UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELLINFORMAZIONE
MASTER THESIS / CORSO DI LAUREA MAGISTRALE IN
ICT for Internet and Multimedia

Calibration and comparison of regressive neural network models for environmental parameter forecasting for sewage water system

Master Candidate/Laureando

Emre DAVUT

Student ID 2041382

Supervisor/Relatore

Prof. Assoc. Gian Antonio Susto

University of Padova

Co-supervisor/Correlatore

Dott. Sergio Lopez Dubon

University of Edinburgh

Academic Year/Anno Accademico 2022/2023
Date of Graduation/Data di Laurea 14/12/2023

To my parents Sengul Davut & Yilmaz Davut and my lovely brother Umut Davut,
for their endless support, especially to my mother Sengul Davut,
I present this MSc degree to her as a gift for his incomplete education.

To my best friends related to Nisantasi Anadolu High School and ITU,
for their existence and instilling in me a vision.

To Mustafa Kemal Atatürk, founder of the Republic of Türkiye,
for being the hope, saviour and guiding light of a nation.

To Mauro Icardi and Lionel Messi,
for being a motivation related to sportive support.

Abstract

In the face of climate change and evolving environmental regulations, effective management and forecasting of sewage water systems (SWS) under various scenarios have become important. These systems wield significant influence over urban flood control and water quality treatments. In this context, a Data-Driven Digital Twin (DT) is developed specifically for a small SWS basin located in northern Italy. This basin encompasses a sewage network, featuring Doppler sensors that measure water velocity, pressure (depth), and temperature every six minutes. Additionally, rain gauges provide minute-by-minute data. Given the operational conditions of these sensors, occasional low-quality measurements are inevitable.

To address this issue, a Neural Network (NN) is designed, trained, and integrated into the DT capable of identifying anomalous values, attributing potential causes (e.g., sensor contamination), and suggesting accurate replacements. This research explores regressive neural network models approaches: a convolutional layer neural network (CNN), CNN with Long Short-Term Memory (LSTM) approach, and a CNN model with residual connection. Models are all replicate the SWS configuration and share the same training data. These models are rigorously evaluated under scenarios involving missing data, such as sensor removal, and both consistently exhibit a high general accuracy rate exceeding 90%. This project not only showcases the successful development and application of the DT but also underscores the importance of collaboration between industry, government, and academia in addressing critical environmental challenges.

Contents

List of Figures

List of Acronyms

CNN	Convolutional Neural Networks
DL	Deep Learning
DT	Digital Twin
hr	Hydraulic Radius
LSTM	Long Short-Term Memory
MSE	Mean Squared Error
ML	Machine Learning
NaN	Not a Number
NN	Neural Network
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SES	Measurement Point
SWS	Smart Water Systems
WRM	Water Resource Management

1 Introduction

1.1 Resource Management in History

Throughout history, managing resources has always been a challenging task for humanity. People have found and developed many methods to store, preserve, and manage resources for later use according to the conditions of the time. In today's conditions, various methods have been developed for the operation of water management systems. In recent years, the growing need for more affordable and efficient water management solutions has led to significant innovations. Investments in smart systems have surged, with the global smart water systems market set to double in the next five years (Rezaei Kalvani and Celico, 2023). These requirements triggered human beings in resource management and always forced them to find a better method. In this way, it has become important for people to use more effective and smarter systems.

1.2 Smart Water Systems in Use

Smart water systems which will be mentioned as SWS, are comprehensive networks consisting of advanced water meters and integrated data systems. These systems are designed to meet the needs of facilities of different sizes and geographical locations, offering a versatile solution for efficient water management. Their main function is to provide continuous and real-time monitoring of water flow in the network. Beyond simply measuring consumption, they are

equipped with the intelligence to detect and promptly address potential issues such as leaks, pressure irregularities, and unusual consumption patterns. Importantly, these systems are not limited to utility operators; they extend their benefits to the end-users as well. With user-friendly interfaces and mobile applications, customers gain the ability to monitor their own water usage in real-time. This not only empowers them with insights into their consumption habits, but also encourages responsible water use, ultimately contributing to conservation efforts and reducing unnecessary waste (Li et al., 2020).

1.3 Machine Learning in Water Management

As the need for efficient water management solutions has grown in the latest years, individuals often design specific frameworks to interpret their systems using a variety of tools. As tools continue to improve in the last decades, the use of neural networks has also gained momentum (Huang et al., 2021). Predictive neural networks, a part of machine learning, are valuable tools for reducing environmental harm in water management. The application of machine learning in water resources has grown recently (Miao et al., 2021). In this study, machine learning models are developed to clean and understand complex interactions in SWS using these approaches, namely Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) combined with CNN, as well as CNN with residual connections. The review was conducted in three different ways. In this way, these approaches are built on these two primary reasons. Firstly, it was to enhance the quality and accuracy of our results by utilizing various techniques. Secondly, it served as a validation process, confirming the reliability and consistency of our findings across different methodologies. Our goal was to provide robust insights tailored to the field of water management, ensuring that our research is both dependable and practical.

2 Background

2.1 Water Resource Management Challenges

Water resource management is a complex field characterized by inherent uncertainties, especially in predicting inflow patterns, which demands a probabilistic approach. To address this challenge, a probabilistic approach has become essential. In recent years, Machine learning has rapidly evolved into a versatile and powerful tool applicable across various scientific domains (Rozos, 2019).

2.2 Use of Machine Learning on Water Resource Management

Machine learning, often abbreviated as ML, is fundamentally defined as a computer's ability to analyze data. This approach allows computers to examine and learn from large amounts of data and use that learning to predict future events. It relies on mathematical and statistical methods to provide intelligent solutions to complex problems such as optimizing sewer systems. It has proven to be a valuable tool in sewage water systems and it provides smart solutions to complex issues (Zhu et al., 2022). ML employs various mathematical and statistical methods to predict how sewage systems function by analyzing past data patterns and relationships. It's especially handy when dealing with intricate problems that involve complex processes and extensive possibilities, which traditional approaches might struggle to manage efficiently. In recent years, ML

has found various uses in water-related challenges, like predicting water quality in rivers, lakes, and groundwater. Different algorithms, such as artificial neural networks (ANNs), support vector machines (SVM), random forests (RF), and decision trees (DT), have been used to enhance water quality predictions (Chen et al., 2020).

In the field of Water Resource Management (WRM), the application of Machine Learning has become increasingly crucial. ML encompasses various algorithms, broadly categorized as supervised, unsupervised, and reinforcement learning, each offering unique capabilities. These algorithms provide predictive power, enabling us to estimate outcomes based on historical data, and they excel in handling complex and uncertain systems, like those in water resource management. The use of ML in WRM offers the capability to predict future events, which is essential for optimizing control, performance evaluation, and decision-making in this dynamic domain. By leveraging ML, WRM can benefit from data-driven insights and more accurate estimations, ultimately leading to improved resource management and planning (Ghobadi and Kang, 2023).

2.3 Use of Supervised Learning

Supervised learning is one of the building blocks of machine learning. In supervised learning, the algorithm is trained on a labeled dataset. The target of the dataset is labeled. Basically, the algorithm learns from examples that match input data with corresponding output labels. The goal is for the model to generalize from these labeled data and make accurate predictions for new, unseen data. In scenarios where water is used, supervised learning can be used to predict certain parameters, for example, predicting water quality or optimizing sewage systems (Xu and Liang, 2021).

Regression is a subset of supervised learning and is used especially when you want to predict a continuous outcome. In the context of water management, regression can be used to estimate the volume of water flow or the concentration of pollutants. The main importance of regression in the machine and deep learning projects is its ability to model the relationship between input features and a continuous target variable, making it possible to make precise predictions for quantitative outcomes in water-related scenarios (Ghobadi and Kang, 2023).

2.4 Use of Machine Learning on Water Resource Management

Deep learning, often abbreviated as DL, is a sub-branch of machine learning based on artificial neural networks and provides a major enhancement in this field. In contrary to the traditional machine learning methods, deep learning involves neural networks with multiple layers, also known as artificial neural networks (ANNs). This method involves deep neural networks designed to automatically extract meaning from large and complex datasets. DL stands out for its ability to better understand data and resolve complex relationships. In this way, DL can be used effectively, especially in large-scale and complex datasets by allowing us to obtain more accurate predictions and inferences. In addition, DL offers the ability to recognize complex patterns through hierarchical learning (Sit et al., 2020). Deep learning in water resources management, particularly represented by Convolutional Neural Networks (CNNs), specializes in understanding complex patterns and relationships in diverse and dynamic water datasets.

2.5 Use of Convolutional Neural Networks

For these resource management and planning issues, Convolutional Neural Networks (CNNs) mark a significant step forward in computational technology. These networks are adept at spotting complex patterns within data, a crucial ability for numerous applications. At the heart of this lies the convolution operation, which extracts time-related features from the data, allowing us to make important predictions (Van et al., 2020).

To harness the power of CNNs, it is essential to define and train the network architecture using suitable data. This process results in a neural network with a convolutional layer, capable of recognizing both simple and complex relationships between input and output. Additionally, CNNs often include pooling layers, which help reduce data size while preserving essential spatial relationships (Ghobadi and Kang, 2023).

Incorporating CNNs and LSTMs into SWS is akin to an infrastructure transformation, enabling these systems to operate intelligently. In a field as dynamic as water resource management, where optimizing control and performance

evaluation is paramount, these technologies provide an innovative edge. By introducing CNNs and LSTMs, we equip SWS with the capacity to learn from its continuous data streams, enabling quasi-real-time decision-making. This transition isn't just about augmenting efficiency; it's a groundbreaking shift in how we manage water resources. With all these, SWS are not just pipe networks; they are intelligent, dynamic resource management solutions that can adapt to an ever-changing environment (Ishida et al., 2021).

2.6 Smart Water Systems

Switching our focus to smart sewage water systems (SWS), these essential infrastructures excel in the efficient collection, transportation, and treatment of wastewater emanating from residential, industrial, and commercial sources. Their primary mission revolves around streamlining the sewage treatment process, mitigating environmental contamination, and preserving public health (Otaki et al., 2007).

SWS is crucial for public health and cleanliness. They work by collecting and transporting wastewater through a system of pipes, sewers, and treatment facilities. These treatment plants clean the wastewater by removing harmful substances and germs before releasing it safely. Properly managing sewage systems is essential to prevent urban flooding, protect water quality, and support sustainable city growth.

3

Network, Data and Sensors

3.1 Data Collection

The data used in this research were sourced from a comprehensive network of measurement instruments that was strategically implemented by Gruppo CAP. Gruppo CAP is a prominent utility company, which manages the water infrastructures of the Milan province. This project has been held for a duration of three years in order to create and monitor the system. The objective of this deployment was to monitor and gain insights into the entire territory managed by Gruppo CAP. The district within the vibrant city of Milan is seen in 3.1.

The focal point of this extensive monitoring initiative was to construct a digital twin (DT) of the sewage network, an ambitious endeavor that holds great promise for the field of smart water management. This digital twin represents a technologically advanced replica of the physical sewage infrastructure. It ingeniously integrates a real-time fluid dynamic model, hence enabling predictive capabilities that can significantly enhance decision-making and overall system efficiency.

The term digital twin refers to the virtual counterpart of a real-world system or process and in this context serves as a breakthrough in sewer network management. The digital twin for the Sesto San Giovanni sewer network includes a highly advanced, real-time fluid dynamics model that mimics the behavior of wastewater as it flows through the network. The use of deep learning regression techniques further increases the capabilities of this digital twin.

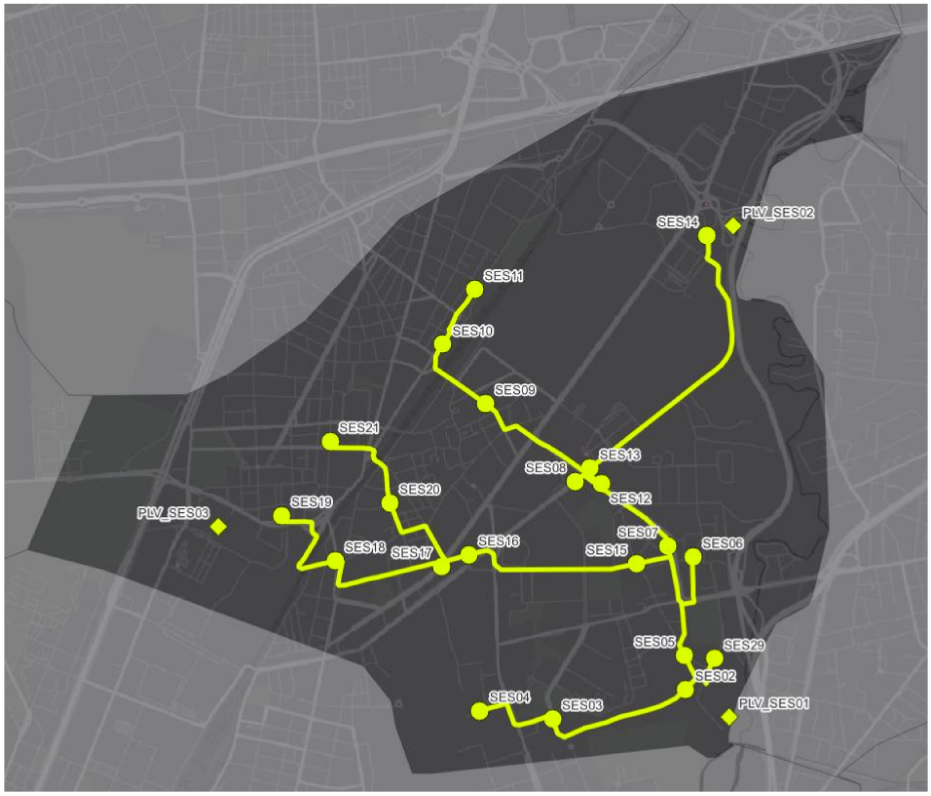

Figure 3.1: Illustration of the layout of the primary pipelines within the sewage network of Sesto San Giovanni.

By constantly processing and analyzing data from various sensors strategically placed throughout the network, the digital twin can predict a range of critical variables. These predictions include flow rates, level changes, and the condition of sewage pipes, among others. Such predictive capabilities offer invaluable advantages to network operators.

The Sesto San Giovanni sewage network under examination extends over an expansive length of 140 kilometers, forming an extended web beneath the city. Within this extensive network, a total of 21 highly advanced flow meters have been installed to collect critical data. The network topology is seen in 3.2. These flow meters employ cutting-edge Doppler ultrasound technology, specifically the Nivus model KDA, and utilize the area-velocity method to determine the flow characteristics of the wastewater.

This comprehensive sensor deployment across the sewage network forms the backbone of the digital twin system. It provides a comprehensive view of the current state of the network by enabling continuous data collection from various critical points within the network.

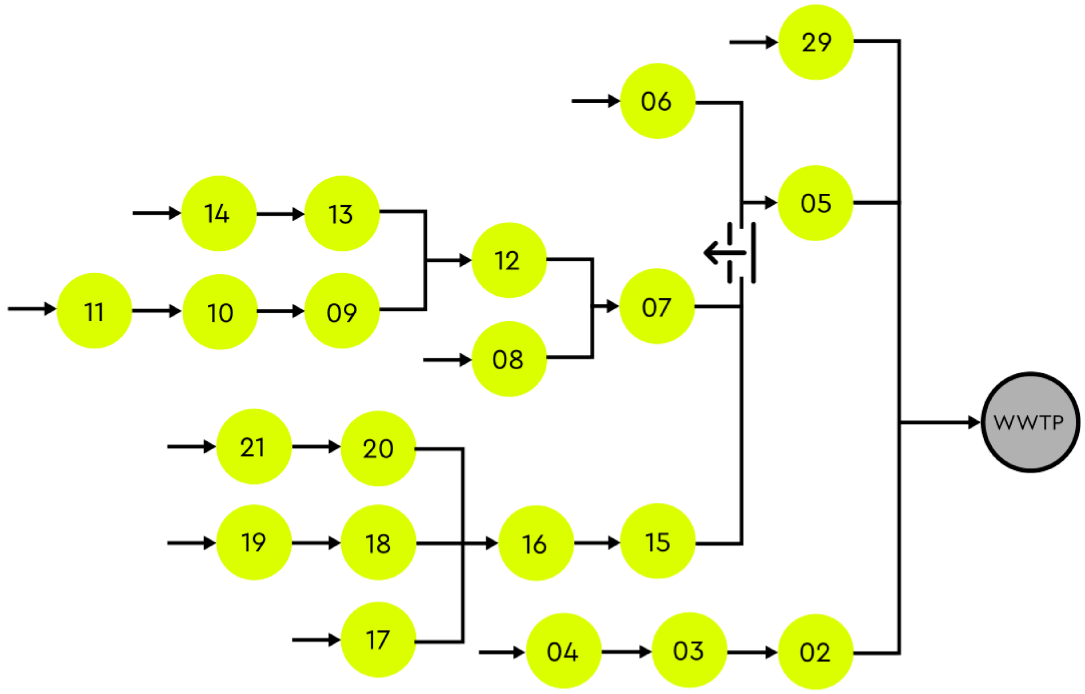

Figure 3.2: Network topology of the sewage water system measurement points.

