# Simulations Analysis of the LSTM Forecasting System for COVID-19 Cases Prediction in the Canary Islands

By Ginevra Iorio

## Abstract

In this study, the LSTM models developed for predicting the number of future COVID-19 cases in the Canary Islands are evaluated based on their training phase results. The hyperparameters are fine-tuned to achieve the lowest possible system's loss and improve model performance. Additionally, overfitting and underfitting are addressed by monitoring the training and validation loss graphs.

## Introduction

After the different LSTM models have been built, it is crucial to ensure that they are performing correctly by simulating the system and adjusting the hyperparameters. This involves running the LSTM models on the data and evaluating their performance examining training and validation losses. However, if the model is not performing well, the hyperparameters will need to be adjusted. These parameters include the learning rate, batch size, number of epochs, number of hidden layers, and number of timesteps. Adjusting them requires multiple evaluations of the LSTM model's performance in which different values will be tested to achieve the desired results.

Monitoring the progress of training and validation losses will also help improve the models and various techniques can be used to effectively do it. One approach is to plot the training and validation losses together on a graph, which will provide a visual representation of the performance of the LSTM model during training. For each set of data and each LSTM model, the graph will be plotted and studied to determine whether the model is overfitting or underfitting the data. For instance, if the training loss is decreasing while the validation loss is increasing, then the model is overfitting to the training data. On the other hand, if both training and validation losses are high, the model may be underfitting. Therefore, carefully monitoring the training and validation loss will be critical to develop accurate LSTM models [1].

## Loss and Validation Loss

For LSTM models, which are typically used for regression problems, accuracy is not the most suitable metric to evaluate the performance of the model, being mainly useful in classification problems [2]. Consequently, in order to evaluate the model and adjust the hyperparameters, the resulting values of loss and validation loss at the end of each training are monitored in this project.

Loss is a measure of how well the model's predictions match the actual values in the training data. For the moment, the mean squared error (MSE) is used but later on the model will be evaluated with different loss functions.

During the training process, the loss function is used to update the model's weights and biases through backpropagation. The goal of training the LSTM model is to minimize the loss function by adjusting the model's parameters in each iteration. When a loss function decreases in value during training, the model is successfully learning [3].

Validation loss is the measure of error between the predicted values and the actual values in the validation dataset. It is calculated by passing the validation dataset through the trained LSTM model and comparing the predicted values with the actual values in the validation dataset. The loss function used in this case is the same as the loss function used during the training process.

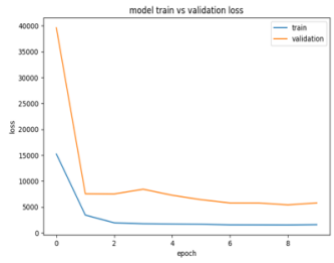
Therefore, the goal of training a LSTM model is to minimize both the training loss and the validation loss, in order to obtain a model that performs well on new, unseen data. To keep track of both values during the training, a new function called *plot_loss()* is introduced into the *data_handler.py*:

```
def plot_loss(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model train vs validation loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper right')
    plt.show()
```

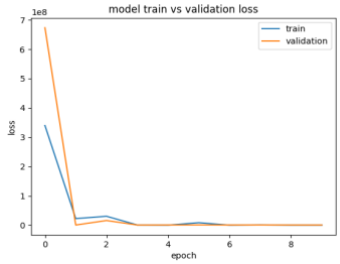
Taking as input the fitting history, this function plots on a graph both the values of loss and val_loss resulting after each epoch and is helpful to detect if the model is overfitting, underfitting or if it's correctly running.

Having a first look at the resulting graphs of different type of data, such as daily COVID cases, accumulated COVID cases or daily deaths, 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 behavior, but accumulatedCases.csv has very high losses values.

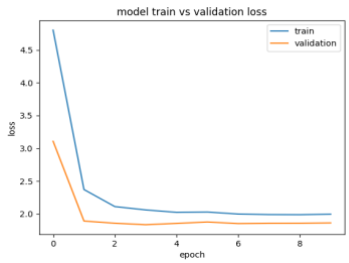
cases.csv




accumulatedCases.csv




deaths.csv




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.

