Flash Floods Forecasting with Deep Learning

Adam Shtrasner, Asif Kagan

November 25, 2023

1 Abstract

Floods pose a danger to populations around the world, causing great economic damage and loss of lives every year [Jonkman, 2005]. Real-time flood forecasting can save lives, enable the preparation of emergency responders, protect infrastructures and more. The common approach to flood forecasting is based on process-based models, but it is difficult to achieve a prediction with a high level of accuracy due to the complexity of the processes involved in the creation of floods. In recent years, the use of machine learning methods in hydrological research has increased, But there are still almost no examples of operational flood forecasting models based on these methods [Nevo et al., 2022]. Moreover, to the best of our knowledge, there is no machine learning model for operative prediction of flash floods, i.e. floods that are a direct result of a rainstorm and the response to them is fast compared to floods in large rivers (riverine floods), and therefore more challenging. In Israel, all floods are Flash floods.

Flood forecasting in Israel is currently carried out at the joint center of the Water Authority and the Meteorological Service. It is based on several process-based models, which receive as input measured and predicted meteorological data, and calculate the expected flows at several sites of interest. A combination of prediction using deep machine learning models with the existing system can lead to its significant improvement, to increase the accuracy and reliability of this system.

2 Introduction

2.1 Background

Floods have consistently posed challenges globally, both in terms of human safety and economic impacts. The issue becomes even more pressing when we consider the quick and unpredictable nature of flash floods, which are particularly prominent in regions like Israel. Traditionally, our approach to tackle this problem has revolved around process-based models. These models (shown in [Emerton et al., 2016], [Krajewski et al., 2017]) take into account various meteorological parameters, aiming to offer a forecast that can help in timely preparations. However, the multifaceted intricacies of hydrological processes sometimes limit the precision of these models.

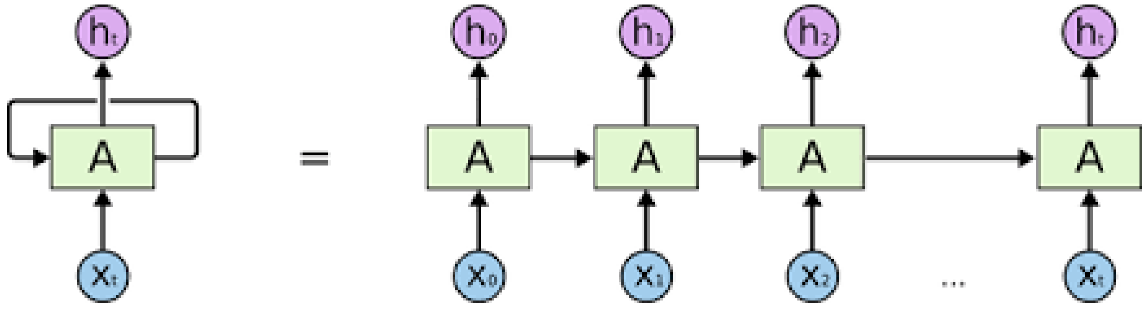
Recently, there's been a shift in the academic community towards exploring machine learning for hydrological predictions. Machine learning, given its data-driven nature, holds the potential to identify patterns and make predictions that might be less intuitive for traditional models. Despite the promise it holds, the practical application of machine learning in flood forecasting remains relatively uncharted. This is especially true for flash floods, which due to their rapid onset following rainstorms, demand a more instantaneous and precise prediction mechanism.

Currently, in Israel, flood predictions are managed jointly by the Water Authority and the Meteorological Service. By leveraging a set of process-based models and incorporating meteorological data, they strive to offer predictions for areas most at risk. Given the predominant threat of flash floods in the region, enhancing the speed and accuracy of these forecasts is essential. An integration of deep learning techniques with the existing infrastructure could potentially bridge this gap, refining the prediction quality and reliability.

2.1.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) represent a class of artificial neural networks designed to recognize patterns in sequences of data. Originating from traditional feedforward neural networks, RNNs possess connections that loop back, enabling them to retain a memory of previous inputs. This looping architecture is particularly apt for handling data with temporal dependencies, like time series or speech.

However, standard RNNs face challenges when it comes to long-term dependencies due to issues like vanishing or exploding gradients. This means that RNNs can struggle to carry information over longer sequences, which can limit their effectiveness in tasks requiring an understanding of longer contexts.



2.1.2 Long Short-Term Memory networks (LSTMs)

Long Short-Term Memory networks (LSTMs) are a special kind of RNN, specifically engineered to avoid the long-term dependency challenges that plague traditional RNNs. Developed by Hochreiter and Schmidhuber in 1997, LSTMs have since been a cornerstone in many successful RNN applications.

LSTMs introduce the concept of gates – specifically, the input, forget, and output gates. These gates determine how much information should be let in, discarded, or passed on, respectively. This mechanism allows LSTMs to regulate the flow of information, ensuring that relevant details are retained over longer sequences, making them particularly useful for tasks where context over extended periods is crucial.

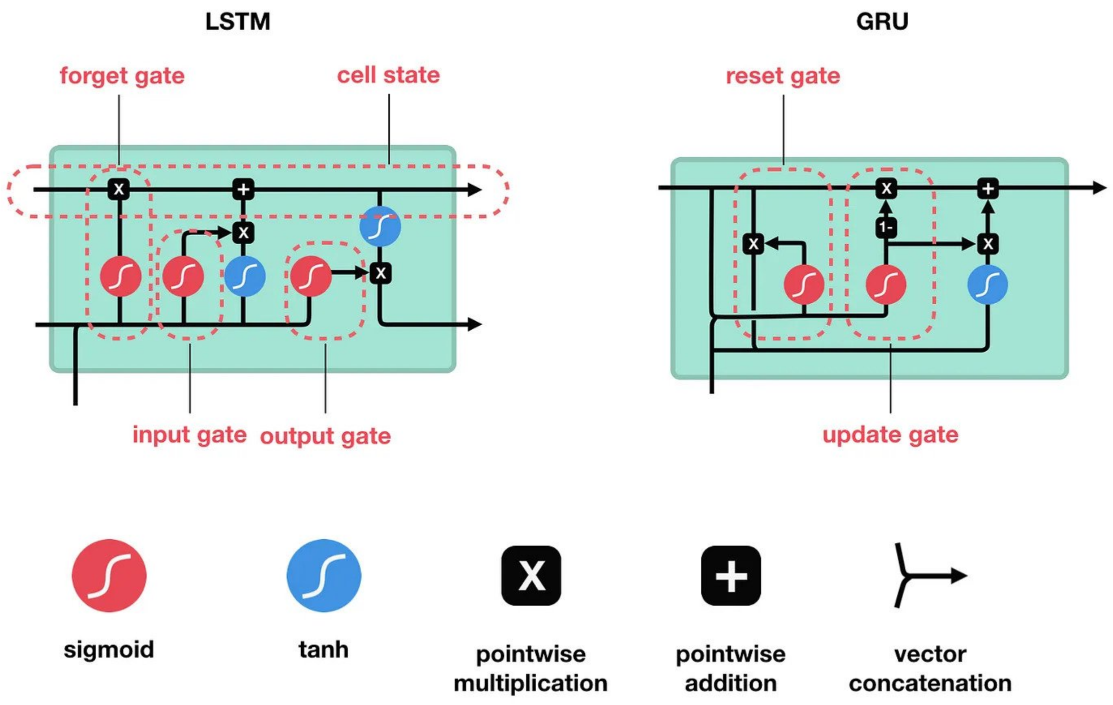
In the realm of time series forecasting, LSTMs have shown significant promise. Given the sequential nature of time series data, LSTMs' inherent capability to remember past information makes them apt for predicting future values based on historical data. This has led to their application in diverse fields, from financial forecasting to predicting weather patterns.

2.1.3 Gated Recurrent Units (GRUs)

Gated Recurrent Units (GRUs) are a type of RNN architecture that seeks to solve the long-term dependency issues encountered by basic RNNs. Introduced by [Cho et al., 2014], GRUs have rapidly gained traction for their efficiency and relative simplicity compared to other recurrent network architectures.

At the heart of GRUs are two critical gates: the update gate and the reset gate. The update gate determines how much of the previous memory to keep around, while the reset gate defines how much of the past information to forget. By judiciously leveraging these gates, GRUs can effectively maintain relevant historical information, even over extended sequences.

The main difference between GRU and LSTM lies in their internal structure, with GRU having a simpler design with two gates compared to LSTM's three gates. This simplicity makes GRU computationally less expensive and potentially more suitable for smaller datasets.



2.2 Related Work

2.2.1 Flood forecasting with machine learning models in an operational framework

Google's operational flood forecasting system, launched in 2018, provides real-time flood warnings, primarily focusing on riverine floods in large, gauged rivers. The system integrates machine learning in its stage forecasting and inundation modeling, utilizing both linear models and long short-term memory (LSTM) networks. During the 2021 monsoon season, it was operational in India and Bangladesh, covering regions home to over 350 million people and dispatching over 100 million flood alerts. The initiative aims to expand its reach and enhance its modeling capabilities in the future.

The article underscores the significance of flood warnings, especially in regions vulnerable to natural disasters. As floods pose a substantial threat to lives and economies, especially in low and middle-income countries, timely and accurate warning systems are paramount. Google's system, which is designed for deployment in large gauged rivers, emphasizes the use of river stage data. By integrating advanced machine learning techniques, such as LSTM networks, with traditional linear models, the system aims to offer a comprehensive solution to flood forecasting. The successful deployment in India and Bangladesh during the 2021 monsoon season serves as a testament to its potential, setting the stage for its broader application in other flood-prone regions.[Nevo et al., 2022]

2.3 Data

Our data contains recorded data from rain stations that are obtained from the meteorological service, and additional rainfall data that are obtained from broadcasting stations of the meteorological service. Moreover, the main data is comprised of 2 datasets, including:
1. Stream flow data - obtained from broadcasting stations of the hydrological service. This data set contains flow data records from 1943 to some stations up until 2021, including time series of the station's stream flow. 2. Drainage basin polygons for each station - obtained from the hydrological service. This data set includes information on 96 stations such as the drainage basin area, exact location of the station, and the name of the station.

2.3.1 Data Characteristics

From the myriad of available stations, we strategically selected five stations, ensuring representation across diverse regions and climatic zones within Israel. The dataset for these stations comprised the following:

Flow Rate and Water Level: All five stations provided flow rate data (measured in m^3/s) and water level data (measured in meters).

Rainfall Data Integration: Four out of these five stations possessed rainfall data sourced from the Israeli Meteorological Service (IMS) spanning the period 2018 to 2021. Notably, the fifth station was exclusively equipped with flow data, covering an extensive timeline from 1964 to 2018, but lacked corresponding rainfall records.

Rain Station Proximity: The rainfall data for the four aforementioned stations were derived from the three geographically closest rain stations. This proximity determination leveraged GIS technology to identify rain stations situated within a 10 km radius of the primary hydrological station.

Data Resampling: To ensure coherence in time resolution across datasets, we undertook a meticulous resampling process. Moreover, the optimal time resolution was individually determined for each station, influenced by the preliminary results and insights gleaned.

