POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Predictive Analysis and Forecasting of COVID-19 Evolution in the Canary Islands based on LSTM Models

Tesi di Laurea Magistrale in
Computer Science Engineering - Ingegneria Informatica

Author: **Ginevra Iorio**

Student ID: 969803
Advisor: Prof. Emanuele Della Valle
Co-advisors: Prof. Carlos M. Travieso-González
Academic Year: 2022-23

Abstract

Being effective in learning from long-term dependencies, Long Short-Term Memory (LSTM) models, a type of Recurrent Neural Network (RNN), are well-suited for time series analysis and forecasting future data based on historical data. This research project focuses on predicting the evolution of the COVID-19 pandemic in the Canary Islands through historical time-stamped data on hospitalized cases. The final performance of these models is finally assessed using two criteria: Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE). The primary objective of this research is to provide a scientifically-grounded prediction of future hospitalisations due to COVID-19 in the Canary Islands. Ultimately, the results of this study have the potential to contribute to better decision-making processes in the management of the pandemic, helping to mitigate its effects on public health and society as a whole.

Keywords: LSTM models, COVID-19, time series forecasting, Root Mean Square Error, Mean Absolute Percentage Error

Abstract in lingua italiana

Essendo efficaci nell'apprendere le dipendenze a lungo termine, i modelli di Long Short-Term Memory (LSTM), un tipo di Rete Neurale Ricorrente (RNN), sono ben adatti all'analisi di serie temporali e alla previsione di dati futuri basati sui dati storici. Questo progetto di ricerca si concentra sulla previsione dell'evoluzione della pandemia da COVID-19 nelle Isole Canarie attraverso dati storici a timestamp sui casi ospedalizzati. La performance finale di questi modelli viene valutata utilizzando due criteri: l'errore quadratico medio (RMSE) e l'errore percentuale medio assoluto (MAPE). L'obiettivo primario è fornire una previsione scientificamente fondata delle future ospedalizzazioni dovute a COVID-19 nelle Isole Canarie. In definitiva, i risultati di questo studio hanno il potenziale per contribuire a processi decisionali migliori nella gestione della pandemia, aiutando a mitigare i suoi effetti sulla salute pubblica e sulla società nel suo complesso.

Parole chiave: LSTM, COVID-19, serie temporali, errore quadratico medio, errore percentuale medio assoluto

Contents

1 | Introduction

During the past two years, the SARS-CoV-2/COVID-19 pandemic has been devastating the world's health care systems, ending up being a major threat for our society. Coronaviruses are positive, single-stranded, enclosed, big RNA viruses that can infect a variety of animals in addition to humans. On the Huanan seafood market in Wuhan, China, the virus appears to have successfully crossed over from animals to people. The virus causes a severe respiratory disease that spreads through the air and that is distinguished by a high risk of infection from person to person. In almost 75% of patients, the infection can proceed to a severe condition with dyspnea and severe chest symptoms similar to pneumonia [38]. Consequently, ever since its upspring in China at the end of 2019, despite the strict controls put in place in the Wuhan region, it has spread steadily both within and outside of the country [16]. On January 30, 2020, the COVID-19 outbreak was deemed a Public Health Emergency of International Concern by the World Health Organization (WHO), and a few months later it was announced as a global pandemic [39].

Accordingly, especially in the first months of the pandemic, hospitals were quickly overcrowding, the medications not always effective and the whole population threatened by this highly contagious virus. The healthcare systems of numerous nations, including Italy, Spain, France, and the United States, have been severely harmed by this disease's exponential global expansion. The COVID-19 epidemic, which has a significant detrimental impact on public health, is still currently one of the most pressing issues facing our society [44]. Several modeling, estimation, and forecasting practices have been proposed in order to comprehend and control this epidemic. The use of hospital resources and the development of treatment strategies to best manage infected patients have now become dependent on accurate forecasting of the number of COVID-19 cases.

The Canary Islands, an archipelago located off the coast of West Africa and one of the autonomous communities of Spain, have been severely impacted by the COVID-19 pandemic. The first case was reported in January 2020, and since then, the virus has spread rapidly throughout the whole archipelago. As of the 22nd of March 2022, the Canary

Islands had accumulated 323,329 cases and 1,617 deaths due to COVID-19. Nowadays, more than two years from the start of the pandemic, COVID-19 keeps threatening the islands. Between July 1st and 5th 2022, hospitalized cases had increased by 13.7% and, in Gran Canaria, the percentage of occupancy of hospital beds by COVID-19 patients was 10.06%, a data that, accordingly to the Public Health Committee, was considered high risk [1]. Moreover, the Canary Islands have a unique geographical and demographic profile, with a high proportion of older people and a significant tourism industry. Therefore, predicting the future trajectory of the pandemic in this region is essential for informing public health policies and decision-making processes. In fact, accurately forecasting hospitalisation rates can help policymakers anticipate the demand for hospital resources and prepare accordingly. Furthermore, understanding the factors that contribute to the spread of the virus can help guide the development of effective public health interventions. Accordingly, the aim of the project is to aid the sanitary system and the public opinion to contain the pandemic in the Canary Islands and undertake appropriate preventative and response measures.

The proposed prediction system is based on temporal series, which are the basis for characterising an observed system and for predicting its future behaviour [40]. Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is particularly effective in learning from these long-term dependencies, making it a promising approach for the purpose of this project. Usually, RNNs are limited to look back in time for approximately ten time-steps. This is due to the fed back signal which is either vanishing or exploding. The vanishing problem, in fact, is a challenge that arises in deep learning models, particularly in RNNs, in which the network architecture allows for the propagation of information from previous time steps to the current time step. This is achieved by using recurrent connections between the hidden states of the network. However, during training, the gradients that are back-propagated through the network can become very small and the weights of the network are updated less and less, resulting in slow or no learning [17]. This can make it difficult for the network to learn long-term dependencies. However, being capable of learning more than 1,000 time-steps and overcoming the vanishing problem, LSTM is considered one of the most powerful dynamic classifiers publicly known [34]. In the context of predicting the evolution of the COVID-19 pandemic in the Canary Islands, LSTM models are indeed a suitable choice because they can effectively capture the complex temporal dependencies in the data, including seasonal patterns and other cyclic behaviours.

[1] https://www.canarianweekly.com/posts/Five-Canary-Islands-now-have-incidence-rates-at-high-risk-for-Covid

To evaluate the performance of the models developed in the project, two criteria will be used: Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE). RMSE is a measure of the average difference between the predicted values and the actual values. It is calculated by taking the square root of the average of the squared differences between the predicted and actual values. One of the advantages of using it as a measure of model accuracy is that it is sensitive to large errors and outliers, penalising models that make large errors more heavily [9]. MAPE, on the other hand, measures the percentage difference between the predicted values and the actual values. It is calculated as the average of the absolute differences between the predicted and actual values, divided by the actual values, and multiplied by 100. MAPE is useful because it allows for easy comparison of the accuracy of models across datasets with different scales or units [11]. For example, it can be used to compare the accuracy of models predicting COVID-19 hospitalisations across different regions or countries, where the number of hospitalisations in each region may vary widely. Therefore, through this metric, a comparison of the results with those of different, larger regions could be made. By using both these criteria, a comprehensive understanding of the accuracy of the LSTM models in predicting the future evolution of the COVID-19 pandemic in the Canary Islands can be obtained. This evaluation will allow to assess the effectiveness of the models and to identify any areas for improvement. Ultimately, it can be ensured that the predictions are scientifically-grounded and can be used to inform decision-making processes and public health policies in the Canary Islands.

The primary objective of this research project is to develop LSTM models to predict the evolution of COVID-19 hospitalisations in the Canary Islands based on historical time-stamped data. The following specific objectives are derived from the general one:

- O1: Analysis of the available data with database metrics.
- O2: Application of time series LSTM methods to build a training set.
- O3: Testing of the forecast data.
- O4: Evaluation of the results through RMSE and MAPE metrics.

The first objective (O1) involves analysing the dataset through database metrics to ensure its reliability and relevance for use in the LSTM models. A check for outliers, missing values and inconsistencies needs to be made, together with an accurate labelling of the data. The second objective (O2) is to apply time series LSTM methods to build a training

set that captures the trends and patterns in the data. This involves identifying the most important variables and determining the appropriate architecture and parameters for the LSTM models. Subsequently, simulating the system to forecast the new data. The third objective (O3) is to test the forecast data generated by the LSTM models to determine their accuracy and reliability. Improvements to the model will be accurately made at this step. Finally, the fourth objective (O4) involves evaluating the results of the LSTM models using the RMSE and MAPE metrics. This allows for a quantitative assessment of the accuracy and performance of the models and provides insights into their potential utility for predicting COVID-19 hospitalisations in the Canary Islands.

Overall, this research project represents an important contribution to the understanding of the COVID-19 pandemic in the Canary Islands and has the potential to inform public health policies and decision-making processes in the region and beyond. Data and code are available at: https://github.com/ginevraiorioo/LSTM-time-series-forecasting-COVID-19.

The rest of this project is organized as follows: Chapter 2 presents the state of the art review, in which related works adopting the above-described LSTM models for time series forecasting of COVID-19 cases will be read and analyzed, in order to acknowledge the current state of art and relative existing works. Chapter 3 provides an overview on the use of machine learning for predicting the spread of infectious diseases and the various steps that have been followed to implement LSTM. Chapter 4 describes the data used in this project, including the sources and methods used for data collection and preprocessing. Chapter 5 outlines the methodology used for predicting hospitalization rates and identifying the factors that contribute to the spread of the virus. Chapter 6 presents the results of the analysis and discusses their implications for public health policies in the Canary Islands. Finally, Chapter 7 provides a summary of the key findings and recommendations for future research.

2 | State of the Art

The relevant studies, techniques, and methodologies that have been proposed in the literature to address the problem of predicting COVID-19 hospitalisations will be discussed in this chapter, highlighting their strengths and weaknesses, and paying attention in identifying any gaps or limitations in the existing works.

2.1. Time Series Analysis and Forecasting Applications

Deep Learning methods for forecasting systems have many different applications, the main of which happen to be the following:

- **Demand Forecasting for Businesses**
  A crucial challenge for companies that handle supply and procurement is predicting the client demand. According to this, another typical application is for the dynamic change of prices and rates for goods and services in response to demand and revenue targets. The advantages of demand forecasting are more effective production scheduling, inventory management and reduction, cost reduction, optimised transport logistics and increased customer satisfaction.

- **Price Prediction for Apps**
  The aim of price prediction in time series forecasting is to generate occasions to improve and personalize the user experience when referring to apps. One relevant case is a flight booking business called *Fareboom.com*[1], that is successful in locating the most affordable tickets for its clients. Because they vary quickly and for no apparent reason, airfares are an issue and having some future price information would be useful for a user. Giving clients the knowledge about whether prices would drop or increase in the future would make customers more likely to make repeat purchases

[1] https:.fareboom.com

and choose *Fareboom.com* as their preferred travel tool.

- **Forecasting Pandemic Spread**
  The primary method used in healthcare to forecast the spread of COVID-19 is time series analysis and forecasting. It was used to predict transmission, calculate mortality rates, track the epidemic's spread, and more. Being a recent event, these techniques have been widely applied lately to gain the backing of the government and the community in order for them to get ready for the upcoming days of the pandemic. On these premises, time series analysis and forecasting can be applied to almost all healthcare fields, from genetics, to medications expenditures, to diagnosis and treatment.

- **Anomaly Detection for Cyber Security**
  One of the most frequent machine learning jobs is anomaly detection, which searches for outliers on the distribution of the data points. Time series data is very handy for this purpose, while it is not a requirement. Detecting outliers is very beneficial when it comes to fraud detection and for instance *PayPal* applies time series analysis to monitor each account's typical operational frequency in order to spot any unusual spikes in the volume of transactions. Subsequently, Cyber Security takes advantage of these techniques to capture and report malware attacks through sensors. These tools are also applied for predictive maintenance, the practice of anticipating equipment failure to cut down on repair and downtime costs. The purpose of an accurate demand forecast is to minimize the deviation between the actual demand and forecasting and using time series for this purpose is appropriate when the demand pattern doesn't vary significally every year.

In conclusion, time series analysis for forecasting systems has a wide range of applications in various fields, including demand forecasting for businesses, price prediction for apps, pandemic spread forecasting in healthcare, and anomaly detection for cyber security. These methods have proven to be highly accurate and effective, providing businesses and organizations with the tools they need to make informed decisions and better serve their customers. Nonetheless, this project will focus on one of the aforementioned practices, which is expressly time series analysis for pandemic prediction.

2.2. Related Works

Having numerous applications, time series analysis and forecasting has been widely applied in many different research projects, many of which being focused on the prediction of patterns in the spread of diseases. Time series analysis is a powerful tool that can be used to identify these patterns and help public health officials make informed decisions about how to respond to the outbreaks. By analysing data on the spread of the disease, including the number of cases, the rate of transmission, and the severity of symptoms, researchers can develop models that predict how an epidemic will evolve over time.

2.2.1. Time Series Analysis for Pandemic Prediction

In R. Allard's study [3], forecasts were compared with observed numbers of cases on different diseases, with the objective of making progress in the decision-making process about whether an apparent increase in cases represents an outbreak of the disease or just a random variation. Based on experience, it has been observed that time series data that exhibit clear periodic patterns, where the number of observations per interval rise and fall in cycles usually over a year, tend to provide more valuable information for accurate forecasting compared to those without any identifiable periodicity. However, it has been found that the ARIMA modelling, which had been selected for the study, despite being a valuable technique for analysing surveillance data, has difficulties to accurately model diseases with low occurrence rates, such as meningococcal meningitis, for example. These findings highlight the need for a continued research and development of more advanced techniques to improve disease surveillance and forecasting.

A comparison between the aforementioned ARIMA method and LSTM models has been made in [37] on predicting an influenza-like illness and respiratory disease. The findings were striking, as the univariate ARIMA model was found to have a significantly higher error rate compared to the multivariate LSTM models. In fact, the error rate for ARIMA was found to be 4 to 10 times higher than that of the LSTM models. The study concluded that all the different types of LSTM prediction models involved had a higher accuracy rate than traditional ARIMA methods, highlighting the potential of machine learning techniques in time series analysis and forecasting.