## Model Optimization

Optimization refers to the process of finding the set of model parameters or weights that minimize a given loss function. Different approaches can be used, such as random searches and grid searches, evolutionary optimization, or Bayesian optimization. In this project, random searches will be applied searching for the best parameters that will result in the best possible performance of the model on the training data [4]. The optimization will consider:

1. **Number of Timesteps:** it determines the length of the input sequence that the LSTM model can process, and it can have a significant impact on the model's ability to learn and make accurate predictions.
2. **Hyperparameters:** what needs to be tuned in a LSTM model, which includes the number of LSTM layers, the number of units in each layer, the batch size, the learning rate, the dropout rate, and the number of epochs.
3. **Activation Functions:** mathematical functions that are applied to the output of each node or neuron in a neural network to introduce non-linearity into the model. Non-linearity is important because many real-world problems are non-linear, and non-linear functions can help neural networks learn complex relationships between inputs and outputs.

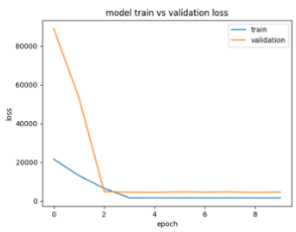
## Number of Timesteps

Choosing the best number of timesteps the LSTM models involves finding a balance between capturing enough historical context while avoiding overfitting or computational complexity. In fact, the number of timesteps is an important hyperparameter that can impact the model's performance. Increasing it can help the LSTM model capture longer-term dependencies in the data. However, it can also increase the computational complexity of the model and make training slower. On the other hand, decreasing the number of timesteps can reduce the computational complexity of the model and make training faster but it may also result in the model not capturing long-term dependencies in the data, which could lead to lower performance.

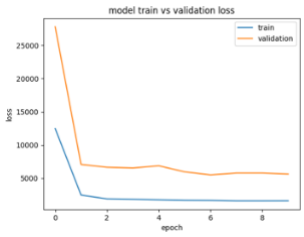
It should be chosen based on the frequency of the data. Since the data is collected daily, then a lower number of timesteps may be necessary to capture the historical context compared to data collected hourly. Moreover, a higher number of timesteps can lead to overfitting, which means that the model may perform well on the training data but poorly on the test data.

It is important to experiment with different values of timesteps and evaluate the performance of the model for each value [5]. Since this number is specified in each one of the five models, these experiments will only be carried out on the Vanilla LSTM and be considered valid for the others.

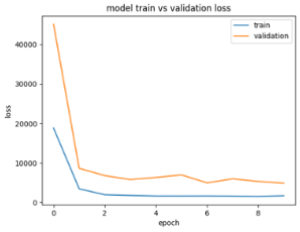
1 timestep:



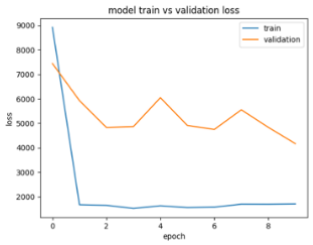

2 timesteps:



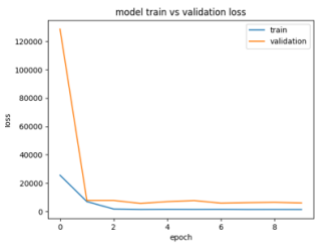

3 timesteps:



4 timesteps:




5 timesteps:




Observing the behavior of the obtained graphs, it is visible that the 3 timesteps case 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.

## Hyperparameters

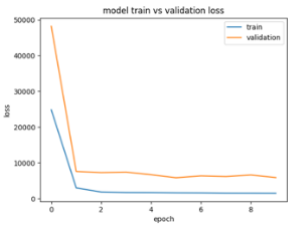
Hyperparameter tuning is the process of finding the optimal combination of hyperparameters for a given machine learning algorithm. Hyperparameters are parameters that are set by the user before training the model, and they can have a significant impact on its performance [6].

Number of hidden units

Increasing the number of hidden units in each LSTM layer can increase the model's capacity to learn complex patterns but may also increase the risk of overfitting and slow down the training process.

Starting with a small number of hidden units (e.g., 32), these are gradually increased (e.g., 64, 128, 256) to compare the performance. Generally, increasing the number of hidden units can help the model learn more complex patterns, but it can also increase the risk of overfitting [7].

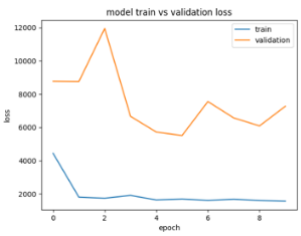
units = 32




units = 64


units = 128


units = 256



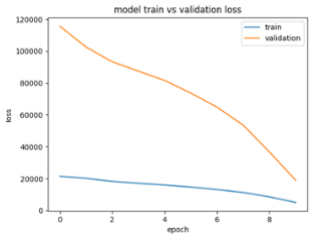
The results show that increasing the number of hidden units the results keep getting worse, meaning that the optimal number of units that can be used is 32.