Model Data Partitioning:

- For the four stations (Ayalon, Gaaton Ben 'Ammi, Qishon and Soreq) including the rainfall data:
  - Model fitting utilized data from the years 2018 and 2019.
  - Validation was performed on the year 2020, adhering to the "leave one year out" principle, a methodology referenced in several foundational papers we consulted.
  - Testing was executed using 2021 data.
- For the fifth station (Bet Lehem) devoid of rainfall data:
  - Model fitting spanned the years 1964 to 2000.
  - The year 2001 was designated for validation.
  - The test phase encompassed the period 2002 to 2018.

3 Results

3.1 Exploring Optimal Lead Time for Flash Flood Predictions

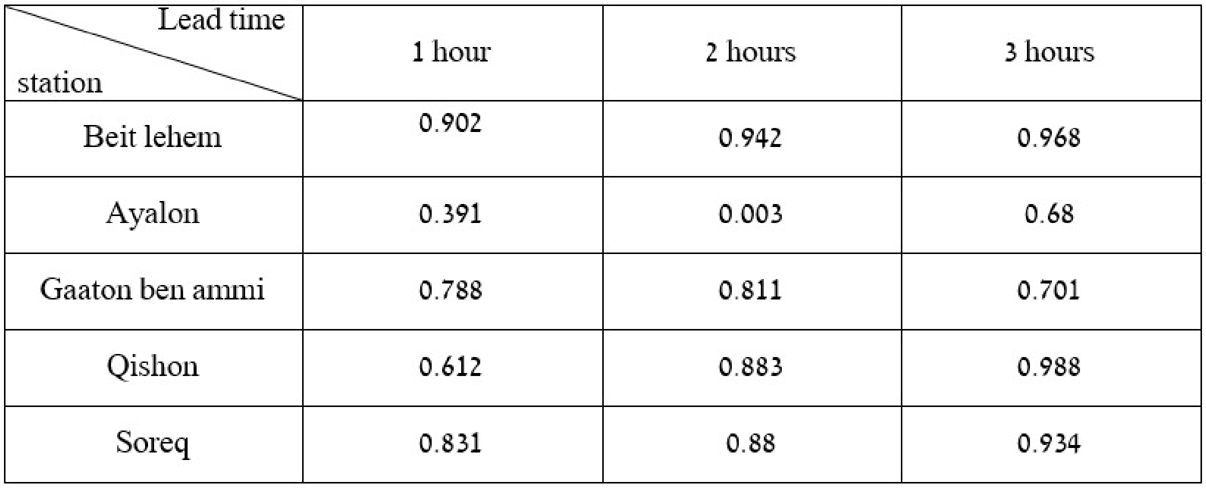
The concept of lead time is central to our predictive modeling; it represents the time window into the future wherein our predictions apply. Determining the most effective lead time requires careful consideration of a critical trade-off: the balance between the temporal advantage it offers to authorities and the accuracy of the prediction itself.

A shorter lead time inherently provides more accurate predictions. For instance, with a lead time of zero, our model can ascertain real-time occurrences of flash floods. While this immediate prediction might be precise, it affords authorities minimal time to alert the populace and mobilize rescue operations.

Conversely, a more extended lead time, while giving authorities an advantage in terms of preparation, could compromise the accuracy of the prediction due to the inherent uncertainties of forecasting over longer periods.

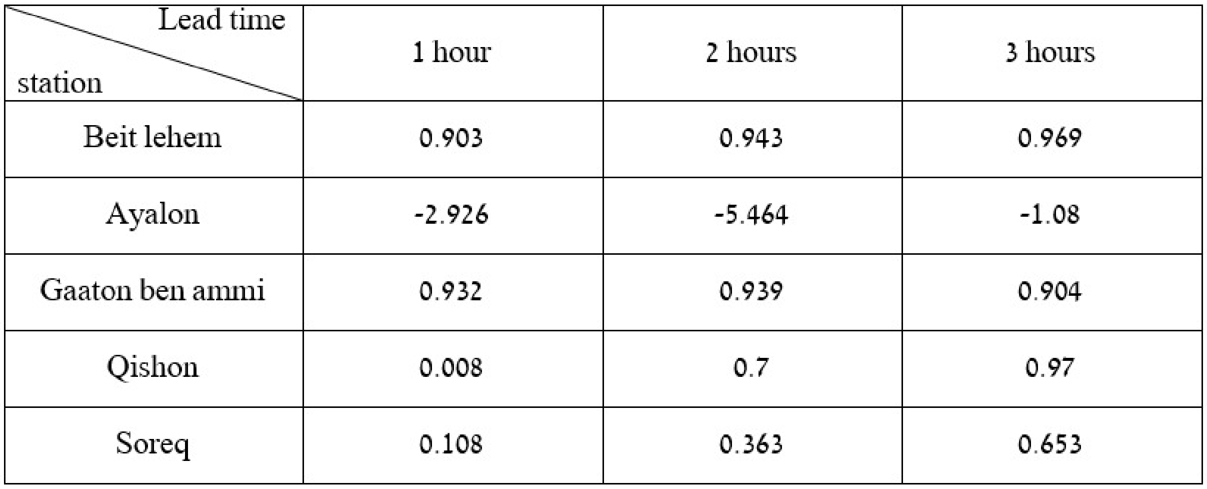
To systematically determine the optimal lead time, we evaluated the Nash-Sutcliffe Efficiency (NSE) and persistNSE metrics across varied lead times, holding the number of iterations constant. We specifically assessed lead times of 1, 2, and 3 hours.

The method for defining the best suited lead-time to use for each station was to train the model using each of the 3 different lead times - 3 times, and calculate the average NSE and persistNSE scores on the test data. The lead time we chose for each station based on the following results was 3 hours:



Lead time station	1 hour	2 hours	3 hours
Beit lehem	0.902	0.942	0.968
Ayalon	0.391	0.003	0.68
Gaaton ben ammi	0.788	0.811	0.701
Qishon	0.612	0.883	0.988
Soreq	0.831	0.88	0.934

(a) NSE



Lead time station	1 hour	2 hours	3 hours
Beit lehem	0.903	0.943	0.969
Ayalon	-2.926	-5.464	-1.08
Gaaton ben ammi	0.932	0.939	0.904
Qishon	0.008	0.7	0.97
Soreq	0.108	0.363	0.653

(b) persistNSE

Figure 1: NSE and peristNSE scores for each of the 3 lead times and for each station

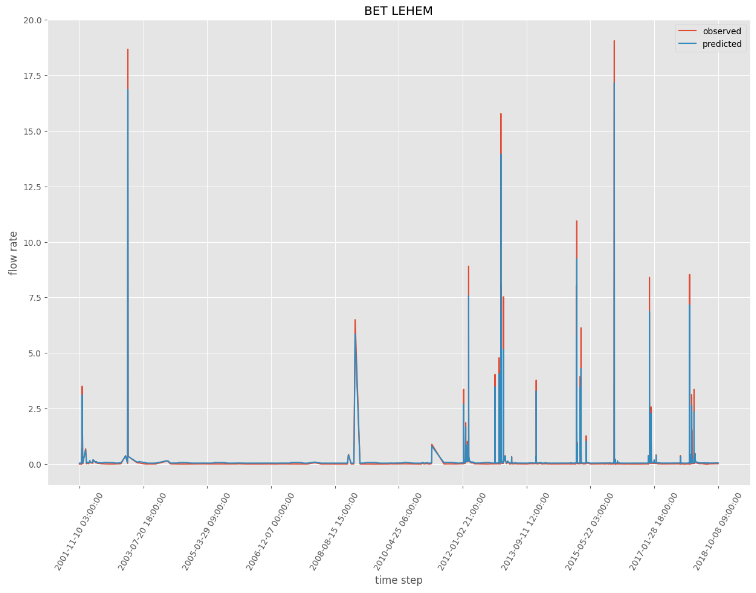
3.2 Bet Lehem Station

The initial values of the hyperparameters for both models are as follows:

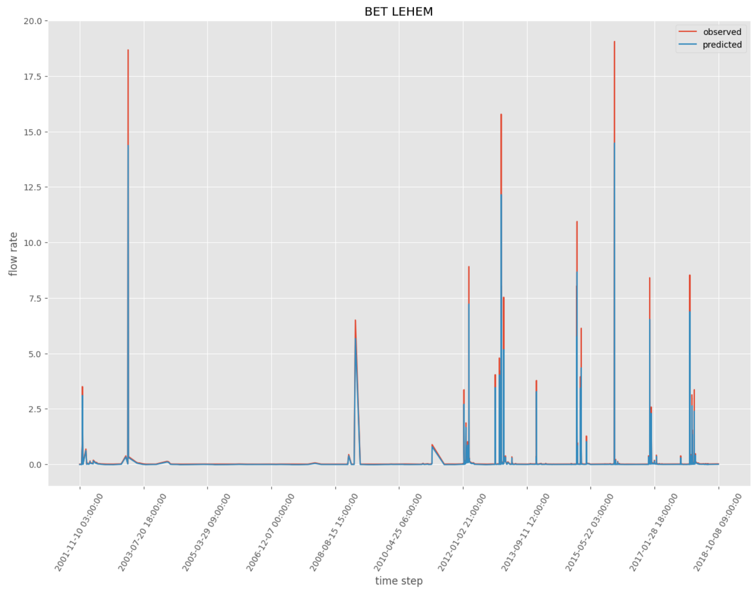
- batch_size = 256
- epochs = 5
- n_neurons = 128

Also, after experimenting with different values for time resolution, the resolution which suited best was $\Delta t = 1\ hour$.

3.2.1 Observed VS Predicted - LSTM and GRU



(a) LSTM



(b) GRU

Figure 2: Observed flow rate (red) vs predicted flow rate (blue)

The test scores for the LSTM model:

Test NSE: 0.972
Test PersistentNSE: 0.973

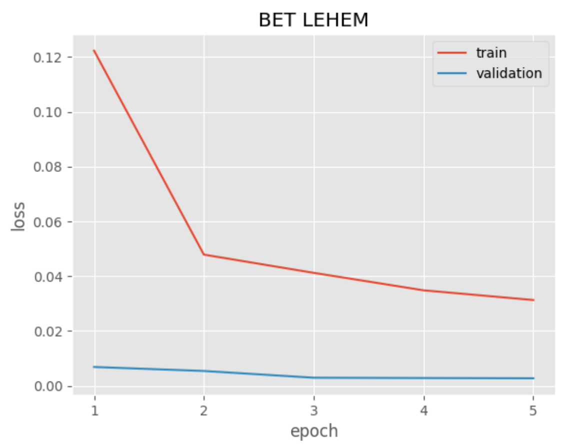
The test scores for the GRU model:

Test NSE: 0.960
Test PersistentNSE: 0.961

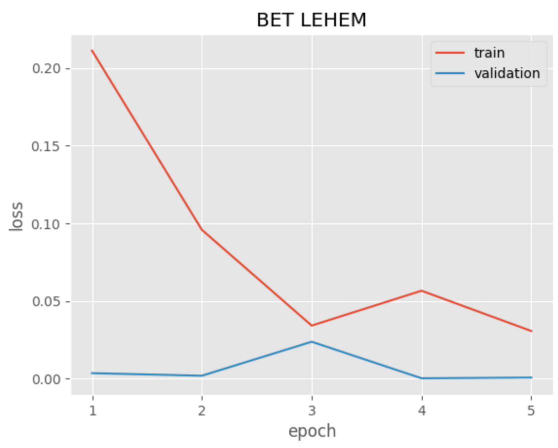
We can see that generally both models performed well while the LSTM showed slightly better results on extreme flow rates.