2.2.2. LSTM Models for Pandemic Prediction

Due to their efficacy, Long Short-Term Memory Recurrent Neural Networks have been proposed in many projects. Given the fact that they take significantly less time to train, it is really convenient to use them for time series analysis. This has been proved by [14], where hospitalisations in French regions and in Belgium were forecast using LSTM approach and obtaining good and truthful results with low efforts. In this project, Google Trends has been used to retrieve the data by its web interface. Then it was divided in training set (seven regions), validation set (three regions) and testing set (four regions). LSTM-based models were experimented on the training dataset and produced as output one single number: the estimated hospitalisations seven days later. For the optimization, the Mean Squared Error (MSE) has been used as error metric and a forecast on the COVID-19 hospitalized cases has been produced, giving support to the government to adjust the measures and restrictions, to hospitals to prepare for the incoming flow of patients, and serving as a warning in case of detection of a second future peak.
In [25] it is demonstrated that LSTM can be utilized as an intelligent clinical decision support system in practical healthcare settings. To prove its efficacy, a vast clinical record dataset is used to validate the model. The results of the experiment demonstrate that the LSTM model surpasses various traditional and deep learning models in terms of prediction performance. Additionally, the study explores the influence of the time interval granularity on the prediction performance and establishes that a one-week granularity yields the best results for the given dataset. The findings suggest, again, that LSTM models can be a promising approach for developing intelligent clinical decisions.

2.2.3. COVID-19 Spread Prediction

Being COVID-19 a major issue for society in the recent years, a great deal of research projects has focused on forecasting the pandemic spread. Every epidemic of an infectious disease has a certain pattern, and these patterns are required to be discovered based on the dynamics of the outbreaks' propagation. At an early stage of SARS-CoV-2, Zhao et al. [45] have immediately started estimating the transmissibility of COVID-19 by the basic reproduction number, which estimates the ability of a new pathogen to spread and is defined as the average number of secondary transmissions from one infected person. This has given the opportunity to promptly understand the trend of the new pandemic and immediately start acting for its containment. The same method has been applied by numerous other researches, among which Shim et al. study on the transmission of

SARS-CoV-2 in South Korea [32]. The goal was to study the evolution of the reproduction number for COVID-19 in Korea using the Korean incidence curves for imported and local cases. The estimations of the reproduction number clearly show that the novel coronavirus was being transmitted continuously in Korea, and the case fatality rate seemed to be higher in males and older populations.

Regardless, any pandemic in a country or province typically happens at varying levels of magnitude throughout time, due to seasonal variations and the virus's ability to adapt. A study conducted in 2020 on the confirmed cases of different countries using the ARIMA model has proven that, indeed, while the confirmed cases' trends in China were going towards a stabilisation in the upcoming weeks, exactly like South Korea and Thailand, Iran and Italy still had very unstable trends [12]. Suitably, different studies have needed to be conducted in different countries using Deep Learning methodologies to help the governments and the public health authorities contain the epidemic.

In [23], LSTM networks have been used to predict the COVID-19 outbreak in Canada based on the past transmission data. To discover the best model that can predict future infections with the least amount of errors, after choosing the network's main parameters, multiple tests were undertaken. The patterns retrieved have then shown that the Canadian public health authorities had been capable to minimize the human exposure, obtaining positive results compared to other countries such as USA and Italy. While transmission rates in the USA were growing exponentially, they were following a linear pattern in Canada. It has also been discovered that provinces that adopted social distance policies prior to the pandemic had fewer confirmed cases than other provinces.

This is not the case of Italy though, where the excess mortality during the COVID-19 outbreak was studied in [30]. Despite the strict lockdown measures implemented soon after the start of the pandemic, it put a tremendous amount of strain on the healthcare system, and it was especially bad in Lombardy and a few other specific provinces in other northern regions, where hospitals were overburdened. General practices and outpatient activities were frequently suspended to stop the spread of the virus. Emergency and intensive care units concentrated almost entirely on COVID-19 patients, with admissions for other causes significantly reduced. Based on the database released by ISTAT, the authors of [30] applied a two-stage time-series model, comparing the risk for mortality during the outbreak and the one of the pre-outbreak. The results revealed a death rate spike of up to 400% in a few particular northern provinces. The mortality rate proved to be higher in men compared to women and substantially lower in people $<$60 years old.

It can be seen that strict lockdown measures, such as interregional travel restrictions, have probably helped to contain the infection's spread and the ensuing excess mortality in other areas, particularly in the South. The retrieved data was useful to explain how risks vary among locations and subgroups, as well as the contribution of local or national policies put in place at the time.

2.2.4. Lockdown Impact Estimations

Additionally, some studies used time series analysis in order to estimate the impact of lockdowns on reducing the number of cases. In [33], for instance, the quarantine measures adopted in Brazil by the president Bolsonaro were studied to evaluate their effectiveness. The data was collected from the Brazilian Ministry of Health website and used to conduct an interrupted time series (ITS) analysis to estimate the impact of social distance measures on reducing the number of COVID-19 cases and deaths in Brazil. It was proved that, in order to stop the spread of contagious diseases, social distancing is a well-established non-pharmaceutical strategy. As a result, social distance policies have shown to have a noticeable impact on the SARS-CoV-2 outbreak, according to studies utilizing the ITS design. According to Tobías [36], who examined how lockdown measures affected the SARS-CoV-2 pandemic in different countries, such Spain and Italy, the first lockdown had decreased incidence in both nations while maintaining favorable trends. The slope of the number of case occurrence shifted and turned negative after the second, harsher lockdown. After these studies it is then possible to conclude that social distancing and restrictions have been very useful to contain the pandemic and give a brake to the exponentially growing number of COVID-19 cases. The less the cases, the more hospitals can be able to tame the spread and avoid overcrowdings, hence, predicting the number of future hospitalisation cases could also be a way of helping the public health authorities.

2.2.5. Study of Pandemic EMS Events

Ever since the end of 2019, COVID-19 has caused overwhelming disruptions to healthcare systems around the globe. Traditional epidemiological models, such as the SIR model, do not give health system officials enough time to plan effectively. The emergency medical services (EMS) demand and emergency department visits increased significantly after the pandemic spread, resulting in a lack of pre-hospital patient assessment, medical resources, and personnel equipment. The seek of [41] is to first look at how the pandemic has affected emergency calls over time and how long it takes an ambulance to be assigned, sent,

and arrive on average and then design a predictive model to forecast the daily number of COVID-19 EMS calls. The study was conducted based on two datasets retrieved on the City of Austin Open Data Portal. In the modelling procedure, time series regression with the change point detection was used to forecast the daily frequency of pandemic EMS events, being the daily frequency of COVID-19 hospitalisation the regressor. The change point detection method was applied and then, to reduce the noise and improve the predictive power, the time series of daily hospitalisation was smoothed by an average of a period of seven days. Later on, autoregressive integrated moving average (ARIMA) models and binomial thinning have been applied after having divided the dataset in training (80%) and testing (20%). As a result, it was seen that since the start of the Covid-19 pandemic and the proclamation of the local state of emergency, the number of non-pandemic EMS incidents per day has sharply decreased. What's more intriguing though is that once the Covid-19 pandemic broke out, the percentage of non-pandemic dead calls increased. Some earlier conducted studies in the United States suggested that many people refused to be transported to hospitals out of fear of contagion. Another observation made was a significant prolongation of the EMS response time, indeed, according to local Austin news reports, the EMS department did not get enough funding to keep enough staff and ambulances on hand throughout the epidemic, which caused these delays. The study carried out in [41] is useful to open up a discussion amongst EMS organizations, government officials, and healthcare partners about how to modify EMS demand shifts and reduce response times during upcoming pandemics.

2.2.6. COVID-19 Patients Estimations

In general, models fitted to hospitalized patients are more trustworthy and accurate than models suited to cases that have been confirmed [19]. The initial objective of [27] is thus to give short-term and medium-term projections for the number of COVID-19 patients who will be admitted to hospitals during the second wave of infections in Italy, or after October 13, 2020. Hospitalisation patterns associated with COVID-19 paint a vivid picture of the overall strain on the national healthcare system. The data used in this project referred to the real- time number of COVID-19 hospitalisations of patients with mild symptoms and patients assigned to the intensive care unit in Italy from February 21, 2020, to October 13, 2020, extracted from the Italian Ministry of Health's website. According to the data, the number of COVID-19 patients admitted to hospitals for minor symptoms and the number of COVID-19 patients placed in the intensive care unit both achieved a first peak on April 4, 2020. After that, they had a declining trend until

the middle of August, before picking up speed once more from the end of September to the middle of October 2020. The author in [27] computed forecasts using four different statistical techniques and their combination, such as linear ARIMA models, ETS models, linear and nonlinear NNAR models and TBATS models. To compare the performances of the single and hybrid prediction models, MAE, MAPE, MASE, and RMSE were used as metrics. In conclusion, the most accurate model for both individuals hospitalized with minor symptoms and patients admitted to ICU between those used, ended up being neural network autoregression (NNAR).

2.3. Summary

The cited works are summarized in Table 2.1.

Project	Application	Involved Technologies
[3]	Infectious disease surveillance	ARIMA
[37]	Forecasting Influenza-like illness and respiratory disease	ARIMA, LSTM
[14]	Hospitalizations forecasting in France and Belgium	Google Trends, LSTM
[25]	Diseases prediction in South East China	LSTM
[45]	COVID-19 prediction in China	Serial Intervals
[32]	COVID-19 prediction in South Korea	Discretized Probability Distribution
[12]	COVID-19 prediction	ARIMA
[23]	COVID-19 prediction in Canada	LSTM
[30]	COVID-19 mortality prediction in Italy	Poisson Regression
[33]	COVID-19 lockdown effects in Brazil	Time Series Analysis
[36]	COVID-19 lockdown effects in Italy and Spain	Poisson Regression
[41]	COVID-19 EMS incidents	Change Point Detection
[27]	COVID-19 hospitalizations in italy	ARIMA, ETS, NNAR, TBATS

Table 2.1: Cited articles with relative application and used technologies' list.

The current COVID-19 pandemic has highlighted the importance of accurate forecasting systems in the healthcare sector. In this regard, the state of the art reveals a significant vacancy in the forecast of COVID-19 cases and hospitalizations in the Canary Islands. While predictions in other countries have shown the potential benefits of using time series forecasting methods to obtain data that could help in future decision making for

authorities and policy makers, the Canary Islands currently lack such a system. With this gap in mind, it is still necessary to employ accurate and efficient forecasting techniques to aid in the management and control of the pandemic. One such technique that has shown promising results is the LSTM model which, compared to other methods, has demonstrated higher accuracy and convincing results. Given the current situation in the Canary Islands, the implementation of LSTM-based forecasting models could be instrumental in predicting the spread of the virus, preparing hospitals for the expected influx of patients, and ultimately, in curbing the spread of the pandemic.

3 | Design and Planning

This chapter outlines the process of designing and planning a COVID-19 case forecasting system for the Canary Islands. To provide a foundation for this approach, it begins with an overview of Long Short-Term Memory Recurrent Neural Networks, exploring their architecture and various applications.

3.1. Deep Learning for Time Series

Deep Learning (DL) is an AI discipline where machines learn from experience using Neural Networks that can create multiple layers to represent data. Artificial Neural Networks (ANNs) are the simplest neural networks, consisting of input, hidden, and output layers of nodes that are connected and activated based on weights and thresholds [22]. There exist distinct types of Neural Networks, each employed for a different purpose and needed for Deep Learning Techniques, between which Recurrent Neural Networks (RNNs), mainly used for sequential input data and can learn from previous iterations during training by keeping a record of processed information.

3.2. Long Short-Term Memory RNNs

Since the late 1990s, several variants of RNNs algorithms have emerged, one of these being LSTM. To forecast time series data with LSTM, a regression network with sequence output is trained. It stores information from all prior steps in a network state, which is updated through time steps. Forecasting can be done through open or closed loop methods. The LSTM uses gated cells to selectively add, withdraw, or preserve information, determined by the importance weights assigned to the data. By adjusting the opening and closing of its gates, the LSTM learns which information is efficient in lowering the prediction error [26].

The 3 gates of the LSTM network are represented with the following equations:

$$I_t = \sigma(\omega_I[h_{t-1}, k_t] + b_I), \tag{3.1}$$

$$F_t = \sigma(\omega_F[h_{t-1}, k_t] + b_F), \tag{3.2}$$

$$O_t = \sigma(\omega_O[h_{t-1}, k_t] + b_O), \tag{3.3}$$

where I_t is the function of the input gate, F_t the function of the forget gate, O_t the function of the output gate, ω_x the coefficients of the neurons at gate x, $h_{\mathrm{t-1}}$ is the result from the previous time step (the hidden state), k_t the input to the current function at time-step t, and b_x the bias of neurons at gate x.
The input gate in 3.1 gives the information that needs to be stored in the cell state, 3.2 throws the information based on the forget gate activation output, and 3.3 combines the information from the cell state and the output of forget gate for generating the output [23].

3.3. Requirements Analysis

The state of the art review has highlighted the importance of accurate forecasting systems in the healthcare sector, particularly in the current COVID-19 pandemic scenario. The ability to accurately predict the spread of the virus can aid public health authorities in taking necessary measures to contain the spread and prepare hospitals for the expected influx of patients.

3.3.1. COVID-19 Prediction System for Canary Islands

At present, there is a lack of predictive modelling on the spread of COVID-19 in the Canary Islands. Given that the pandemic has not yet come to an end, such models could be highly beneficial in containing the virus and efficiently organizing the hospitals in the region. Time series prediction analysis and forecasting, as demonstrated by countries like China, Italy, and Canada, has proved to be an effective tool in this regard. By accurately predicting future hospitalisation rates, public health authorities in the Canary Islands would be able to better prepare healthcare facilities and ensure that they are equipped to handle the expected influx of patients. This would not only reduce the risk of being caught unprepared during a potential peak in cases, but also facilitate the process of returning to normalcy. By preventing high-risk situations from arising, the Canary Islands

could gain valuable insights into how to effectively respond to such crises in the future.