Learning rate

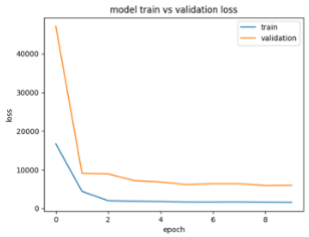
The learning rate determines the step size at which an optimizer adjusts the weights of the model during training. A higher learning rate can lead to faster convergence but may also cause the optimizer to overshoot the optimal weights and lead to instability. On the other hand, a low learning rate can cause the optimizer to take small steps towards the optimal weights, resulting in slow convergence of the training process and a longer time to reach the optimal solution.

Starting with a small learning rate (e.g., 0.0001), it is increased (e.g., 0.001, 0.01) and the performance is tracked. If the training loss is fluctuating or increasing, it may be a sign that the learning rate is too high [8].

learning_rate = 0.0001




learning_rate = 0.001



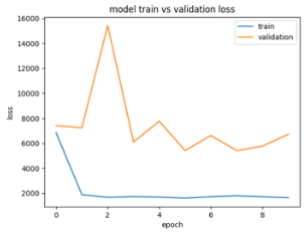

learning_rate = 0.1


The optimal loss behavior, in this case, is given by a learning rate of 0.001, which is consequently chosen as a parameter for the Adam optimizer.

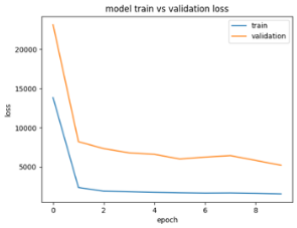
Batch size

The batch size determines the number of samples that are processed at once during training. A larger batch size can lead to faster convergence but may also require more memory and decrease the model's ability to generalize to new data. Different hardware may be optimized for different batch sizes. For example, GPUs are often optimized for larger batch sizes, while CPUs may perform better with smaller batch sizes [9].

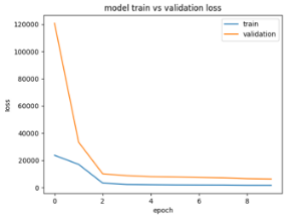
batch_size = 1



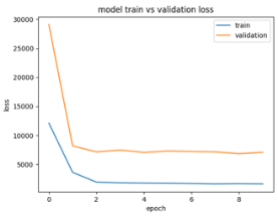

batch_size = 8



batch_size = 16




batch_size = 32




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.

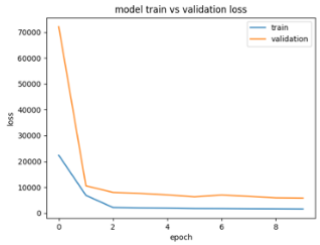
Number of Epochs

The number of epochs refers to the number of times the entire dataset is passed forward and backward through the neural network during training. In other words, it is the number of complete cycles the training algorithm goes through the entire dataset [10].

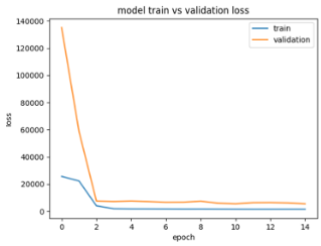
Starting with a small number of epochs (e.g., 10), these are gradually increased keeping an eye on the loss.

Generally, a larger number of epochs will result in better performance on the training set, but it may also lead to overfitting and poorer generalization. Training for too many epochs can be time-consuming, so the trade-off between training time and performance needs to be considered when choosing the number of epochs.

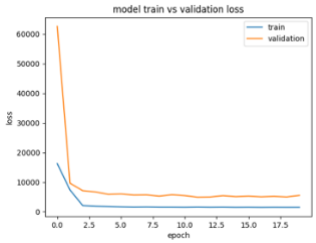
epochs = 10



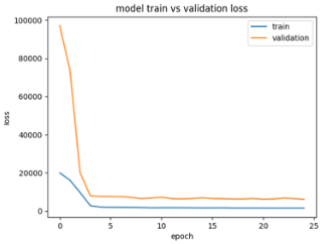

epochs = 15




epochs = 20




epochs = 25




The number of epochs that best fits the model is 15.

