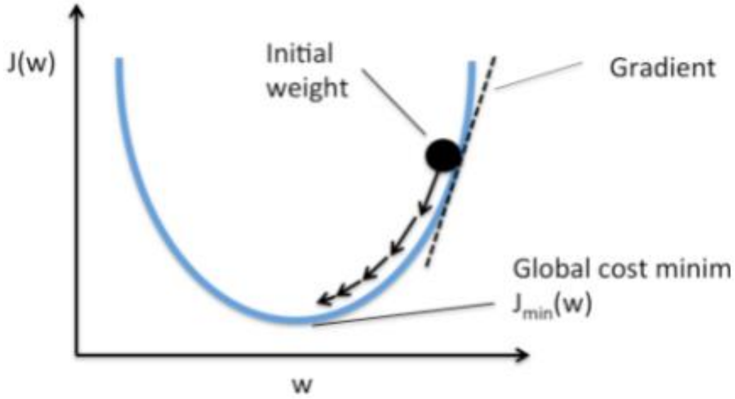
$$y = \gamma y_{norm} + \beta$$

$$L_{total} = L + \frac{\lambda}{2} \|w\|^2$$

Hyperparameter optimization

- Two types of parameters: model parameters and hyperparameters

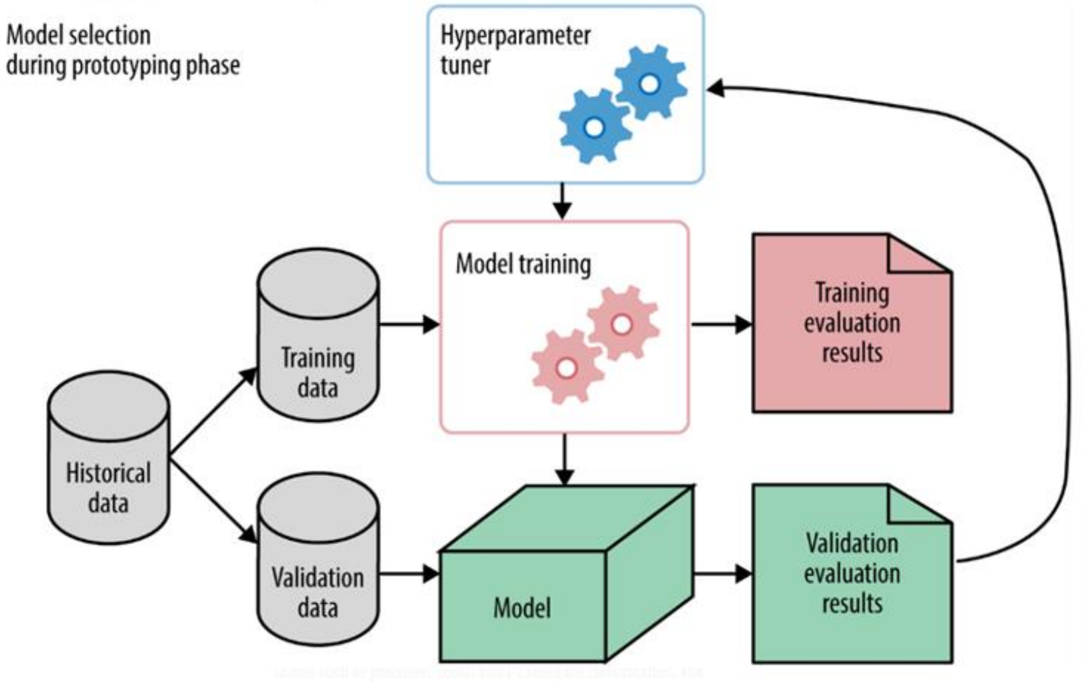
- Model parameters: learned



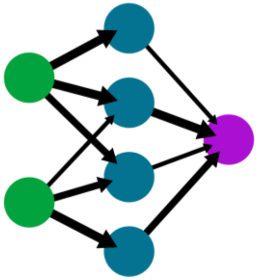
Hyperparameters: set by the developers

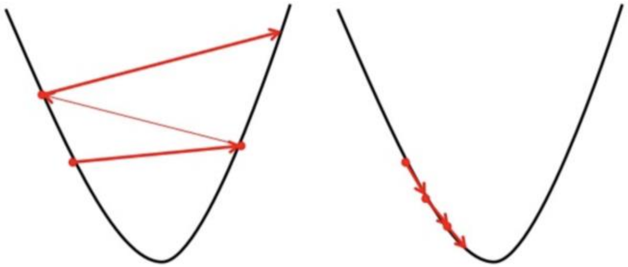
epochs **n_nodes**
kernel_size **patience**
batch_size **n_timesteps**
learning_rate pooling_size
total_hidden_layer Momentum
weight_regularizer **filter**
activation_function
dropout

Hyperparameter optimization



Categories of hyperparameters





Model
- Size of layers
- Dropout probability
- Number of layers

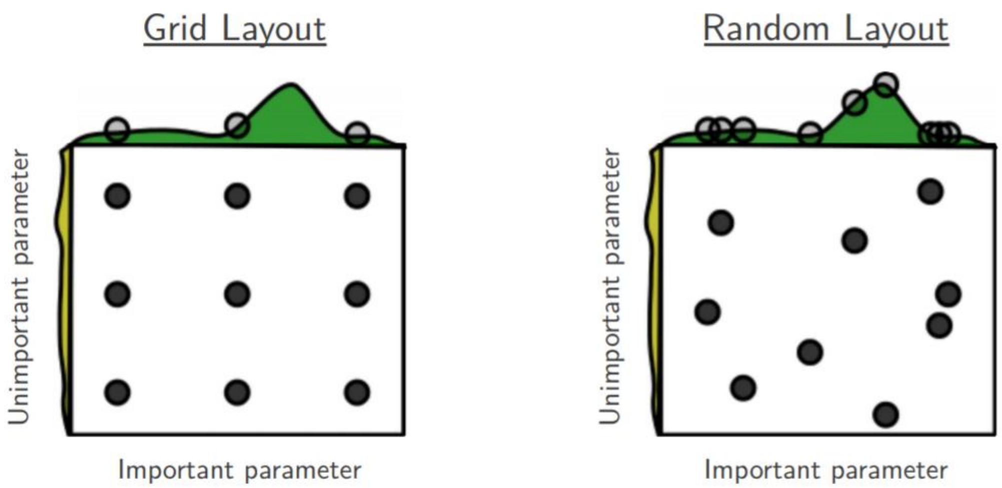
Optimizer
- Learning rate
- Mini-batch size
- Early stopping (patience)

Data and others
- Input length
- Preprocessing

Why hyperparameter tuning / optimization?