In conclusion, the Canary Islands are in the need of a prediction method that, as seen after its use in other countries, could aid in case of new COVID-19 peaks and hospitals overflow. Accordingly, in order to solve this issue, it will be necessary to adopt the same Deep Learning methods that have been applied in many other countries, such as LSTM, which has been proven to be the most effective, being able to overcome the vanishing gradient problem. In fact, it has been made clear that every epidemic of an infectious disease has a certain pattern, and these patterns are required to be discovered in order to evaluate the outbreak and develop effective intervention strategies.

3.3.2. Requirements

It will be possible to accomplish the aforementioned goals through various tasks.

Task 1: entails conducting a thorough research on Recurrent Neural Networks and Deep Learning techniques specifically focusing on temporal series prediction, particularly in LSTM models. This research will also dive into the various implemented technologies associated with these models. By exploring these cutting-edge technologies, the research team will gain an in-depth understanding of the latest advancements in the field, which will be crucial in developing a robust forecasting system for the Canary Islands.

Task 2: involves the comprehensive analysis of the available CSV files containing the datasets, which will be meticulously labeled to ensure accurate data description. This step is critical in generating a valid and reliable dataset that can be used for forecasting purposes. The team will work meticulously to ensure that the data is error-free and easy to interpret.

Task 3: the team will design the architecture of the forecasting system, incorporating the Long Short-Term Memory (LSTM) technique, which was identified as the most effective model in Task 1, on the previously labeled dataset. The LSTM model is particularly well-suited for time-series prediction tasks and is known for its ability to handle the complexities associated with this type of data.

Task 4: the team will design the architecture of the forecasting system, incorporating the Long Short-Term Memory (LSTM) technique, which was identified as the most effective model in Task 1, on the previously labeled dataset. The LSTM model is

particularly well-suited for time-series prediction tasks and is known for its ability to handle the complexities associated with this type of data.

Task 5: the results obtained during the simulations of the forecasting system will be evaluated using two commonly used criteria: the Root Mean Square Error (RMSE) and the Mean Absolute Percentage Error (MAPE). The team will then make any necessary adjustments to improve the performance of the system, ensuring that it provides the best results possible. By following this multi-step approach, the team will be able to develop a reliable and effective forecasting system that will benefit the healthcare system in the Canary Islands.

3.3.3. Experimental Environment

- **Hardware Resources:**
  - RHW-1: Personal laptop with Internet connection (Apple M1 Chip), RAM memory 16GB.
- **Software Resources:**
  - RSW-1: PyCharm CE
  - RSW-2: MathLab
  - RSW-3: Microsoft Excel

3.4. Environment Setup

After the study of the state of the art and the analysis and preparation of the available datasets, the forecasting system can finally be implemented. The datasets are univariate time series with only the recorded cases changing over time, recorded sequentially over equal time increments. This suggests the use of univariate time series models, which model and forecast variables using past values and an error term. The project implements various LSTM univariate models in Python to compare and select the best model for future data. PyCharm is the Python Integrated Development Environment (IDE) chosen to write the code that models it.

The first step to be followed for the use of this tool is the setup of the environment, that involves the check, and eventually update, of the current Python version and the

installation of the various packages together with that of a package manager. The Python version installed and used for this project is *3.8.9*, which is compatible with all the packages to be used.

3.4.1. Package Manager

Conda[1] is the package manager that is used to install the packages needed for the project. It is a popular cross-platform package and environment manager used for machine learning, data science, and scientific computing. It simplifies managing and deploying packages from the Anaconda[2] repository and Anaconda Cloud, and it has a large collection of free and open-source libraries and packages available for use.

The installation of Conda has been made through Miniconda[3], a small, bootstrap version of Anaconda that only includes Conda, Python, the packages they depend on, and a small number of other useful packages, including *pip*, *zlib* and a few others.

3.4.2. Needed Packages

TensorFlow: The main package needed for a Deep Learning application like this one is TensorFlow, an open-source library that uses data flow graphs to work with multi-dimensional arrays known as tensors [5].

Nonetheless, Conda was chosen for its benefits such as easier installation and automatic installation of compatible dependencies. Additionally, Conda's TensorFlow packages use *Intel MKL-DNN*, improving the performance of computationally intensive deep learning applications by more than 8x. The TensorFlow's version utilized in this project is *2.10.0*, which means that the performance improvement will be recorded thanks to the use of Conda (Figure 3.1)[4].

[1]https://medium.com/analytics-vidhya/why-conda-install-instead-of-pip-install-ba4c6826a0ae
[2]https://blog.hubspot.com/website/anaconda-python
[3]https://docs.conda.io/en/latest/miniconda.html
[4]https://www.anaconda.com/blog/tensorflow-in-anaconda

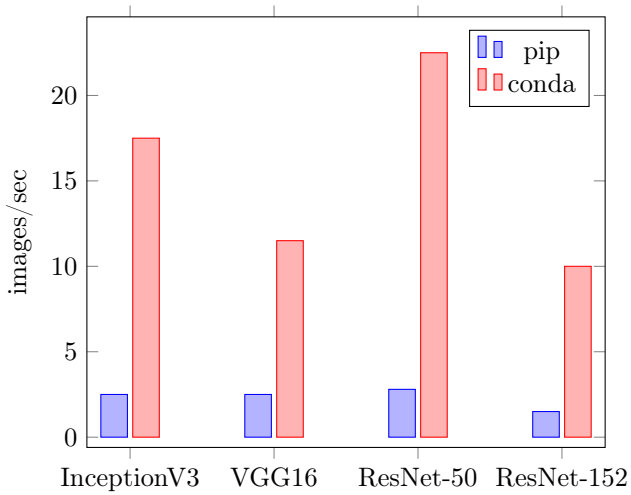


Figure 3.1: Chart on Tensorflow's Training Performance on Synthetic Data.

Keras: Running on top of TensorFlow, Keras[5] is a deep learning API that thanks to its simplicity, flexibility and scalability enables fast experimentation. Its core data structures are layers and models, between which the simplest and to-be-used one is the Sequential model, which is sufficient for a simple stack of layers, where each layer has precisely one input tensor and one output tensor. This model is imported in the project inside the *univariate_ models.py* program together with all the layers that will be needed. Table 3.1 describes in detail which are the layers that have been used with a short definition and shows which LSTM models take advantage of them in the program:

[5]https://keras.io/about/

Layer	Definition	LSTM models
Dense	A layer that is deeply connected with its preceding layer, meaning that the neurons of the layer are connected to every neuron of the preceding one. It is used to get the output in the format needed by the user, changing the dimensionality of the preceding layer.	Vanilla, Bidirectional, CNN, Stacked, Convolutional.
LSTM	This layer learns long-term relationships between time steps. It performs additive interactions, which might enhance gradient flow over lengthy periods during training.	Vanilla, Bidirectional, CNN, Stacked.
Bidirectional	This is a layer needed to connect two hidden layers of the opposite directions to the same output. This way the output layer gets the information from past and the future states simultaneously.	Bidirectional.
Flatten	Layer that flattens the input with no effect to the batch size.	CNN, Convolutional.
TimeDistributed	It is a wrapper, that helps in adding a layer to each temporal slice of an input.	CNN.
ConvLSTM2D	It is comparable to an LSTM layer, but with a convolutional input and recurrent transformations.	Convolutional.
Conv1D	This layer generates a convolution kernel that is twisted with the layer input over a single spatial (or temporal) dimension, in order to generate a tensor of outputs.	CNN.
MaxPooling1D	By picking the highest value over a spatial window of the size pool size, it downsized the input representation.	CNN.

Table 3.1: List of Keras layers needed in the LSTM models.

NumPy: NumPy[6] is a core scientific computing library for Python that provides a multidimensional array object, *ndarray*, and various routines for quick operations on arrays. It is preferred over standard Python sequences due to its fixed size, homogeneous data types, advanced mathematical operations, and compatibility with other Python packages.

Pandas: Pandas[7] is a Python module for working with labeled data, providing quick,

[6]https://numpy.org/doc/stable/user/whatisnumpy.html
[7]https://pandas.pydata.org/docs/getting_started/overview.html

adaptable, and expressive data structures. Its main features include simple handling of missing data, data alignment, size mutability, and flexible reshaping, subsetting, and merging of datasets. It is widely used in data analysis and aims to be the most robust and adaptable open-source data analysis tool available in any language.

4 | Analysis and Labelling of Data

To forecast future COVID-19 cases in the Canary Islands, daily hospital statistics were provided and analyzed through time series analysis. Data cleaning is necessary to ensure accuracy and efficiency.

4.1. Data Cleaning

The reliability of a dataset is indicated by its data quality, which refers to ensuring that data is fit for a specific project by using quality management techniques. Issues such as incomplete, inconsistent, and inaccurate data can compromise the data quality. Six dimensions are essential for high-quality datasets:

1. **Completeness:** the degree to which the necessary data is available for use;
2. **Uniqueness:** the degree to which the data is unique and cannot be mistaken for other entries;
3. **Consistency:** the degree to which the data is equal within and between datasets;
4. **Validity:** the degree to which the data is within defined requirements like format, type and range;
5. **Accuracy:** the degree to which the data represents the reality;
6. **Timeliness:** the degree to which the data is available at the time it is needed [15].

The data to be studied is recorded in different .xlsx files. One at a time, they are analyzed following the six data quality dimensions through MATLAB[1], a programming platform created specifically for engineers and scientists to analyse and design systems through the MATLAB language, a matrix-based language that allows for the most natural expressions of computational mathematics.

[1]https://it.mathworks.com/discovery/what-is-matlab.html

The MATLAB environment is prepared to work with the available data by loading the sheets of the *.xlsx* files one at a time through the *readtable()* function. Then, the first column of each is deleted as it only contains descriptions of the entries, which can be represented by the table name, while the remaining two columns are renamed as "Date" and "NumCases". At this point, the 'Date' column is transformed to *datetime* format after having fixed the months' names from Spanish to English.

4.2. Data Labeling

The datasets are now ready and saved in MATLAB workspaces for labelling. Data labelling involves adding informative labels to raw data, such as images, text files, or audio, to provide context for machine learning models. Common types of data labelling include computer vision, natural language processing, and audio processing.

Natural language processing is used to manually label dataset entries through MATLAB. The available datasets will be labeled in two groups based on their features:

- COVID-19 Cases Data: the collected data on COVID-19 cases of the Canary Islands, such as new positivities, deaths and discharges. The two available datasets, *Acumulado.xlsx* and *Diario.xlsx*, range from January 31st (or 20th in some daily cases), 2020, to March 29th, 2022, covering a total of 789 days (or 800) of pandemic. The data reported in the two datasets have two different formats:

    1. Accumulated Cases (Acumulado.xlsx): Data representing the total number of COVID-19 cases, deaths and discharged patients through time. For each of the three different datasets, *accumulatedCases.csv*, *accumulatedDeaths.csv* and *accumulatedDis.csv*, the same procedure is followed.

    2. Daily Cases (Diario.xlsx): Data representing daily COVID-19 cases, deaths and discharged patients. For each of the three different datasets, *cases.csv*, *deaths.csv* and *discharged.csv*, the same procedure is followed.

- Care Capacity Data: data on hospital capacity including incoming patients and

occupied beds. There are tree datasets, *Ingresos.xlsx*, *Pacientes Covid 19.xlsx*, and *Pacientes no Covid 19.xlsx*, covering a total of 928 days from March 23rd, 2020 to October 6th, 2022.

The different datasets are loaded on MATLAB and observed. The obtained results are reported in Table 4.1.

Data	Dataset	Missing Entries	Dots	To Double	Inconsistent	Seasonal
cases.csv	Diario.xlsx	Y	N	N	N	N
discharged.csv	Diario.xlsx	Y	Y	N	N	N
allCOVbeds.csv	Pacientes Covid 19.xlsx	N	Y	N	N	N
accumulatedCases.csv	Acumulado.xlsx	Y	N	N	N	N
accumulatedDis.csv	Acumulado.xlsx	Y	N	N	N	N
inCOVpatients.csv	Ingresos.xlsx	N	N	N	N	N
inTOTpatients.csv	Ingresos.xlsx	N	N	N	Y	Y
resPatients.csv	Pacientes no Covid 19.xlsx	N	Y	N	N	N
deaths.csv	Diario.xlsx	Y	N	N	N	N
COVnoRes.csv	Pacientes Covid 19.xlsx	N	Y	N	N	N
COVwithRes.csv	Pacientes Covid 19.xlsx	N	Y	N	N	N
allBeds.csv	Pacientes no Covid 19.xlsx	N	Y	Y	N	Y
noResPatients	Pacientes no Covid 19.xlsx	N	Y	N	N	Y
accumulatedDeaths.csv	Acumulado.xlsx	Y	N	N	N	N

Table 4.1: Results of the study of the datasets made through the labeling of the data.

Table 4.1 summarizes the datasets that contain missing entries, inconsistencies, string entries that require conversion to doubles, and special seasonalities that need to be addressed. In terms of data quality, various measures have been taken to solve the issues:

- Missing Entries: it is observed that the number of entries of the datasets is not equal to the number of days of the time period taken into consideration. In order to deal with this lack, the available table needs to be first transformed into a timetable.

Subsequently, using the MATLAB *retime()* function, the missing days are inserted using the value of the previous entry.

- Dots: in some cases, the 'Date' column needs to be fixed before being able to start working with the data. In fact, some of the month names are abbreviated with a dot, which doesn't allow to transform them to date-time format correctly. In order to solve the issue, the dot is replaced with an empty space character.

- To Double: the number of occupied beds is in string format and needs to be convert it in double with the *str2double()* function.

- Inconsistent: looking at the 'inCOVpatients' table, which holds the data about the incoming COVID-19 patients, it is easy to notice an inconsistency in the data. The COVID-19 patients of March 24th and 25th result to be equal to 1, which means that the incoming total patients on the same days should at least be 1. Dealing with the inconsistency from a data quality point of view, the best option is deleting the first three entries of the 'inTOTpatients' table.