The test scores are good, and we've decided that there's no need for hyperparameter optimization.

3.2.2 Comparison - LSTM and GRU



(a) LSTM

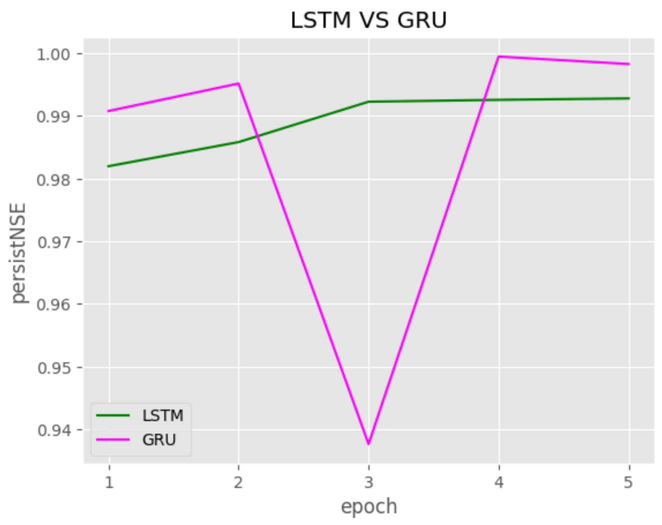


(b) GRU

Figure 3: NSE* loss: train (red) vs validation (blue)

The loss graphs align with the results above, where we see that the train and validation loss in the LSTM model were more consistent with its decrease while the loss in the GRU model increased for the train in the 4th epoch and the validation in the 3rd epoch.

The validation score of both models is as follows:



We can see that the validation score is more consistent in the LSTM model than in the GRU model, although except for the large decrease in the 3rd epoch - the validation score in the GRU model were slightly higher.

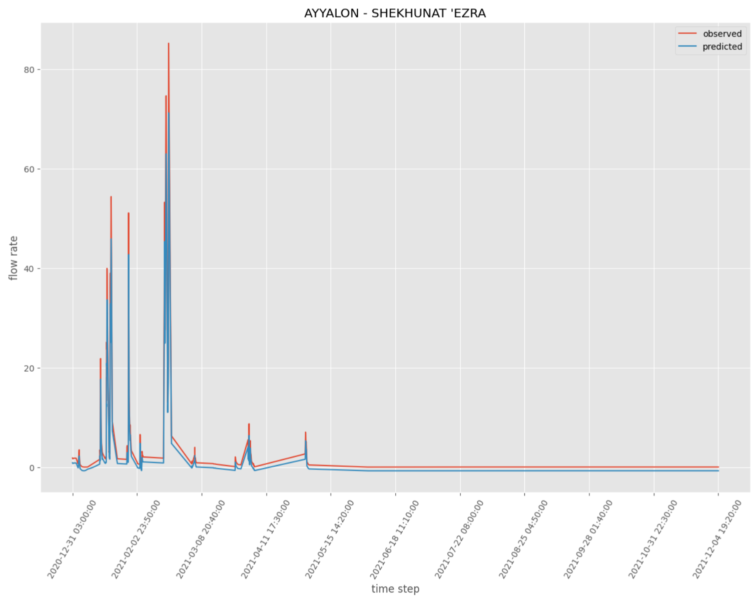
3.3 Ayalon Station

The initial values of the hyperparameters for both models are as follows:

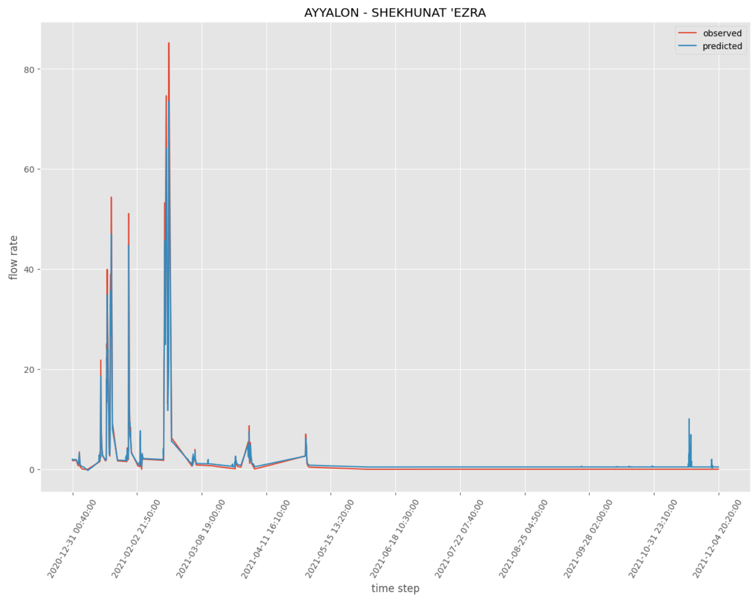
- batch_size = 128
- epochs = 5
- n_neurons = 128

Also, after experimenting with different values for time resolution, the resolution which suited best was $\Delta t = 10\ minutes$.

3.3.1 Observed VS Predicted - LSTM With and Without Rainfall Data



(a) LSTM without rainfall data



(b) LSTM with rainfall data

Figure 4: Observed flow rate (red) vs predicted flow rate (blue)

The test scores for the LSTM model without rainfall data:

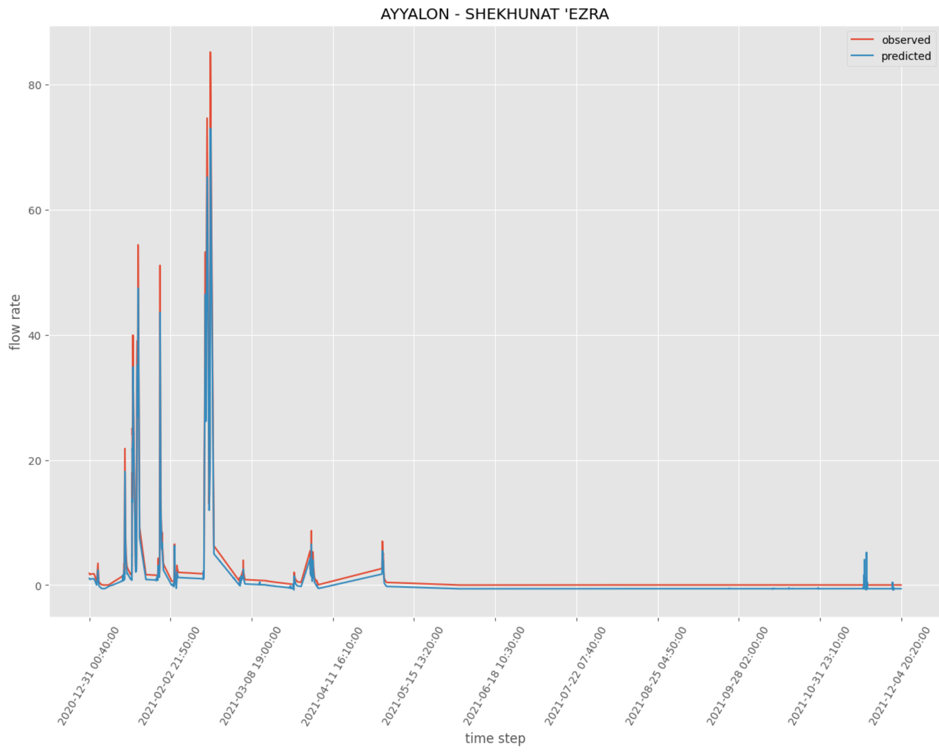
Test NSE: 0.957
Test PersistentNSE: 0.957

The test scores for the LSTM model with rainfall data:

Test NSE: 0.975
Test PersistentNSE: 0.975

We can see that in both cases, the model had trouble predicting the extreme values, but was successful on predicting trends. Moreover, after we added the rainfall data to the LSTM model, it predicted more accurately.

3.3.2 Observed VS Predicted - GRU



The results of the GRU model are quite similar to the results given by the LSTM model. The test scores for the GRU model:

Test NSE: 0.972
Test PersistentNSE: 0.971

Although the results are good, we wanted to see whether we can improve both of the models, and get more accurate predictions on the extreme values. After conducting hyperparameter optimization, we noted an improvement in the performance of the LSTM model, in contrast to the GRU model where we exhibited a decline in performance post-optimization.

This outcome can be attributed to several factors inherent to the nature of LSTM and GRU architectures and their interaction with hyperparameter tuning.

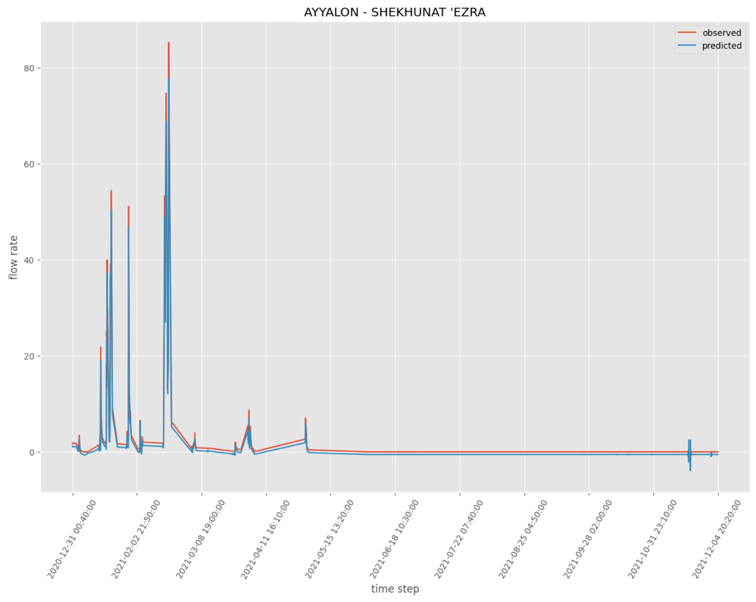
It's also crucial to consider the possibility of overfitting in the context of hyperparameter optimization. In our case, the LSTM might have benefited from a fine-tuned balance between model complexity and generalization, whereas the GRU could have diverge towards overfitting or underfitting, leading to a decrease in performance.

3.3.3 Hyperparameter Optimization - LSTM

LSTM's best hyperparameters:

- batch_size = 128
- epochs = 15
- n_neurons = 128

3.3.4 Observed VS Predicted - Optimized LSTM



(a) LSTM

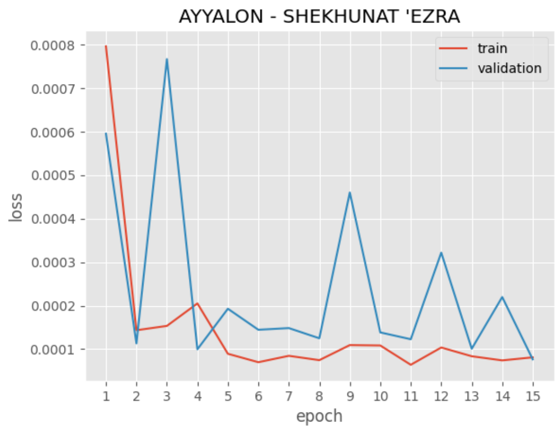
Figure 5: Observed flow rate (red) vs predicted flow rate (blue) after hyperparameter optimization

The test scores for the optimized LSTM model:

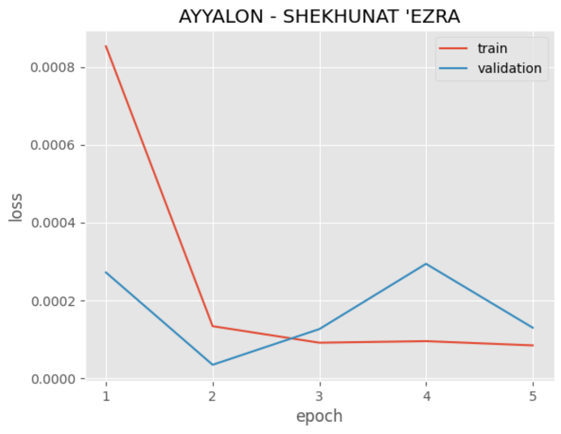
Test NSE: 0.982
Test PersistentNSE: 0.981

We can see that the model is slightly improved - the test scores increased by 0.001, and the optimized LSTM was able to predict the extreme high flow rates better.