Types of hyperparameter optimization

- Grid search
- Random search
- More advanced algorithms



Manual hyperparameter optimization

Epochs	Loss	Batch size	Optimizer	Repeats	MAE	RMSE
50	MSE	32	Adam	5	18.78	26,06
50	MSE	32	Adam	2	19.26	25.07
30	MSE	32	Adam	20	23.56	29,85
100	MSE	32	Adam	2	25.15	34.31
40	MSE	64	Adam	3	27.86	38.33
20	MSE	64	Adam	20	31.88	42.92
100	MSE	32	Adam	5	35,157	49,662

Automatic hyperparameter optimization

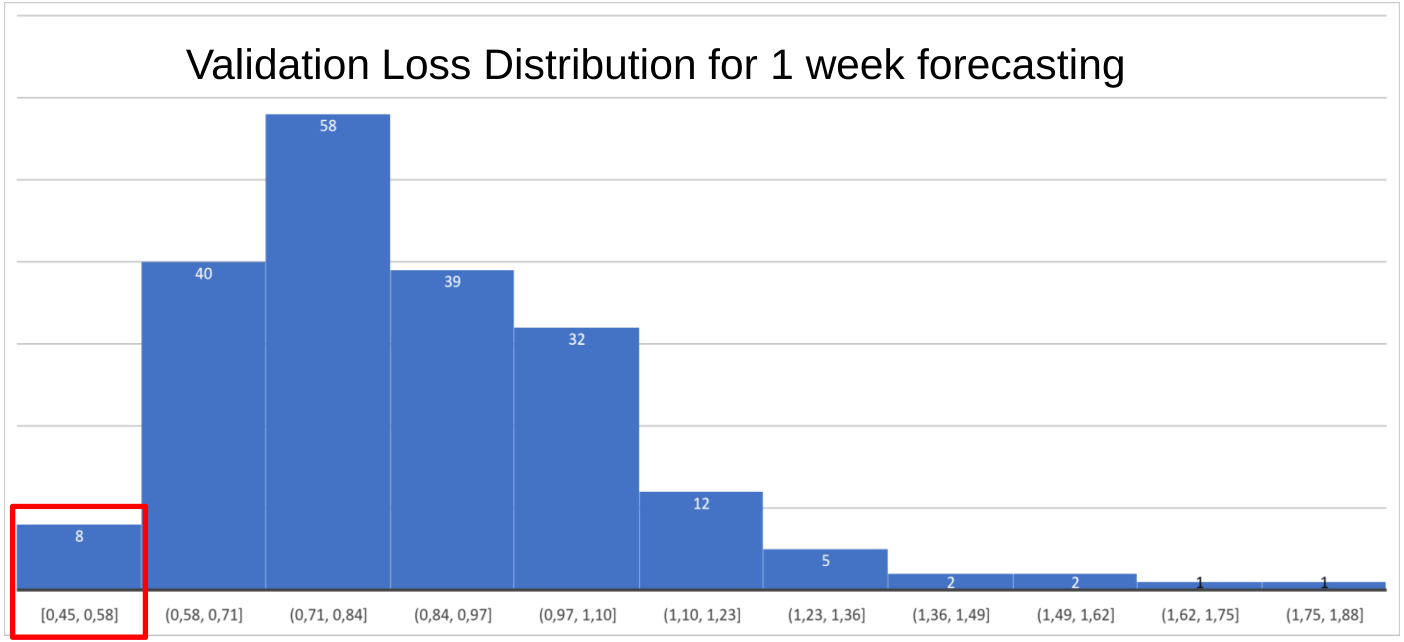
Features of Optuna

- Automatization
- Easy to use and good documentation
- Different strategies are implemented (Tree-structured Parzen Estimator)
- Support for different data types and distributions
  - suggest_categorical()
  - suggest_int()
  - suggest_uniform()
  - suggest_loguniform()
  - …

How to use Optuna?

```
config["n_timesteps"] = trial.suggest_int('n_timesteps', 8, 102)
config["batch_size"]= trial.suggest_categorical('batch_size', [64, 128])
config["epochs"] = trial.suggest_int("epochs", 25, 100)
config["n_nodes1"] = trial.suggest_int("n_nodes1", 4, 128)
config["n_nodes2"] = trial.suggest_int("n_nodes2", 4, 64)
config["filter1"] = trial.suggest_int("filter1", 2, 128)
config["filter2"] = trial.suggest_int("filter2", 2, 64)
config["kernel_size"] = trial.suggest_int("kernel_size", 3, 9)
config["learning_rate"] = trial.suggest_loguniform('learning_rate', 1e-5, 1e-3)
config["dropout_probability"]= trial.suggest_float('dropout_probability', 0, 0.5)
config["use_batch_normalization"]= trial.suggest_categorical('use_batch_normalization', [True, False])
config["patience"] = trial.suggest_int("patience", 5, 50)
config["weight_regularizer"] = trial.suggest_loguniform("weight_regularizer", 1e-9, 1e-5)
```

Why Optuna?