- Seasonal: after plotting the data in a MATLAB graph, it is easy to conclude that, in some cases, there is a seasonality. In fact, every 6 or 7 days, there is a peak on the 3rd day and there are lows on the 1st and last days. This seasonality that recurs weekly can be attributed to the fact that some hospitals might be closed on the weekend or might not register the patients until the next Monday.

4.3. Data Preparation

Datasets are located in the *csv_files* directory and are retrieved using the *csv_read()* function from the Pandas library. They are then indexed by date and transformed to date-time format. The 'NumCases' column is used to create a univariate sequence for the LSTM model.

To prepare the sequence, a *split_sequence()* function is created in *data_handler.py* to split the observations into instances with *n_steps* input samples for the LSTM model. CNN and Convolutional LSTM models need *n_steps* to be an even number, since the

samples obtained by the *split_sequence()* function will later need to be reshaped, being split into two sub-samples of two time-steps each. Accordingly, for these two special cases, *n_steps* has been chosen equal to 4. The rest of the model types have been set to have, on the other hand, *n_steps = 3*, since a smaller number provides more samples for the LSTM input, which entails obtaining a more accurate result from the forecasting.

Let's consider the case where *n_steps = 3*: it means that the series is split into samples made up of three elements, that are then stored inside the array X. In the meanwhile, another array y containing the following values in the sequences of the samples is created and will be used as output for the prediction.

The chosen number of timesteps for each model is summarized in Table 4.2.

LSTM Model	**n_steps**
Vanilla LSTM	3
Bidirectional LSTM	3
Stacked LSTM	3
CNN LSTM	4
ConvLSTM	4

Table 4.2: Results of the study of the datasets made through the labeling of the data.

4.4. Data Splitting

The obtained sequence of samples will be split into three different datasets:

1. Training Data: Training set is used for Machine Learning algorithms to generate predictions or complete tasks. The accuracy of the model improves with more training data, hence using a high volume of data for training is preferred. In this project, 60% of the dataset will be used for training.

2. Validation Data: Validation set is used to evaluate model's performance and adjust its hyperparameters during training. It helps prevent overfitting and underfitting. In this project, 20% of the available data is reserved for validation.

3. Testing Data: A subset of data used to evaluate how well a final model fits the

training dataset, chosen as the last 20% portion of the dataset.

A function called *split_samples()* is used to divide the samples obtained from *univariate_models.py* into training, validation, and test sets. This function takes X and y sequences and splits them into the respective sets:

```
1 def split_samples(sequence1, sequence2):
2     train_length, val_length = define_lengths(sequence1)
3     X_train, y_train = sequence1[:train_length], sequence2[:train_length]
4     X_val, y_val = sequence1[train_length:train_length + val_length], sequence2[
      train_length:train_length + val_length]
5     X_test, y_test = sequence1[train_length + val_length:], sequence2[train_length +
      val_length:]
6
7     return X_train, y_train, X_val, y_val, X_test, y_test
```

Listing 4.1: Splitting the data into training

To define the lengths of the training, validation, and testing sets, the function *define_lengths()* is called. This function uses the percentages mentioned earlier to determine the dimensions of the training and validation sets, while the testing set's length is determined by the remaining portion of the input sequence.

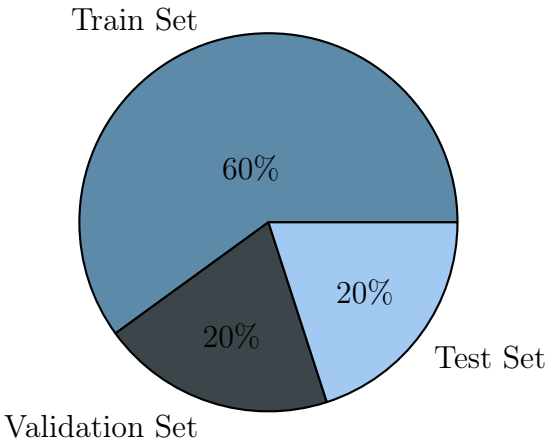


Figure 4.1: Training, Validation and Test Sets.

5 | Implementation

After the study of the state of the art and the analysis and preparation of the available datasets, the forecasting system can finally be implemented. Data and code are available on GitHub[1]. The number of recorded cases of every dataset is the only variable changing over time in all the available datasets, while the various observations are single and recorded sequentially over equal time increments (everyday). This suggests that the data should be treated as a *univariate time series*, using the suitable existing univariate models. Univariate time series models are a type of specifications in which one seeks to model and forecast certain variables using just information contained in their own past values and possible present and past values of an error term [7].

Various LSTM univariate models are implemented in order to provide a useful comparation. A walkthrough of all the steps followed to achieve the aforementioned system will be presented in this chapter.

5.1. LSTM Models

There exist multiple types of univariate LSTM models: Vanilla, Bidirectional, Stacked, CNN and Convolutional. In order to forecast the COVID data in the best way possible and have a comparison of the different models, they have all been implemented in the project and the user is given the possibility to choose which model to use at each run, after having chosen the dataset to study. All these models are defined inside the *univariate_models.py* program.

[1] https://github.com/ginevraiorioo/LSTM-time-series-forecasting-COVID-19

5.1.1. Vanilla LSTM

In a vanilla LSTM, the prediction is made using the output layer, which has only one hidden layer of LSTM units.

```
model = Sequential()
model.add(LSTM(units=32, activation='relu', input_shape=(n_steps, n_features)))
```

Listing 5.1: Vanilla LSTM implementation

The LSTM Keras layer is added to the model with the specification of the following hyperparameters:

- units, which is the dimension of the hidden state, also known as output. It also determines the number of parameters in the LSTM layer. Having more units is beneficial since it enables the network to retain more complex patterns due to the higher dimension of hidden states;
- activation, that indicates which activation function to use. ReLu has been chosen since, in comparison to the sigmoid and TanH, it is the most advanced, totally eliminating limitations like the vanishing gradient problem;
- input_shape, the shape of the input, or the amount of time steps and features the model requires as input for each sample.

5.1.2. Stacked LSTM

This model is made up by multiple hidden LSTM layers stacked one on top of the other. It is possible to use the 3D output from a hidden LSTM layer as input to a subsequent layer by using the *return sequences=True* option on the layer.

```
model = Sequential()
model.add(LSTM(units=32, activation='relu', return_sequences=True, input_shape=(n_steps,
    n_features)))
model.add(LSTM(units=32, activation='relu'))
```

Listing 5.2: Stacked LSTM implementation

5.1.3. Bidirectional LSTM

BiLSTM can produce a more meaningful output, combining LSTM layers from both the past and the present: one takes the input in a forward direction and the other in a back-

wards direction. A Bidirectional LSTM may be implemented by enclosing the first hidden layer in a wrapper layer called Bidirectional.

```
model = Sequential()
model.add(Bidirectional(LSTM(units=32, activation='relu'), input_shape=(n_steps,
    n_features)))
```

Listing 5.3: Bidirectional LSTM implementation

5.1.4. CNN LSTM

Convolutional Neural Network (CNN) layers are used in the CNN LSTM architecture to extract features from input data, assisted by LSTMs in sequence prediction. This particular kind of neural network was created to be used use with two-dimensional picture inputs, but when features are automatically extracted and learned from one-dimensional sequence data, such as univariate time series data, it can also be quite successful.

```
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=32, kernel_size=1, activation='relu'),
    input_shape=(None, n_steps, n_features)))
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(units=32, activation='relu'))
```

Listing 5.4: CNN LSTM implementation

By wrapping the CNN model in a *TimeDistributed()* wrapper, the same model can be used when separately reading in each sub-sequence of the data. The first layer is a convolutional layer, which reads over the subsequence and requires some hyperparameters to be specified:

- filters, the times the input sequence is read or interpreted;
- kernel_size, which is the number of time steps contained in each "read" operation of the input sequence.

The CNN also needs to be made of a max pooling layer, which reduces the filter maps to half their size and incorporates the most important features. These structures are then reduced to a single one-dimensional vector thanks to the Flatten layer before being utilized as the LSTM layer's input.

5.1.5. ConvLSTM

It's another type of CNN LSTM, where each LSTM unity can perform a convolutional reading of the input. This model may be used for univariate time series forecasting even though it was designed for reading two-dimensional spatial-temporal data.

```
model.add( ConvLSTM2D(filters=64, kernel_size=(1, 2), activation='relu', input_shape=(
    n_seq, 1, n_steps, n_features)))
model.add(Flatten())
```

Listing 5.5: ConvLSTM implementation

Regarding the quantity of filters and the size of the kernel, we may characterize the ConvLSTM as a single layer and since we are dealing with one-dimensional series, the kernel's row number is always fixed to 1. The model's output must then be flattened in order to be understood and a forecast to be made.

5.1.6. Output Layer

A Dense layer is finally added to every model, setting its units to 1 in order to have a one-dimension output.

```
model.add(Dense(units=1))
```

Listing 5.6: Output layer implementation

5.2. Building the Models

After the models and all their layers have been defined, the function *build_model()* of the *data_handler.py* is called.

```
def build_model(model, X_train, y_train, X_val, y_val, X_test, y_test):

    # compiling model
    model.compile(loss='mse', optimizer=Adam(learning_rate=0.001))

    # fitting model
    cp1 = ModelCheckpoint('model1/', save_best_only=True)
    history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=15,
    callbacks=[cp1], batch_size=16)

    # modeling the testing dataset and printing the results
    model1 = load_model('model1/')
    test_predictions = model1.predict(X_test, batch_size=16).flatten()
```

```
13
14      return test_predictions
```

Listing 5.7: Function that builds each model

This function executes three different actions on the model:

1. **Compile.** This Keras function is used to define the *optimizer*, *loss* and the *metrics* that will be used by the training function.

   Optimizers are techniques that modify the weights and learning rates of the neural network in order to minimize losses. The Adam algorithm adjusts the learning rate for each weight in the neural network using estimates of the first and second moments of the gradient. It is chosen as it is the most effective stochastic optimization, needing just first-order gradients when memory is insufficient [2].

   The *loss* function analyzes how effectively the neural network represents the training data by comparing the target and the output values. The goal is to reduce this difference during training. The chosen loss function, MSE, calculates the average of the squared difference between the target and the predicted outputs (5.1). The loss will grow dramatically if a projected value is much larger than or less than its target value since this loss function is particularly sensitive to outliers [42].

   $$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \widehat{y}^{(i)}\right)^2 \tag{5.1}$$

   At this stage of the study, no *metrics* were used to evaluate the output, but they will be added later on.

2. **Fit.** Function called to train the model by slicing the data into *batches* and repeatedly iterating over the entire dataset for a given number of *epochs*. It determines how effectively a machine learning model predicts data that is similar to that on which it was trained. Its first three arguments are the input and target training datasets and the validation data previously defined as validation datasets, that will be used to check the performance of the model after each epoch of training.

   The number of *epochs* is used to determine how many times the input data will be go through the whole training process, which should be an optimal number in order to avoid overfitting or underfitting.

The *callbacks* are a collection of operations used to receive a glimpse of the model's internal states and statistics. The *ModelCheckpoint()* callback was used in order to save the model in a checkpoint file *cp1* at the end of every epoch if it's the best seen so far. The best model will be then used for prediction.

3. **Predict.** On the basis of the trained model, which is taken as input, this function predicts the labels of the data values. In order to map and forecast the labels for the test data, it works on top of the training model and using the previously learnt labels.

Finally, the test predictions that have just been made are returned by the *build_model()* function.

5.3. Plotting the Predictions

The last action performed by every model is calling the *plot_results()* function of the *data_handler.py*.

```
1  def plot_results(y_test, test_predictions, test_dates, title):
2      # plotting the testing dataset in comparison with the actual dataset
3      test_results = pd.DataFrame(data={'Test Predictions': test_predictions, 'Actuals':
       y_test})
4      plt.plot(test_dates, test_results['Test Predictions'], label='Test Predictions')
5      plt.plot(test_dates, test_results['Actuals'], label='Actual Values')
6      plt.xlabel('Dates')
7      plt.ylabel('Cases')
8      plt.legend()
9      plt.title(title)
10     plt.show(block=True)
```

Listing 5.8: Function that plots the prediction results

This function takes as input the target testing dataset, y_test, the test predictions returned by the *build_model()* function, which will represent the y-axis, the title of the to-be-built graph and the dates for which the testing dataset has been described, that will be displayed on the x-axis and are retrieved inside each model. The function uses *Pandas DataFrame* structure to store and label the previously made predictions and the actual target values. The structure is then used to plot both these values on the y-axis of a graph, being the dates of the testing dataset chosen as x-axis. The legend for the graph and the title will also be shown.

5.4. MAIN Function

The main function is called inside the *init.py* script. When the program is run, the list of all *csv* files containing the datasets to be studied is shown to the user in a numbered list, printed on the console by a for loop inside the *csv_files* directory.

```
# asking the user to choose the dataset to study
print("\n\n Which csv file would you like to open?\n")
i = 1
for file in os.listdir(currDir + '/csv_files'):
  if file == '.DS_Store':
    continue
  print(str(i) + ' - ' + file)
  i = i+1

# asking for an input
print("\nInsert number: ")
file_num = input()
```

Listing 5.9: Main function

The user is then required to choose one file inputting its corresponding number of the numbered list. The check on the correctness of the input, which can only be an integer between 1 and 14, is then made by:

```
    # checking for wrong inputs
    if not file_num.isdigit():
        file_num = 0

    while not (1 <= int(file_num) <= 14):
        print("Invalid number! Try again: ")
        file_num = input()

        if not file_num.isdigit():
            file_num = 0
```

Listing 5.10: Checking for wrong inputs

Once the file to be opened has been chosen, it is loaded inside the variable *df*, while its name is loaded inside *dataset_name*, by calling the function *switch_file()* of the *data_handler.py*. Based on the number inputted by the user, this function reads the correct csv file and returns it together with its defined title.