Essentially, the deployment of these advanced sensors and the development of the digital twin are transformative steps towards the future of sewer network management. The insights gained from this initiative are invaluable in optimizing network performance, increasing efficiency and ensuring the highest standards of environmental responsibility. Gruppo CAP's strategic approach to data collection and analysis sets an example for smarter and more sustainable urban infrastructure management, as demonstrated in this study.

3.2 Sensors

The Nivus KDA sensor provides a cutting-edge technology solution for flow measurement in sewage water systems that are suitable for both lightly and heavily polluted environments. This sensor type ensures precision and reliability in flow rate measurement using fourth-generation smart Doppler technology. What sets it apart is its cost-effectiveness, making it an attractive choice for a variety of applications. The wedge-shaped design of the KDA sensor allows for versatile installation options, whether positioned on the duct wall or the duct bottom, making it adaptable to different duct cross-sections.

This sensor is known for its ease of installation, which reduces the need for additional construction. Moreover, it offers the flexibility to extend cables without any problems. The sensor performs superiorly even in harsh conditions, including very dirty and corrosive environments. Additionally, its compatibility with transmitter types such as PCM F and OCM F creates a comprehensive flow measurement system. This advanced technology plays a fundamental role in this study's quest to improve sewage system management.

However, like many precision devices, these sensors face various challenges. These include problems such as contamination of the sensor, which can lead to loss of velocity readings, and failures in the pressure compensator, which lead to drifts in water levels. These meters may also encounter challenges with varying levels of suspended sediment density. Their performance tends to decrease as they approach their operating limits, especially in applications involving level and velocity measurements. The sensor is illustrated in 3.3.

Figure 3.3: Nivus KDA Doppler Sensor.

Additionally, the network is equipped with three rain gauges. Data collection intervals on the network range from 3 to 6 minutes, with data transfers occurring every 4 hours. The HD2013 rain gauge is a robust and reliable tool for measuring rainfall. Its durability allows it to withstand even the harshest weather conditions, ensuring accurate and reliable performance in harsh environments. The working principle of the tipping bucket rain gauge is simple. It uses the tipping bucket mechanism that fills and unloads according to the amount of precipitation. Each time the tilting mechanism moves, it activates a reed contact that records the amount of rainfall. An important advantage of this design is that it is self-sufficient. It works without needing a constant power supply. Power is only required at low ambient temperatures that may require heating. The rain gauge is illustrated in 3.4.

Figure 3.4: HD2013 Rain Gauge.

3.3 Kriging

Kriging is a robust geostatistical interpolation technique that finds valuable application in environmental data analysis. This method becomes particularly advantageous when dealing with unevenly distributed data points, such as observations obtained in various geographical locations. This situation is often encountered when rain gauges are used to measure rainfall amounts in different regions. In this project, ordinary kriging is used to adjust the rainfall intensity according to the distance to the measurement nodes, as shown by equation 3.1, 3.2, and 3.3. This approach not only improves the neural network models ability to estimate missing data but also increases the precision of the neural network models estimates by minimizing errors. Ordinary Kriging serves as a valuable asset when it comes to handling complex spatial variations in environmental data. It provides an efficient and effective solution to tackle these complex challenges, providing more accurate and reliable results in environmental data analysis.

$$\epsilon(x_0) = \hat{Z}(x_0) - Z(x_0) \tag{3.1}$$

$$= [W^T - 1] \cdot \begin{bmatrix} Z(x_1) & \cdots & Z(x_N) & Z(x_0) \end{bmatrix}^T \tag{3.2}$$

$$= \sum_{i=1}^{N} w_i(x_0) \cdot [Z(x_i) - Z(x_0)] \tag{3.3}$$

For this Kriging process, in the data engineering part, Kriging calculation function is custom defined. OrdinaryKriging method is available for Python in the pykrige.ok library. The function takes the dataset, measurement points,

and rain gauge data as input for the function. Equal precipitation is checked in the data. If the data are equal throughout the rain gauges, then Kriging is not needed and missing values are filled in by averaging the rain gauges that have been measured. Kriging is performed for cases where there is a difference in the data. For each measurement point, the precipitation value at that point is estimated using a model based on observations from other points. Kriging estimates for each measurement point are stored and returned in a dictionary. These forecasts include the amount of precipitation at specific coordinates. In summary, this code uses the Kriging method to estimate rainfall amounts at specific measurement points based on observations from other points.

4 Methods

4.1 Data Preparation

Data preparation is a basic and important stage in data analysis. This is the process where raw data is carefully organized and formatted to ensure it is ready for meaningful analysis. In the context of this research, data preparation plays a very important role in terms of data reliability and accuracy. Through this rigorous process, data is refined, ensuring that it is not only clean and consistent but also reliable as the basis for the subsequent analysis work.

One of the first steps in data preparation is to resample the data at consistent 6-minute intervals. This standardization is very important because some measurement sites initially operated at a 3-minute sampling frequency, then switched to 6 minutes during the measurement campaign.

Additionally, data preparation includes addressing inconsistencies or inaccuracies in measurements. For example, issues such as measurement resets and occasionally empty cumulative values are fixed, especially in the context of precipitation measurements. The importance of handling null values correctly during these operations cannot be denied. It is very important to avoid any confusion between the original null values and the null values that may have arisen temporarily during data processing. This comprehensive data preparation process forms the basis of the research as it ensures that the data is both accurate and in a format suitable for meaningful analysis. This process makes research findings more robust and reliable, allowing us to move forward with confidence in further detailed analysis.

4.2 Data Engineering

Data engineering is a collection of data processing techniques that form the basis of improving the performance of deep learning models. Essentially, this is the stage where the work on the raw data is completed. Raw data is collected for improvement by potentially creating new parameters to improve the overall performance of the models. This preliminary data curation is a critical step in this research and paves the way for more advanced data analysis techniques.

It is crucial to ensure the reliability of the data before it is used to support any conclusions. This highlights the importance of effective data cleansing in this context. Each measurement point periodically records information on various properties. Among the most important are the velocity of water flow (v) and water level (l), as well as the quality diagnostic parameter (q), which gives insight into the accuracy of the velocity measurement. In addition, parameters such as water temperature (t) and rain intensity (i) are also monitored. The rain intensity recorded using rain gauges is calculated for each measurement point via Kriging interpolation. When rain occurs, it affects the level and velocity of water, and temperature can act as an indicator of precipitation, as fluctuations in temperature can be attributed to rainy conditions. There is also a concentration period, which refers to the time it takes for rain to reach the measurement point in the sewage since it has touched the ground. This concentration period inherently depends on local urbanization factors, in particular the impermeability of the soil.

Mainly, the data engineering coding process starts with data upsampling/-downsampling in this study. With the defined get entire range function, it enables the data frame to be expanded to a full-time range at a certain frequency. This function expands the data frame at the specified frequency using a specified time interval. Then, timestamps are edited and converted into the appropriate time period. Upsampling is done with the resampling of the data. The defined function takes data from sensors with a certain sampling rate and expands the time series according to these rates. Besides, the function fills NaN values with the created values by upsampling. Upsampling/downsampling is done in order to process time series data by sampling at a certain frequency and filling in missing data appropriately basically.

Also, another important point of view on data is wet geometry data. Wet geometry data need to be defined and data engineering needs to be done on

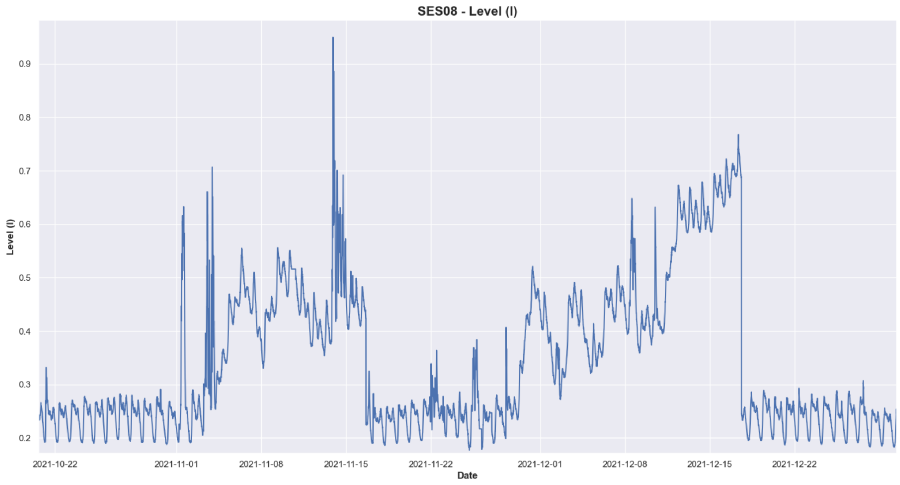

Figure 4.1: Water level data of the SES 08 measurement point.

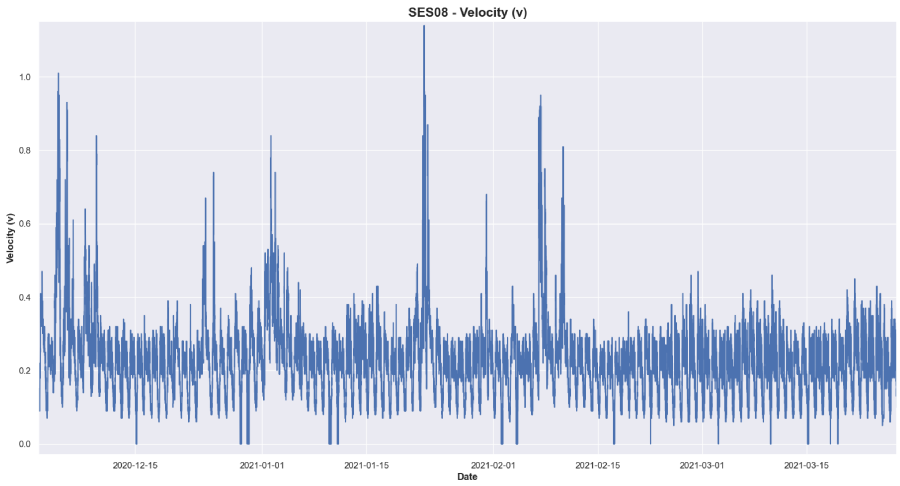

Figure 4.2: The velocity of water flow data of the SES 08 measurement point.

that. The defined function that calculates wet geometries calculates wet areas and related properties based on the geometric properties for each given measure point. The function retrieves geometric information from the database. Depending on the geometry of the measurement point, it calculates wet areas and the properties of these areas with a specific algorithm. Then, it adds these calculated features and fields to the values of the measurement point within certain time periods and returns a dictionary containing wet area calculations and related properties.

Another defined calculate wet time function calculates the wet time (the time the levels the sewage are affected by a rain event) for each sensor. The function determines the starting and ending points of precipitation from the database data. It determines the wetness on the ground due to the effect of precipitation within a certain period, estimated considering the dimension of the tributary basin. A certain level, a threshold of wetness on the ground determines the start and end times. Then, it creates a mask containing this information and adds this mask to the relevant data frame. These two specified functions perform the operations of editing the data of each given measurement point, calculating its geometric properties, and determining the wet time.

Data scaling is the process of editing variable values to present data in a format suitable for machine and deep learning models. Because of this, data scaling improves model performance by standardizing variables in different units and ranges. First, non-dimensionalized scaling brings data into a specific range or dimensionless form. This means expressing data in a non-unit (dimensionless) form over a certain range of scales.

In order to scale the velocity (v) data, firstly, any raw negative values are excluded, because the particular condition of this network leads to consider negative velocity highly unlikely. Also, the values smaller than the measurement error of the sensor are excluded, i.e. values smaller than 0.06 m/s. The velocity is then scaled to the 95th percentile of all measured values in the measurement campaign. Because although extremely fast flows are possible in theory, it is not reasonable to scale these values because they are so rare. These affect learning metrics as they can greatly affect model learning by increasing the skewness of the distribution (Jeni et al., 2013).

$$\varepsilon = \text{95th Percentile of } V \tag{4.1}$$

$$v_{\text{scaled}} = \frac{v}{\varepsilon} \tag{4.2}$$

V contains each value of all velocity values greater than 0.06 m/s here. The symbol ε represents the 95th percentile from measurements of velocity values. That is, 95% of the speed values are less than this threshold value. This marks the limit of speed values that are generally rare, falling outside the top 5% of the overall distribution of the velocity.

In the second formula, the symbol v represents a single velocity value. This formula gives a proportional value of a speed value to that measurement relative to the overall percentile limit of velocity measurements. This indicates the position of a particular velocity within the overall velocity distribution. For example, if this proportional value is greater than 1, this velocity value is an abnormal value above the 95% limit of the overall velocity distribution. Conversely, if this proportional value is less than 1, this speed value is considered a normal value below the 95% limit of the general velocity distribution.

Non-dimensionalize scaling is a scaling technique that ensures the homogeneity of the dataset by transforming variables into a specific range. The custom defined scale linear function performs a normalization operation by linearly scaling a variable to a specified range, usually in the range [0, 1]. Specifically, it performs this scaling using the minimum and maximum values of the given variable. The other custom defined scale non dimension function specifically handles features and inputs such as 'area' and 'hr'. This function makes these features dimensionless by scaling them to specific values over specific time periods. Then they are scaled by the number of observations, and other columns are scaled by values in certain range as [0, 1]. This scaling information is stored for later to check the model's result in terms of physical unit measures.

Seasonality is a concept that generally refers to the change of a time series within a certain periodic pattern or repetition. Seasonality of a time series refers to regular changes or patterns within a specific temporal period. Seasonality usually occurs during specific periods of time, such as months, weeks, or days in the year. Also, seasonality can be considered a separate component from the trend of a time series or random fluctuations. Seasonal patterns often reflect recurring events or impacts over a period of time. Thats why based on this con-

cept, the seasonality interval data are separated within the dataset. In order to do seasonality analysis, observations within a certain time period are examined. Regular patterns and changes over a period of time are identified. These patterns often show periodic repetitions. These differences are so important based on typical season weather behaviors or continuous rainy intervals. The seasonal pattern is distinguished from the general trend of the time series and random fluctuations. Thus, the specific effects of the seasonal pattern can be better understood. In the end, defined calculate seasonality function applies seasonality calculations such as adding virtual values or detrending. Since seasonality is an important component in time series analysis, defining and managing it correctly can contribute to a better understanding of the time series and more accurate forecasting of future values.

In the last part of the data engineering process, tensors are generated for both input features and output targets by taking into account specified look-back and look-ahead times. Input tensors are generated with a custom function for the neural network model. The function contains a specific look-back and look-ahead time and saves them as numpy files. Then, it also creates charts for visualization. Also, another tensor-making function creates y_tensors, regression target tensor based on the specified output parameters, and saves them.

4.3 Input Data Explanation

The neural network model input are given below. These parameters are the basic features used for training the model and making predictions.

Water level is the first feature for the input of the neural network model. The water level is indicated by "l" and is the value that determines the water level in the pipe. The water level is important to understand the degree of filling of the pipe, and it is an important parameter for further calibrate fluid-dynamic models. The water level indicates how full or empty the pipe is.

The water flow's velocity is the value that determines the speed of water in the pipe. Flow rate indicates how fast water moves from one point to another at a given time. This parameter is important to calculate the flow-rate, and it is an important parameter for further calibrating fluid-dynamic models. The water flow's velocity is indicated by "v".

The third feature "area" represents the wetted cross-sectional area inside the pipe and indicates the portion of the inner surface of the pipe in contact with

the fluid. This value is calculated depending on the shape and dimensions of the pipe. When evaluating water flow, the internal area of the pipe is important because it determines the flow rate.