Trial #	Val loss
178	0,45
134	0,51
164	0,52
126	0,54
136	0,56
129	0,57
130	0,57
181	0,58
61	0,58
88	0,58

Target data (Inflow)

How data is passed to the model

	Øvre Leirbotn/ swe	Øvre Leirbotn/ temperatur	ØVERBYGD II/ swe	ØVERBYGD II/ tem
2010-01-10T00:00:00.000000000	0.38617	0.15537	0.56620	0.54211
2010-01-17T00:00:00.000000000	0.39339	0.35696	0.58043	0.69832
2010-01-24T00:00:00.000000000	0.39531	0.33020	0.60512	0.66109
2010-01-31T00:00:00.000000000	0.39609	0.19727	0.62484	0.61714
2010-02-07T00:00:00.000000000	0.39683	0.29920	0.63303	0.64318
2010-02-14T00:00:00.000000000	0.40387	0.27508	0.64842	0.57799
2010-02-21T00:00:00.000000000	0.41083	0.00000	0.66178	0.59543
2010-02-28T00:00:00.000000000	0.41294	0.07971	0.67325	0.54873
2010-03-07T00:00:00.000000000	0.41479	0.23351	0.68709	0.57728
2010-03-14T00:00:00.000000000	0.43359	0.30457	0.72078	0.67190
2010-03-21T00:00:00.000000000	0.44455	0.19223	0.73443	0.60287
2010-03-28T00:00:00.000000000	0.45390	0.35224	0.75189	0.67221
2010-04-04T00:00:00.000000000	0.45431	0.48973	0.76755	0.75088
2010-04-11T00:00:00.000000000	0.44692	0.55830	0.76432	0.76569
2010-04-18T00:00:00.000000000	0.44761	0.52236	0.77809	0.76301
2010-04-25T00:00:00.000000000	0.44214	0.48323	0.76255	0.76249
2010-05-02T00:00:00.000000000	0.44633	0.52595	0.77350	0.75641
2010-05-09T00:00:00.000000000	0.45348	0.57230	0.75549	0.79128
2010-05-16T00:00:00.000000000	0.44258	0.65878	0.71057	0.86028
2010-05-23T00:00:00.000000000	0.39338	0.70584	0.53504	0.86291

	Aursunden
2010-05-16T00:00:00.000000000	0.13301
2010-05-23T00:00:00.000000000	0.91384
2010-05-30T00:00:00.000000000	0.38160
2010-06-06T00:00:00.000000000	0.29442
2010-06-13T00:00:00.000000000	0.26628
2010-06-20T00:00:00.000000000	0.28875
2010-06-27T00:00:00.000000000	0.19065
2010-07-04T00:00:00.000000000	0.19365
2010-07-11T00:00:00.000000000	0.11728
2010-07-18T00:00:00.000000000	0.09063
2010-07-25T00:00:00.000000000	0.06513
2010-08-01T00:00:00.000000000	0.06140
2010-08-08T00:00:00.000000000	0.07032
2010-08-15T00:00:00.000000000	0.05141
2010-08-22T00:00:00.000000000	0.02757

	Aursunden
2010-05-23T00:00:00.000000000	0.91384
2010-05-30T00:00:00.000000000	0.38160
2010-06-06T00:00:00.000000000	0.29442
2010-06-13T00:00:00.000000000	0.26628
2010-06-20T00:00:00.000000000	0.28875
2010-06-27T00:00:00.000000000	0.19065
2010-07-04T00:00:00.000000000	0.19365
2010-07-11T00:00:00.000000000	0.11728
2010-07-18T00:00:00.000000000	0.09063
2010-07-25T00:00:00.000000000	0.06513
2010-08-01T00:00:00.000000000	0.06140
2010-08-08T00:00:00.000000000	0.07032
2010-08-15T00:00:00.000000000	0.05141
2010-08-22T00:00:00.000000000	0.02757
2010-08-29T00:00:00.000000000	0.03402

Input of model (downloaded data: SWE, temperature...)

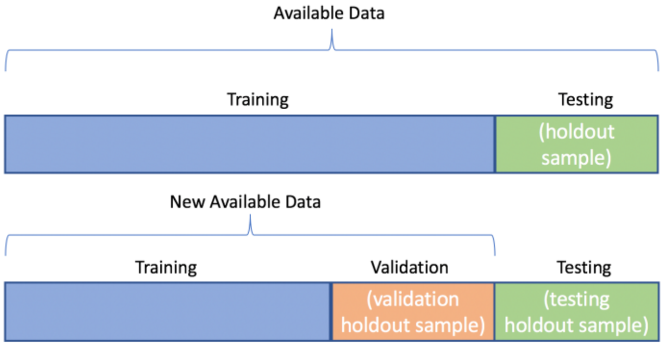
Model settings

Model configurations:

- Loss function: MSE
- Optimizer: Adam
- Validation split: 0.2
- Repeats: 3

2 different sources for our target data (inflow time series):

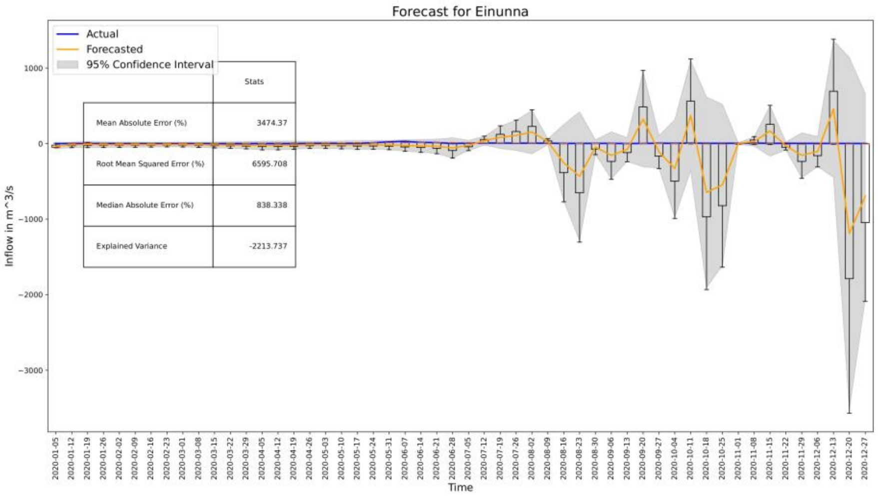
- Time series containing 18 reservoirs
- Time series containing 1 reservoir