The *init.py* continues its execution by asking the user which LSTM univariate model to use and presenting a numbered list of the implemented models: Vanilla LSTM, Stacked LSTM, Bidirectional LSTM, CNN LSTM and ConvLSTM. The correctness of the input,

which is required to be an integer between 1 and 5, is checked again. Ultimately, the function *switch_model()* of the *data_handler.py* is called inside the main function. Based on the last number inputted by the user, it calls the corresponding model function of the *univariate_models.py*, with the file and dataset name as parameters. This function will execute the previously mentioned data preparation, apply the correct LSTM model to the dataset and train it, obtaining the required prediction of the testing dataset, which will be shown in a plot together with the actual testing dataset values.

5.5. Simulation Analysis and Hyperparameter Tuning

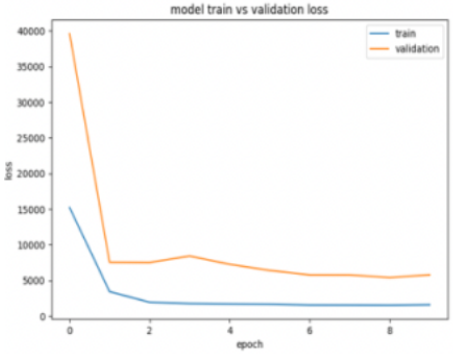
To ensure the accuracy of the LSTM models, simulation and hyperparameter adjustments are crucial. This involves evaluating the performance of the models based on training and validation losses. The hyperparameters that may need adjusting include the learning rate, batch size, number of epochs, hidden layers, and timesteps. Techniques such as plotting the losses on a graph can be used to monitor progress and identify overfitting or underfitting. Careful monitoring of the training and validation losses[2] is critical to develop accurate LSTM models.

5.5.1. Loss and Validation Loss

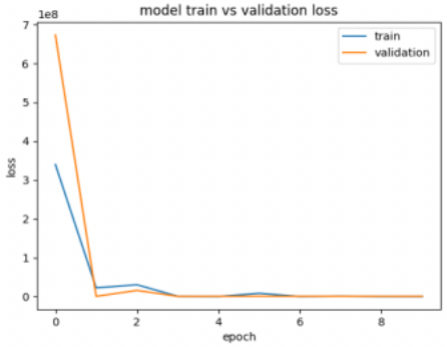
For LSTM models, accuracy is not the best metric for evaluating performance as it is typically used for classification problems. Instead, the mean squared error (MSE) is used to evaluate the performance of the model and adjust the hyperparameters. During training, the goal is to minimize the loss function by adjusting the model's parameters in each iteration through backpropagation. Validation loss is calculated by comparing the predicted values with the actual values in the validation dataset, using the same loss function as during training. The aim is to minimize both training and validation loss to obtain a model that performs well on unseen data. To track both values during training, a *plot_loss()* function is introduced into *data_handler.py*. Taking as input the fitting history, this function plots on a graph both the values of loss and val_loss resulting after each epoch and is helpful to detect if the model is overfitting, underfitting or if it's correctly running.

[2]https://www.baeldung.com/cs/training-validation-loss-deep-learning

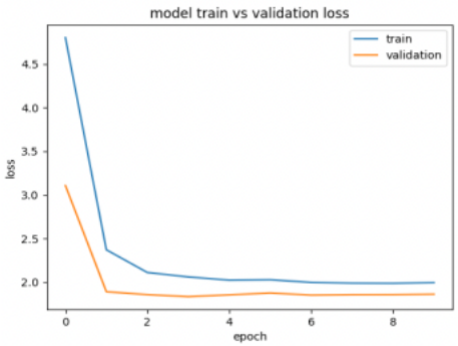
Having a first look at the resulting graphs of different type of data (Figure 5.1), such as daily COVID-19 cases (Figure 5.1a), accumulated COVID-19 cases (Figure 5.1b) or daily deaths (Figure 5.1c), on a Vanilla LSTM model, it is visible that the results are very different. Specifically, the *cases.csv* model is underfitting, having a higher validation loss compared to its training loss, the other two appear to have a correct behaviour, but *accumulatedCases.csv* has very high losses values.



(a) Daily Cases



(b) Accumulated Cases



(c) Deaths

Figure 5.1: Loss and Validation Loss plots of different datasets.

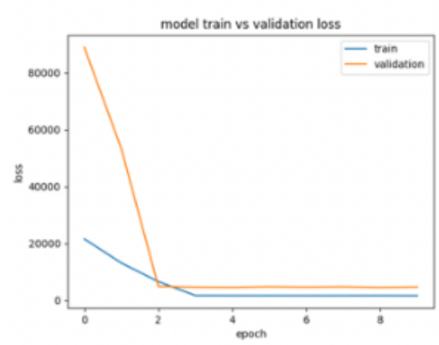
The experiments will be conducted on the *cases.csv* file, using a Vanilla LSTM model. At the end of the optimization process, a check on all the different combinations of models and data types will be done in order to make sure the results are all accurately optimized.

5.5.2. Model Optimization

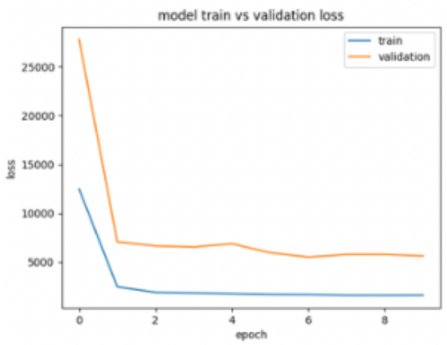
Optimization in machine learning involves finding the set of model parameters or weights that minimize a given loss function. This can be achieved using various approaches such as random and grid searches, evolutionary optimization, or Bayesian optimization [35]. In this project, random search will be utilized to find the best parameters that result in optimal performance of the LSTM model on the training data. The optimization process will consider the number of timesteps, hyperparameters (units per layer, batch size, learning rate and number of epochs), and activation functions.

1. **Number of Timesteps**

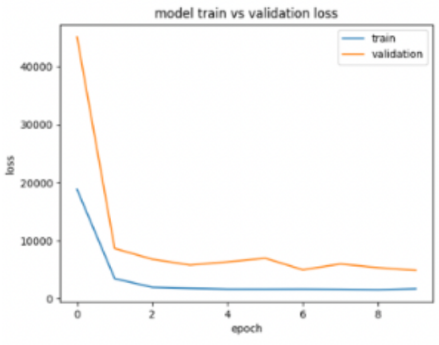
   Choosing the optimal number of timesteps for LSTM models requires a balance between capturing sufficient historical context and avoiding overfitting or computational complexity. Increasing the number of timesteps can help capture longer-term dependencies in the data, but it can also increase computational complexity and training time. Conversely, decreasing the number of timesteps can reduce computational complexity but may result in lower model performance. The frequency of data should be considered when choosing the number of timesteps. Experimenting with different values and evaluating the model's performance for each value is important [8]. This experiment will be conducted on the Vanilla LSTM and the results applied to the other models.



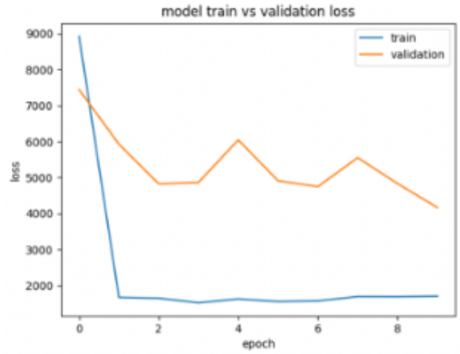
(a) 1 timestep



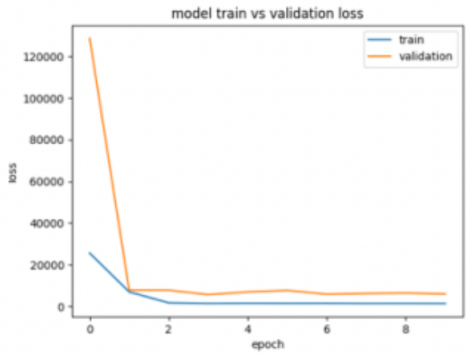
(b) 2 timesteps



(c) 3 timesteps



(d) 4 timesteps



(e) 5 timesteps

Figure 5.2: Loss and Validation Loss plots for different timesteps values.

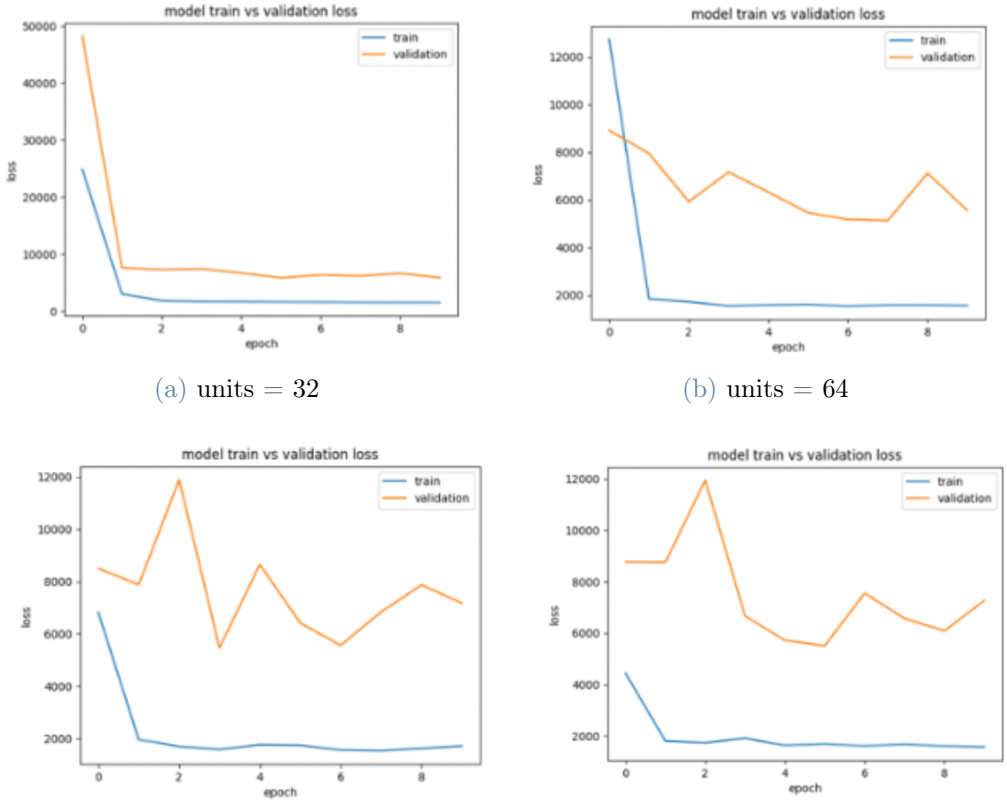
Observing the behavior of the obtained graphs (Figure 5.2), it is visible that the 3 timesteps case (Figure 5.2c) is the best option in this case. In fact, in its graph, both the losses are gradually decreasing and have the smallest values. The validation loss is still bigger than the training loss, resulting in an underfitting that will be considered later on.

2. **Hyperparameters**

Hyperparameter tuning is the process of finding the optimal combination of hyperparameters for a given machine learning algorithm. Hyperparameters are parameters that are set by the user before training the model, and they can have a significant impact on its performance [4].

- Number of hidden units

  Increasing LSTM layer's hidden units can enhance model's ability to learn complex patterns but may lead to overfitting and slow down training. Gradually increasing the hidden units (e.g., 32, 64, 128, 256) is a common approach to compare performance, but overfitting risks may increase with higher numbers [6].



(a) units = 32 (b) units = 64

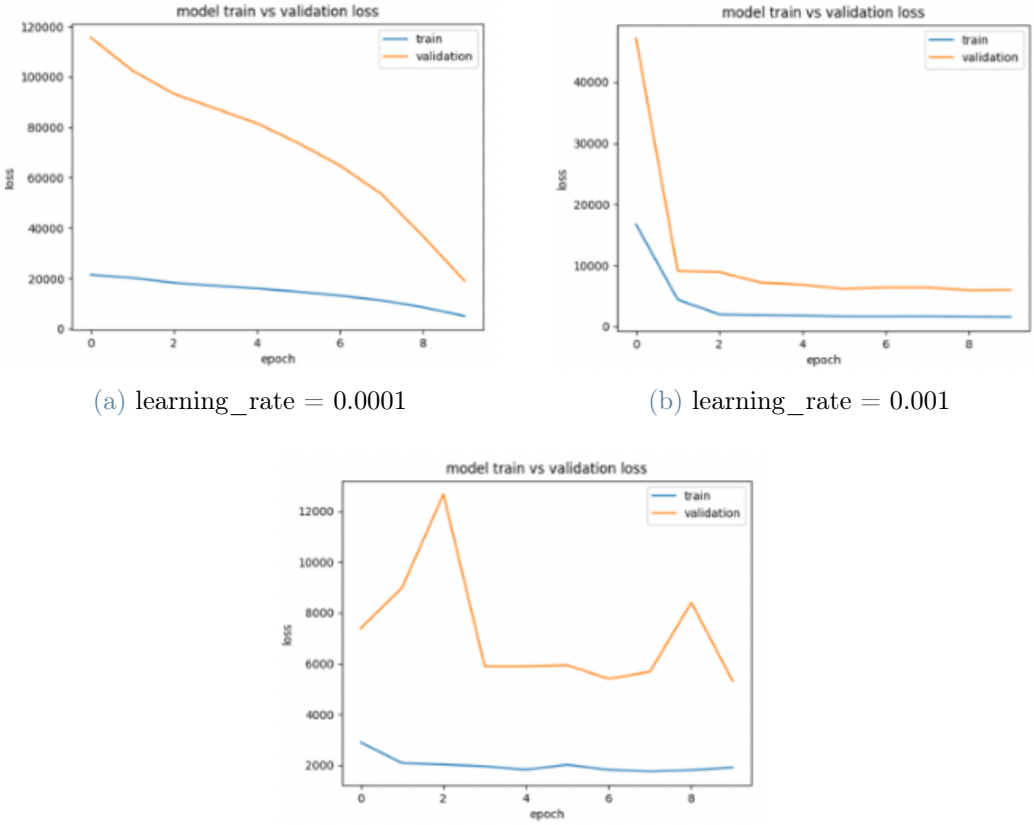
(c) units = 128 (d) units = 256

Figure 5.3: Loss and Validation Loss plots for different hidden units values.