On the other hand, "hr" (hydraulic radius) represents the ratio between the wet cross-sectional area and the wetted perimeter of the pipe. It signifies the proportion of the perimeter for which the wet area is in contact with the wall of the pipe. In this project, it is calculated with this formula 4.3.

$$\text{HR} = \frac{\text{Area}}{\text{Wetted Perimeter}} \tag{4.3}$$

There is this physical quantity is used in the Manning equation. The Manning Equation is a fluid mechanics equation that includes factors such as the area, the hydraulic radius, a coefficient that takes into account the wall roughness, and the slope of the pipe, if the uniform motion approximation is applicable, hence if both spatial and temporal acceleration is negligible. This equation is used to evaluate the movement and flow rate of liquid inside the pipe. "Area" and "hr" are frequently used parameters in the Manning Equation 4.4 and are important indicators for understanding the behavior of the liquid in the pipe.

$$Q = \frac{1}{n} A R^{2/3} S^{1/2} \tag{4.4}$$

In this equation, Q is the flow rate, n is the Manning friction coefficient, A is the wet cross-sectional area (area), R, hydraulic radius, and S represents the energy gradient. With this equation, the relation of these two parameters can be understood and also determined.

The Fourier transform in equation 4.5 is basically used to separate a time-varying signal into frequency components. In this case, the current level data already exhibits a time-varying nature. However, thanks to the Fourier transform, periodic or recurring patterns can be detected underlying this change. The virtual level is obtained by Fourier transform in order to better represent the recurring patterns within this time series, that is, daily, weekly, or seasonal changes. This indicates what time of day or period the time falls on, which determines periods of time when human activity is busy or not. This added virtual level better represents changes over different periods of time, allowing the model to learn time-dependent patterns more precisely. This allows changes over time to be detected more accurately and to react more sensitively to these

changes. The virtual level is labeled as "vir_l".

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-i\omega t}\, dt \tag{4.5}$$

Rain intensity is shown as "i" and is a parameter showing the rainfall intensity obtained by the Kriging method. The precipitation data obtained by the sensors are passed through the Kriging algorithm to make better sense and are added to the input to become a model feature.

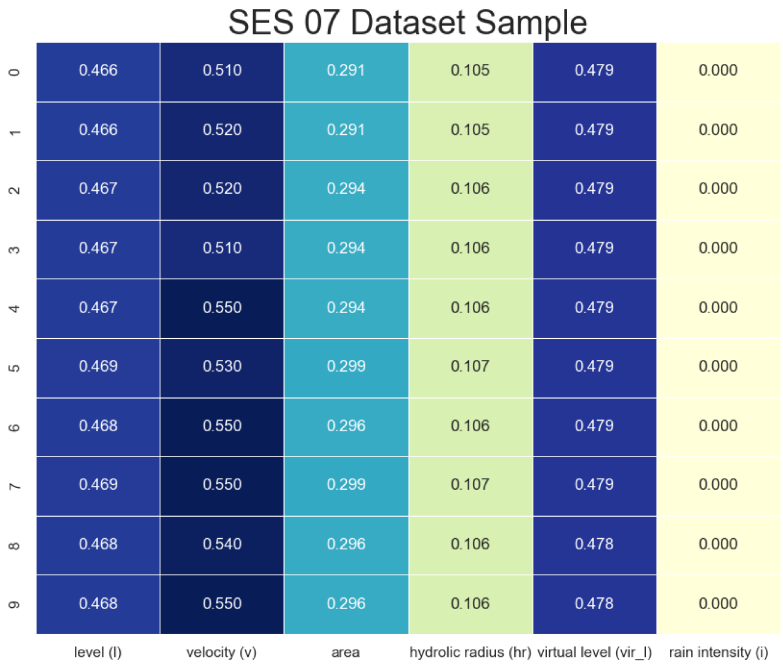

Figure 4.3: Dataset sample for SES 07 measurement point.

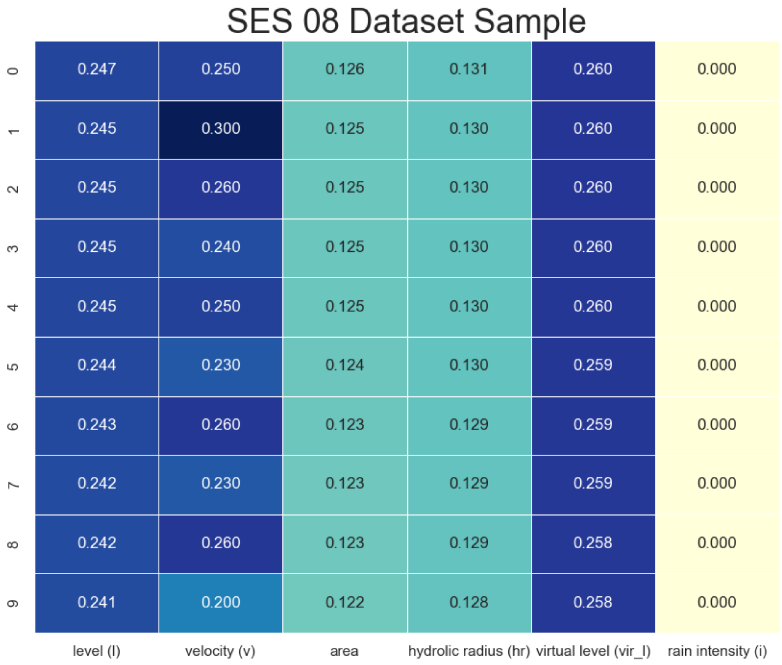

Figure 4.4: Dataset sample for SES 08 measurement point.

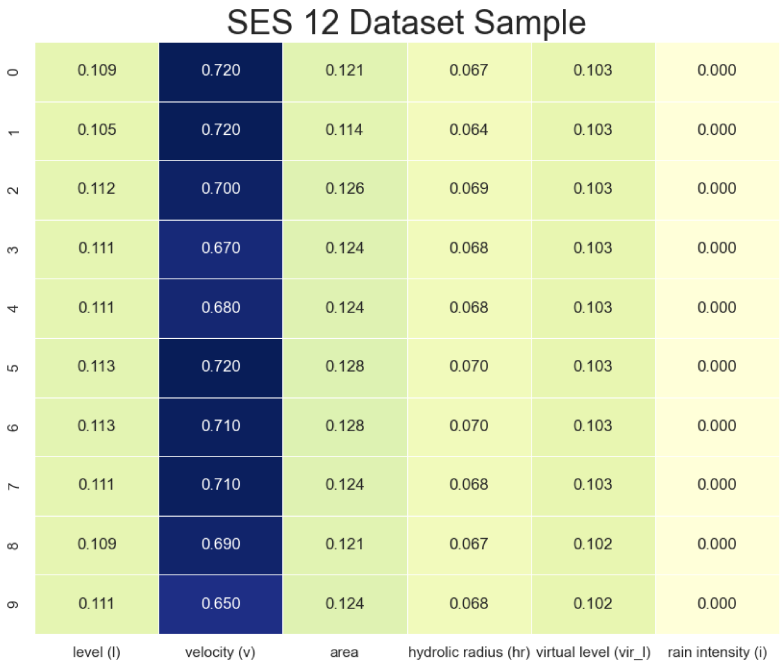

Figure 4.5: Dataset sample for SES 12 measurement point.

5 Regression Models

5.1 Data Loading

The data loading process is a fundamental step in data science and machine learning projects and serves as the first stage for data preparation and analysis. This process involves obtaining and preparing data for subsequent modeling and analysis.

The data, stored in numpy format, is accessed from specific data files corresponding to each measurement point and data type. Numpy is a library in Python for performing high-performance computations on multi-dimensional arrays and matrices. Numpy format, on the other hand, refers to a specific file format used for storing data in a way that's compatible with the numpy library. It allows efficient storage and sharing of data, making it particularly useful for handling large datasets. These data are loaded into a Python data structure, typically a dictionary. This data structure contains components for each measurement point, including "X" (input data), "Y" (target data), and "dates" (date information).

5.2 Data Cleaning

Data cleaning is applied primarily to "X" (input data). If the data contains missing or nonsensical values, such as "NaN" or "-9999," these values are replaced with "NaN" or filled forwards and backwards to ensure that the dataset

is free from incomplete or corrupted entries.

NaN, refers to Not-a-Number, is a term used in mathematics to represent the undefined or unrepresentable value of a number or data. It is commonly used in numerical data analysis to indicate situations where a value is invalid, missing, or uncomputable. NaN serves as a symbol or notation to indicate that a particular value does not produce a valid result in mathematical operations such as dividing by zero.

-9999, on the other hand, is a synthetic or placeholder data value frequently used in the literature. It is often used to indicate missing or unidentified data in a dataset. This value relates to situations where actual data is missing or cannot be measured for any reason. It is often used to account for measurement errors, missing data points, or technical problems during data collection.

These placeholder values, such as -9999, should be taken into account during data cleansing and processing. Developing strategies to handle missing or erroneous data is crucial in many data science applications where such placeholder values are frequently encountered during data analysis or modeling.

5.3 Data Partition

The dataset is usually divided into training, validation, and testing subsets. %70 of the data is used for training and %15 of the data is used for validation and %15 of the data is used for test. This part allows the machine learning model to be trained using one subset, while the performance of the model is evaluated on another subset. The data loading process is responsible for creating and preparing these partitioned datasets.

This process marks a critical initial step in any data analysis or machine learning project. These steps are necessary to ensure proper loading, cleaning, and splitting of data, with consequences for the overall success and reliability of the project.

5.4 Hyperparameters

Hyperparameters can be considered as the settings of a machine/deep learning model. Hyperparameters are the essential values that user can control how a model or algorithm behaves, it they also can be considered as setting a person-

ality for machine/deep learning models. For instance, these settings basically manage the number of neurons in each layer (nodes), the size of the data/feature windows (kernel), how the model approaches the data (strides), the learning rate (how fast the model learns), and many other aspects.

Choosing the correct hyperparameters is crucial because these hyperparameters significantly impact how well the models perform. The right settings cause the machine learning models to work effectively and produce good results. However, incorrect settings can make your model act weirdly. That's why tuning these hyperparameters correctly is an important decision. It often involves some trial and error.

Essentially, experimenting is done with different settings in order to see what set works the best. These settings can also help with speeding up learning, preventing overfitting (when the model fits the training data too closely), and other important aspects of model training.

In summary, hyperparameters are the internal settings of machine/deep learning models. Setting them correctly helps the machine learning model understand data better and generate better results.

5.5 Overfitting

In the machine learning world, one of the most important phenomena called overfitting, happens when a machine learning model learns the training data too well. As an example, it is studying for a test by memorizing the exact answers, without understanding the underlying concepts. This is an important issue because when the model encounters new, unseen data, it struggles to give the correct answers.

In other words, overfitting occurs when a model becomes too specialized in the training data. It starts capturing the noise or random fluctuations in the data, thinking these are important patterns. As a result, the predictions will not perform well on data it has not seen before because it is too focused on the specific quirks of the training data.

To avoid overfitting, it is crucial to strike a balance. The model is preferred to learn the general patterns in the data without getting lost in the details. This can be done by adjusting hyperparameters, increasing the amount of training data, or using techniques like dropout or regularization. The goal is to ensure that the model can apply what it has learned to new, unseen data effectively, just like

understanding the concepts behind the test questions rather than memorizing answers.

Based on these aspects, this study underscores the significance of hyperparameter calibration and configuration as fundamental components of the research methodology. These issues serve as a compass that guides the next stages of the study in the model setting part. Adjusting the network layers and arranging the hyperparameters reflects our determination to ensure the robustness and effectiveness of the model.

5.6 Importing Hyperparameters

Before the model setting, hyperparameters are important respectively how they will be used in the model. This is a fundamental step in configuring a neural network, and it introduces and accesses the key hyperparameters critical to the model's architecture and performance. These hyperparameters are respectively, including the number of nodes, kernel size, strides, learning rate, factor, patience, minimum ratio, epochs, and batch size, play a pivotal role in shaping the neural network's capabilities. Selecting these hyperparameters for our model, a task efficiently executed through the utilization of TensorFlow is pivotal in crafting a neural network that can robustly meet the specific demands of our application. The strategic choice of hyperparameter values profoundly impacts the network's ability to learn, generalize, and perform with the utmost accuracy. By delving into the specifics of each hyperparameter and enlightening their significance and influence on the model's behavior and outcomes, respectively they are underscored.

5.6.1 Nodes

The hyperparameter of nodes determines the number of neurons used in a neural network for one layer. A higher number of nodes allows the neural network model to capture more complex patterns and relationships within the data, but it can also lead to increased computational complexity and potential overfitting. The practice of selecting the number of nodes as powers of 2 in a neural network is rooted in computational efficiency and structural simplicity. By sticking to this practice, calculations align with the binary system, enhancing computational efficiency and memory usage. Moreover, it results in a straight-

forward, symmetric structure, aiding in model management and understanding. Even though this guideline is common for lots of studies and projects, it is not a strict rule and can be adjusted based on the specific requirements of the problem in use. In this study it is examined to determine to be 16 or 32 after testing both configurations. Using 16 nodes provided similar results while being computationally less demanding, making it a more practical choice instead of using as 32.

5.6.2 Kernel Size

The kernel hyperparameter refers to the size of the filter used in convolutional layers. It plays a crucial role in feature extraction. Larger kernel sizes capture more extensive features, while smaller sizes focus on finer details in the data. In this work, the kernel is set to 3. In this case, a kernel size of 3 means a 3x3 filter is applied in the convolution operation. Larger kernel sizes capture broader features but come at a higher computational cost. A 3x3 kernel is effective in capturing local patterns in the data. For instance, it is adept at recognizing small, intricate details in the input data. Additionally, it's computationally efficient and requires fewer parameters compared to larger kernel sizes, which can be advantageous for faster model training and less risk of overfitting.

5.6.3 Strides

Strides hyperparameter determine how the filter moves across the input data during convolution. A smaller stride captures more information but increases computational requirements, while a larger stride reduces the output size and might result in information loss. The strides value is set to 2. A value of 2 implies that the filter moves 2 pixels at a time. Larger strides reduce the output size but might lead to information loss. Choosing 2 is also a common behavior to help in simplifying the model and speeding up the computation.

5.6.4 Learning Rate

The learning rate is a crucial hyperparameter in the training of neural networks because the learning rate determines the step size at which the model's weights are updated during the optimization process. It plays a pivotal role in

controlling overfitting, one of the key challenges in machine learning as mentioned before. In this project, overfitting is observed and controlled, which is a significant practice. The learning rate is a critical tool in this context. To ascertain the optimal learning rate, a tool like TensorBoard is employed to observe overfitting and its effects closely. Through a visual examination of metrics such as loss and accuracy curves in TensorBoard, a learning rate that effectively balances model convergence speed and stability can be selected, thereby enhancing the overall performance of the model.

Typically, learning rates such as 1e-3, 5e-4, 1e-4, 5e-5, or 1e-5 are employed. These values correspond to 0.001, 0.0005, 0.0001, 0.00005, and 0.00001, respectively. A smaller learning rate, such as 1e-5, implies smaller steps in weight updates. While it leads to more accurate results by carefully fine-tuning the model, it slows down the training process considerably, especially dataset is extensive, or the neural network model is deep as in this work.

On the other hand, a learning rate of 1e-3 or 1e-4 allows for faster convergence and is less likely to get stuck in local minima. However, it might also pose a risk of overshooting the global minimum, making the training process less stable. However, in this work, a learning rate of 1e-3 (0.001) is chosen for a practical reason. This choice allows the model to be tuned more quickly due to the larger steps taken during training. As a result, it reduces the number of epochs required and, subsequently, the time needed for training. This approach strikes a balance between efficiency and the risk of instability, providing a suitable compromise for this work.

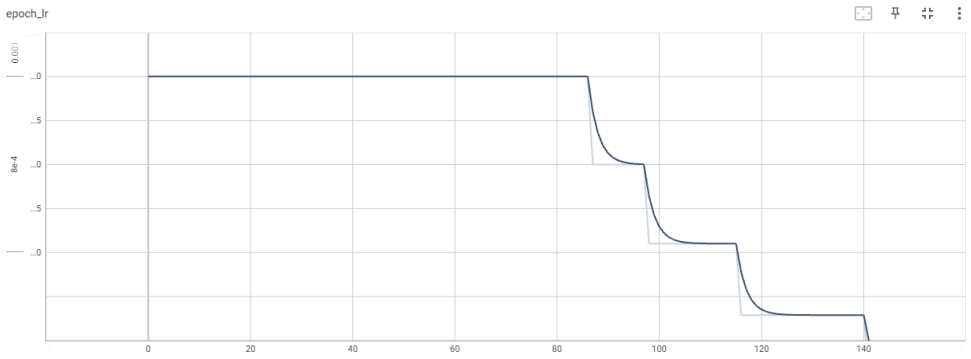

Figure 5.1: Epoch - learning rate curve when learning rate is 1e-3 on TensorBoard.