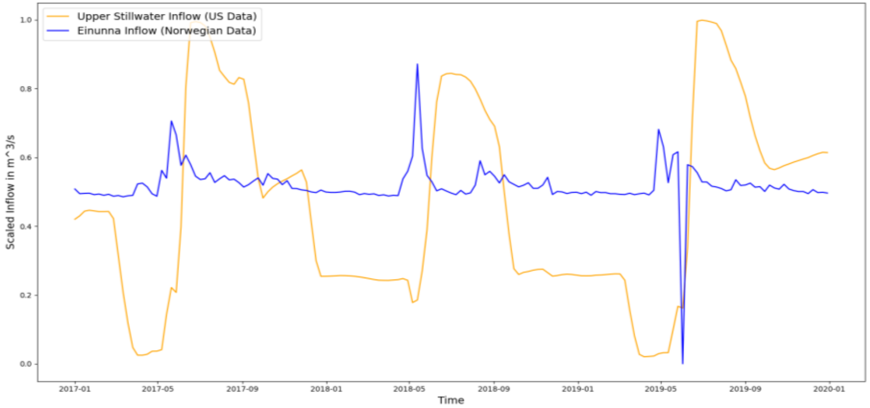


For "real" future forecasts no test set

Data quality

- Inflow (today) = reservoir volume (tomorrow) - reservoir volume (today) + outflow (today)
- Errors in measurement of the inflow ➡ inflow often small compared to outflow and changes in reservoir volume ➡relatively small errors and inaccuracies in water level or outflow can result in large errors in calculated inflow

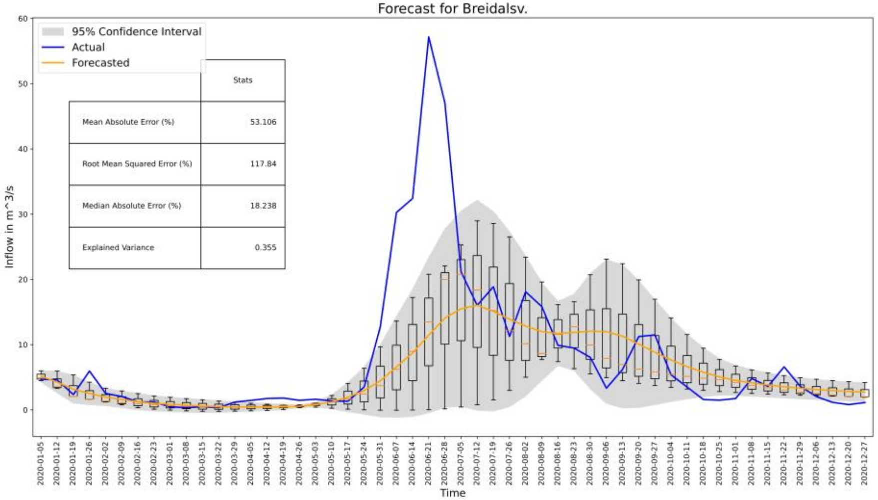




Yearly forecasts 2020

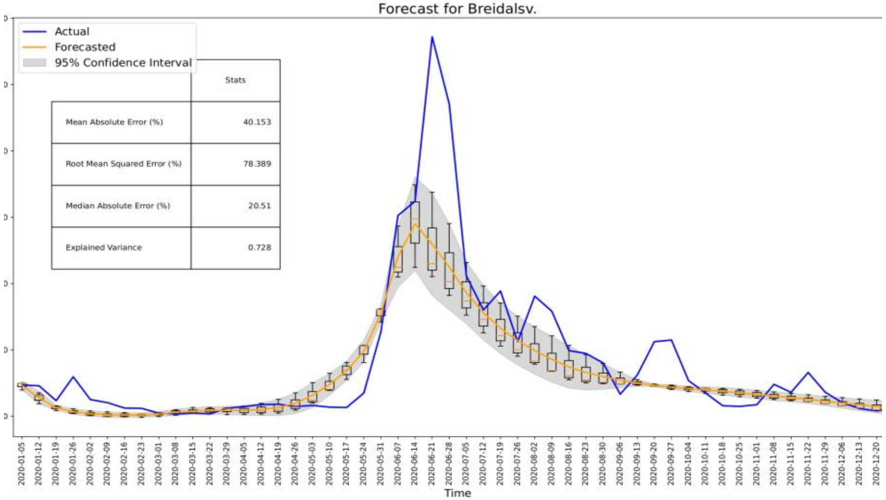
Without hyperparamter tuning

(➡predicting 52 weeks based on the previous 40 weeks)



With hyperparamter tuning

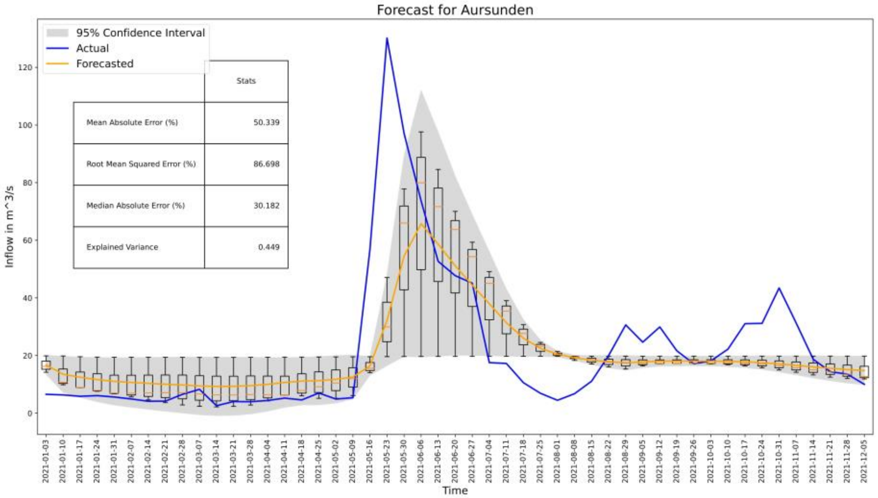
(➡predicting 52 weeks based on the previous 98 weeks)