3.3.5 Comparison - LSTM and GRU



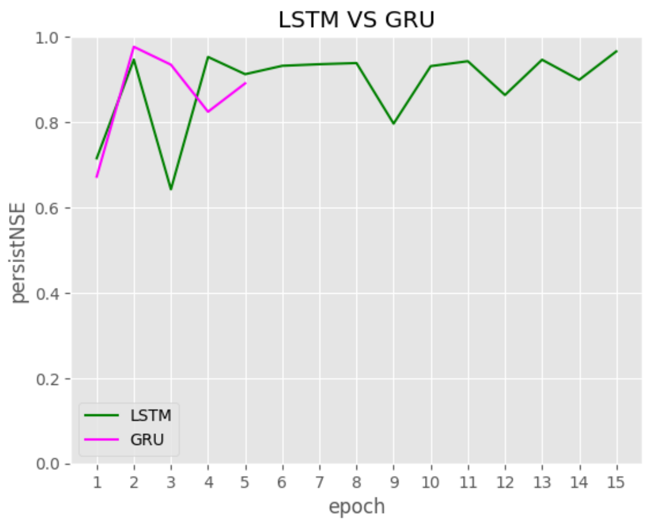
(a) LSTM



(b) GRU

Figure 6: NSE* loss: train (red) vs validation (blue) after hyperparameter optimization for the LSTM

The validation scores for both the LSTM and GRU are:



We can conclude that both models performed well, with the LSTM being slightly better.

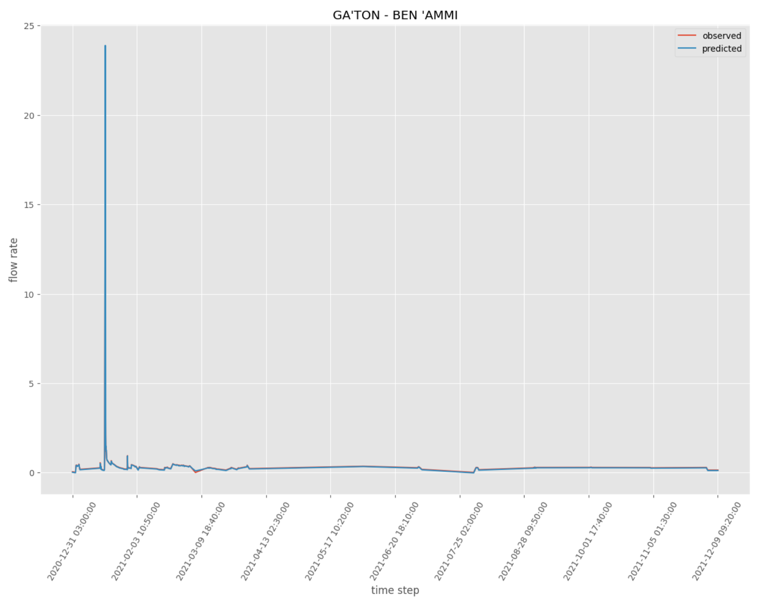
3.4 Gaaton Ben 'Ammi Station

The initial values of the hyperparameters for both models are as follows:

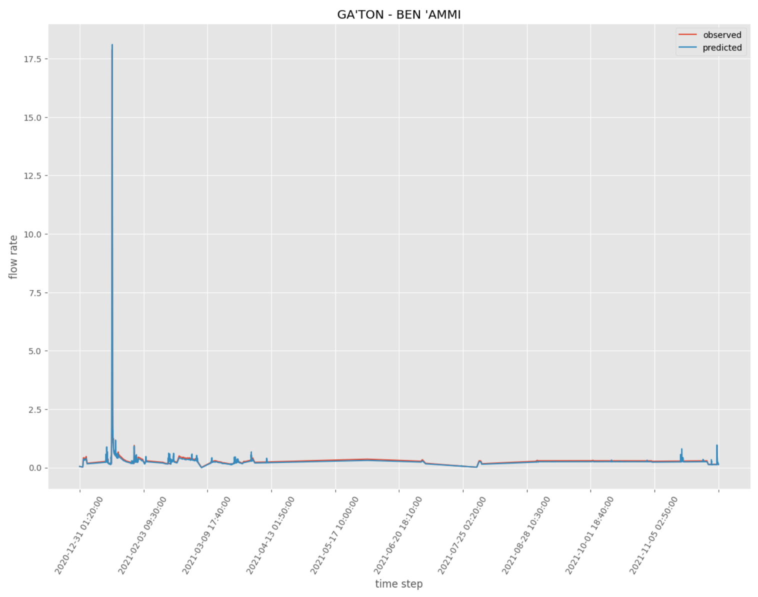
- batch_size = 128
- epochs = 5
- n_neurons = 128

Also, after experimenting with different values for time resolution, the resolution which suited best was $\Delta t = 10\ minutes$.

3.4.1 Observed VS Predicted - LSTM with and without rainfall data



(a) LSTM without rainfall data



(b) LSTM with rainfall data

Figure 7: Observed flow rate (red) vs predicted flow rate (blue)

The test scores for the LSTM model without rainfall data:

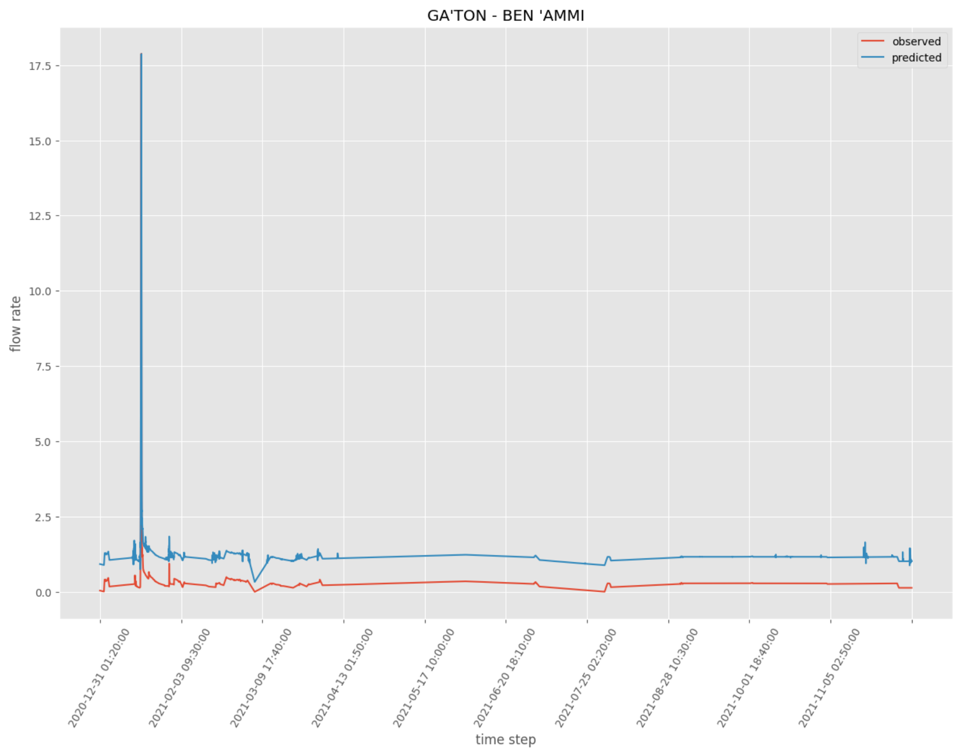
Test NSE: 0.846
Test PersistentNSE: 0.862

The test scores for the LSTM model with rainfall data:

Test NSE: 0.963
Test PersistentNSE: 0.966

We can see that the LSTM managed to predict the flow almost perfectly in both cases, while the test scores are better when rainfall data is added. For that reason, we've decided not to perform hyperparameter optimization on the LSTM model.

3.4.2 Observed VS Predicted - GRU



The test scores for the GRU model:

Test NSE: -3.931
Test PersistentNSE: -3.429

We can see that the GRU model was not successful at predicting the flow rates correctly, and the model is bad (negative score).

3.4.3 Hyperparameter Optimization - GRU

GRU's best hyperparameters:

- batch_size = 32
- epochs = 5
- n_neurons = 64

3.4.4 Observed VS Predicted - Optimized GRU

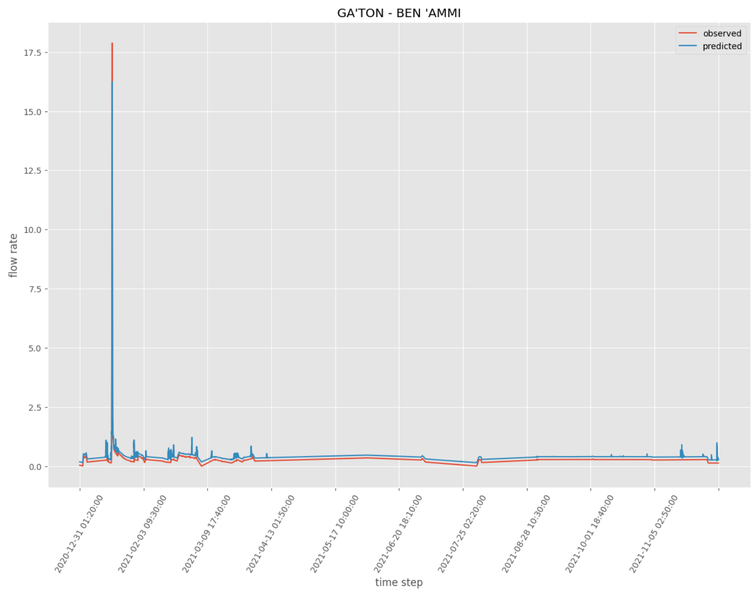


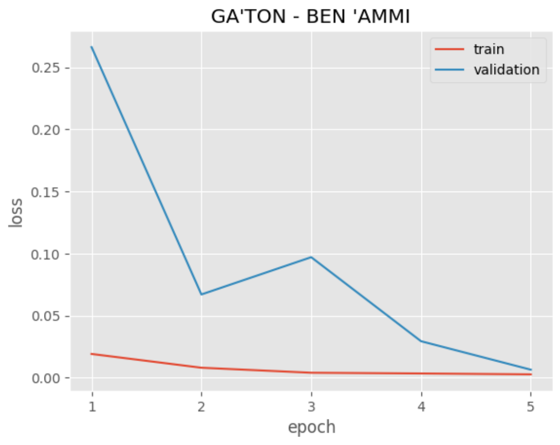
Figure 8: Observed flow rate (red) vs predicted flow rate (blue) after hyperparameter optimization

The test scores for the GRU model:

Test NSE: 0.872
Test PersistentNSE: 0.885

After hyperparameter optimization the GRU is much better and was able to predict the flow rates pretty well.