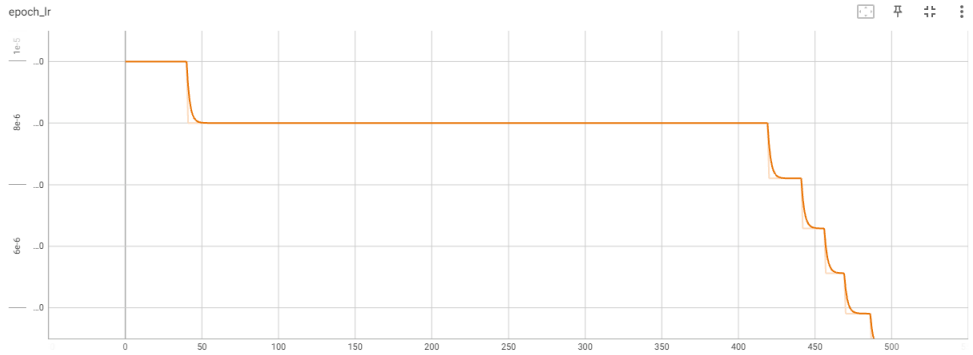

Figure 5.2: Epoch - learning rate curve when learning rate is 1e-5 on TensorBoard.

5.6.5 Factor

The factor hyperparameter is employed to modulate the learning rate during training. Typically, it assumes a value within the range of 0 to 1 and governs the extent to which the learning rate diminishes as the model's performance levels off during training. In this context, the factor is configured to 0.9, influencing the adaptive learning rate reduction based on the patience criteria. This adaptive learning rate strategy and its impact are closely monitored in TensorBoard.

5.6.6 Patience

Patience denotes the number of epochs in which the neural network model's performance does not exhibit any enhancement before a learning rate adjustment takes place, guided by the factor. It is a vital parameter in early stopping strategies, serving as a key indicator of how soon the training process should adapt. In this project, the patience value is set to 10, indicating that should the model's accuracy remain stagnant or saturated over 10 epochs, a learning rate reduction is initiated based on the factor.

5.6.7 Minimum Ratio

The minimum ratio is a threshold for loss reduction; therefore, it is a criterion for minimizing loss. It specifies the minimum improvement required to continue training. When the loss does not decrease beyond this threshold, training stops. In this study, a minimum ratio of 10 is judiciously selected as a reference point.

5.6.8 Epochs

Epochs are a pivotal element of the training process for neural network models. Epochs represent the number of complete passes the neural network model makes through the entire training dataset. Training for an increased number of epochs allows the model to ingest and adapt from the data more comprehensively, potentially resulting in improved performance. However, it is vital to note that this improvement is not boundless. Excessive epochs can have negative consequences, leading mainly to overfitting.

In this study, the neural network model is intentionally trained for 2000 epochs. The selection of this substantial epoch count is founded on careful consideration of various hyperparameters like learning rate, factor, and patience. The model continues to train until it reaches a saturation point determined by these hyperparameters. It is crucial to strike a balance; the number of epochs is judiciously set to optimize performance while mitigating the risk of overfitting. An epoch signifies one complete cycle through the entire training dataset. The key principle here is that extending training to more epochs can bolster performance, yet it simultaneously heightens the potential of overfitting, a challenge meticulously addressed in this work.

5.6.9 Batch Size

Batch size is also an important hyperparameter of the training process of a neural network, determining the number of samples utilized in each iteration. Its selection plays a crucial role in model training. In this work, the batch size is configured to be 1024, indicating that during each iteration of training, the model processes 1024 samples.

Choosing this particular batch size is the result of an intentional decision. A larger batch size can potentially speed the training process up by reducing the frequency of weight updates. This leads to a more efficient utilization of computational resources, as fewer updates consume less memory. However, a larger batch size also requires a higher memory capacity.

Backward, smaller batches provide more frequent weight updates, which can enhance the model's adaptability. Nevertheless, they require more memory and can lead to more extended training durations due to the increased update frequency.

In this study, a batch size of 1024 is adopted for several reasons. Firstly, the server's Random Access Memory (RAM) is robust, easily accommodating this batch size. Secondly, it strikes a balance between training efficiency and computational resource utilization. By opting for a larger batch size than common selections like 64, 128, 256, or 512, the training process remains relatively speedy while efficiently leveraging the server's high RAM capacity. This choice is made with the goal of optimizing both computational resource use and training efficiency. In the conclusion part, there is a comparison graph of 1024 and 2048 batch sizes.

5.6.10 Dropout

Dropout is a technique often employed in neural network training to mitigate overfitting. It involves temporarily removing, or "dropping out," a fraction of neurons during each training iteration. This procedure prevents the network from becoming overly reliant on specific neurons and, in turn, enhances its generalization capabilities. In the present study, a dropout rate of 0.2 is implemented. This means that during training, approximately 20% of neurons are randomly deactivated in each iteration, providing a controlled level of regularization to prevent overfitting. This technique complements the other hyperparameters, such as learning rate, in fine-tuning the model and is a fundamental element in ensuring its robust performance.

Dropout is not typically considered a hyperparameter in the same category as learning rate, batch size, or the number of hidden layers. Instead, dropout is a regularization technique used during the training of neural networks.

Hyperparameters are settings or configurations that are determined before the training process begins, and they directly impact the training process. Examples of hyperparameters include learning rate, batch size, and the number of neurons in a layer.

Dropout, on the other hand, is a regularization technique that is applied during training to prevent overfitting. It is a technique used to regulate the flow of information in the neural network by randomly deactivating a certain percentage of neurons during each training iteration. While dropout involves setting a dropout rate (e.g., 0.2 to deactivate 20% of neurons), it's more of a regularization method than a hyperparameter.

So, in summary, dropout is not typically considered a hyperparameter; it's a

regularization technique used alongside hyperparameters to improve the training and generalization of a neural network.

5.7 Loss Function in Neural Networks

Loss function, also known as the cost function, is an important phenomenon in the training process of neural networks. The loss function acts as the guide for the neural network during the learning process, quantifying the error or cost associated with the model's predictions. The objective of training aims to minimize this cost, narrowing the gap between predicted outputs and actual inputs of the neural network model.

For regression tasks, the common loss functions are Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). MSE measures the average of the squared differences between predicted and actual values. MSE is widely used to assess the accuracy of regression models. On the other hand, RMSE is the square root of MSE, providing a measure of the standard deviation of these differences. RMSE is often preferred as it presents errors in the same units as the predicted values, offering a more intuitive understanding of the neural network model's performance.

5.8 Custom Loss Function In This Project

A special loss function has been prepared for a unique purpose in this project and this regression task. This specialized custom loss function computes the RMSE, a common metric for assessing the variance between model predictions and actual values in regression tasks. In addition to this common technique, a mask is used. This mask is a reasonable addition that handles scenarios where the true values (y_true) are indicated as -9999, which is a common practice when dealing with missing or invalid data. The mask ensures that only valid data points (where y_true is not equal to -9999) contribute to the loss calculation. This feature is particularly valuable when working with real-world datasets where missing or invalid data points can pose challenges. This new custom loss function ignores the -9999 values, as they should not influence the loss computation. This approach effectively handles missing data during the training of your model.

In summary, this specialized custom loss function, with its mask integration, enables the calculation of RMSE while delicately managing situations where true values are designated as -9999. This makes it a suitable choice for scenarios where the consideration of missing or invalid data points is vital during the neural network model training. This loss function acts like a compass in training neural networks, steering the model to learn from data and make better predictions with each training step.

Figure 5.3: Epoch - loss graph on TensorBoard.

5.9 Callbacks

Callbacks are essential tools in training neural networks. They are functions that can be applied at specific stages of the training process to monitor, control, or customize how the training proceeds. In your provided code, you're using several different callbacks to enhance and monitor your training process.

5.9.1 LearningRateLogger

This custom callback, the LearningRateLogger, is introduced to keep track of the learning rate during training. At the end of each epoch, this callback logs the current learning rate used for optimization. Monitoring the learning rate is important because it helps us understand how the model adapts its weights during training, and it's a key element of training process transparency.

5.9.2 ReduceLROnPlateau

The ReduceLROnPlateau callback dynamically adjusts the learning rate based on the model's performance. It does this by monitoring the validation loss. If no improvement is observed in the validation loss for a specific number of epochs (as defined by the patience parameter), the learning rate is decreased by a factor (factor). This process aims to help the model converge effectively by fine-tuning the learning rate as training progresses.

5.9.3 TensorBoard

The TensorBoard callback enables powerful visualization during training. It logs various metrics such as loss, accuracy, and more. These metrics are then visualized using TensorBoard, which provides insights into the model's performance and helps identify potential issues like overfitting.

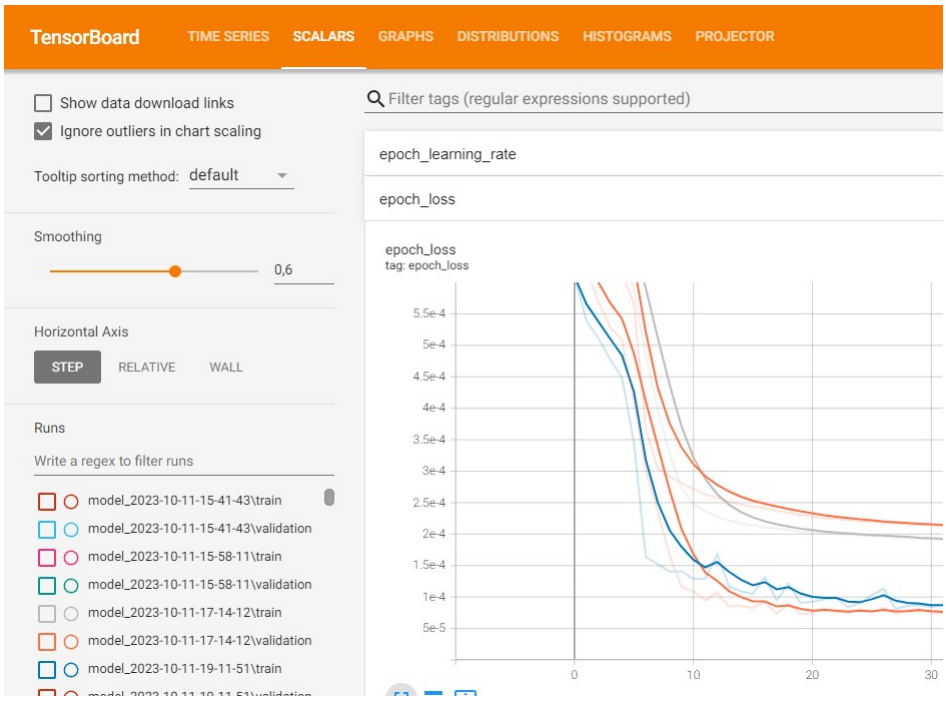

Figure 5.4: TensorBoard interface.

5.9.4 EarlyStopping

The EarlyStopping callback is designed to prevent overfitting. It monitors the validation loss, and if there is no improvement over a specified number of epochs (controlled by patience), it stops training early. Additionally, this callback allows you to restore the model's weights to the best achieved during training, ensuring that you save the most optimal model.

Collectively, these callbacks are crucial in optimizing and monitoring the training process of your neural network model, ensuring that you achieve the best performance while preventing overfitting and other potential issues. This comprehensive use of callbacks is fundamental to a successful deep learning pipeline.

5.10 Convolutional Neural Network Model Architecture

The first model is designed as the CNN regression model for time-series data and uses Conv1D layers for feature extraction. The Conv1D function is part of the Keras library, which is integrated into the TensorFlow deep learning framework. It is accessed and imported through TensorFlow and Keras libraries. This function is a one-dimensional convolution function, and it is a neural network operation that applies a filter to input data along a single dimension, capturing local patterns or features within sequential data. The general model architecture includes input layers for multiple measurement points, convolutional layers for feature extraction, and output layers for predictions.

Convolutional layers are primarily useful for time series analysis due to their effectiveness in recognizing temporal patterns within data. This is achieved by sharing parameters, reducing complexity, and facilitating robust feature learning even across variations. Additionally, in deeper architectures, these layers capture hierarchical features by starting from basic models and gradually learning more complex data features.

Conv1D is a type of convolutional layer used in Convolutional Neural Networks, which are primarily associated with sequential data such as time series and text data rather than image data. CNNs are designed to handle grid-structured data. Images and other multi-dimensional data can be given as an

example to this. However, when sequential data or time series data is achieved or encountered in a project or study, such as audio signals, text, or even one-dimensional sensor data, a modification is required. In this case, Conv1D is used to adapt the convolution operation to the one-dimensional nature of the data.

Conv1D implements one-dimensional convolutions and it is specifically designed for sequences and time series. Filters of Conv1D are applied to a small part of the input data at a time by sliding this filter across the entire array. This operation allows the neural network to detect local patterns in the data when making it suitable for capturing features within that sequential data. Conv1D is applied using libraries like TensorFlow or Keras in Python and in this study, it is applied in this way. Since sequential data in time series format was used as input from various measurement points in this study, Conv1D is chosen as an effective layer for feature extraction from these sequential data sets. The aim is to capture meaningful temporal patterns and dependencies in the data.

At first, the shape of the input data is determined and it will be used as the tensor shape. This tensor shape variable represents the shape of the training data for a single measurement point. The reason for using tensors is fundamental in deep learning due to their ability to represent and process input and output data efficiency. Tensors are multi-dimensional arrays that provide an all-around structure for handling complex mathematical operations in neural networks.

Afterward, three lists store the input layers, output layers, and intermediate layers to be concatenated, respectively. Also, the measurement points are looped for each measurement point (mp) to be performed in these layers. For each measurement point, an input layer is created. These input layers receive the data for each measurement point from iterative mp loops.

In the following step is the most crucial one. For each measurement point, a series of five convolutional layers is added. These layers are important for extracting features from the input data. In these convolutional layers, there are set key parameters such as kernel size, activation function, strides, and padding. These parameters define how the convolution is performed. Nodes, strides, and the kernel size are mentioned in the previous hyperparameter section.

5.10.1 Activation Function

An activation function is the fundamental component that makes the neural network become non-linear. Besides, it allows the neural network to learn from complex patterns and make decisions for the prediction output. It determines the output of a node or neuron and whether it is activated or not. Without activation functionality, a neural network becomes limited to linear transformations, making it less capable of learning complex patterns and relationships in data. There are several types to define the activation function. Basically, they are the sigmoid function, hyperbolic tangent (tanh) function, Rectified Linear Unit (ReLU) function, and Leaky ReLU function.

Sigmoid function is represented as (x) and it is a mathematical function that maps any real-valued number to a value between 0 and 1. That is why the output of the sigmoid function range is accepted as (0, 1). The real reason for using it in the final phase of the layers is, that it maps the output to a probability range. Therefore, the sigmoid function is used in the output layer of binary classification and logistic regression models. The reason for this study is a regression task, the sigmoid function is used in the last, final convolution layer of each model architecture. Its formula 5.1 is given below.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.1}$$

The Hyperbolic Tangent (tanh) function is a mathematical function that is also used in neural networks. It is similar to the sigmoid function but it maps the outputs as real-valued numbers to a range between -1 and 1, (-1, 1). This characteristic makes it zero-centered, which can be beneficial for optimization in determined conditions.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{5.2}$$

In neural networks, tanh is often used to ensure nonlinearity in hidden layers. It performs particularly well when the data distribution has negative and positive values. Like the sigmoid function, tanh is used to compress the output of a neuron, a node, into a specific range, allowing the network to model complex relationships in the data.