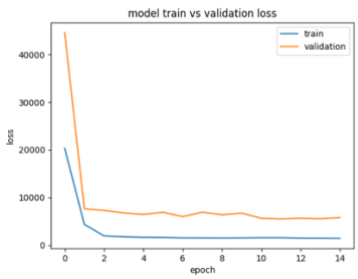
## Activation Functions

There are several types of activation functions that are commonly used in neural networks, including:

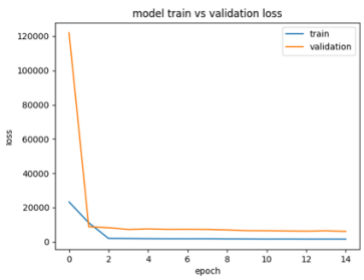
- **Sigmoid:** it is a commonly used activation function that maps any input value to a value between 0 and 1. It is often used in the output layer of binary classification problems, where the output represents the probability of belonging to the positive class.
- **Linear:** it is used for regression 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 [11]. Given that this problem is addressing a regression problem, the only activation functions that can be used are ReLU and Linear.

activation = linear




activation = relu

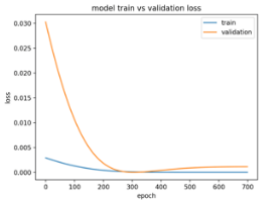



The results show that the activation function that best fits the model is ReLu.

## Overfitting and Underfitting

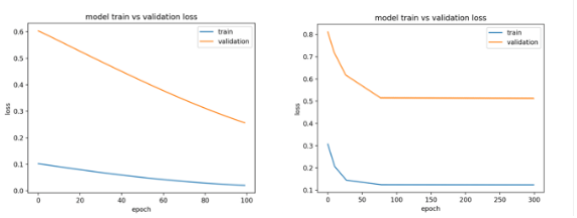
Overfitting and underfitting are common problems in machine learning that can lead to poor performance of models. Overfitting occurs when the model is too complex and fits the training data too well, resulting in poor performance on new data. Underfitting occurs when the model is too simple and cannot capture the underlying patterns in the data, resulting in poor performance on both the training and new data. Detecting whether a model is overfitting or underfitting is an important step in evaluating its performance.

One way to detect overfitting is to plot the model's learning curves for both the training and validation sets. If the training error is significantly lower than the validation error, it indicates that the model is overfitting.



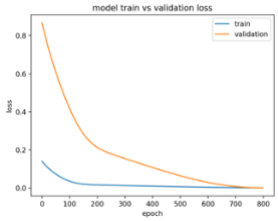

To solve overfitting, several techniques can be used such as regularization, dropout, and early stopping. Regularization adds a penalty term to the loss function during training, discouraging the model from having large weights and making it more generalizable. Dropout randomly drops out some neurons 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.




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. Hyperparameter 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 [12].

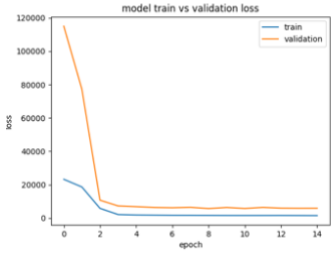


## Results

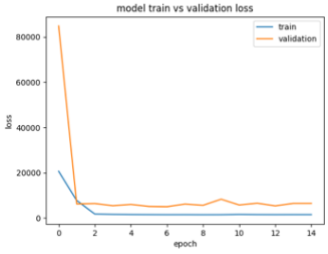
For each csv file of the dataset, all the different LSTM methods will be applied in order to have a look on the behavior of the training and validation losses of the models.