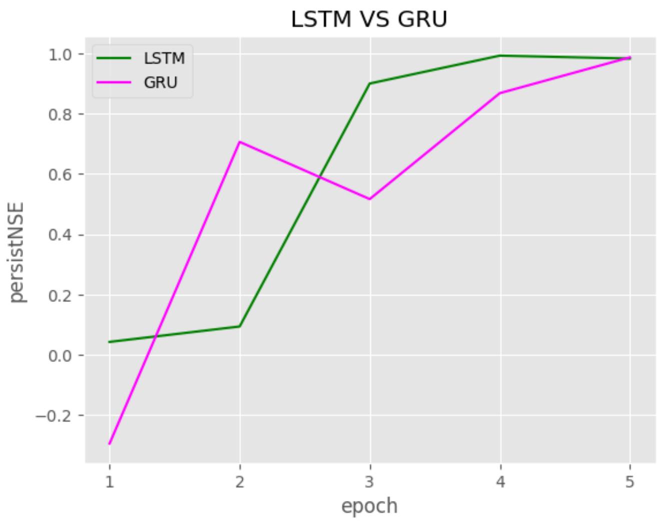
3.4.5 Comparison - LSTM and GRU



(a) GRU

Figure 9: NSE* loss: train (red) vs validation (blue) after hyperparameter optimization

The validation score of both models is as follows:



We can conclude that for the Gaaton station, the LSTM model is more suitable for flash flood prediction than the GRU.

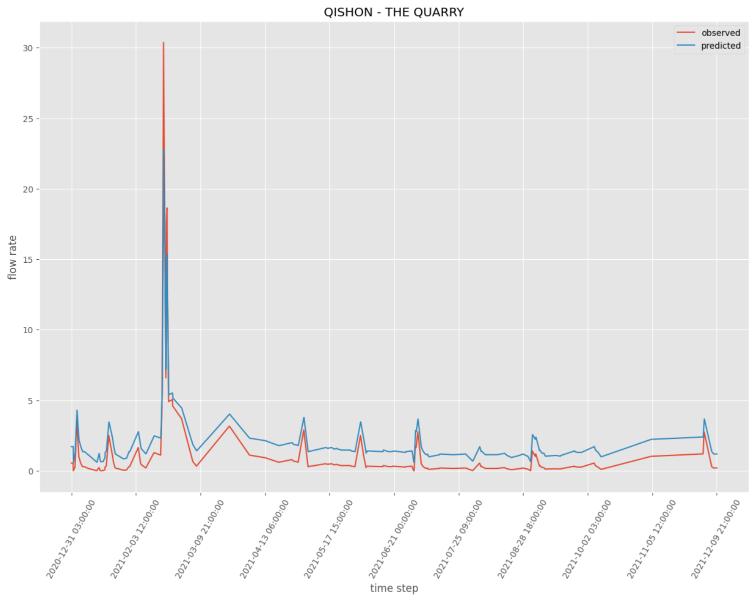
3.5 Qishon Station

The initial values of the hyperparameters for both models are as follows:

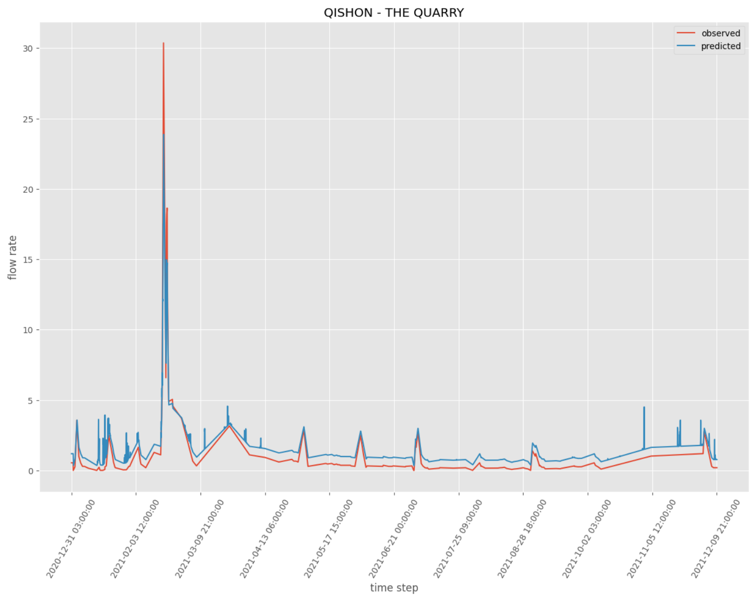
- batch_size = 64
- epochs = 5
- n_neurons = 128

Also, after experimenting with different values for time resolution, the resolution which suited best was $\Delta t = 1\ hour$.

3.5.1 Observed VS Predicted - LSTM with and without rainfall data



(a) LSTM without rainfall data



(b) LSTM with rainfall data

Figure 10: Observed flow rate (red) vs predicted flow rate (blue)

The test scores for the LSTM model without rainfall data:

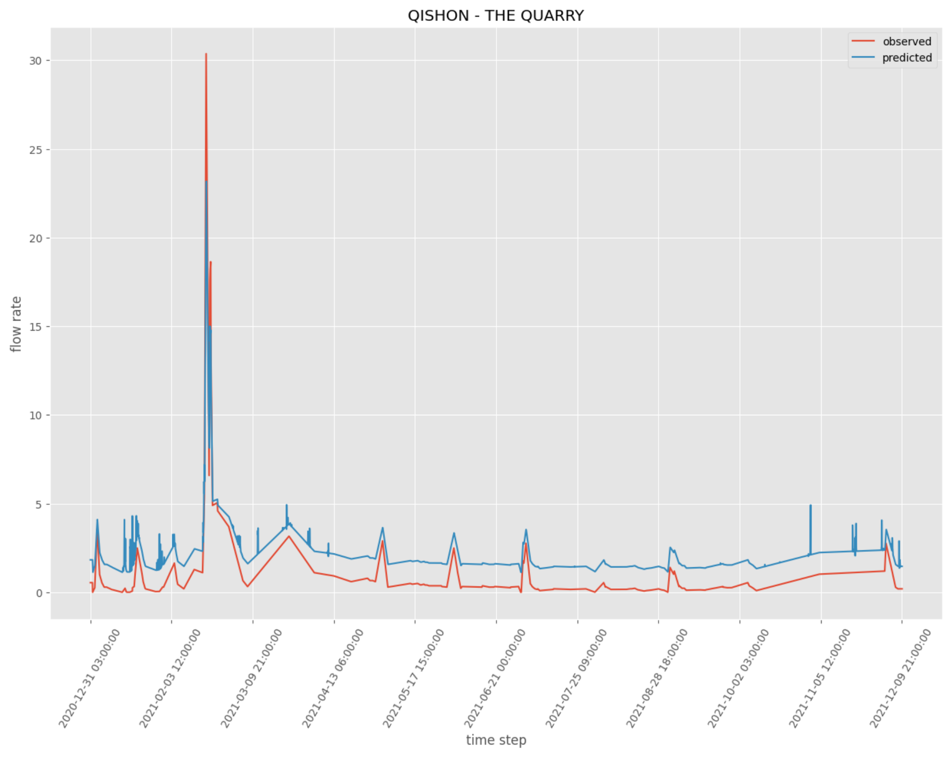
Test NSE: 0.738
Test PersistentNSE: 0.693

The test scores for the LSTM model with rainfall data:

Test NSE: 0.899
Test PersistentNSE: 0.881

We can clearly see that adding the rainfall data contributed to the accuracy of the prediction - the blue line is closer to the red line for the LSTM model with rainfall data. The test scores for the LSTM model with rainfall data are also higher. We can still see that the extreme high values are hard to predict in both cases.

3.5.2 Observed VS Predicted - GRU



The test scores for the GRU model:

Test NSE: 0.654
Test PersistentNSE: 0.595

The GRU performance was bad - the test scores are much lower than the LSTM model's scores, with and without rainfall.

3.5.3 Hyperparameter Optimization - LSTM and GRU

LSTM's best hyperparameters:

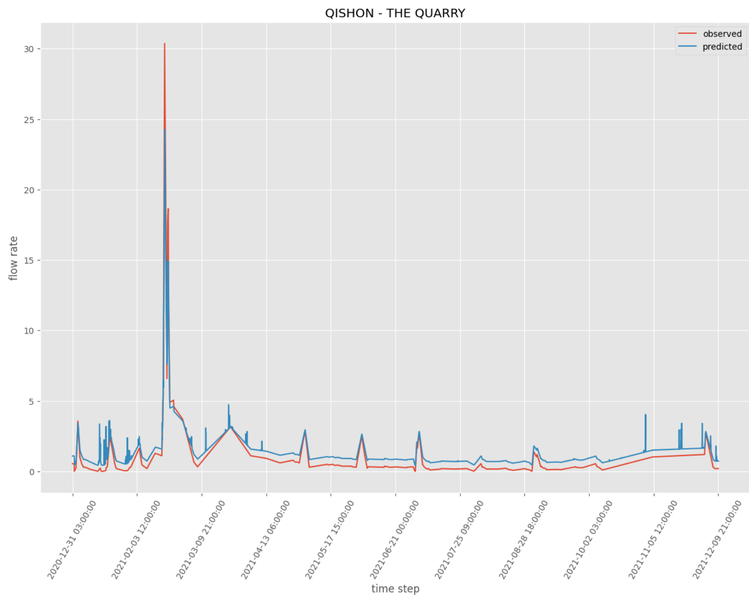
- batch_size = 64

- epochs = 15
- n_neurons = 128

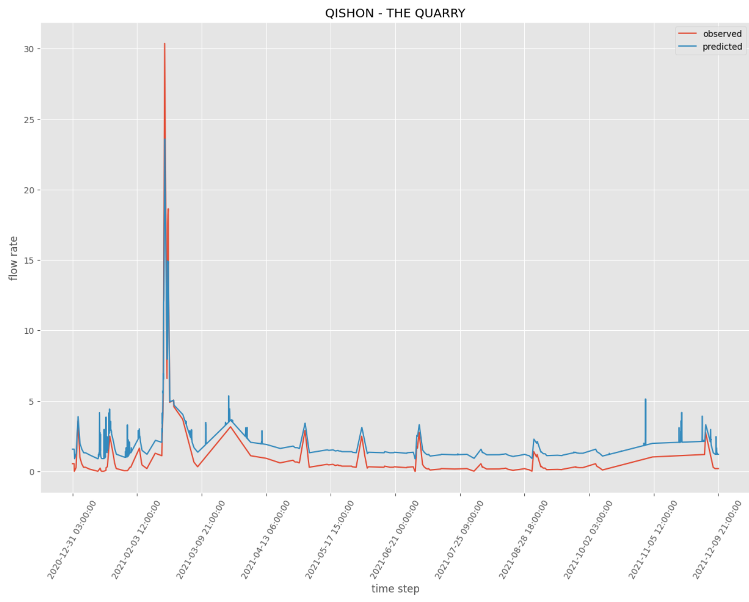
GRU's best hyperparameters:

- batch_size = 64
- epochs = 15
- n_neurons = 128

3.5.4 Observed VS Predicted - Optimized LSTM and GRU



(a) LSTM



(b) GRU

Figure 11: Observed flow rate (red) vs predicted flow rate (blue) after hyperparameter optimization

The test scores for the optimized LSTM model:

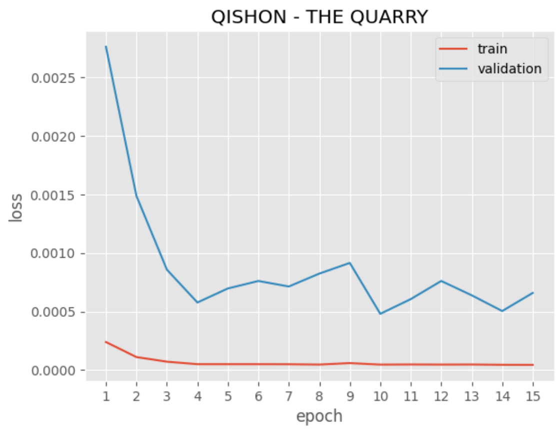
Test NSE: 0.919
Test PersistentNSE: 0.905

The test scores for the optimized GRU model:

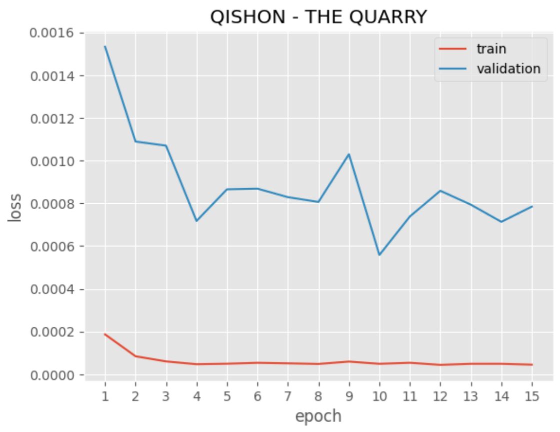
Test NSE: 0.769
Test PersistentNSE: 0.730

We can see that the LSTM was more successful than the GRU, as seen in the graphs - the blue and red lines are closer to each other in graph (a) than in graph (b), and both test scores of the LSTM model are greater than the GRU model.

3.5.5 Comparison - LSTM and GRU



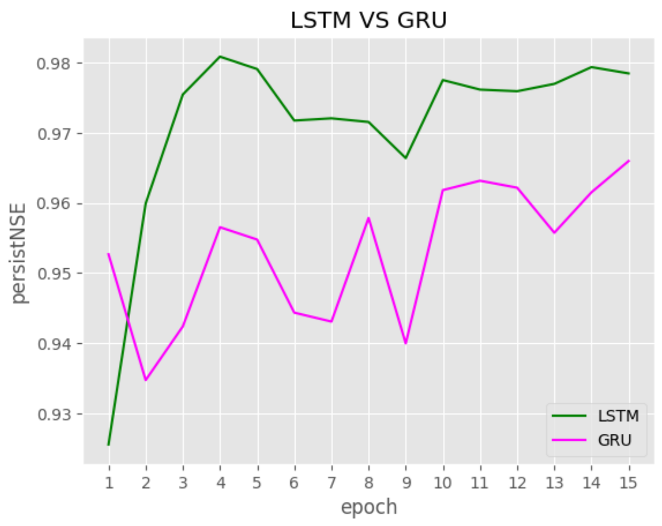
(a) LSTM



(b) GRU

Figure 12: NSE* loss: train (red) vs validation (blue) after hyperparameter optimization

The validation score of both models is as follows:



As seen in the graph, the validation score for the GRU model is lower than the LSTM's validation score for all epochs except the first one. We can conclude that the LSTM model is more fit to deal with Qishon station.

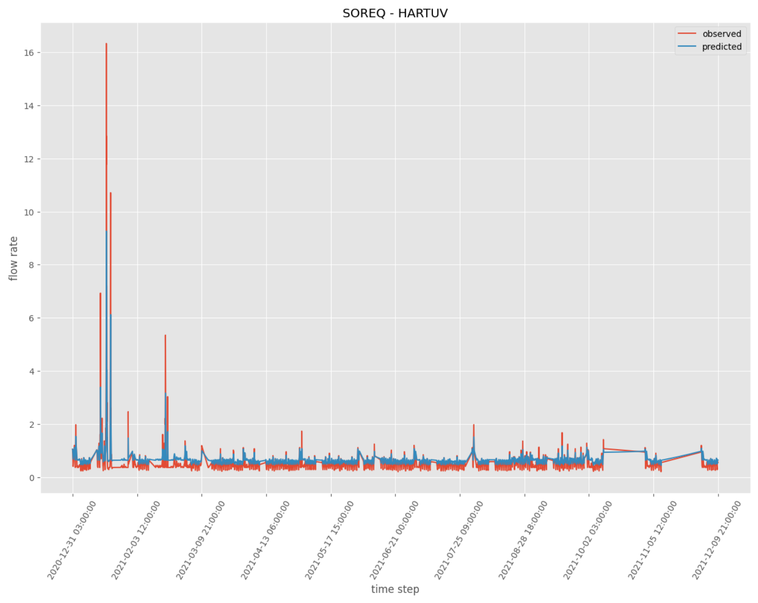
3.6 Soreq Station

The initial values of the hyperparameters for both models are as follows:

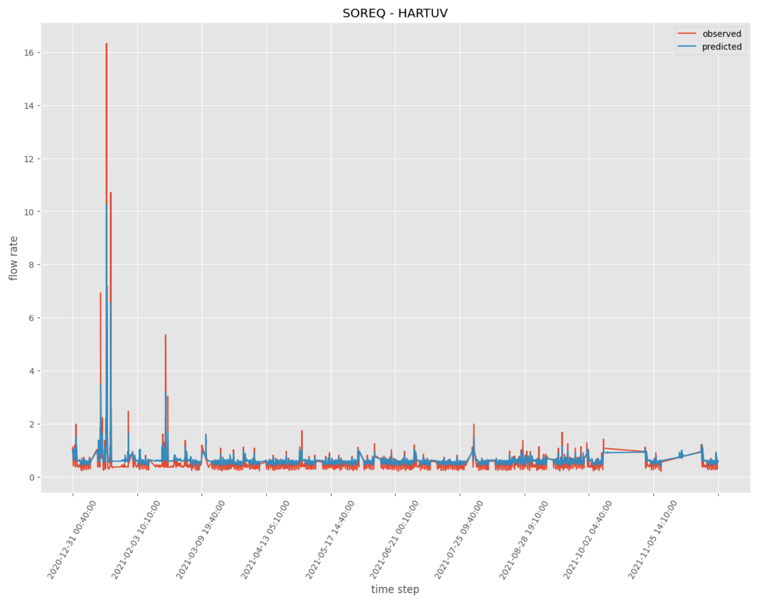
- batch_size = 128
- epochs = 5
- n_neurons = 128

Also, after experimenting with different values for time resolution, the resolution which suited best was $\Delta t = 10\ minutes$.

3.6.1 Observed VS Predicted - LSTM with and without rainfall data



(a) LSTM without rainfall data



(b) LSTM with rainfall data

Figure 13: Observed flow rate (red) vs predicted flow rate (blue)

The test scores for the LSTM model without rainfall data:

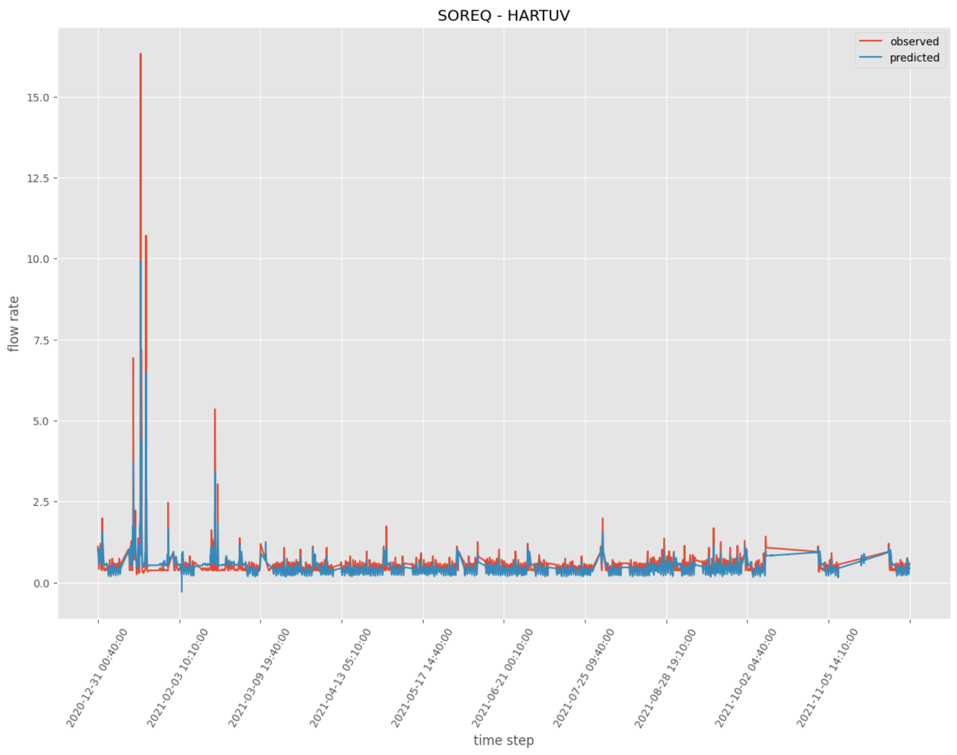
Test NSE: 0.95
Test PersistentNSE: 0.734

The test scores for the LSTM model with rainfall data:

Test NSE: 0.962
Test PersistentNSE: 0.797

Similarly to other stations we can see that adding rainfall data improves the model's accuracy. We can still see, as in previous stations, that the peak values are hard to predict.

3.6.2 Observed VS Predicted - GRU



The test scores for the GRU model:

Test NSE: 0.961
Test PersistentNSE: 0.794

The GRU was as successful as the LSTM with the rainfall data, with almost equal test scores.