The Rectified Linear Unit (ReLU) function is a very commonly used activation function in the hidden layer of the neural networks. Contrary to the sigmoid

function, the range of the ReLU function is [0,). For any positive input, ReLU returns the input value, and for any negative input, it returns zero. ReLU ensures non-linearity by ensuring that the input signal is directly positive and zero otherwise. This function is defined as below.

$$f(x) = \max(0, x) \tag{5.3}$$

ReLU has been widely used in neural network architectures due to its simplicity and efficiency. It helps reduce the problem of lost slope, allowing faster and more effective training. Zero outputs for negative inputs also cause sparsity in the network, which can be useful in certain cases. Due to its advantageous cases, it is also used in the hidden layers of all neural network architectures of this study.

Leaky ReLU (Rectified Linear Unit) is an activation function that is also used in neural networks. It operates similarly to the traditional ReLU but with a key differentiation. This difference is that it allows a small, non-zero gradient for negative inputs, preventing the dying ReLU problem. In this function, if the input is positive, it remains unchanged, but if it is negative, i is multiplied by a small positive constant (). This controlled linearity for negative values helps avoid issues with inactive neurons in deep networks, making Leaky ReLU an important option for several applications.

Padding adds extra pixels around the input, preventing information loss at the borders. 'valid' (no padding) can reduce output size, while 'same' (zero-padding) maintains spatial dimensions. In this study, with padding='causal', it is specific padding useful for sequence data because sequence data is used as the input, preventing future data influence. Basically, causal padding is used to maintain the temporal order of data.

After all these iterative layers for each measurement point, these layers are concatenated. In this step, the concatenation process combines the features extracted from each measurement point into a single tensor.

Afterwards, two additional convolutional layers are added. The first one has 32 nodes and the second convolutional layer has 64 nodes. The aim of this step in order to enhance the model's capacity to capture progressive complex patterns and features in the last convolutional layer before the Global Max Pooling Layer. Between these two layers, there is so important regularization layer used, it is called as Dropout layer.

The following Global Max Pooling layer is used to reduce the dimensionality of the data and extract basic information from the previous convolutional layers, and it causes retaining the most notable features.

In the last part which can be called as the output layers, for each measurement point, an output layer is created with the length of output prediction parameters' number of nodes and a sigmoid activation function. These layers produce the final predictions for each measurement point.

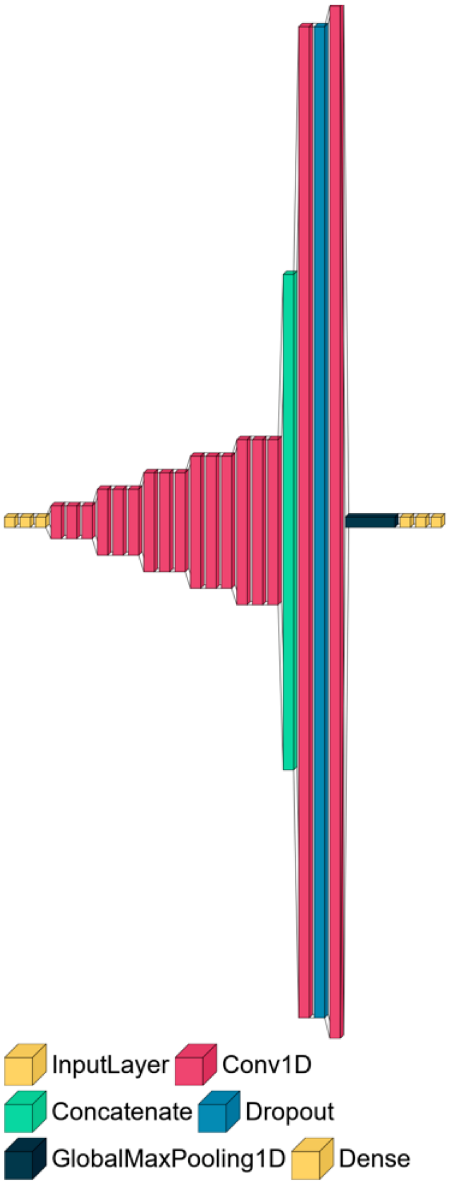

Figure 5.5: CNN model with visualkeras layered diagram.

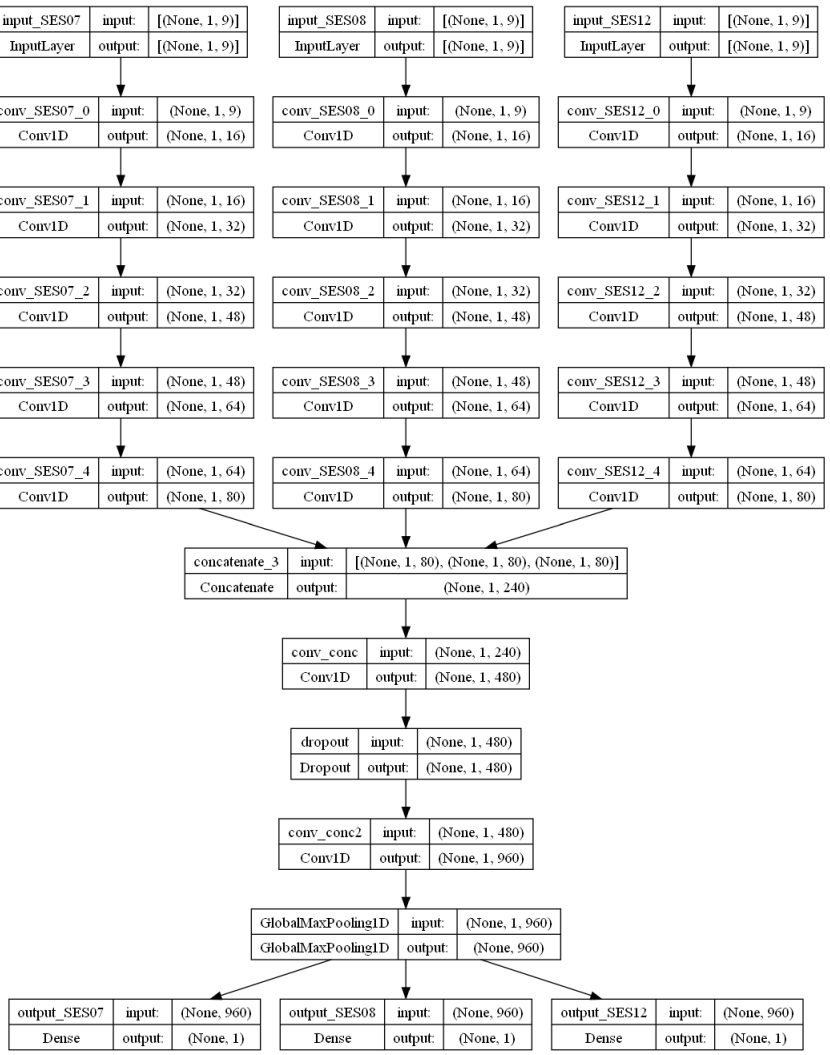

Figure 5.6: CNN model with detailed diagram.

In summary, this convolutional neural network model architecture takes input data from multiple measurement points and processes it through a series of convolutional layers for feature extraction. Then remarkably it concatenates the results of each measurement point and further processes them through convolutional and pooling layers. Finally, it produces output predictions for each measurement point.

5.11 CNN - LSTM Model Architecture

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) layer. It particularly stands for processing and making predictions based on sequential data. LSTM is highly effective at capturing dependencies over time, making it an excellent choice when working with time series data. The reason for this is that LSTM and CNN combine this architecture and is also being investigated.

As the second neural network model architecture, LSTM layers at the beginning of this neural network are initialized. The LSTM layer is utilized in Keras and TensorFlow by importing the library and calling the layer directly using tensorflow.keras.layers.LSTM. It is a part of the Keras library within TensorFlow and can be easily accessed and configured within a neural network model.

In the beginning, the input layer and the LSTM layer are defined for each measurement point (mp) in the network for the dataset. The LSTM layer has 64 units and returns sequences, which means it provides output at each time step. This setup is suitable for learning from the sequential nature of your data. For the function, the number of 64 is used for LSTM units input. This number specifies the number of memory cells or units within the LSTM layer and it can be changed due to the use cases. A higher number of units allows the LSTM layer to capture more complicated patterns within the sequential data. A higher number than 64 may cause computational complexity and a less number than 64 may cause the data understandability of the LSTM layers. Due to this situation, the number of 64 is selected in this study. There is another attribute called as return sequences that is marked as True. This signifies that the LSTM layer returns sequences rather than just the final output. This is an important point that the subsequent layers also require sequential data as input, as is often the case in this kind of sequence prediction task.

In the following, after the LSTM layer, the process goes with a similar architecture to the previous CNN model. Convolutional layers are used for feature extraction and they are especially beneficial for identifying local patterns in the dataset. Therefore, five convolutional layers again are applied to each measurement point's input. Between those CNN layers, in order to regularize the neurons, the Dropout technique is applied with the rate of 0.2 here. Afterward, the GlobalMaxPooling1D layer takes the stage after the consecutive convolutional layers. It is again in order to reduce the dimensionality of the data by

selecting the maximum value from the feature maps. As in the previous model, the output layers are added for each measurement point. These layers are designed to make the prediction of the values based on the learned features in the previous CNN layers. In the last stage, another LSTM layer is applied right before the GlobalMaxPooling1D. This additional LSTM layer is set to capture longer-term dependencies in the data and provide another level of abstraction in the feature learning process.

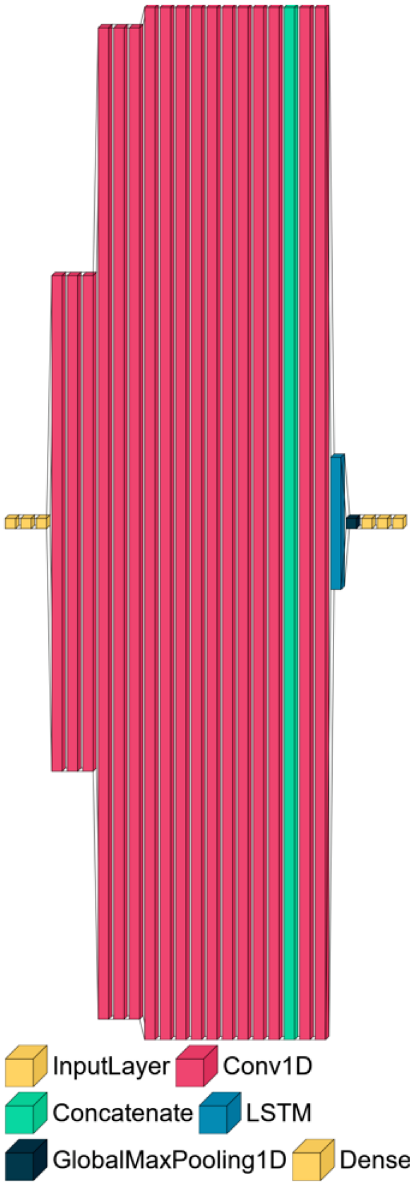

Figure 5.7: CNN - LSTM model with visualkeras layered diagram.

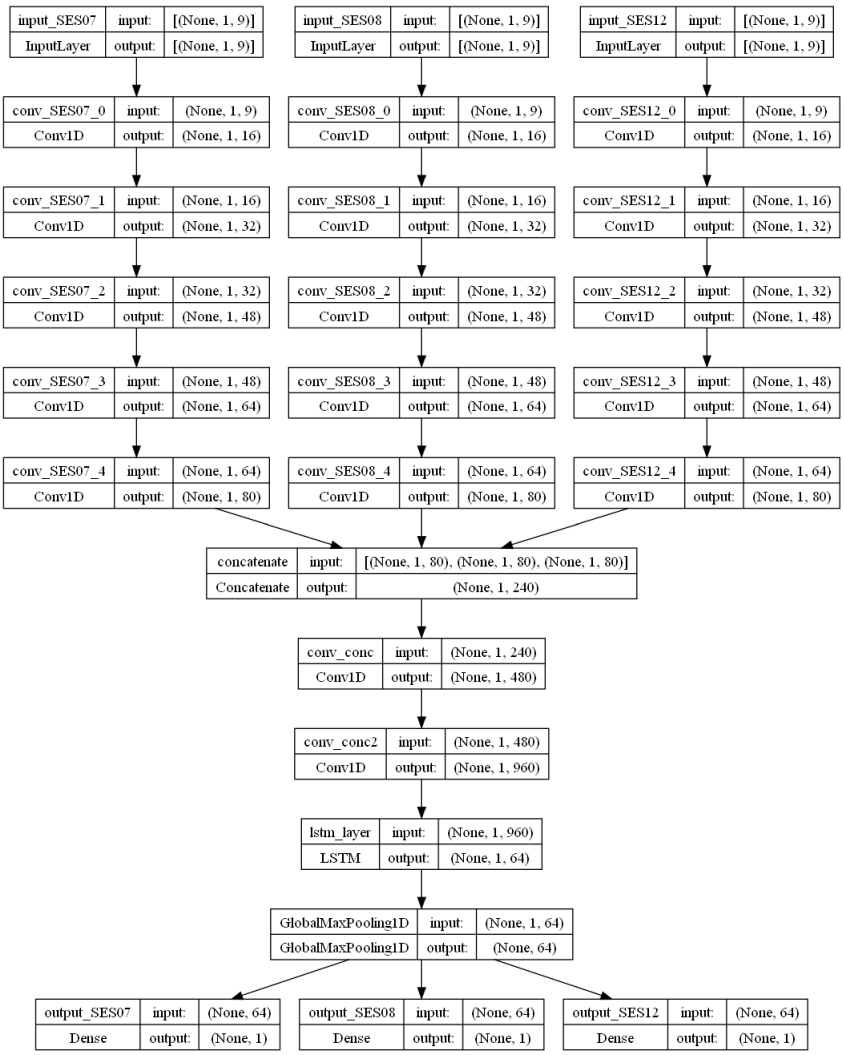

Figure 5.8: CNN model with detailed diagram.

To sum up, for this model, the addition of the LSTM layers is a strategic choice and another point of view to deal with the sequential data in this study. These detailed LSTM layers are proficient at capturing temporal dependencies which becomes crucial when working with time series data or sensor data. The LSTM layers ensure this neural network model is capable of understanding the data changes over time, which is often an important feature of accurate predictions in these domains. Overall, this architecture is well-suited for tasks where the order and timing of data points are important.

5.12 CNN Model With Residual Connections Architecture

Residual connections also can be named as skip connections, provide alternate ways for information flow and transition in a neural network. Characteristically, each layer in a neural network receives the output of the previous layer. However, in a residual connection, the input from an earlier layer is directly added to the output of the following layer. This creates a shortcut or redundant path that allows the network to bypass certain layers in the neural network.

By mentioning the advantages, the main advantage of these connections occurs in reducing the vanishing gradient problem that is often encountered in very deep networks during training. When the neural networks grow deeper, the gradients can decrease during the back propagation which can prohibit effective learning. Residual connections provide shorter paths for gradient propagation, facilitating smoother gradient flow and thus aiding the optimization process. This method makes it easier to train deeper neural networks by preserving gradients, allowing the neural network model to learn more complex features and improving overall performance. In this model architecture, residual connections are incorporated and mainly they are applied as follows.

Residual connection in a neural network implemented in Python using libraries like Keras or TensorFlow. In Keras or TensorFlow, the Add() function is used to sum the tensor outputs from different layers. This function creates a Keras tensor or TensorFlow operation that adds the values of the inputs, and in the case of residual connections, it allows the addition of the output of one layer to another. The Add() function is part of the functional API in Keras, and it's utilized to merge layers or tensor outputs within a neural network model.

The model implementation starts again with the definition of the input layers as in previous models. These are applied for each measurement point. For each measurement point, five convolutional layers are added similarly to previous models. The distinguishable part of this architecture from others is the usage of residual connections. After the five convolutional layers, the results are concatenated using the Concatenate layer. This step brings together the features extracted from each convolutional layer.

Afterward, two convolutional layers in a sequence are followed by the insertion of the output from the previous Concatenate layer. This sequence is called

as a residual block. The purpose of the residual block is to skip over one or more layers by inserting the input into the output. This process helps to maintain gradient flow, making it easier to train deep networks. In this architecture, two residual blocks, residual 1 and residual 2 are used as follows.

Then the flow continues with the GlobalMaxPooling1D layer, which extracts again the most important features from the data. In the final stage, as with the previous models, the flow ends with output layers for each measurement point to make predictions.