Yearly forecasts 2021

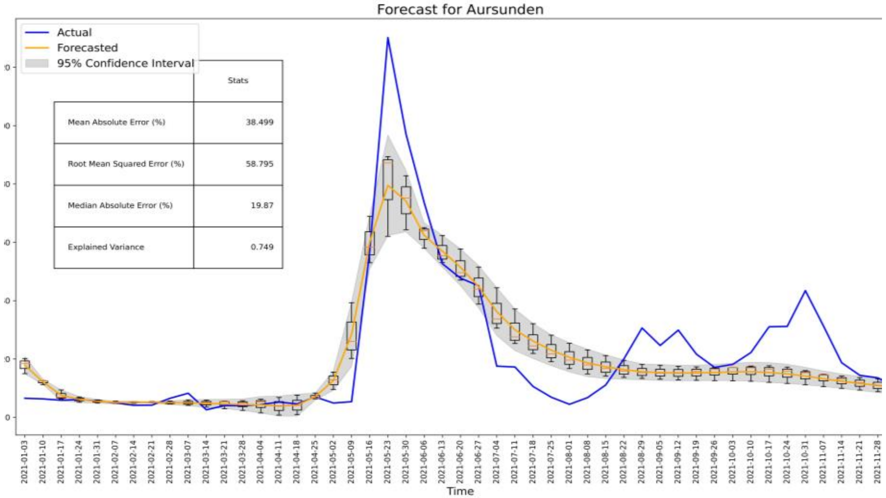
Without hyperparamter tuning

(➡predicting 49 weeks based on the previous 40 weeks)



With hyperparamter tuning

(➡predicting 49 weeks based on the previous 98 weeks)

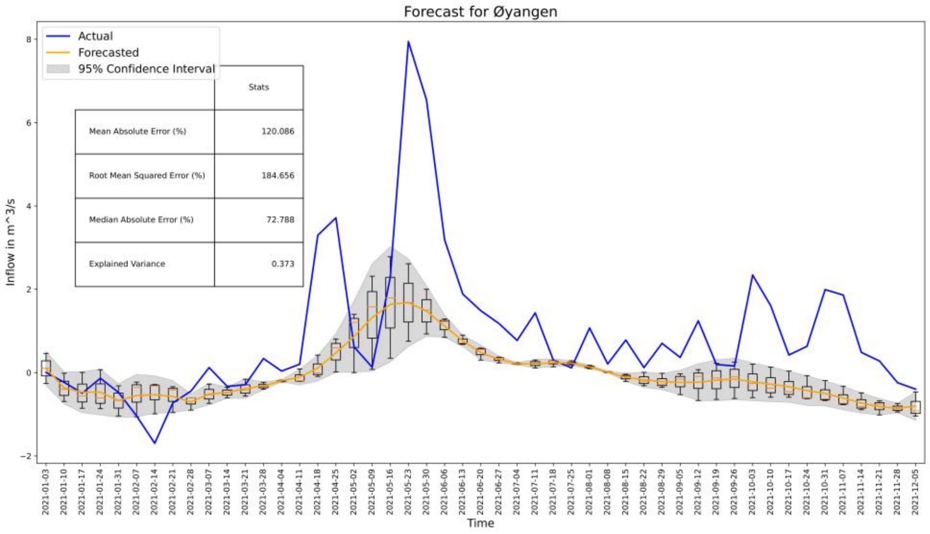


Note: Inflow time series ends in November 2021

Bad station – with hyperparamter tuning

- Negative values (also in previous years)
- Many ups and downs

➡Hard to make accurate predictions



Performance on annual forecasts 2021 over all 18 stations

Without hyperparameter tuning

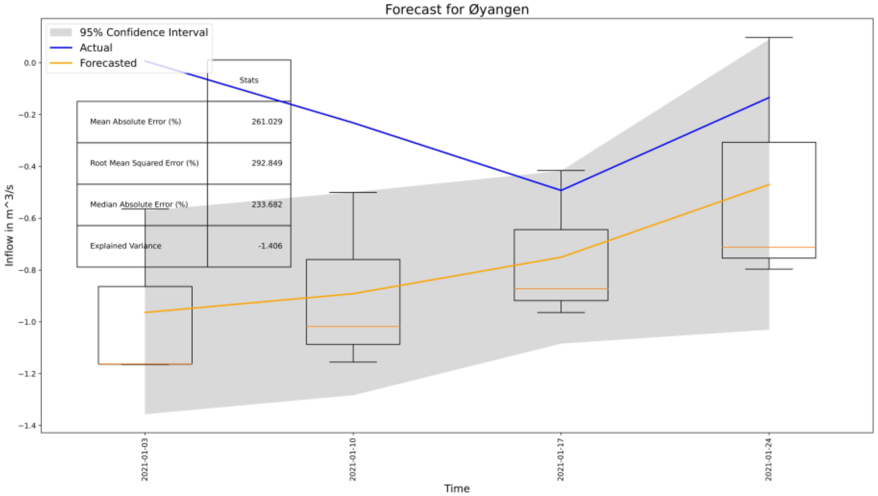
Metric	Average	Median
Mean Absolute Error (%)	72.32	73.95
Root Mean Sqaured Error (%)	112.17	112.15
Median Absolute Error (%)	41.67	47.03
Explained Variance	0.35	0.3

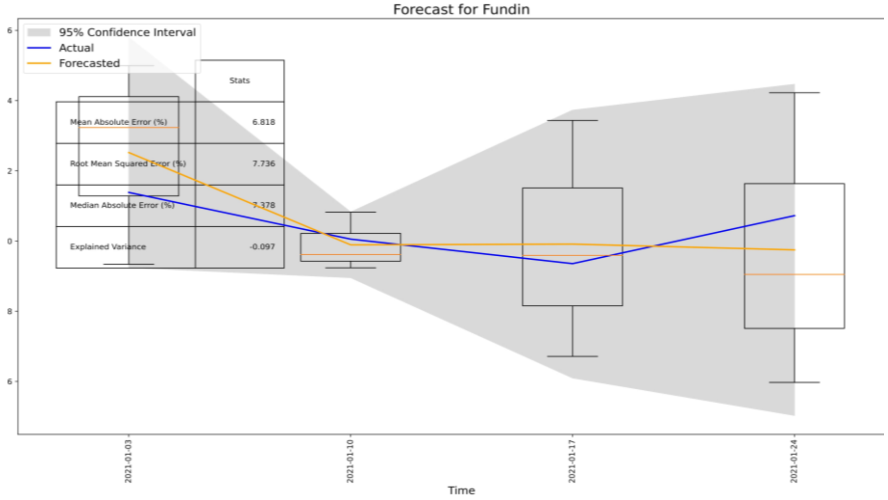
With hyperparameter tuning

Metric	Average	Median
Mean Absolute Error (%)	66.85	62.62
Root Mean Sqaured Error (%)	100.21	101.95
Median Absolute Error (%)	41.13	36.77
Explained Variance	0.48	0.45