3.6.3 Hyperparameter Optimization - LSTM and GRU

LSTM's best hyperparameters:

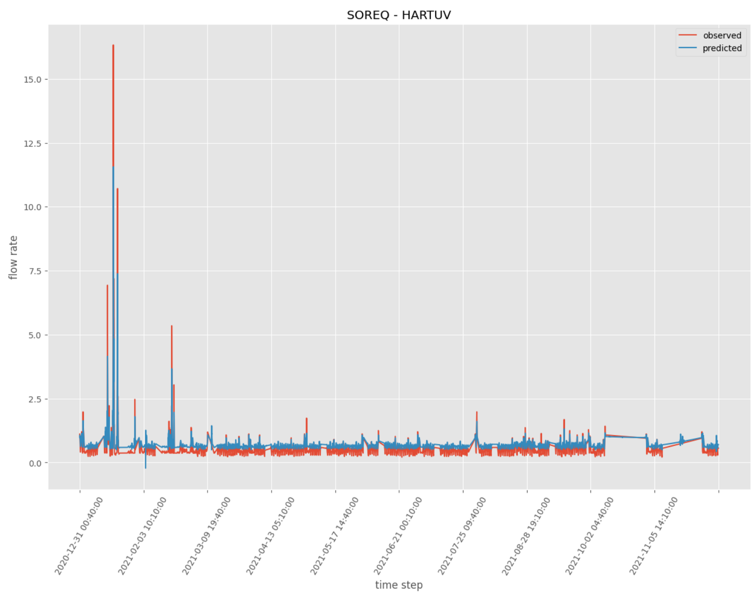
- batch_size = 128
- epochs = 5

- n_neurons = 64

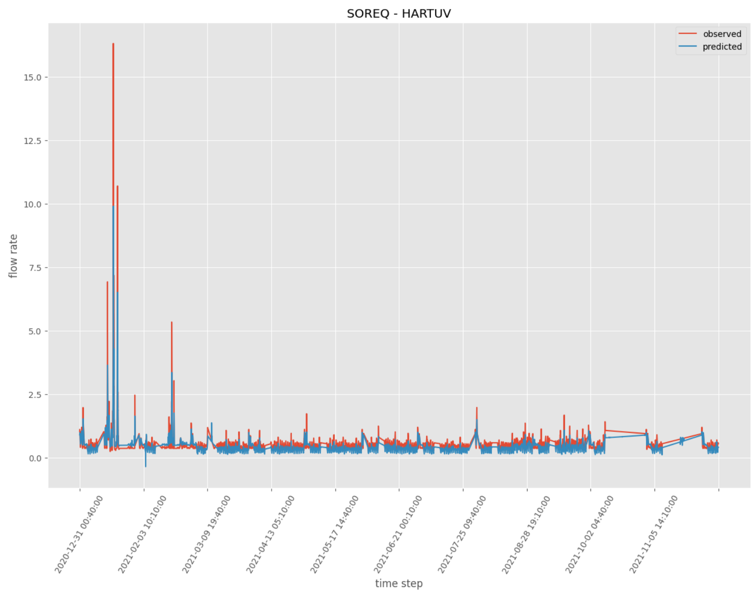
GRU's best hyperparameters:

- batch_size = 128
- epochs = 5
- n_neurons = 64

3.6.4 Observed VS Predicted - Optimized LSTM and GRU



(a) LSTM



(b) GRU

Figure 14: Observed flow rate (red) vs predicted flow rate (blue) after hyperparameter optimization

The test scores for the optimized LSTM model:

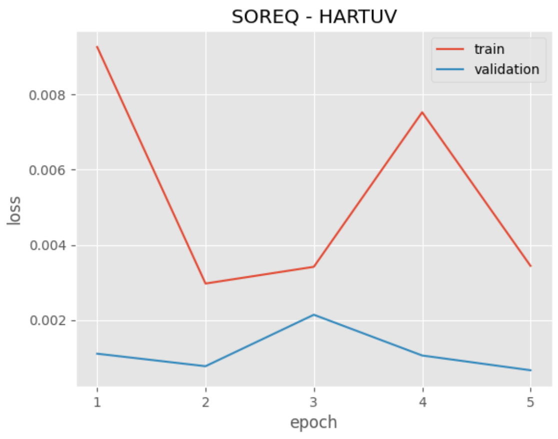
Test NSE: 0.957
Test PersistentNSE: 0.77

The test scores for the optimized GRU model:

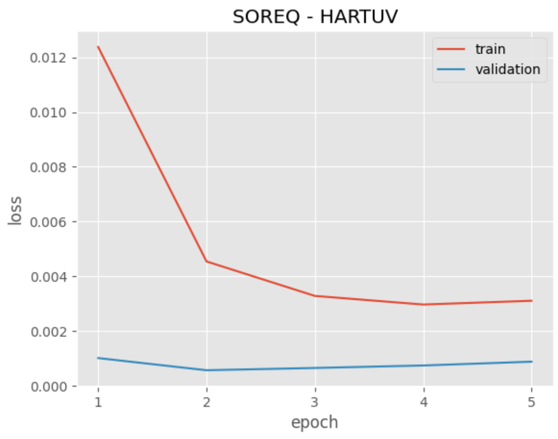
Test NSE: 0.955
Test PersistentNSE: 0.761

We can see that we got a lower score for both LSTM and GRU models after hyperparameter optimization, but not significantly. As stated in the Ayalon Station section, this might be due to the architectures of both models, or overfitting.

3.6.5 Comparison - LSTM and GRU



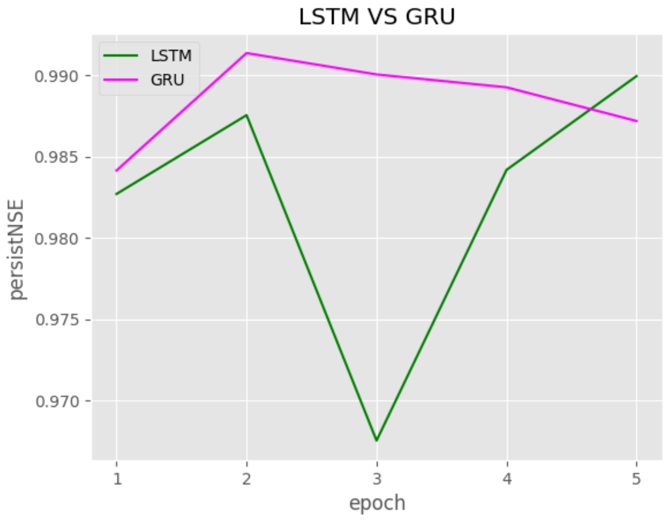
(a) LSTM



(b) GRU

Figure 15: NSE* loss: train (red) vs validation (blue) after hyperparameter optimization

The validation score of both models is as follows:



We can see that the GRU model is more consistent when it comes to the decrease in loss versus the LSTM model. Also, the GRU's validation score is greater than the LSTM's at each epoch except the last one. We conclude that both models can predict the flow rates quite accurately.

4 Discussion

4.1 A Brief Summary and Conclusions Based on the Results

In our study, both the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) models demonstrated impressive performance across a diverse range of monitoring stations, strategically chosen from different parts of Israel. This selection was intended to test our models in varied environmental conditions, ensuring a comprehensive evaluation of their capabilities.

Specifically, at the 'Bet Lehem' station, both models performed well in terms of prediction accuracy, with the LSTM model showing a slight edge. This station, interestingly, utilized only flow data, highlighting the robustness of both models even with limited data inputs.

For the 'Ayalon' station, the addition of rainfall data proved beneficial for both the LSTM and GRU models, enhancing their predictive capabilities. This trend was also observed in the 'Ga'aton', 'Qishon', and 'Soreq' stations, where the inclusion of rainfall data alongside flow rates was a common factor. In these cases, the LSTM model consistently outperformed the GRU, particularly at the 'Ga'aton' and 'Qishon' stations. However, both models yielded satisfactory results at the 'Soreq' station.

In summary, our findings indicate that while both LSTM and GRU models are effective for hydrological prediction, the LSTM model generally exhibits a slight advantage, especially when augmented with additional data inputs like rainfall. This advantage is most pronounced in stations where complex data patterns may exist, as evidenced by our results from the 'Ga'aton' and 'Qishon' stations. The diverse geographical spread of the stations across Israel, encompassing different environmental conditions, further underscores the adaptability and robustness of these models in varied hydrological contexts.

4.2 Limitations

4.2.1 Challenges with the IMS API

We faced considerable difficulty when querying the Israeli Meteorological Service(IMS) API. The data returned were nested within JSON structures with embedded dictionaries, making parsing a challenge. The API presented limited querying capabilities, restricting the flexibility and convenience of data retrieval. Data integrity issues were evident, with numerous instances of missing values, indicating incomplete rainfall records over several years.

4.2.2 Data Gaps in Hydrological Stations

Several hydrological stations exhibited missing data. For instance, the "Dalia - Bat Shlomo" station (located near Haifa, Israel) only provided records up to 2018. This is in contrast with our meteorological data which was consistent up to 2021. Due to this inconsistency, we were unable to employ our optimal model, necessitating the use of alternative stations to compensate for the data gaps.

4.2.3 Resource Constraints in Model Execution

Implementing our model for every station across diverse regions in Israel required substantial computational resources. Despite securing the advanced subscription of Google Colab, we quickly exhausted the available resources. Consequently, we strategically selected representative hydrological stations from varied climatic regions in Israel to evaluate our model's efficacy.

4.2.4 Exclusion of Upstream Flow Data

Initially, our model considered the inclusion of upstream flow data from proximate stations—particularly those at higher altitudes, as water flowing through these stations could potentially continue towards our selected station. However, the absence of a relational database detailing the interconnections between stations posed challenges. In consultation with Professor Efrat Morin, we decided to set aside this component for potential future research.

4.2.5 Limited Expertise

While Professor Morin provided invaluable guidance throughout our research, our own expertise in this intricate domain is nascent. Our relative inexperience could influence data interpretation, possibly overlooking nuances critical to expert practitioners in the field. A deeper understanding might reveal insights related to aspects such as the computation of polygon areas for stations, extreme data values, interconnected parameters, outliers, among others.

It is paramount to consider these limitations when interpreting the study's results, as they could influence the generalizability and applicability of our findings.

4.3 Future Work

4.3.1 Enhanced Integration with IMS API

Given the challenges experienced with the current structure of the Israeli Meteorological Service (IMS) API, future endeavors could involve working closely with the service provider to enhance the API's querying capabilities. Developing customized scripts or tools that adeptly parse nested JSON structures could streamline the data retrieval process, making it more efficient.

4.3.2 Addressing Data Inconsistencies in Hydrological Stations

To rectify the data gaps identified in specific stations, collaboration with local authorities or stakeholders might yield additional datasets or alternative data sources. Further, advanced imputation methods, possibly using machine learning, could be explored to fill in missing data and thereby enable the use of the optimal model.

4.3.3 Optimizing Model Structure and Computational Resources

In our current approach, we employed individual models for each station, driven by the clarity, readability, and superior results this method provided. However, future research might consider developing a unified model that encompasses all stations collectively. Such an approach could potentially streamline computations, reduce processing time, and even yield improved outcomes by leveraging inter-station correlations.

To support this potential shift in modeling strategy, it would be crucial to secure more robust computational infrastructure. Collaborations with cloud providers or investments in dedicated computational setups could provide the necessary scalability to undertake such comprehensive analyses.

4.3.4 Incorporating Upstream Flow Data

Recognizing the potential value of upstream flow data, future work should prioritize the development of a relational database that maps station interconnections. Collaborative initiatives with experts in hydrology could aid in identifying these interrelations and refining the model accordingly.

Addressing these points in subsequent research will not only mitigate the current limitations but will also significantly enhance the validity and applicability of the results, paving the way for a more comprehensive understanding of hydrological patterns in Israel.

5 Resources and Methods

5.1 Preprocessing

We obtained the following datasets from our supervisor, Professor Efrat Morin:

- flow_in_hydro_stations1
- flow_in_hydro_stations2
- flow_in_hydro_stations3
- hydro_stations_catalog

Subsequently, we used Geographic Information System (GIS) with the help of a student of Professor Efrat who did guided work for her in the previous semester, to spatially associate each hydro station with its five nearest meteorological (rain) stations, ensuring that they were situated within a 10-kilometer radius (distances file). In addition, we combined the three flow files into one file that contains all the stations, dates and flow rates and performed imputation on null values (preprocess notebook).