1. cases.csv

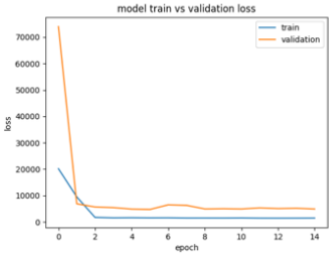
vanilla LSTM



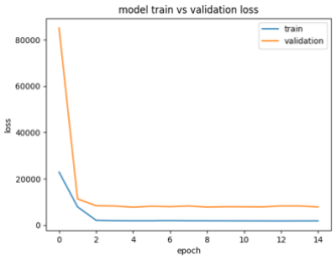

stacked LSTM



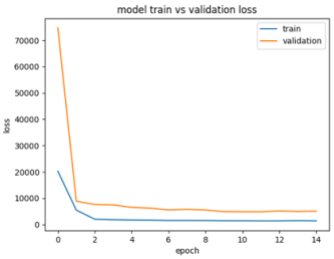

bidirectional LSTM




CNN LSTM

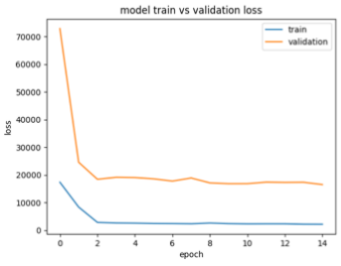



ConvLSTM

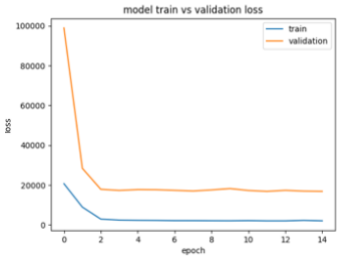



2. discharged.csv

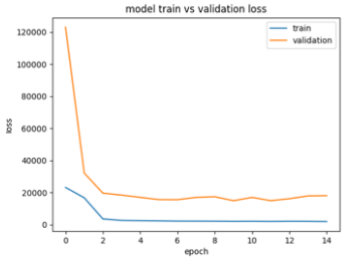
vanilla LSTM



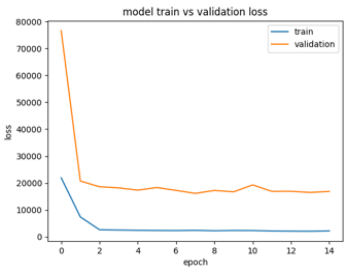

stacked LSTM



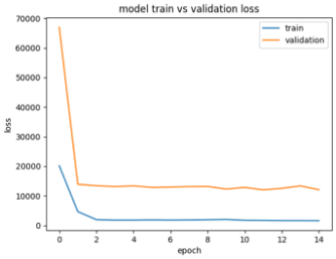

bidirectional LSTM




CNN LSTM

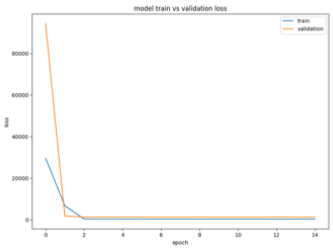



ConvLSTM

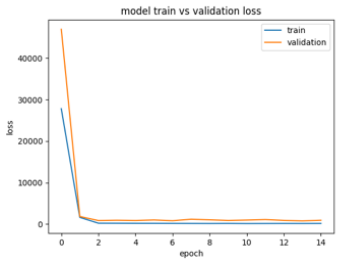


3. allCOVbeds.csv

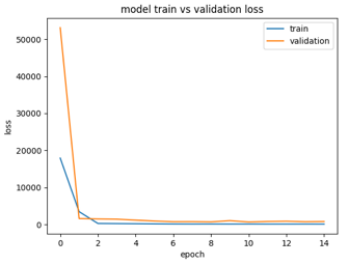
vanilla LSTM



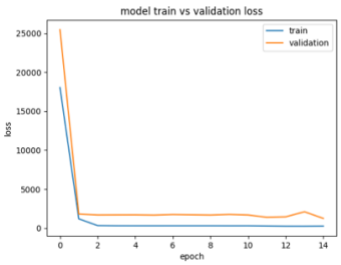

stacked LSTM



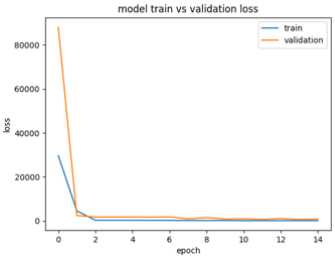

bidirectional LSTM




CNN LSTM



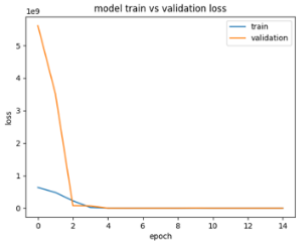

ConvLSTM




4. accumulatedCases.csv

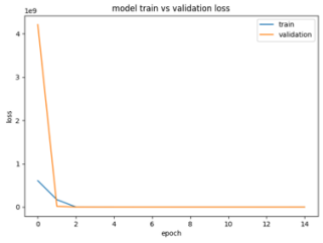