Monthly forecasts – with hyperparameter tuning

☹ ☺





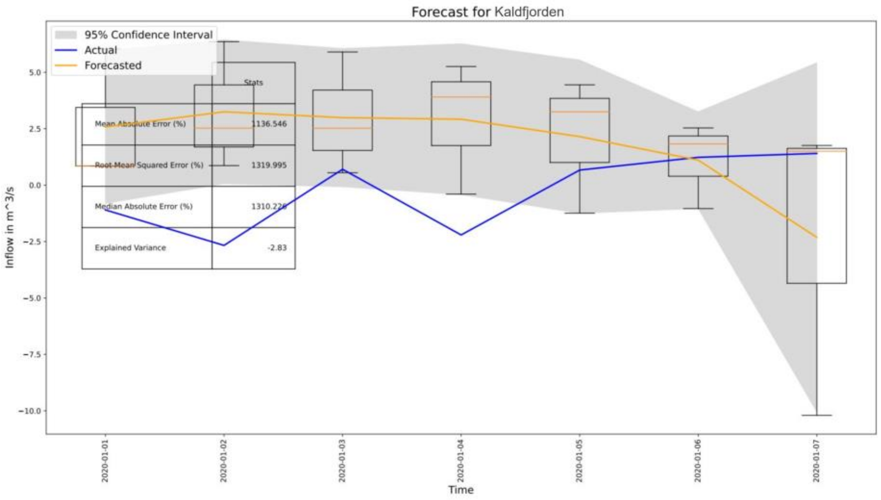
➡ Again station Øyangen

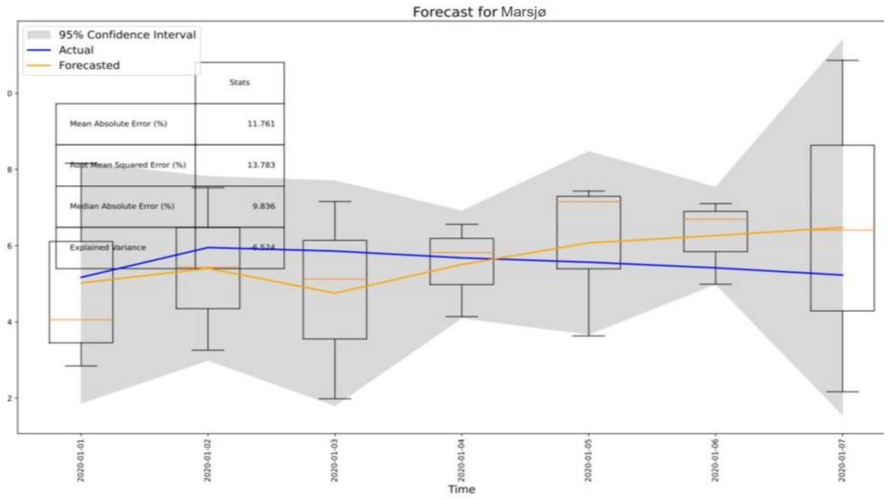
Performance on monthly forecasts over all 18 stations

Metric	Average
Mean Absolute Error (%)	75.41
Root Mean Sqaured Error (%)	84.79
Median Absolute Error (%)	70.07
Explained Variance	-1.37

Weekly forecasts – with hyperparameter tuning

☹ ☺





➡ Negative values and mean of actual values close to 0

Performance on weekly forecasts over all 18 stations

Metric	Average
Mean Absolute Error (%)	199.24
Root Mean Sqaured Error (%)	236.12
Median Absolute Error (%)	209.02
Explained Variance	-1.15

Weaker performance on weekly forecasts?

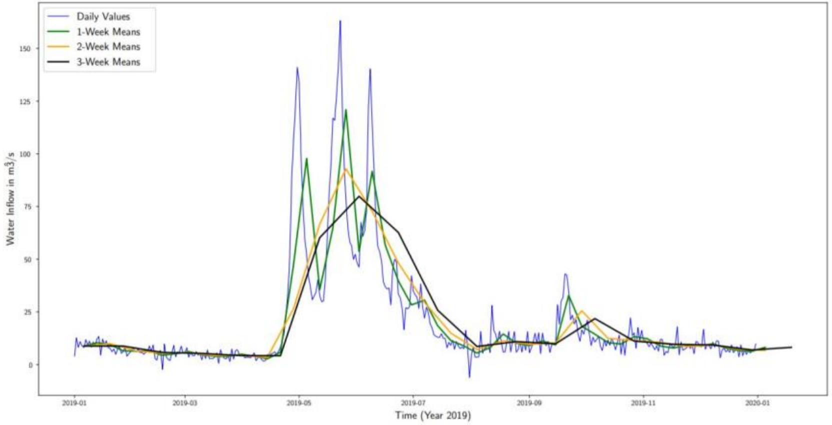
- Data quality:
  1. More noise in the data (Not resampled to weekly averages)
  2. More fluctuations
  3. More negative values ➡ Normalized metrics = divided by mean of actual values (which might be close to 0 due to negative values) ➡ "Worse" accuracy based on metric ➡ same holds for monthly forecasts (mean closer to 0)

$$MAEn = \frac{\frac{1}{N}\sum_{i=1}^{N}|Y_i - \hat{Y}_i|}{\frac{1}{N}\sum_{i=1}^{N} Y_i}$$

$$RMSE = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2}}{\frac{1}{N}\sum_{i=1}^{N} Y_i}$$