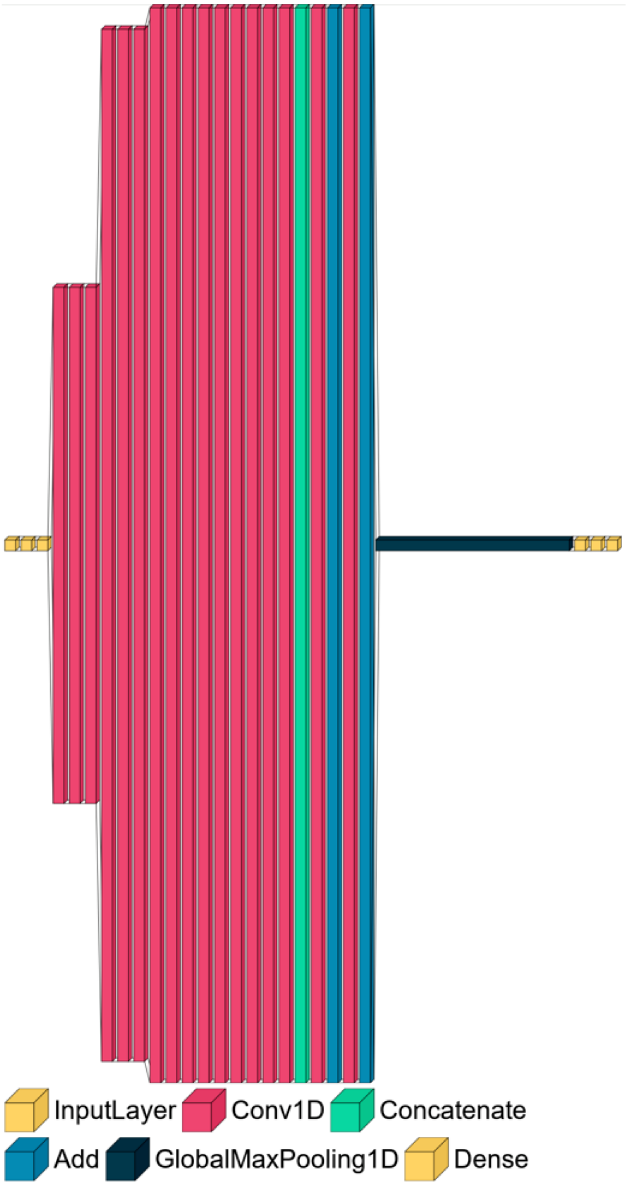

Figure 5.9: CNN with residual connection model with visualkeras layered diagram.

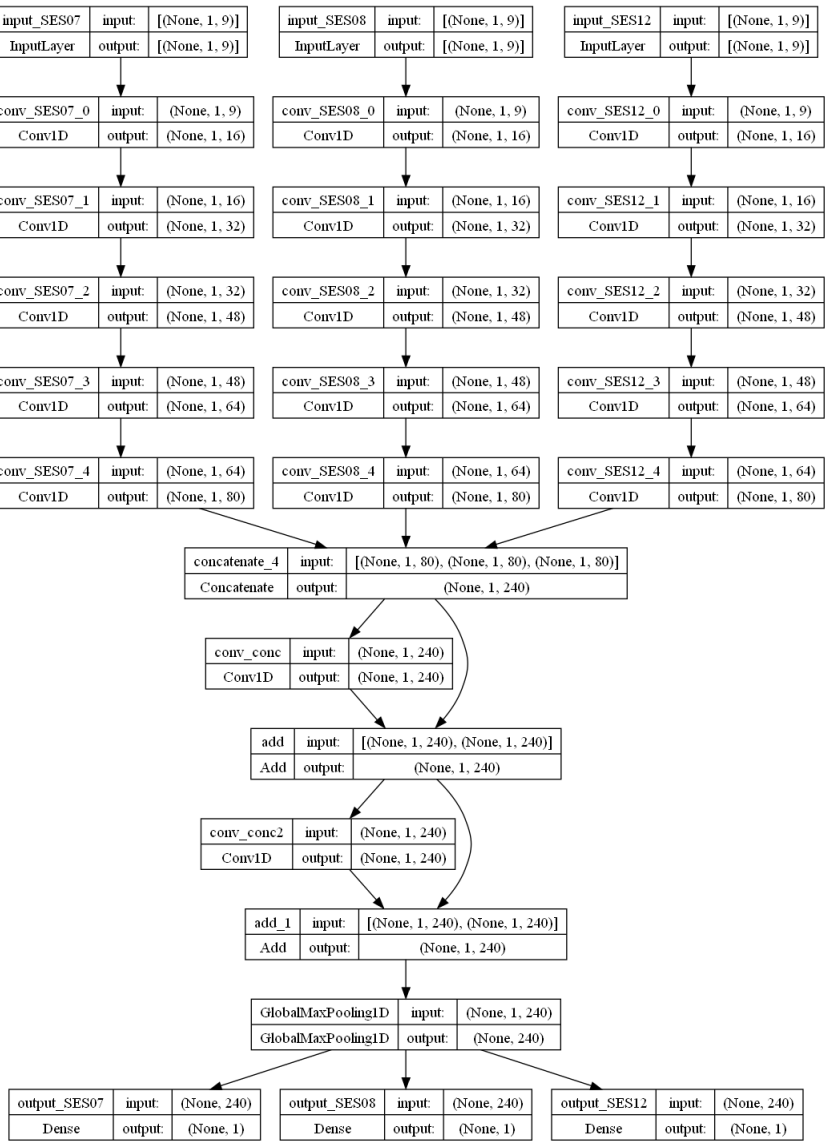

Figure 5.10: CNN model with detailed diagram.

In summary for this architecture, the residual connections effectively capture complex features from the dataset. The concatenation layer and the addition of residuals play a vital role in enabling your network to understand and learn from data more effectively. This makes it suitable for tasks where complex patterns need to be identified.

5.13 Model Compilation

In order to configure a neural network before the training step, compilation is the essential process. Depending on the library usage, there is a method provided by Keras as part of TensorFlow that is specifically used to compile the Keras model. This function from the Keras library within TensorFlow, allows the user to define fundamental parameters required for the training phase. It requires three primary arguments: the optimizer function, the loss function, and optional metrics.

The choice of optimizer defines the method used to update the model parameters during training to minimize the resulting and computed loss function. The optimizer can be chosen from a variety of options and each has different characteristics that affect training speed and model performance. The adaptive nature of the Adam optimizer is particularly useful in scenarios where the data distribution may change over time or the gradients of different parameters may be poorly distributed. This adaptability often helps to converge faster and more efficiently than traditional optimizers such as Stochastic Gradient Descent (SGD). Furthermore, the Adam optimizer adaptively adjusts the learning rates for each parameter based on the average of its past gradients. This adaptability is particularly suited to dynamic or changing data conditions, which is crucial when dealing with a system such as a sewage network where data patterns may change depending on environmental factors or specific network conditions.

The Adam optimizer combines the advantages of both AdaGrad and RMSProp. For sewage networks, conditions, and data points can have varying importance and frequent changes. Adam's combination of adaptive learning rates and momentum allows for more balanced and efficient updates to the parameter space of the neural network model. Besides, in practical use cases in the deep learning world, Adam optimizer has showcased efficient performance, especially in environments where data characteristics fluctuate. The self-adapting learning rate can handle varying gradient magnitudes, and this optimizer is capable of improving model convergence. Lastly, the Adam optimizer provides customization by enabling the user in order to fine-tune parameters like learning rates and betas for different scenarios. This flexibility strengthens the optimization process that is specifically designed for the subtle dynamics of networks, meeting potential changes and requirements in the system.

To summarize, the choice of Adam optimizer in complex network-related

projects provides adaptability, efficiency, and the ability to handle potential variations and changes in system data, making it a suitable choice to optimize the performance of the neural network. The other input of the compilation process is the loss function. It measures how well the model performs on training data. During training, the goal is to minimize this function, and the selection for this regression task is made and defined as a specific loss function in the earlier stages.

The last input is the metrics and it is optionally used. Normally, metrics like accuracy or precision can be used to monitor the model's performance during training. However, for this study, observation is made based on the loss function.

At the end of the model compilation, to sum up, the compile function allows customization with optional arguments for metrics or loss weights. By configuring the neural network model using this function, the neural network becomes ready for the subsequent training phase, defining how it learns from the provided data.

5.14 Model Fitting

Model fitting is the training phase of a deep learning model on a given dataset, where the model adjusts its parameters to make predictions and capture underlying patterns. All the sections described earlier have been organized for this part of the study. Through an iterative process, the model learns from the training data, aiming to minimize the difference between predicted and actual outcomes, consequently optimizing its ability to make accurate predictions on new, unseen data.

Data inputs and outputs are initialized according to a multi-stage iteration of training, validation, and testing partition. As the training parameters, the batch size and the epoch number are given to the fit function. Batch size specifies the number of samples processed before the neural network model's internal parameters are updated. The epochs variable denotes the number of complete passes through the training dataset. The details of these parameters are given before. Fit function has a verbose mode that the verbose argument determines the level of output displayed during the training process. In this task, it is set to 1. It means that it shows a progress bar for each epoch. Besides, the shuffle parameter is set to False, and it provides that the data's order remains unchanged during the training process. When working with time-based data, the shuffle

is generally set to False setting. This move ensures that the sequence of events remains in its original order. For tasks involving time series, like forecasting in sewage network systems in this work case, maintaining this time order is important. This allows the neural network model to accurately learn patterns and dependencies over time, helping to make more reliable forecasts.

In summary, this fit method runs the training loop for the defined number of epochs by updating the neural network model's weights based on the defined optimizer function and loss function set during the compilation stage. The performance metrics as custom loss function are evaluated and recorded on both training and validation datasets throughout the training process.

5.15 Model Evaluation and Prediction

Predict function is used from the Keras and TensorFlow libraries. This function works by using a trained neural network to make predictions based on new or unseen data. This method plays a significant role in evaluating model performance and generating forecasts.

At the first stage, evaluate function is used from the Keras and TensorFlow library. This assesses the neural network model's performance on the test dataset. This function typically computes loss and accuracy metrics, depending on the model's configuration, providing insight into how well the model generalizes to unseen data. For this study, it goes on the custom loss function.

In the prediction phase, the predict function generates the predictions on the dataset. It employs the trained model to produce forecasted outputs for the three different phases: training, validation, and testing datasets. These predictions are made and evaluated against the actual outputs to assess the model's accuracy, generalization, and suitability for the specific task at hand.

6

Conclusions

In the conclusion part, the aim is mentioned again and the related results and their explanations are explained. In this SWS, data from the measurement points corresponding to this project task SES 07, SES 08, and SES 12 are utilized. As depicted in the network diagram, the data from these three measurement points (SES 07, SES 08, and SES 12) holds significance. Challenges arising from sensor issues, measurement errors, or environmental complexities are meticulously addressed to ensure a comprehensive understanding of the network and data, minimizing errors in the process.

Before presenting the optimal results, illustrative examples of well-predicted parameters are provided for each model. These examples aim to demonstrate the effective performance of the three distinct model architectures.

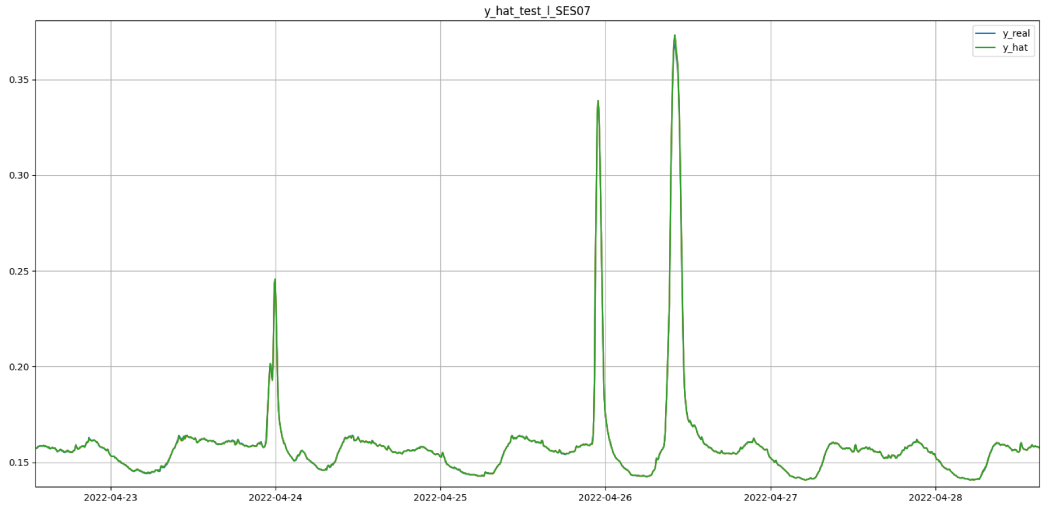

Figure 6.1: Real and predicted water level data comparison with CNN model on measurement point SES 07.

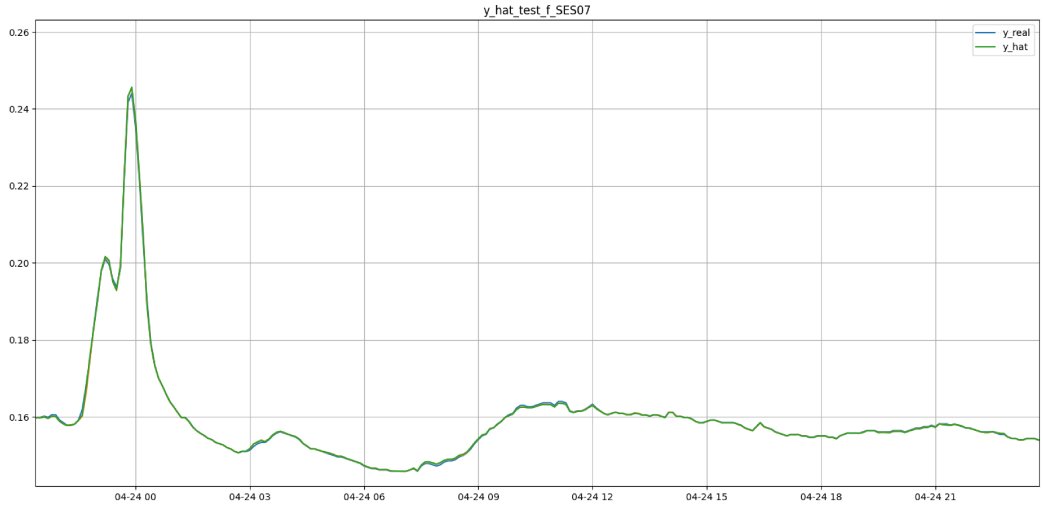

Figure 6.2: Real and predicted water flow rate data comparison with CNN model on measurement point SES 07.

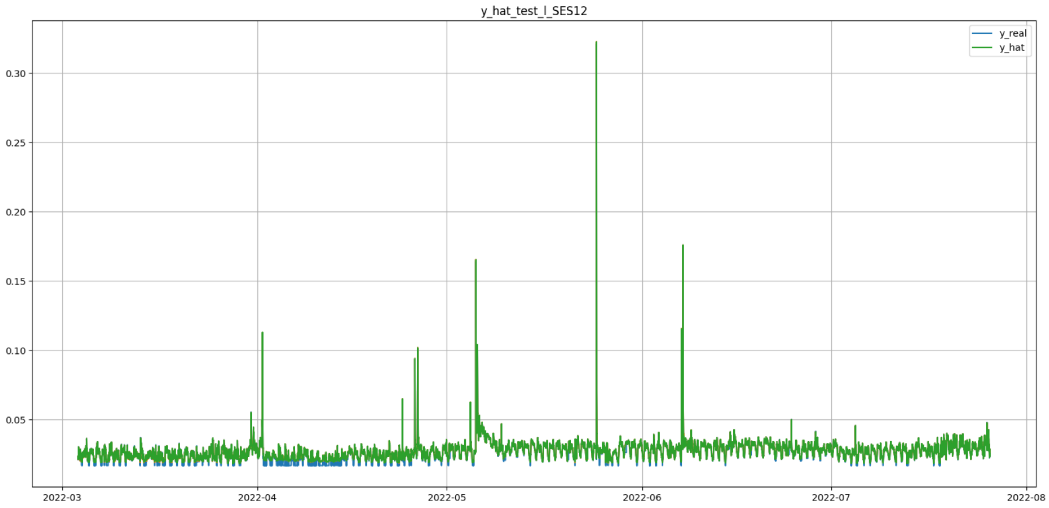

Figure 6.3: Real and predicted water level data comparison with CNN with LSTM model on measurement point SES 12.