The results show that increasing the number of hidden units the results keep getting worse (Figure 5.3), meaning that the optimal number of units that can be used is 32 (Figure 5.3a).

- Learning Rate

The learning rate determines the step size of weight adjustments during training. A higher learning rate leads to faster convergence but can cause instability, while a lower rate can result in slow convergence. The project starts with a small learning rate and increases it to track the performance, but if the loss increases, it may indicate a high learning rate [24].



(a) learning_rate = 0.0001

(b) learning_rate = 0.001

(c) learning_rate = 0.01

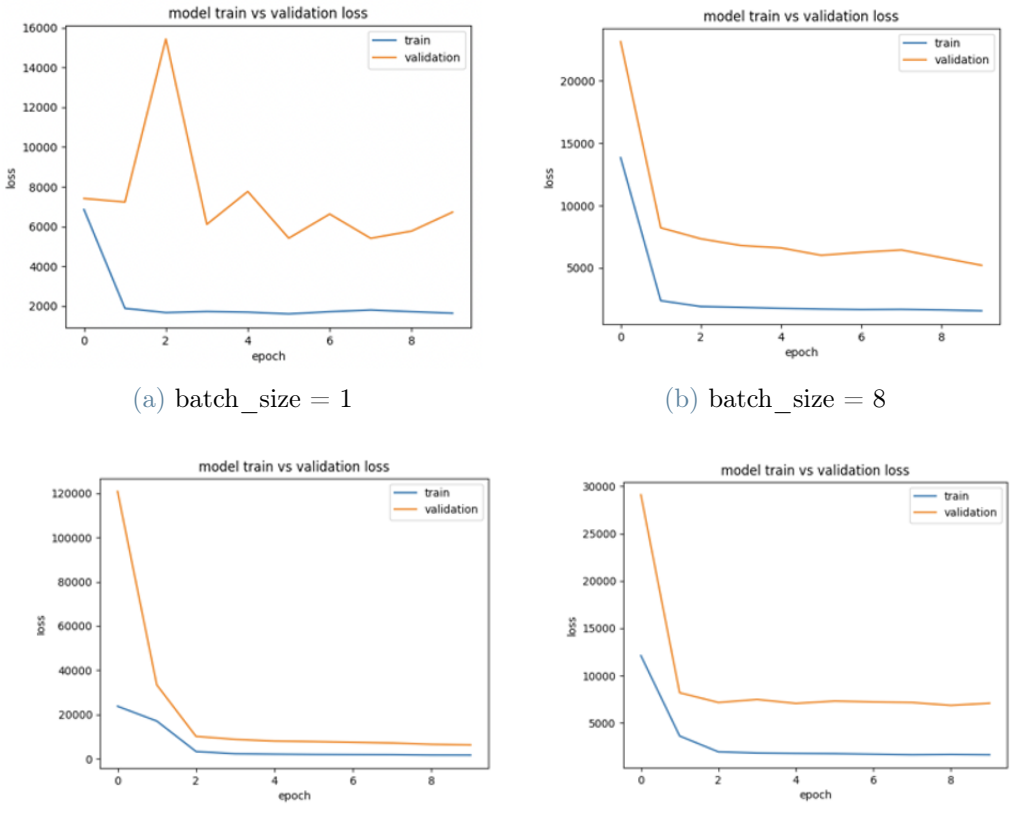
Figure 5.4: Loss and Validation Loss plots for different learning rate values.

The optimal loss behavior, in this case, is given by a learning rate of 0.001 (Figure 5.4), which is consequently chosen as a parameter for the Adam opti-

mizer.

- Batch Size

  Batch size refers to the number of samples processed during each training iteration. Larger batch sizes can result in faster convergence but may require more memory and reduce the model's ability to generalize. The optimal batch size may depend on the hardware used for training. GPUs are often optimized for larger batch sizes, while CPUs may perform better with smaller batch sizes [20].



(a) batch_size = 1 (b) batch_size = 8

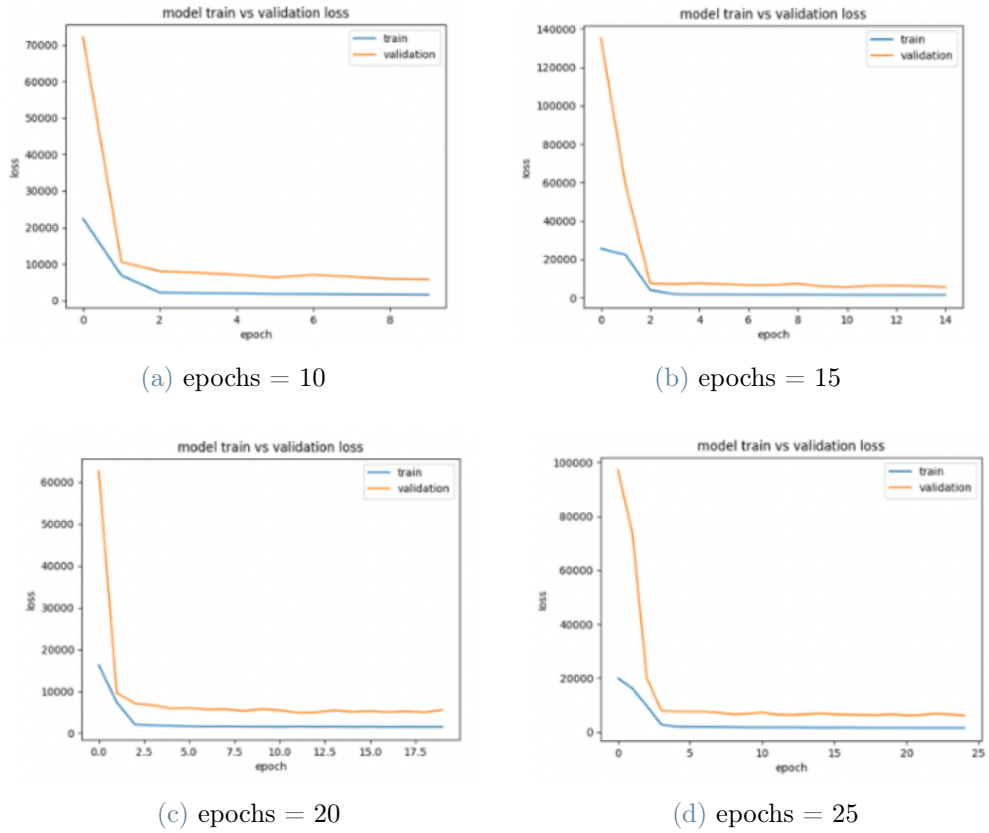
(c) batch_size = 16 (d) batch_size = 32

Figure 5.5: Loss and Validation Loss plots for different batch size values.

It is visible that in this case choosing a batch size of 16 will reduce the gap between the two losses and consequently reduce the underfitting problem (Figure 5.5c).

- Number of Epochs

  Epochs are complete cycles of the training algorithm through the entire dataset. Starting with a small number of epochs (e.g., 10), they are gradually increased to improve performance. A larger number of epochs can lead to overfitting and slower training, so the trade-off between performance and training time must be considered [29].



(a) epochs = 10 (b) epochs = 15

(c) epochs = 20 (d) epochs = 25

Figure 5.6: Loss and Validation Loss plots for different numbers of epochs.

The number of epochs that best fits the model is 15 (Figure 5.6b).

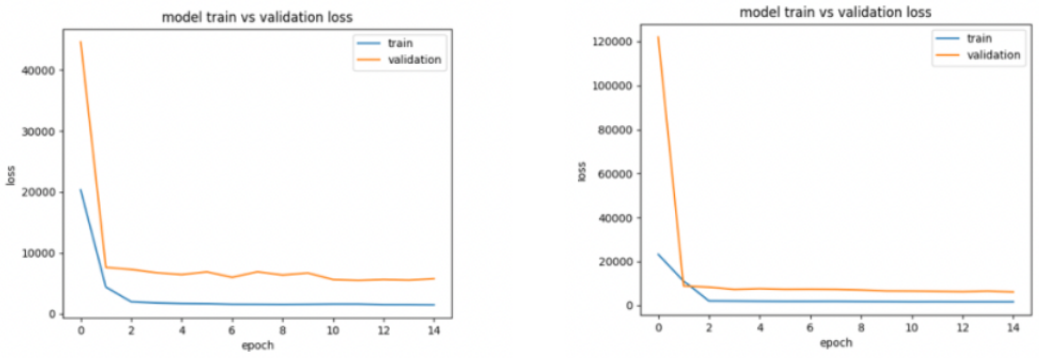
3. **Activation Functions**

There are several types of activation functions that are commonly used in neu-

ral networks, including:

- Sigmoid: it is a commonly used activation function that maps any input value to a value between 0 and 1. It is often used in the output layer of binary classification problems, where the output represents the probability of belonging to the positive class.
- Linear: it is used for regression problems where the output is a continuous value. It is also used in the output layer of a neural network to scale the output to a specific range.
- ReLU (Rectified Linear Unit): it maps any input value to 0 if it is negative, or the input value itself if it is positive. ReLU is the most commonly used activation function in deep neural networks because it is computationally efficient and can help to mitigate the vanishing gradient problem.
- Tanh (Hyperbolic Tangent): the tanh function is similar to the sigmoid function, but it maps any input value to a value between -1 and 1. It is often used in the hidden layers of neural networks.
- Softmax: the softmax function is a generalization of the sigmoid function that maps any input vector to a probability distribution over multiple output classes. It is commonly used in the output layer of multi-class classification problems.
- Leaky ReLU: it is similar to the ReLU function, but it introduces a small positive slope for negative input values. This can help to address the dying ReLU problem, where some nodes in the network become inactive and stop learning during training.

The choice of activation function can have a significant impact on the performance of the neural network, and it is important to choose an appropriate function for the specific problem being addressed [31]. Given that this problem is addressing a regression problem, the only activation functions that can be used are ReLU and Linear:



(a) linear (b) relu

Figure 5.7: Loss and Validation Loss plots for different numbers of epochs.

The results show that the activation function that best fits the model is ReLu (Figure 5.7b).

5.5.3. Overfitting and Underfitting

Overfitting and underfitting are common issues in machine learning that can harm model performance. Overfitting happens when a model is too complex and fits the training data too closely, resulting in poor performance on new data. Underfitting occurs when the model is too simple and cannot capture underlying patterns in the data, leading to poor performance on both training and new data. Detecting these problems is vital to evaluating model performance. One way to detect overfitting is by comparing the learning curves of a model on the training and validation sets.

If the training error is significantly lower than the validation error, then the model is overfitting (Figure 5.8).

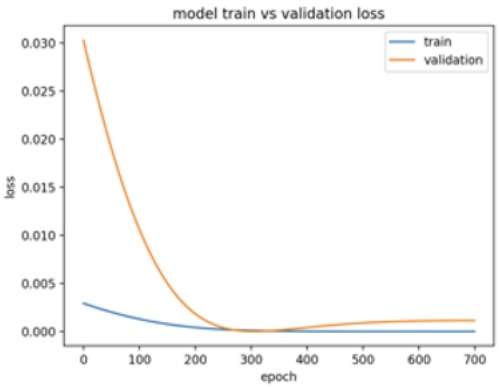
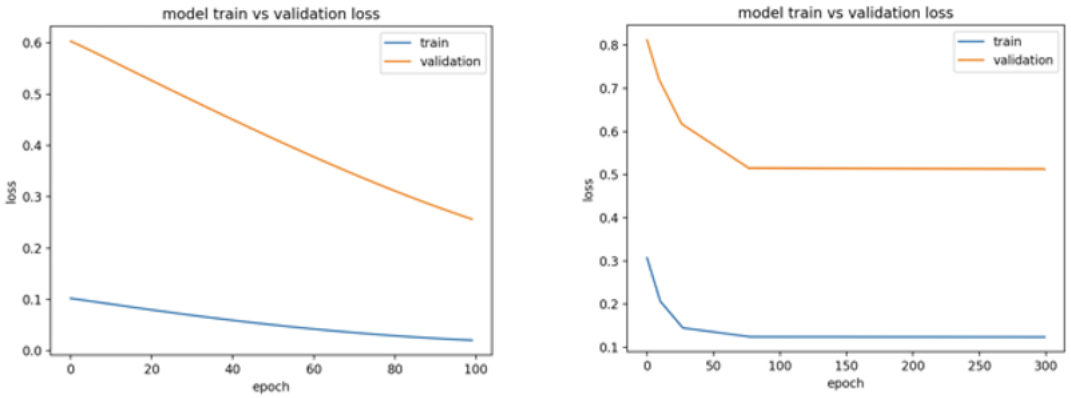


Figure 5.8: Overfitting Model.

To solve overfitting, several techniques can be used such as regularization, dropout, and early stopping. Regularization adds a penalty term to the loss function during training, discouraging the model from having large weights and making it more generalizable. Dropout randomly drops out some neurons during training, forcing the model to learn more robust features. Early stopping stops the training process when the model's performance on a validation set starts to degrade, preventing it from overfitting the training data.

On the other hand, underfitting can be detected if both curves converge to a high error rate, it suggests that the model is underfitting (Figure 5.9).



(a) (b)

Figure 5.9: Underfitting Model.

To solve underfitting, increasing the complexity of the model can be helpful, for example, by increasing the number of hidden units or adding more layers. Additionally, increasing the training data can help the model learn more complex patterns in the data. Hyperpa-

rameter tuning can also be used to find the optimal combination of hyperparameters for the model to achieve better performance.

Obtaining a good fit for a model is a critical task in machine learning, and it depends on several factors, including the quality and quantity of the data, the model's architecture, and the hyperparameters selected. To achieve a good fit, the model's training loss should be low, indicating that it is accurately predicting the training data. However, the validation loss should also be low, indicating that the model can generalize well to new data. A small gap between the training and validation loss is often a good sign of a well-fit model [21] (Figure 5.10).