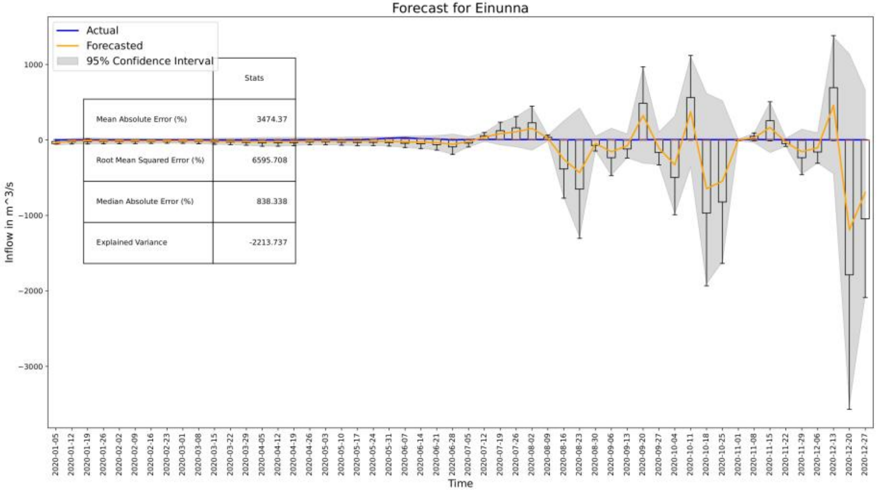
$$MedianAbsoluteError = \frac{median(|Y_1 - \hat{Y}_1|, ..., |Y_N - \hat{Y}_N|)}{\frac{1}{N}\sum_{i=1}^{N} Y_i}$$

$$ExplainedVariance = 1 - \frac{\sum_{i=1}^{N}|Y_i - \hat{Y}_i|}{\sum_{i=1}^{N}(Y_i - (\frac{1}{N}\sum_{j=1}^{N} Y_i))^2}$$