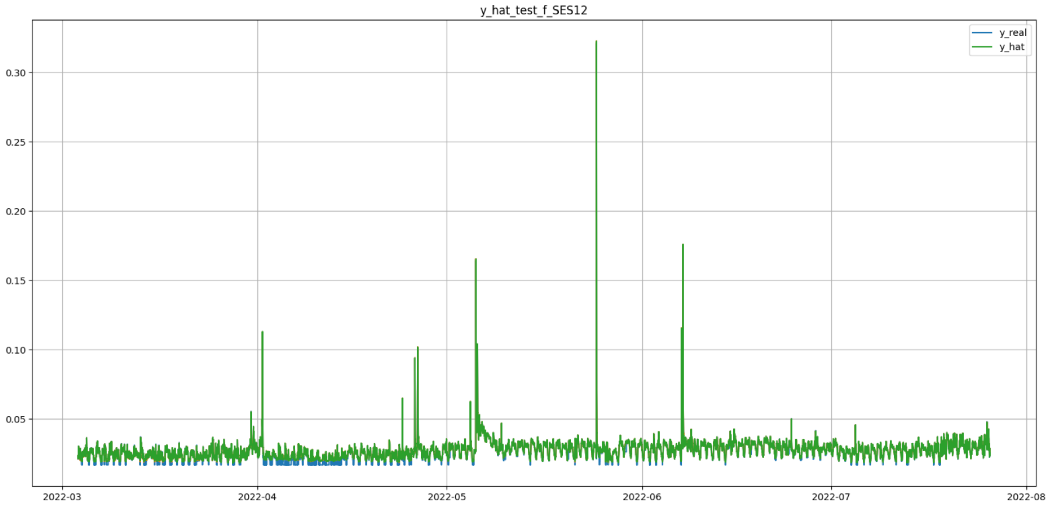

Figure 6.4: Real and predicted water flow rate data comparison with CNN with LSTM model on measurement point SES 12.

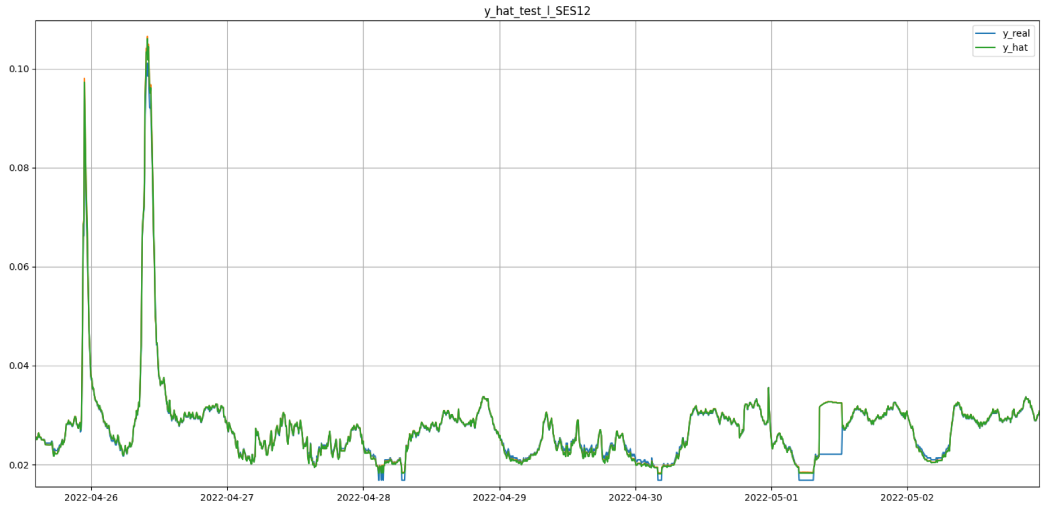

Figure 6.5: Real and predicted water level data comparison with CNN with residual connection model on measurement point SES 12.

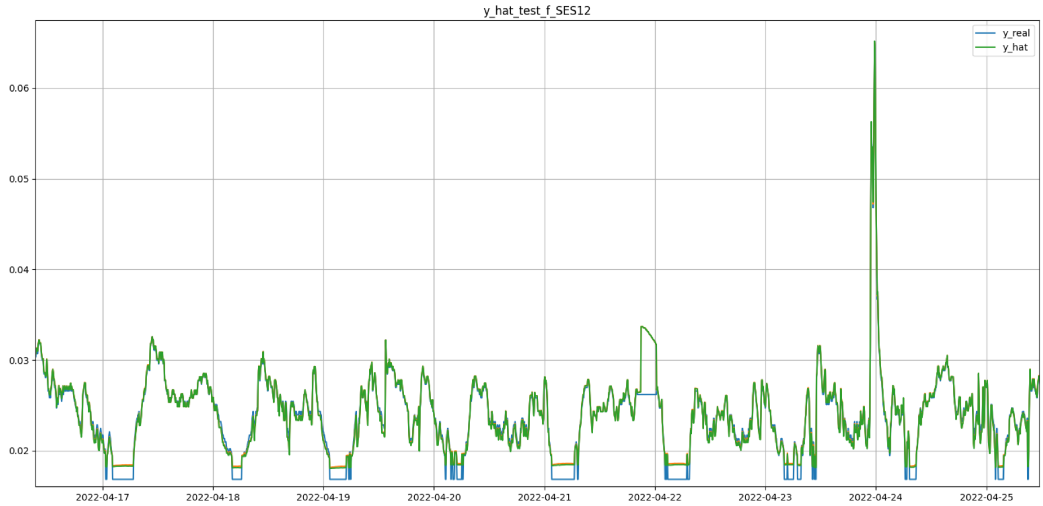

Figure 6.6: Real and predicted water flow rate data comparison with CNN with residual connection model on measurement point SES 12.

The results of the main task of the study represent a valuable evaluation of CNN's predictive capabilities for the selected sewage network. The loss part of the results, shown in Figure 6.7, dives into the RMSE analysis by presenting quantitative information on the CNN model's predictive performance. A good

number of RMSEs are obtained for each measurement point. This indicates very good prediction performance for each. When RMSE percentage calculation is made, results are obtained as 0.38% for SES07, 0.71% for SES08 and 0.47% for SES12, respectively. RMSE is just one of the possible metrics to evaluate model performance and has a certain level of generalization as it provides an average without taking other considerations into account. Therefore, a qualitative chart is also used to further check the behavior of the forecast across the 3-time series.

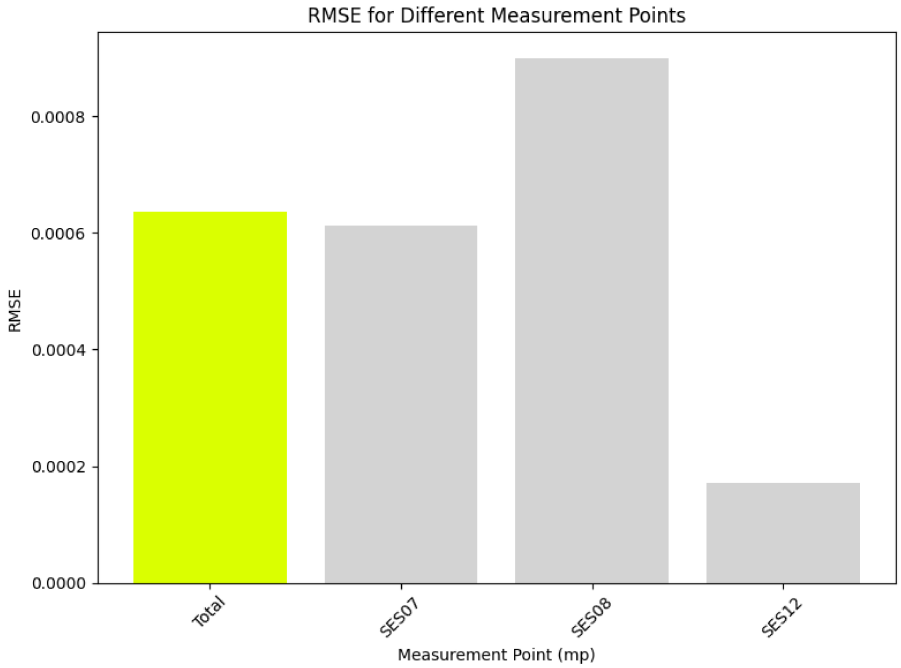

Figure 6.7: RMSE for each measurement point with CNN model.

In the level comparison subplots, the first shown in Figure 6.8 is the rain intensity. This hyetograph is structured in 1-hour intervals and visually depicts precipitation occurrences, revealing temporal precipitation patterns that significantly affect sewer network water levels.

The third section below in this subgraph shows level comparisons for SES07, SES08 and SES12 respectively. Specifically, predicted levels are compared with raw levels, i.e., levels before data engineering preprocessing. We notice a strong correlation between the actual level data and the model's SES07 predictions. These images show that the model effectively captures the natural dynamics of these sewer networks. It is noteworthy that, especially for SES07 and SES08, deviations at many levels were automatically corrected throughout the process.

These represent the optimal outcomes achieved in this section, designed

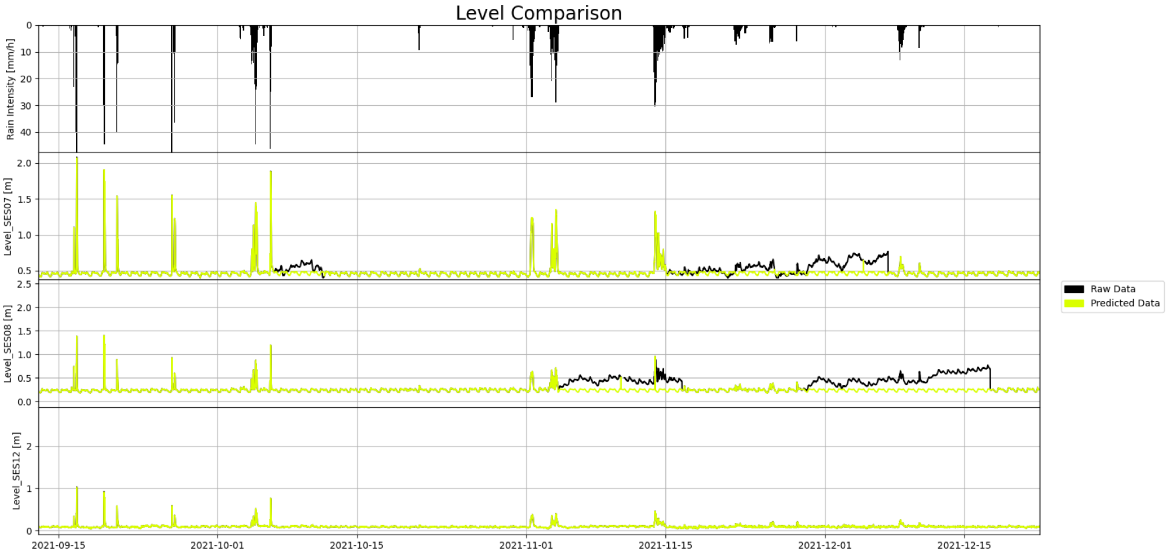

Figure 6.8: Rain intensity and the level comparison with CNN model for each measurement point.

to deliver maximum benefits to the concerned stakeholders. In the following sections, a comprehensive comparison will be made between the models and between the corresponding hyperparameters to each other.

The loss values for the CNN architecture are presented above. Additionally, the loss calculations for the CNN-LSTM model and CNN with residual connections are illustrated in Figure 6.9 and Figure 6.10, respectively.

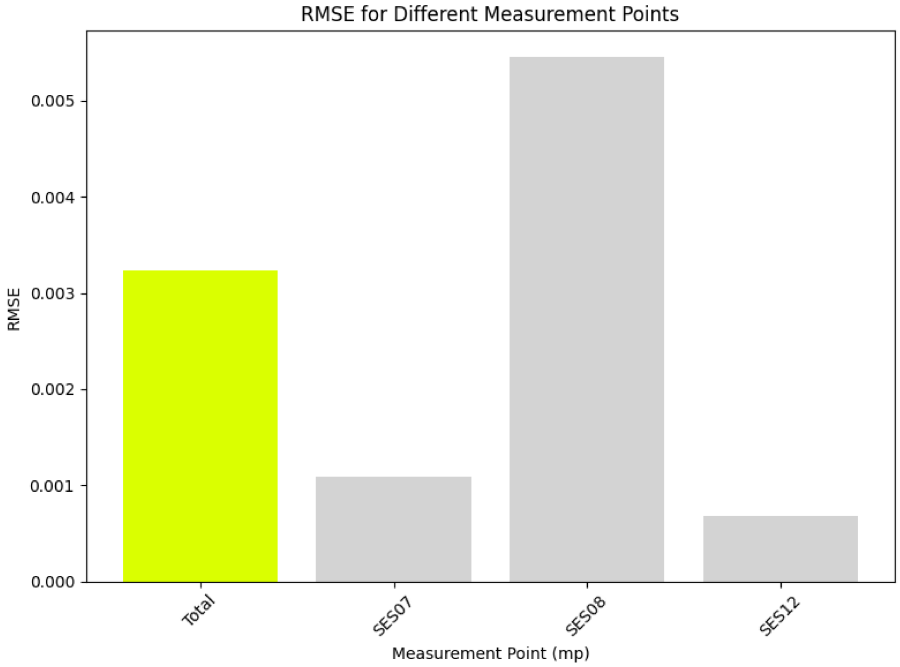

Figure 6.9: RMSE for each measurement point with CNN with LSTM model.

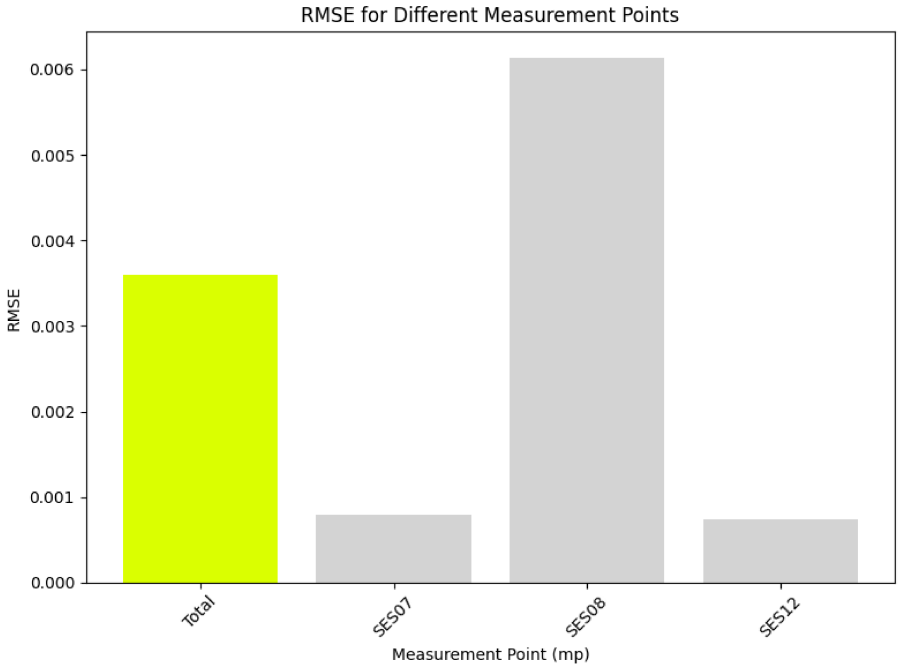

Figure 6.10: RMSE for each measurement point with CNN with residual connection model.

Furthermore, the subsequent graphs provide an in-depth exploration of the implications and nuances related to the learning rate hyperparameter. Exam-

ining the figures, it becomes evident that a learning rate of 1e-3 yields more favorable loss calculations. Additionally, this setting demonstrates an efficient number of epochs, indicating a reduced computational load.