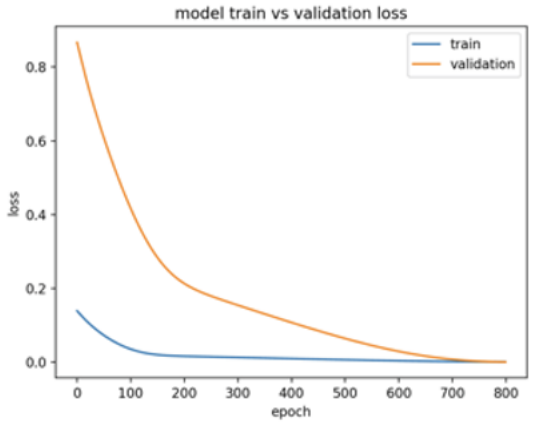


Figure 5.10: Well Fitted Model.

5.5.4. Results

For each *csv* file of the dataset, all the different LSTM methods are applied in order to have a look on the behaviour of the training and validation losses of the models, which are represented in Table 5.1.

Dataset	Underfitting/Overfitting	Comment
cases.csv	Underfitting	Models and datasets are too simple
discharged.csv	Underfitting	Models and datasets are too simple
allCOVbeds.csv	No	The data has a weekly pattern that helps the model's fitting
accumulatedCases.csv	Underfitting	The accumulated data doesn't present any pattern
accumulatedDis.csv	Underfitting	The accumulated data doesn't present any pattern
inCOVpatients.csv	Overfitting	Regularization needed
inTOTpatients.csv	Overfitting	Regularization needed
resPatients.csv	No	The data has a weekly pattern that helps the model's fitting
deaths.csv	Overfitting	Regularization needed
COVnoRes.csv	Overfitting	Regularization needed
COVwithRes.csv	No	The data has a weekly pattern that helps the model's fitting
allBeds.csv	No	The data has a weekly pattern that helps the model's fitting
noResPatients	No	The data has a weekly pattern that helps the model's fitting
accumulatedDeaths.csv	Underfitting	The accumulated data doesn't present any pattern

Table 5.1: Evaluation of models through training and validation losses.

As a result, it is visible that almost all the different LSTM models are underfitting, resulting in very high validation and training losses, meaning that a larger and more complex training dataset would be needed. Another solution would be adding more layers to the models, increasing their complexity. As a matter of fact, the datasets representing COVID data of the Canary Islands hospitals that have been used don't have a large number of entries, due to the fact that the number of cases has only been reported daily from January 2020 to March 2022. Moreover, it is clear that the accumulated cases type of datasets results in the worst performance, having really high losses. The reason of this behavior is that it is not possible to capture the underlying patterns and trends in the data, in fact the daily changes or fluctuations in the data is lost.

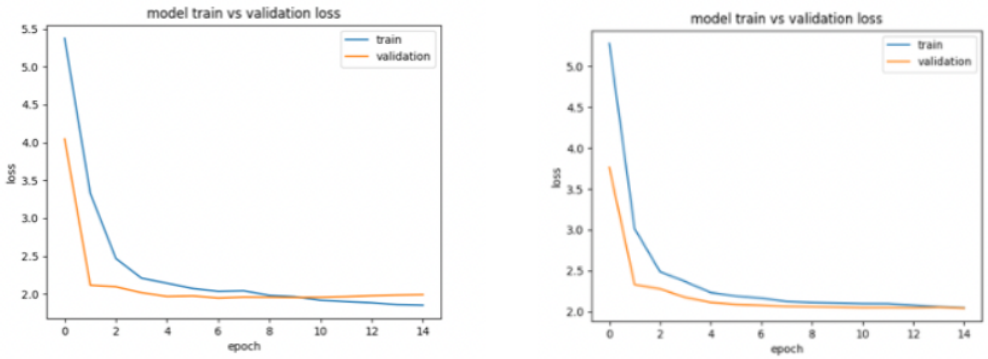
In the *deaths.csv* instead, some models are overfitting since the dataset contains many zeros and small values, which can create noise in the data that the model can learn to fit

too closely, resulting in poor generalization to new data. To address this issue, regularization techniques such as L1 and L2 regularization can be used:

```
1    model.add(
2        ConvLSTM2D(filters=64, kernel_size=(1, 2), activation='relu', input_shape=(n_seq,
     1, n_steps, n_features), kernel_regularizer=regularizers.L1L2(l1=0.02, l2=0.05)))
```

Listing 5.11: Adding regularization

Adding them to the convolutional layer, for example, definitely improves the model performance:



(a) no regularization (b) with regularization

The regularization is also applied to the CNN and Bidirectional models in order to further improve the results of their fitting.

6 | Performance Analysis

This chapter aims to introduce two new evaluation metrics, MAPE and RMSE, for evaluating the performance of the LSTM COVID-19 forecasting models, in addition to the commonly used MSE. MAPE is useful for data with significant value differences, while RMSE estimates the spread of the error distribution.

6.1. LSTM Models Evaluation Methods

There are several evaluation methods that can be used to assess the performance of LSTM models:

- Mean Squared Error (MSE): it is a commonly used evaluation metric for LSTM models that measures the average squared differences between the predicted and actual values. A lower value of MSE indicates better performance of the model.
- Root Mean Squared Error (RMSE): it is the square root of MSE and provides an estimate of the standard deviation of the differences between the predicted and actual values.
- Mean Absolute Error (MAE): it measures the average absolute differences between the predicted and actual values. It provides a measure of the magnitude of the errors, but it does not take into account their direction.
- R-squared (R2): it is a measure of the proportion of variance in the dependent variable that is explained by the independent variables. It ranges from 0 to 1, with higher values indicating a better fit of the model to the data.
- Mean Absolute Percentage Error (MAPE): it measures the average percentage differences between the predicted and actual values. It is commonly used when the magnitudes of the predicted and actual values are significantly different.
- Symmetric Mean Absolute Percentage Error (SMAPE): it is similar to MAPE but takes into account the direction of the errors. It is calculated as the average of the

absolute percentage differences between the predicted and actual values, divided by the sum of the predicted and actual values.

The choice of the evaluation method depends on the specific requirements of the forecasting task and the nature of the data. It is often recommended to use multiple metrics to evaluate the performance of LSTM models and compare their results [10]. So far, the model has been evaluated through MSE in the experimentation phase. Through the evaluation, hyperparameters have been fixed and changed with the goal of obtaining the best results possible.

Two additional evaluation methods, MAPE and RMSE, will now be introduced, and a study on the model will be conducted comparing the different metrics. By using these evaluation methods, the aim is to provide a more comprehensive analysis of the performance of the LSTM models. Ways to modify and improve the models will also be explored, based on the obtained results.

6.1.1. Root Mean Square Error (RSME)

The Root Mean Square Error is a widely used metric for evaluating the accuracy of predictive models, particularly in the field of time series analysis. It measures the differences between the predicted values and the actual values, by calculating the square root of the average squared differences between the two.

$$RMSE = \sqrt{\overline{(f-o)^2}} \tag{6.1}$$

RMSE is an important metric for evaluating the accuracy of predictive models, as it measures the overall deviation of predicted values from actual values. It is expressed in the same units as the target variable and is similar to standard deviation. It is particularly useful for time series analysis as it considers the order of data points. However, RMSE can be sensitive to outliers and does not indicate the direction of errors [9]. Outliers have been removed from the datasets to avoid issues with this metric.

RMSE has several advantages, including its simplicity, computational efficiency, and ease of differentiation. Additionally, because of the square root, it does not penalize large errors as much as MSE. However, it is sensitive to outliers and can be influenced by the

size of the data. As the size of the test sample grows, RMSE can also increase, which can be problematic when comparing results across different test samples.

6.1.2. Mean Absolute Percentage Error (MAPE)

MAPE stands for Mean Absolute Percentage Error. It is a commonly used evaluation metric for forecasting models that measures the average percentage differences between the predicted and actual values of a variable. It is calculated as the average of the absolute differences between the predicted and actual values, divided by the actual values, multiplied by 100 [11].

$$MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \widehat{y_i}}{y_i} \right| \tag{6.2}$$

MAPE provides a measure of the accuracy of the forecasting model, considering the magnitude of the actual values. It is useful when the data has significant variations in magnitude, such as when dealing with data on different scales or units. A lower value of it indicates better performance of the forecasting model. However, it has some limitations, such as being sensitive to extreme values and outliers in the data. In this case, a check for outliers has already been made, and, if present, they have been taken out from the datasets.

MAPE is a simple and easy-to-understand evaluation method that standardizes all errors on a common scale and expresses them in terms of percentages. However, its bias towards underestimating errors and its inability to handle zero values in the denominator are limitations [1].

6.2. Tests

In order to conduct the evaluation, the Root Mean Squared Error and the Mean Absolute Percentage Error metrics are used to compile the model. Being a loss, the RMSE needs to be first imported from the Keras' library in order to be used as a metric. Moreover, two more graphs are now printed during the execution: one representing the train and validation RMSE and one for the train and validation MAPE. These graphs will help out for the

detection of overfitting and underfitting models and for the adjustment of the hyperparameters. The functions that are called that build the graphs also need to be added to the code. They are identical to the already existing *plot_loss()* function that, taking as input the fitting history, plots on a graph both the values of loss and val_loss, except for the fact that one takes the RMSE and the other the MAPE values from the compiling history.

The Python program tests all implemented LSTM models on different datasets, and a study is conducted on the behavior of RMSE and MAPE metrics during training and validation phases. Results with decreasing metrics, low values, and slightly higher validation values are considered good.

One run using Vanilla LSTM is sufficient for the study since the results obtained on different models for a specific dataset are similar. However, some cases result in high metric values due to the small number of entries in the dataset covering COVID data from January 2020 to March 2022. Adding more layers to LSTM cells or input features would be a possible solution. Moreover, in a few cases the MAPE seems to grow instead of decreasing. The reason of this behavior is that MAPE is not a good metric when in the dataset compare one or more zeros. Observing its formula, it is visible that a zero in the data would end up at the denominator of a division, resulting in the Mean Absolute Percentage Error not to work properly.

6.2.1. Final Evaluation

After the compiling phase, a final evaluation on the model is made using the train and test datasets. The *evaluate()* function is called, and the results are saved in two variables, called *train_metrics* and *test_metrics*. Subsequently, train and test losses, RMSE and MAPE are printed on the screen in order to be collected and studied.

```
1 # final evaluation of the model
2     train_metrics = model.evaluate(X_train, y_train, return_dict=True)
3     test_metrics = model.evaluate(X_test, y_test, return_dict=True)
4 
5     print('\n\nTrain loss: {:.2f}, Test loss: {:.2f}'.format(train_metrics['loss'],
      test_metrics['loss']))
6 
7     print('Train RMSE: {:.2f}, Test RMSE: {:.2f}'.format(train_metrics['rmse'],
      est_metrics['rmse']))
8 
```

```
9     print('Train MAPE: {:.2f}, Test MAPE: {:.2f}\n\n'.format(train_metrics['mape'],
      test_metrics['mape']))
```

Listing 6.1: Model evaluation

The Python program is now executed various times in order to evaluate all the different datasets, taking into consideration only the Vanilla LSTM model because, as seen previously, all the different LSTM models give similar results when used on the same dataset. The results are reported in Table 6.1.

Dataset	Train Loss	Test Loss	Train RMSE	Test RMSE	Train MAPE	Test MAPE
cases.csv	1478.01	653019.00	38.44	808.10	69511944.00	5591667712.00
discharged.csv	2048.25	451161.59	45.26	671.69	141947936.00	26.03
allCOVbeds.csv	134.42	242.56	11.59	15.57	6.80	4.99
accumulatedCases.csv	60265.69	10610372.00	245.49	3257.36	1.08	4.14
accumulatedDis.csv	21346.02	7952625.50	146.10	2820.04	6112884.50	0.98
inCOVpatients.csv	34.80	43.07	5.90	6.56	9238986.00	24.38
inTOTpatients.csv	4119.00	4755.48	64.18	68.96	13.81	12.56
resPatients.csv	94.19	109.14	9.71	10.45	5.40	4.63
deaths.csv	1.99	6.94	1.41	2.63	354511392.00	137399568.00
COVnoRes.csv	0.65	0.21	0.81	0.46	137520144.00	177197776.00
COVwithRes.csv	8.22	5.38	2.87	2.32	2245033.00	13.28
allBeds.csv	23493.45	19949.93	153.28	141.24	3.71	3.14
noResPatients	56.81	37.41	7.54	6.12	27.70	26.62
accumulatedDeaths.csv	3.76	43.77	1.94	6.62	6311135.00	0.45

Table 6.1: Final evaluation of Vanilla LSTM model through RMSE and MAPE.

The results of the RMSE and MAPE evaluation provide insight into the performance of the machine learning model. A low RMSE or MAPE value indicates that the model is accurate in its predictions, while a higher value suggests that the model may not be performing as well. However, it is important to keep in mind that the results of these evaluation metrics can be influenced by the range of data being analyzed and the presence of outliers or extreme values. Additionally, the comparison of RMSE and MAPE results across different models or datasets should be done with caution, as these metrics may not be directly comparable due to their different calculation methods.

6.2.2. Improvements

Besides the problems connected to the chosen evaluation metrics, a major concern resulting from the evaluation is overfitting. There are several methods to remove overfitting in a machine learning model, among which increasing the size of the data, using regularization techniques, early stopping, dropout or augmentation [43].

In order to improve the existing model, some of these techniques will be applied and a new loss function will be introduced to evaluate the model.

1. Early Stopping

   Early stopping is a technique used to prevent overfitting and improve the model's generalization performance. It stops the training process of the model when its performance on a validation set stops improving, rather than continuing until the training loss is minimized. This improves the model's ability to generalize to new, unseen data [28]. However, early stopping may also cause underfitting if stopped too early, which is why a patience of 3 epochs is recommended. The *EarlyStopping()* function from Keras' callbacks library is imported and called in the code before fitting the model:

```
1    es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=3)
```

Listing 6.2: Adding early stopping

2. Regularization

   L1 and L2 regularization can be used together in machine learning to improve the generalization performance of a model. This technique is known as Elastic Net regularization, and it is powerful for preventing overfitting.