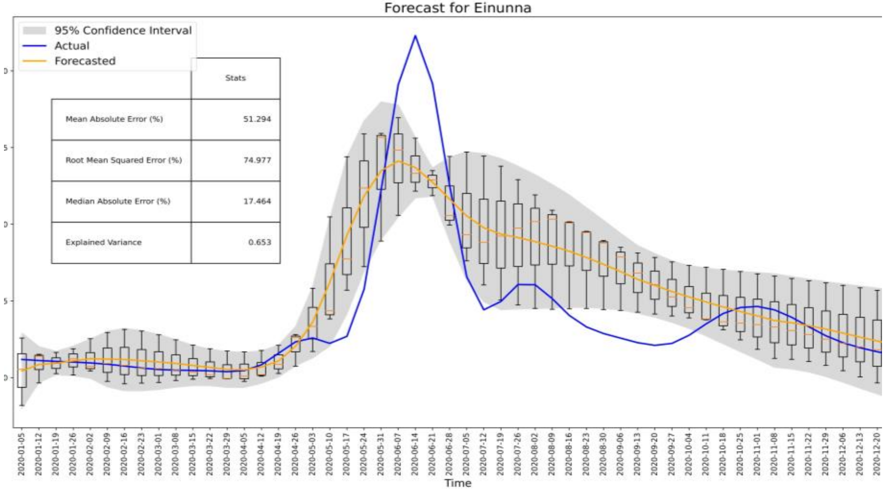


Other resampling methods

1W-average: ☹



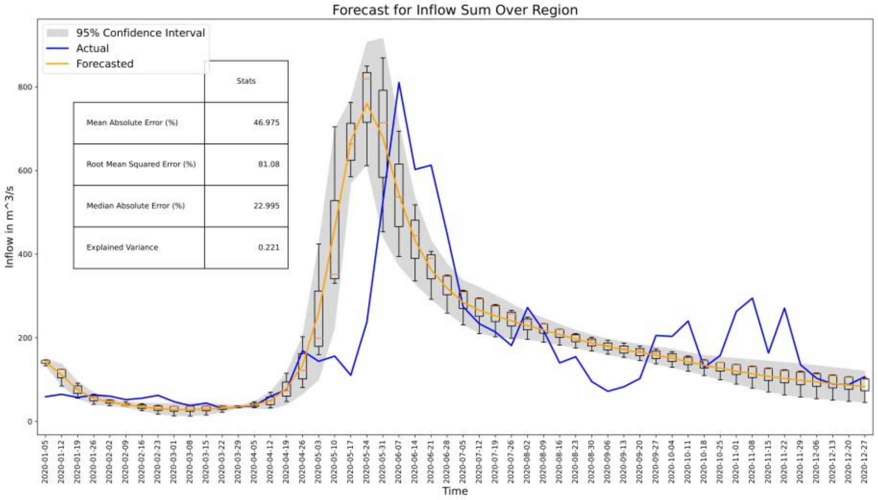
3W-average: ☺

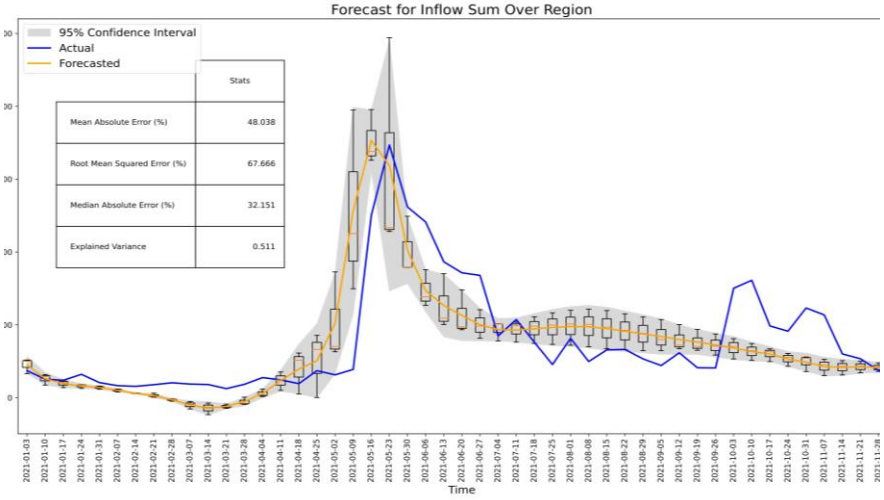


➡Remove noise through coarser resampling methods

Columns summed up - region view

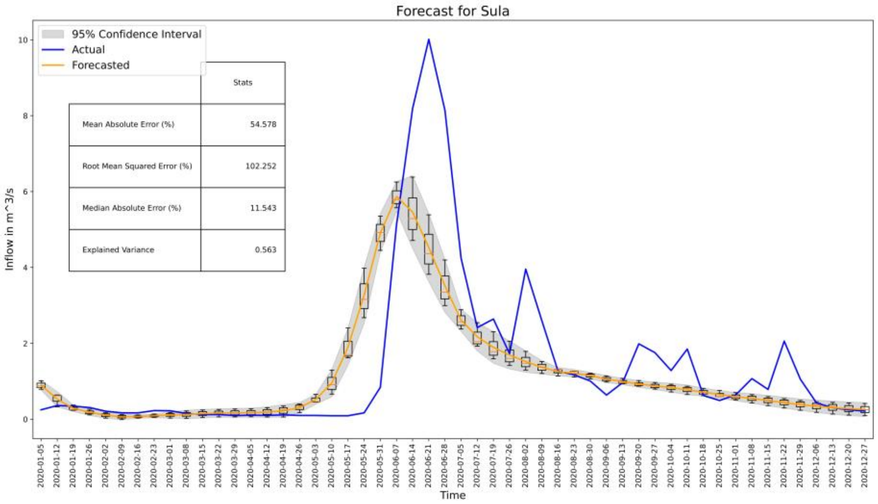
☺ ☺

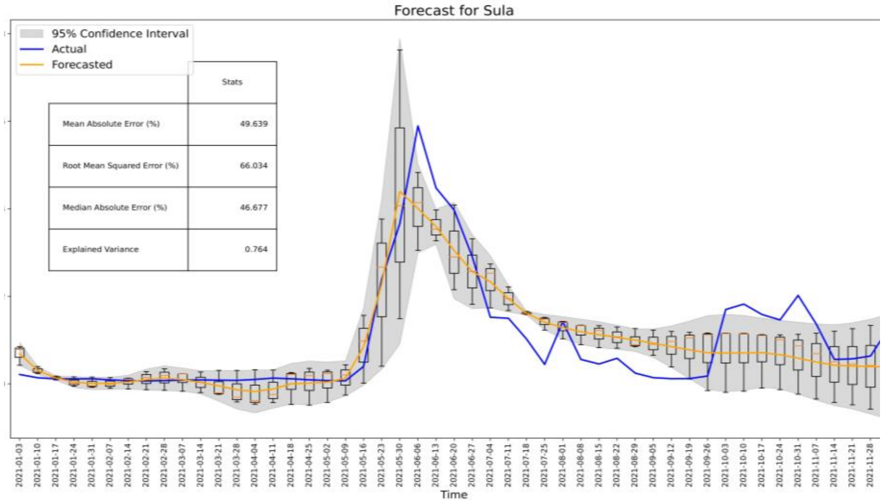




Second datasource (reservoir Sula)

☺ ☺





Learnings

- Tensorflow and also improved coding knowledge in general in Python
- Real-world data is not always nice → how to get most value out of it (imputing methods)
- ML tools

- Real project using Tensorflow
- Working on multi-disciplinary project
- Automatic hyperparameter tuning: Optuna

- The real world is not Kaggle
- Good code quality is important
- Use the features of GitLab/GitHub

- Learned how to work on a coding project
- Use of GitLab
- Structure and building blocks of a ML model
- Data visualization
- Combat overfitting

- Using requests model to download data from website
- Real world data is not always perfect
- Data preprocessing is very important to train a model

Conclusion

Researching related work

Developing the *DataLoader*

- Filling in the missing values in the downloaded data
- Time resolution of the data

Model architecture

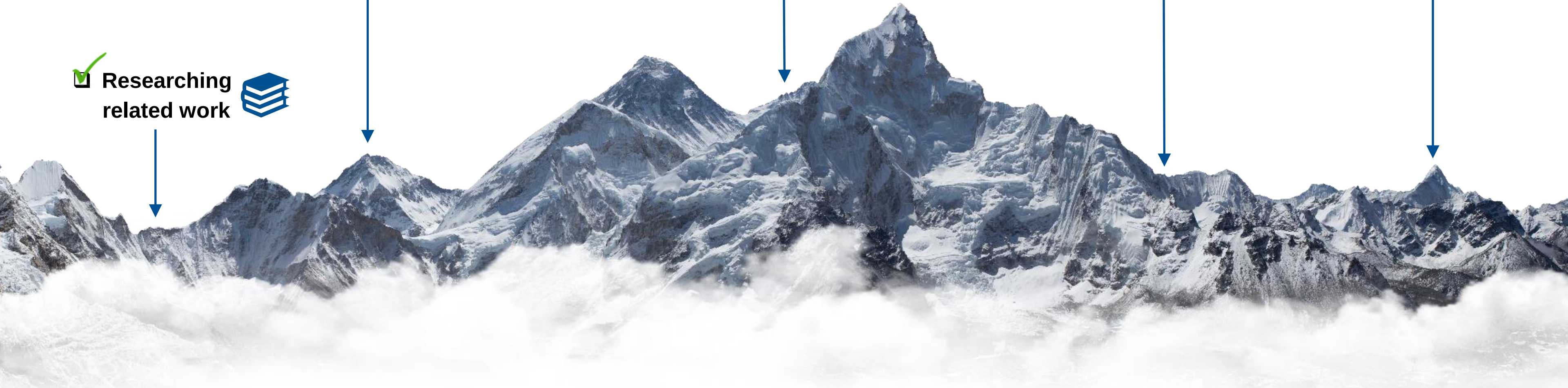
- Improving the code quality, flexibility and readability
- Creating building blocks to remove code duplicates
- Creating config files
- Combat overfitting

OPTUNA

Automatic hyperparameter tuning

Testing

- Yearly forecasts
- Monthly forecasts
- Weekly forecasts
- Sum over all stations



Conclusion

- Norwegian data was more challenging
- We did not expect this many missing values
- Code quality was greatly improved
- Forecast for different time resolutions and lengths possible
- Flexible and easy to use
- Positive feedback from all team members and our mentor Dr. Juliane Sigl

Thank you for your attention
We are looking forward to your questions ?

Robin

Forhad

Florian

Fabienne

Wudamu