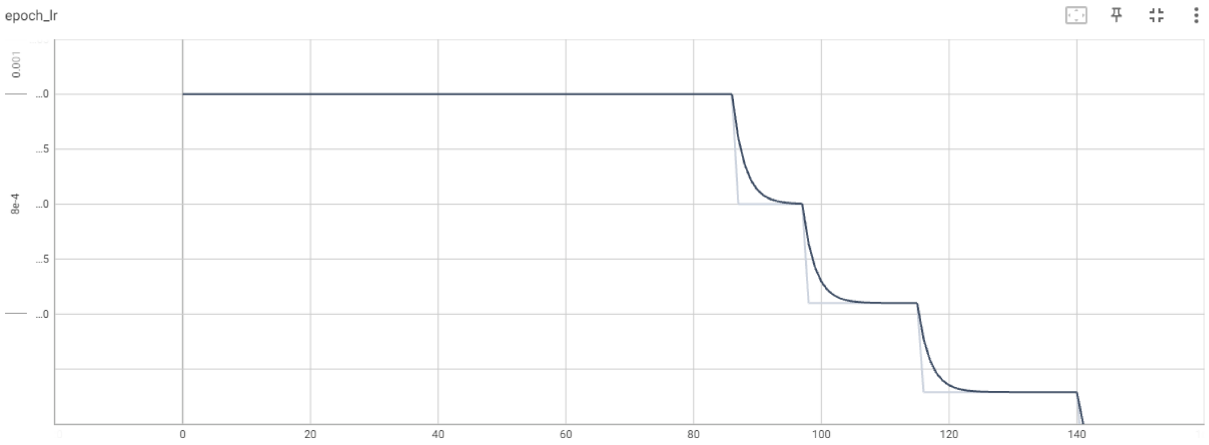

Figure 6.11: Epoch - learning rate graph when the learning rate is set to 1e-3 on TensorBoard.

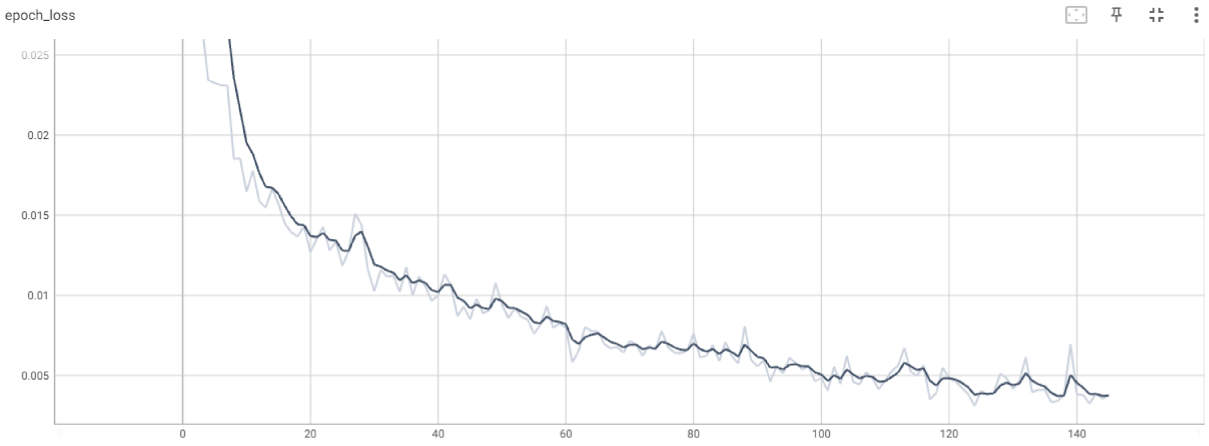

Figure 6.12: Epoch - loss graph when the learning rate is set to 1e-3 on TensorBoard.

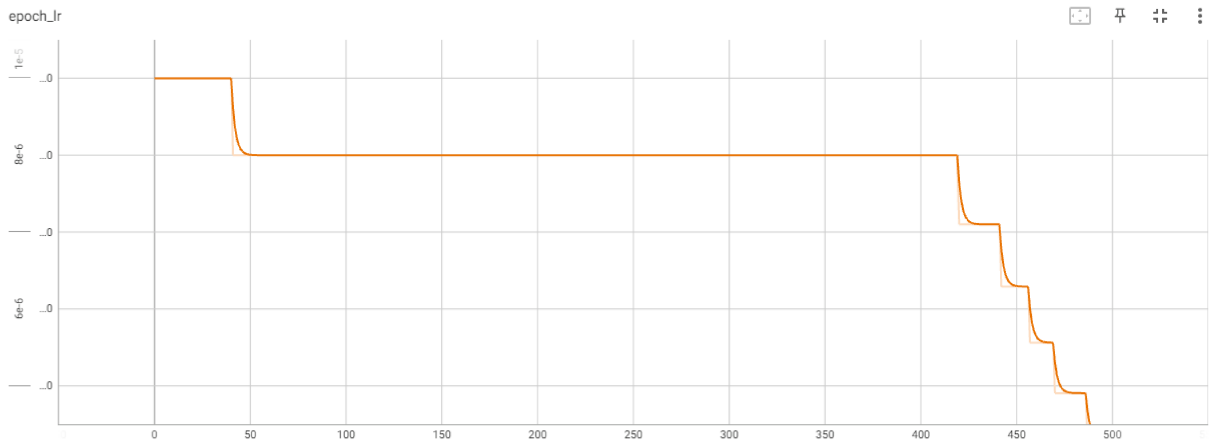

Figure 6.13: Epoch - learning rate graph when the learning rate is set to 1e-5 on TensorBoard.

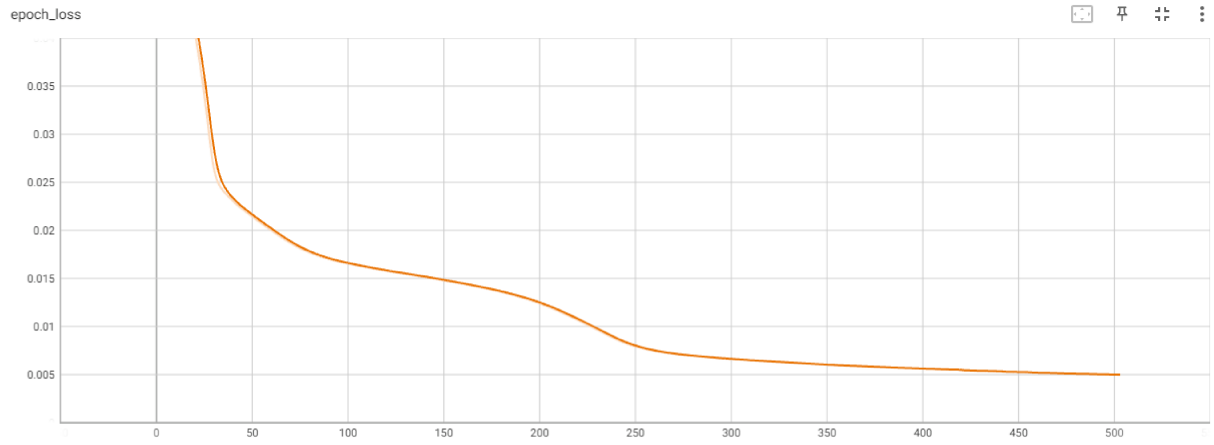

Figure 6.14: Epoch - loss graph when the learning rate is set to 1e-5 on TensorBoard.

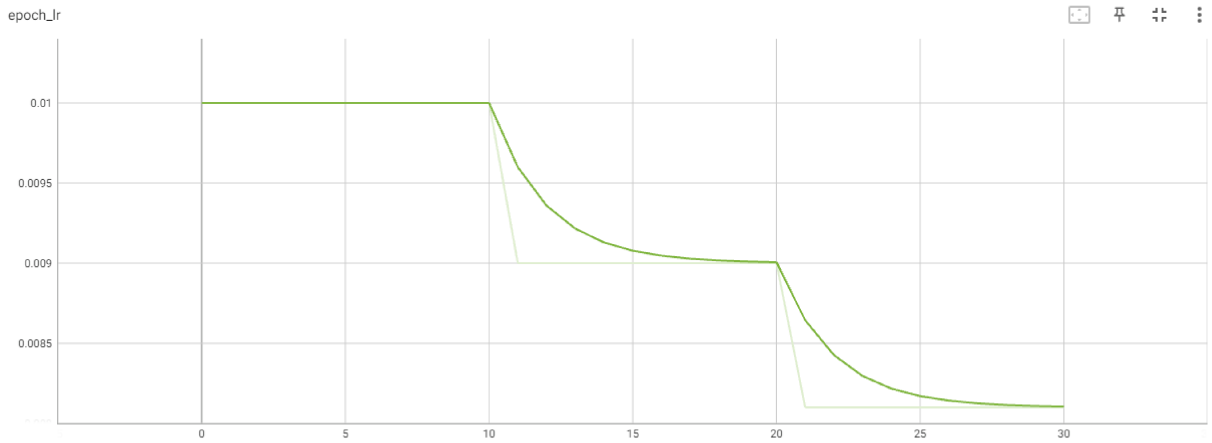

Figure 6.15: Epoch - learning rate graph when the learning rate is set to 1e-2 on TensorBoard.

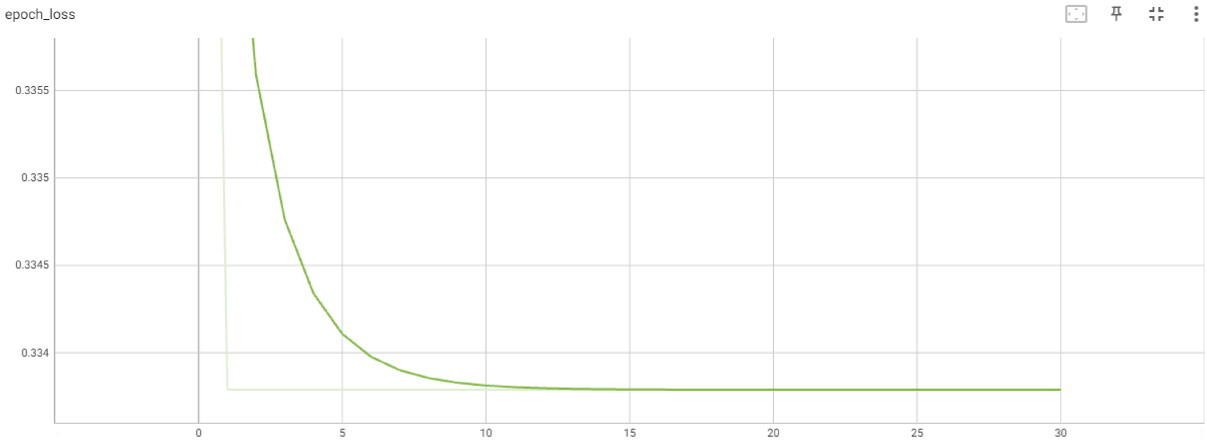

Figure 6.16: Epoch - loss graph when the learning rate is set to 1e-2 on TensorBoard.

In the final graph, the impact of batch size is illustrated on the RMSE. The results indicate a deterioration in performance, suggesting that an increase in batch size does not yield improved outcomes. This observation emphasizes the importance of optimizing batch size for better model performance, as larger batch sizes may not necessarily lead to enhanced predictive accuracy.

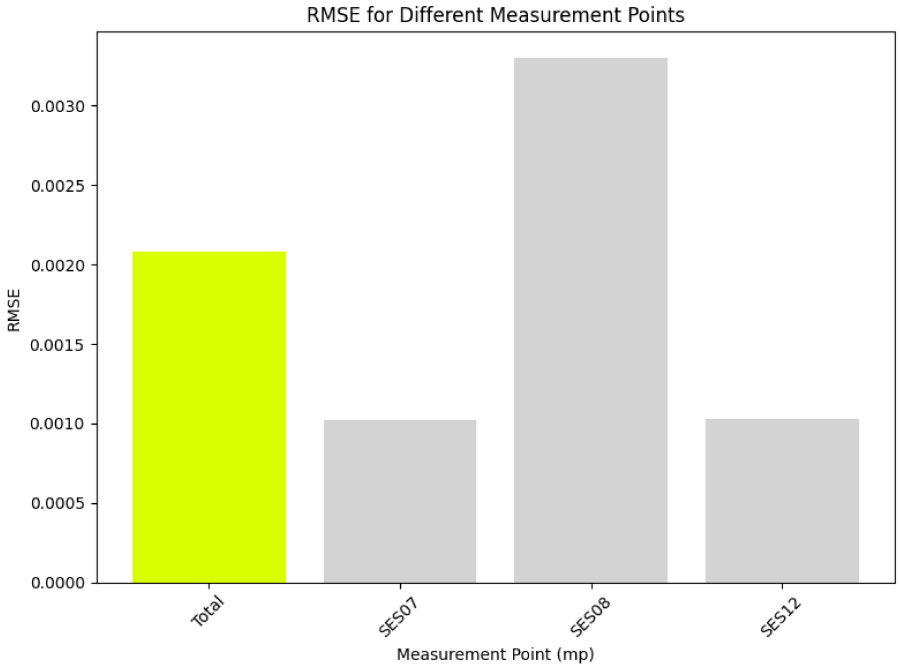

Figure 6.17: RMSE for each measurement point with CNN model when batch size is set to 2048.

References

Chen, Y., Song, L., Liu, Y., Yang, L., & Li, D. (2020). A review of the artificial neural network models for water quality prediction. *Applied Sciences*. https://api.semanticscholar.org/CorpusID:225248517

Ghobadi, F., & Kang, D. (2023). Application of machine learning in water resources management: A systematic literature review. *Water*, *15*(4). https://doi.org/10.3390/w15040620

Huang, R., Ma, C., Ma, J., Huangfu, X., & He, Q. (2021). Machine learning in natural and engineered water systems. *Water Research*, *205*, 117666. https://doi.org/https://doi.org/10.1016/j.watres.2021.117666

Ishida, K., Ercan, A., Nagasato, T., Kiyama, M., & Amagasaki, M. (2021). Use of 1d-cnn for input data size reduction of LSTM in hourly rainfall-runoff modeling. *CoRR*, *abs/2111.04732*. https://arxiv.org/abs/2111.04732

Jeni, L., Cohn, J., & De la Torre, F. (2013). Facing imbalanced data - recommendations for the use of performance metrics. *Proceedings - 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction, ACII 2013*, *2013*. https://doi.org/10.1109/ACII.2013.47

Li, J., Yang, X., & Sitzenfrei, R. (2020). Rethinking the framework of smart water system: A review. https://doi.org/10.3390/w12020412

Miao, S., Zhou, C., AlQahtani, S. A., Alrashoud, M., Ghoneim, A., & Lv, Z. (2021). Applying machine learning in intelligent sewage treatment: A case study of chemical plant in sustainable cities, 103009. https://doi.org/10.1016/j.scs.2021.103009

Otaki, Y., Otaki, M., & Sakura, O. (2007). Water systems and urban sanitation: A historical comparison of tokyo and singapore. *Journal of water and health*, *5 2*, 259–65. https://api.semanticscholar.org/CorpusID:25524284

Rezaei Kalvani, S., & Celico, F. (2023). The waterenergyfood nexus in european countries: A review and future perspectives. https://doi.org/10.3390/su15064960

Rozos, E. (2019). Machine Learning, Urban Water Resources Management and Operating Policy. *Resources*, *8*(4), 1–13. https://ideas.repec.org/a/gam/jresou/v8y2019i4p173-d286880.html

Sit, M., Demiray, B. Z., Xiang, Z., Ewing, G. J., Sermet, Y., & Demir, I. (2020). A comprehensive review of deep learning applications in hydrology and water resources. *Water Science and Technology*, *82*(12), 2635–2670. https://doi.org/10.2166/wst.2020.369

Van, S. P., Le, H. M., Thanh, D. V., Dang, T. D., Loc, H. H., & Anh, D. T. (2020). Deep learning convolutional neural network in rainfallrunoff modelling. *Journal of Hydroinformatics*, *22*(3), 541–561. https://doi.org/10.2166/hydro.2020.095

Xu, T., & Liang, F. (2021). Machine learning for hydrologic sciences: An introductory overview. *WIREs Water*, *8*. https://doi.org/10.1002/wat2.1533

Zhu, M., Wang, J., Yang, X., Zhang, Y., Zhang, L., Ren, H., Wu, B., & Ye, L. (2022). A review of the application of machine learning in water quality evaluation. *Eco-Environment Health*, *1*(2), 107–116. https://doi.org/https://doi.org/10.1016/j.eehl.2022.06.001

Acknowledgments

I would like to express my gratitude and warmest thanks to my project manager, Riccardo de Socio, and my co-supervisor, Sergio Dubon, for their invaluable guidance and insightful advice throughout the analysis and writing stages of my thesis. Their unwavering support and expertise have been instrumental in the completion of this work. I am truly thankful for the opportunity they provided me, and I have gained a wealth of knowledge and experience through their mentorship.