   The regularization term is a linear combination of the L1 and L2 penalties. The L1 penalty encourages sparsity in the model's parameters, while the L2 penalty encourages small parameter values [13]. During the simulation analysis phase, L1 and L2 regularizations had already been applied to the LSTM models that were overfitting. To further improve the results, they are now added to the remaining ones:

```
model.add(LSTM(units=32, activation='relu', input_shape=(n_steps, n_features),
kernel_regularizer=regularizers.L1L2(l1=0.05, l2=0.005)))
```

Listing 6.3: Adding regularization to every model

3. MSLE

MSLE stands for Mean Squared Logarithmic Error, which is a loss function used to evaluate regression models. Like MSE (Mean Squared Error), MSLE measures the difference between the predicted and actual values of a model, but it takes the logarithm of the predicted and actual values before calculating the squared difference. The use of logarithms is useful when dealing with predictions that cover a large range of values, as it helps to reduce the impact of outliers and extreme values on the loss function. In most of the used datasets, in fact, this is the case. By taking the logarithm of the values, the loss function focuses more on the relative differences between predicted and actual values rather than the absolute differences [18].

```
model.compile(loss='msle', metrics=[RootMeanSquaredError(name='rmse'), 'mape',
'accuracy'], optimizer=Adam(learning_rate=0.001))
```

Listing 6.4: Choosing MSLE as loss function

After the application of the above-described methods to reduce overfitting and minimize the loss function, the model is evaluated one last time. Table 6.2 represents the old values of MSE train and test losses next to the new values, calculated with the newly introduced MSLE function:

Dataset	MSE Train Loss	MSE Test Loss	MSLE Train Loss	MSLE Test Loss
cases.csv	1478.01	653019.00	0.33	0.56
discharged.csv	2048.25	451161.59	0.38	0.26
allCOVbeds.csv	134.42	242.56	0.06	0.05
accumulatedCases.csv	60265.69	10610372.00	0.09	0.08
accumulatedDis.csv	21346.02	7952625.50	0.06	0.05
inCOVpatients.csv	34.80	43.07	0.22	0.17
inTOTpatients.csv	4119.00	4755.48	0.11	0.11
resPatients.csv	94.19	109.14	0.06	0.07
deaths.csv	1.99	6.94	0.30	0.29
COVnoRes.csv	0.65	0.21	0.10	0.12
COVwithRes.csv	8.22	5.38	0.10	0.12
allBeds.csv	23493.45	19949.93	0.07	0.07
noResPatients	56.81	37.41	0.26	0.22
accumulatedDeaths.csv	3.76	0.09	1.94	0.06

Table 6.2: Evaluation of Vanilla LSTM model through MSE and MSLE losses.

When the target variable represents a count, such as in traffic flow, sales data or, in this case, hospital patients, MSLE is very effective. Since the logarithm is used in the calculation, larger errors are still weighted more heavily, but the scale of the target variable is now taken into account without the need of normalizing the data, resulting in more cohesive loss values. By evaluating the model with MSLE, its performance can be assessed in a way that is appropriate for count data, leading to more accurate predictions of hospital patient flow.

7 | Conclusions and Future Developments

In this study, the use of LSTM models for COVID-19 forecasting in the Canary Islands has been examined. The primary objective was to develop an accurate and reliable model that could predict the spread of the virus in the region, helping public health officials to take proactive measures to control its spread.

The results have been promising, demonstrating that LSTM models can effectively predict the number of COVID-19 cases, deaths and hospitalizations in the islands. By using appropriate data preprocessing techniques and hyperparameter tuning, the model's performance has been improved significantly, and the training and validation losses were monitored to avoid overfitting. Moreover, the model's performance has been evaluated through different metrics, such as MAPE and RMSE, finding that RMSE is a more optimal metric to use in this case, as it considers the range of the dataset and reduces the loss through its squared root, not being affected by zero values.

The importance of carefully selecting appropriate evaluation metrics and considering the dataset's characteristics in developing and evaluating LSTM models for COVID-19 forecasting has been emphasized by the findings. Furthermore, this study provides insights and guidelines that can be useful for public health officials and policymakers in managing and controlling the spread of the virus. In conclusion, this study has contributed to the field of COVID-19 forecasting by demonstrating the effectiveness of LSTM models in predicting the spread of the virus in the Canary Islands.

Further research could focus on further improving the performance of the LSTM models by exploring different architectures, hyperparameters, and optimization techniques. Moreover, additional data sources could be incorporated, such as demographic and environmental factors, to enhance the accuracy and robustness of the models. Another area of future research could be to develop ensemble models that combine multiple forecasting

models to improve the overall prediction performance. Furthermore, the models' interpretability could be enhanced by incorporating visualization techniques that enable users to better understand the models' underlying patterns and factors that drive the predictions. Finally, the models' real-world applicability could be evaluated by integrating them into decision-making processes and evaluating their effectiveness in reducing the spread of COVID-19.

Bibliography

[1] R. Agrawal. Know the best evaluation metrics for your regression model. *Analyticsvidhya*, September 2022.

[2] A. Ajagekar. Adam. *Cornell University Computational Optimization*, 2021.

[3] R. Allard. Use of time-series analysis in infectious disease surveillance. *Bull World Health Organ*, 76(4):327–333, 1998. ISSN 0042-9686 (Print); 0042-9686 (Linking).

[4] R. Andonie. Hyperparameter optimization in learning systems. *Journal of Membrane Computing*, 1(4):279–291, 2019. doi: 10.1007/s41965-019-00023-0.

[5] M. Banoula. *What is Tensorflow? Deep Learning Libraries Program Elements*. Simplilearn, February 2021.

[6] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[7] C. Brooks. *Univariate time series modelling and forecasting*, pages 206–264. Cambridge University Press, 2 edition, 2008. doi: 10.1017/CBO9780511841644.006.

[8] J. Brownlee. How to use timesteps in lstm networks for time series forecasting. *Machine Learning Mastery*, April 2017.

[9] T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)? – arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014. doi: 10.5194/gmd-7-1247-2014.

[10] D. Chicco, M. J. Warrens, and G. Jurman. The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation. *PeerJ Computer Science*, 7:e623, 2021.

[11] A. De Myttenaere, B. Golden, B. Le Grand, and F. Rossi. Mean absolute percentage error for regression models. *Neurocomputing*, 192:38–48, 2016.

[12] T. Dehesh, H. Mardani-Fard, and P. Dehesh. Forecasting of covid-19 confirmed cases

in different countries with arima models. *medRxiv*, 2020. doi: 10.1101/2020.03.13.20035345.

[13] O. Demir-Kavuk, M. Kamada, T. Akutsu, and E.-W. Knapp. Prediction using stepwise l1, l2 regularization and feature selection for small data sets with large number of features. *BMC bioinformatics*, 12:1–10, 2011.

[14] G. Derval, V. Franc ois Lavet, and P. Schaus. Nowcasting covid-19 hospitalizations using google trends and lstm. *Center for Research on Computation and Society*, 2020.

[15] C. Fox, A. Levitin, and T. Redman. The notion of data and its quality dimensions. *Information Processing Management*, 30(1):9–19, 1994. ISSN 0306-4573. doi: https://doi.org/10.1016/0306-4573(94)90020-5.

[16] D. Giuliani, M. M. Dickson, G. Espa, and F. Santi. Modelling and predicting the spatio-temporal spread of covid-19 in italy. *BMC Infectious Diseases*, 20(1):700, 2020. doi: 10.1186/s12879-020-05415-7.

[17] S. Hochreiter. Recurrent neural net learning and vanishing gradient. *International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, 1998.

[18] T. O. Hodson, T. M. Over, and S. S. Foks. Mean squared error, deconstructed. *Journal of Advances in Modeling Earth Systems*, 13(12):e2021MS002681, 2021.

[19] I. Holmdahl and C. Buckee. Wrong but useful — what covid-19 epidemiologic models can and cannot tell us. *New England Journal of Medicine*, 383(4):303–305, 2020. doi: 10.1056/NEJMp2016822. PMID: 32412711.

[20] I. Kandel and M. Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, 6(4):312–315, 2020. ISSN 2405-9595. doi: https://doi.org/10.1016/j.icte.2020.04.010.

[21] W. Koehrsen. Overfitting vs. underfitting: A complete example. *Towards Data Science*, pages 1–12, 2018.

[22] A. Krogh. What are artificial neural networks? *Nature Biotechnology*, 26(2):195–197, 2008. doi: 10.1038/nbt1386.

[23] V. Kumar, R. Chimmula, and L. Zhang. Time series forecasting of covid-19 transmission in canada using lstm networks. *Chaos, Solitons Fractals*, 135:109864, 2020. ISSN 0960-0779. doi: https://doi.org/10.1016/j.chaos.2020.109864.

[24] Y. Li, X. Ren, F. Zhao, and S. Yang. A zeroth-order adaptive learning rate method

to reduce cost of hyperparameter tuning for deep learning. *Applied Sciences*, 11(21), 2021. ISSN 2076-3417. doi: 10.3390/app112110184.

[25] L. Men, N. Ilk, X. Tang, and Y. Liu. Multi-disease prediction using lstm recurrent neural networks. *Expert Systems with Applications*, 177:114905, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2021.114905.

[26] H. Onnen. Temporal loops: Intro to recurrent neural networks for time series forecasting in python. *Towards Data Science*, October 2021.

[27] G. Perone. Comparison of arima, ets, nnar, tbats and hybrid models to forecast the second wave of covid-19 hospitalizations in italy. *The European Journal of Health Economics*, 23(6):917–940, 2022. doi: 10.1007/s10198-021-01347-4.

[28] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[29] R. Rawat, J. K. Patel, and M. T. Manry. Minimizing validation error with respect to network size and number of training epochs. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2013. doi: 10.1109/IJCNN.2013.6706919.

[30] M. Scortichini, R. Schneider dos Santos, F. De' Donato, M. De Sario, P. Michelozzi, M. Davoli, P. Masselot, F. Sera, and A. Gasparrini. Excess mortality during the COVID-19 outbreak in Italy: a two-stage interrupted time-series analysis. *International Journal of Epidemiology*, 49(6):1909–1917, 10 2020. ISSN 0300-5771. doi: 10.1093/ije/dyaa169.

[31] S. Sharma, S. Sharma, and A. Athaiya. Activation functions in neural networks. *Towards Data Science*, 6(12):310–316, 2017.

[32] E. Shim, A. Tariq, W. Choi, Y. Lee, and G. Chowell. Transmission potential and severity of covid-19 in south korea. *International Journal of Infectious Diseases*, 93: 339–344, April 2020.

[33] L. Silva, D. Figueiredo Filho, and A. Fernandes. The effect of lockdown on the covid-19 epidemic in brazil: evidence from an interrupted time series design. *Cadernos de Saúde Pública*, 36, 2020.

[34] R. C. Staudemeyer and E. Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks. *arXiv*, 2019.

[35] S. Sun, Z. Cao, H. Zhu, and J. Zhao. A survey of optimization methods from a

machine learning perspective. *IEEE Transactions on Cybernetics*, 50(8):3668–3681, 2020. doi: 10.1109/TCYB.2019.2950779.

[36] A. Tobías. Evaluation of the lockdowns for the sars-cov-2 epidemic in italy and spain after one month follow up. *Sci Total Environ*, 725:138539, Jul 2020. ISSN 1879-1026 (Electronic); 0048-9697 (Print); 0048-9697 (Linking). doi: 10.1016/j.scitotenv.2020.138539.

[37] Y.-T. Tsan, D.-Y. Chen, P.-Y. Liu, E. Kristiani, K. L. P. Nguyen, and C.-T. Yang. The prediction of influenza-like illness and respiratory disease using lstm and arima. *International Journal of Environmental Research and Public Health*, 19(3), 2022. ISSN 1660-4601. doi: 10.3390/ijerph19031858.

[38] T. P. Velavan and C. G. Meyer. The covid-19 epidemic. *Trop Med Int Health*, 25 (3):278–280, Mar 2020. ISSN 1365-3156 (Electronic); 1360-2276 (Print); 1360-2276 (Linking). doi: 10.1111/tmi.13383.

[39] A. Wei, Z. Chew, Y. Pan, Y. Wang, and L. Zhang. Hybrid deep learning of social media big data for predicting the evolution of covid-19 transmission. *Knowledge-Based Systems*, 233:107417, 2021. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2021.107417.

[40] A. S. Weigend. Forecasting the future and understanding the past. *Time Series Prediction*, page 663, 1994.

[41] Y. Xie, D. Kulpanowski, J. Ong, E. Nikolova, and N. M. Tran. Predicting covid-19 emergency medical service incidents from daily hospitalisation trends. *International Journal of Clinical Practice*, 75(12):e14920, 2021. doi: https://doi.org/10.1111/ijcp.14920.

[42] V. Yathish. Loss functions and their use in neural networks. *Towards Data Science*, August 2022.

[43] X. Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[44] A. Zeroual, F. Harrou, A. Dairi, and Y. Sun. Deep learning methods for forecasting covid-19 time-series data: A comparative study. *Chaos, Solitons Fractals*, 140: 110121, 2020. ISSN 0960-0779. doi: https://doi.org/10.1016/j.chaos.2020.110121.

[45] S. Zhao, Q. Lin, J. Ran, S. S. Musa, G. Yang, W. Wang, Y. Lou, D. Gao, L. Yang, D. He, and M. H. Wang. Preliminary estimation of the basic reproduction number of novel coronavirus (2019-ncov) in china, from 2019 to 2020: A data-driven analysis

in the early phase of the outbreak. *Int J Infect Dis*, 92:214–217, Mar 2020. ISSN 1878-3511 (Electronic); 1201-9712 (Print); 1201-9712 (Linking). doi: 10.1016/j.ijid.2020.01.050.

List of Figures

List of Tables

Listings