Following this spatial analysis, we initiated API requests to the Israeli Meteorological Service(IMS). These requests necessitated the provision of station numbers, the specific measurement parameter (in our case, rainfall), and the desired date range. The data retrieved from these requests was subsequently processed and tabulated in Excel spreadsheets. Each spreadsheet encompasses columns indicating the measurement time, hydrological station number, and for every rain station, its respective number and corresponding precipitation value for the designated time.

During our data validation phase, we identified gaps in the records, indicative of missing values from some stations. Fortunately, Professor Efrat Morin graciously provided supplemental rainfall data which had to be extracted and merged to the flow data (IMS file), enabling us to achieve a comprehensive dataset crucial for the subsequent model development.

With the consolidated data in hand, it became imperative to synchronize the timestamps from the meteorological stations with those of the hydrological stations. This required intricate time resampling and adjustments to ensure temporal alignment, a crucial step before proceeding with the model execution.

5.2 Loss Function

The loss function for training the models will be the one used in [Kratzert et al., 2019] and in other studies. This function is defined as such:

$$NSE^* = \sum_{n=1}^{N} \frac{\left(f_b(n * \Delta t) - pred\_f_b(n * \Delta t)\right)^2}{(std(b) - \epsilon)^2}$$

Where:
b - Basin number b.
Δt - Time resolution.
N - Number of sequences.
$f_b(n * \Delta t)$ - The true flow rate at time $n * \Delta t$
$pred\_f_b(n * \Delta t)$ - The predicted flow rate at time $n * \Delta t$
$std(b)$ - The standart deviation of the true flows in basin b.

5.3 Metrics

Two metrics were used during the evaluation of the model. The first metric is the Nash-Sutcliffe Efficiency (NSE) function:

Where $\bar{f}_b$ is the mean of true flows for basin b, and the rest of the parameters are as explained above.
NSE is a widely accepted metric in hydrology, where 1 represents a perfect fit and as the value gets smaller the fit is worse. Negative values indicate an ineffective model (A prediction of the average will give a higher metric).

$$NSE = 1 - \frac{\sum_{n=1}^{N}\left(f_b(n * \Delta t) - pred\_f_b(n * \Delta t)\right)^2}{\sum_{n=1}^{N}\left(f_b(n * \Delta t) - \overline{f_b}\right)^2}$$

The second metric is the Persistent NSE (persistNSE) function:

$$PersistNSE = 1 - \frac{\sum_{n=1}^{N}\left(f_b(n*\triangle t) - pred_{f_b}(n*\triangle t)\right)^2}{\sum_{n=1}^{N}(f_b(n*\triangle t) - f_{last\,b})^2}$$

Where f_{lastb} is the last observation, and the rest of the parameters are as explained above. PersistNSE is a metric that is particularly suitable when forecasting based on the station's past data is used. In this situation, NSE will usually be higher and less informative (which is why during training we get NSE values closer to 1 while the persistNSE is smaller). PersistNSE is a metric that compares the model's prediction to the prediction using the last measured value (i.e L time steps back).

5.4 Models and ML Methods Used

5.4.1 Time Series to Supervised Learning

Converting time series data into a supervised learning problem is very important as it enables the application of powerful machine learning algorithms to extract valuable insights and make predictions.

By structuring time series data into input-output pairs, with historical observations serving as inputs and future values or target variables as outputs, we equip ourselves to forecast future trends, anomalies, or events.

Our function - series_to_supervised does just that - it reframes the problem as a supervised learning problem by defining input features (water level, rainfall) and a target variable (flow rate):

- Input Features: The features or input variables for each time step are typically derived from past observations. For example, if we are predicting the value of a time series at time 't', and the lead time is n hours while the time resolution is 1 hour, the input features include data from time 't-1', 't-2', 't-3', and so on, up to 't-n'.
- Target Variable: The target variable is the value at time 't' that we want to predict. This becomes the output for the input features in the same time window.

5.4.2 LSTM and GRU Architectures

The architecture for both models are similar: 1 layer with an initial value of 128 of neurons, with dropout=0.2, and a dense layer.

For both models, the activation function is tanh, and the recurrent activation function is sigmoid.

5.4.3 Hyperparameter Optimization

The method used for the hyperparameter optimization is Grid Search. The parameters (and the different values in brackets) that were chosen to optimize are:

- Batch size [32, 64, 128]
- Epochs [5, 10, 15]
- Number of neurons in the layer [64, 128]

5.4.4 Time Resolution and Interpolation

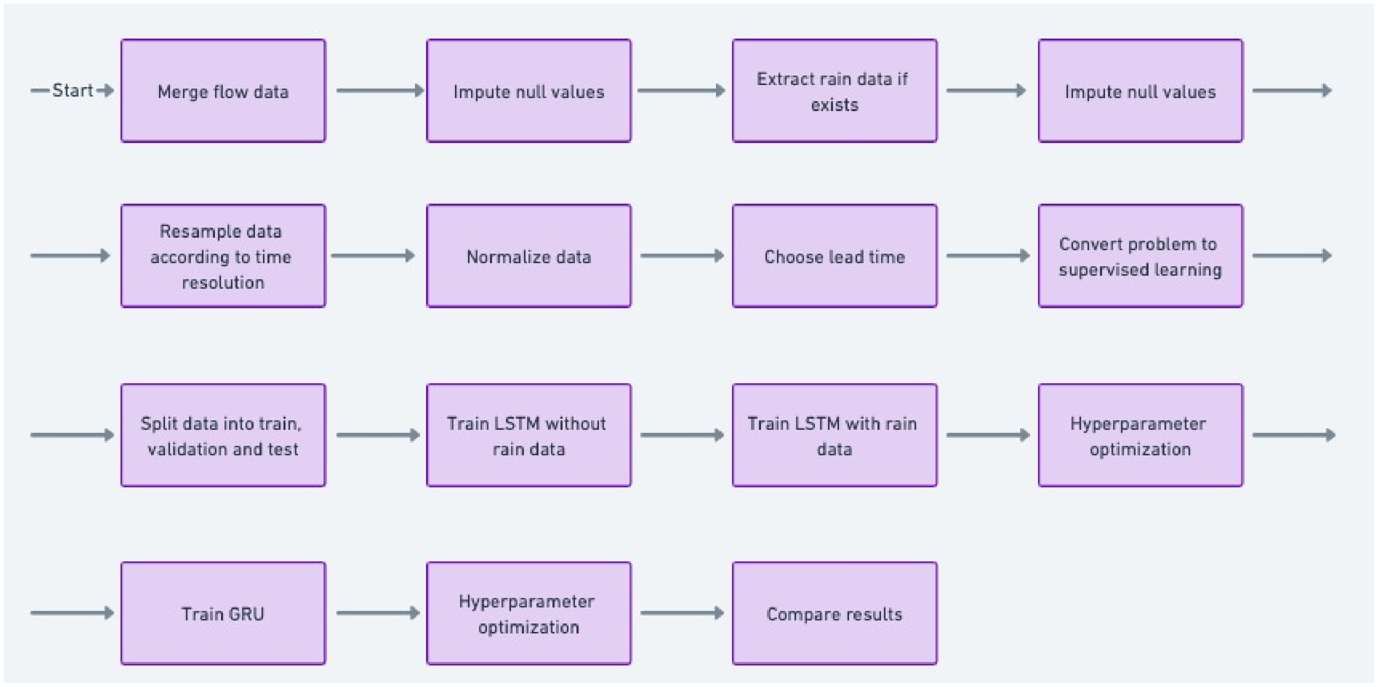
Time resolution in time series data refers to how finely or coarsely time intervals are recorded or measured. The choice of time resolution for each station was done by experimenting different time resolutions and their impact on the results.
Moreover, since the time intervals in the given data were not equal, an interpolation of the data had to be done accordingly. We chose to interpolate the data linearly.

5.5 Resources

- Kaggle notebook that explains time-series data analysis using LSTM
- An explanation on how to convert a time series to a supervised learning problem in Python

5.6 Work Pipeline

A description and a flowchart of the work pipeline


Start
Merge flow data
Impute null values
Extract rain data if exists
Impute null values
Resample data according to time resolution
Normalize data
Choose lead time
Convert problem to supervised learning
Split data into train, validation and test
Train LSTM without rain data
Train LSTM with rain data
Hyperparameter optimization
Train GRU
Hyperparameter optimization
Compare results

A brief description of the pipeline:

1. Merging all 3 flow data files into 1 file.
2. Impute null values for flow rate and water level by replacing them with zeros.
3. Extracting rain data from the folders professor Efrat gave us and add the data to the correct flow station according to distance from rain station (distances.csv consists of 5 closest rain stations for each flow station).
4. Impute null values of rain data by replacing them with zeros.
5. Resample data according to time resolution using a linear interpolation.
6. Normalize data using MinMaxScaler.
7. Choose lead time - discussed at section 3.1.
8. Convert problem to supervised learning - discussed at section 5.4.
9. Split data into train, validation and test sets - discussed at section 2.3.1.
10. Train an LSTM model without rain data, then train an LSTM model with rain data and compare.
11. Optimize hyperparameters with Grid Search and train the new model.
12. Do the same with the GRU model but only with the rainfall data.
13. Compare optimized LSTM and GRU models' results.

6 Git

Link to git of the project

References

K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In A. Moschitti, B. Pang, and W. Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL `https://aclanthology.org/D14-1179`.

R. E. Emerton, E. M. Stephens, F. Pappenberger, T. C. Pagano, A. H. Weerts, A. W. Wood, P. Salamon, J. D. Brown, N. Hjerdt, C. Donnelly, C. A. Baugh, and H. L. Cloke. Continental and global scale flood forecasting systems. *WIREs Water*, 3(3):391–418, 2016. doi: https://doi.org/10.1002/wat2.1137. URL `https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wat2.1137`.

S. N. Jonkman. Global perspectives on loss of human life caused by floods. *Natural Hazards*, 34(2):151–175, Feb 2005. ISSN 1573-0840. doi: 10.1007/s11069-004-8891-3. URL `https://doi.org/10.1007/s11069-004-8891-3`.

W. F. Krajewski, D. Ceynar, I. Demir, R. Goska, A. Kruger, C. Langel, R. Mantilla, J. Niemeier, F. Quintero, B.-C. Seo, S. J. Small, L. J. Weber, and N. C. Young. Real-time flood forecasting and information system for the state of iowa. *Bulletin of the American Meteorological Society*, 98(3):539 – 554, 2017. doi: https://doi.org/10.1175/BAMS-D-15-00243.1. URL `https://journals.ametsoc.org/view/journals/bams/98/3/bams-d-15-00243.1.xml`.

F. Kratzert, D. Klotz, G. Shalev, G. Klambauer, S. Hochreiter, and G. Nearing. Towards learning universal, regional, and local hydrological behaviors via machine learning applied to large-sample datasets. *Hydrology and Earth System Sciences*, 23(12):5089–5110, 2019. doi: 10.5194/hess-23-5089-2019. URL `https://hess.copernicus.org/articles/23/5089/2019/`.

S. Nevo, E. Morin, A. G. Rosenthal, A. Metzger, C. Barshai, D. Weitzner, D. Voloshin, F. Kratzert, G. Elidan, G. Dror, and G. Begelman. Flood forecasting with machine learning models in an operational framework. *Hydrology and Earth System Sciences*, 2022. URL `https://doi.org/10.5194/hess-2021-554`.