vanilla LSTM


stacked LSTM

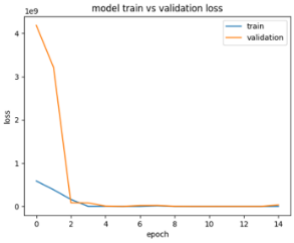



bidirectional LSTM


CNN LSTM

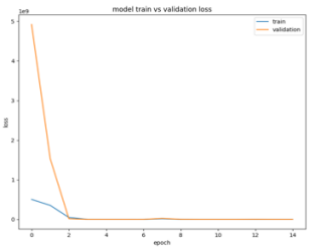



ConvLSTM

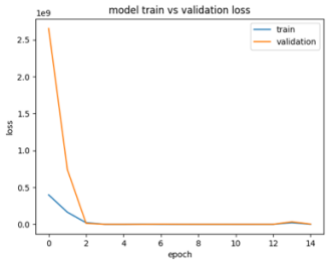


5. accumulatedDis.csv

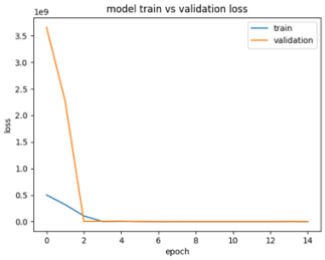
vanilla LSTM




stacked LSTM




bidirectional LSTM




CNN LSTM

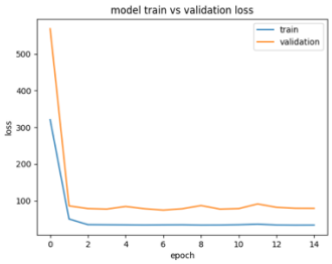

ConvLSTM


6. inCOVpatients.csv

vanilla LSTM

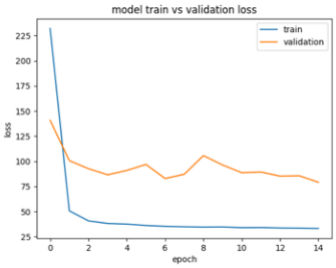

stacked LSTM


bidirectional LSTM



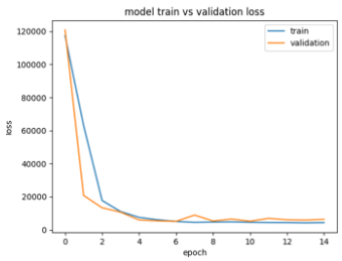

CNN LSTM


ConvLSTM

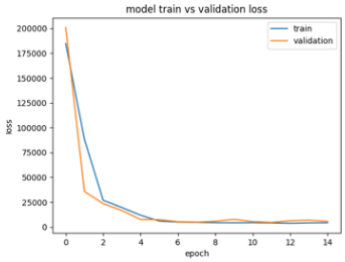


7. inTOTpatients.csv

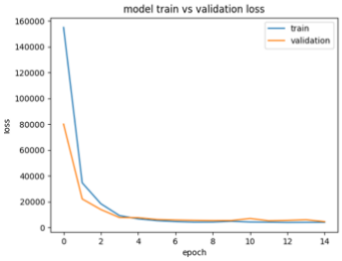
vanilla LSTM



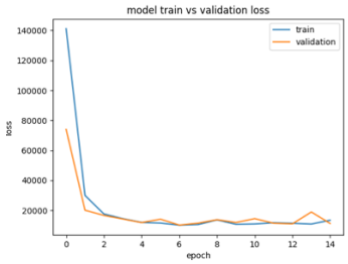

stacked LSTM



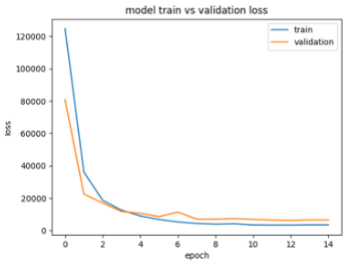

bidirectional LSTM




CNN LSTM

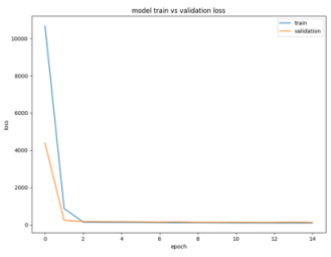



ConvLSTM

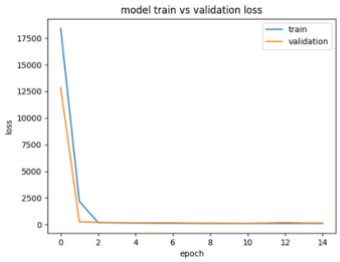



8. resPatients.csv

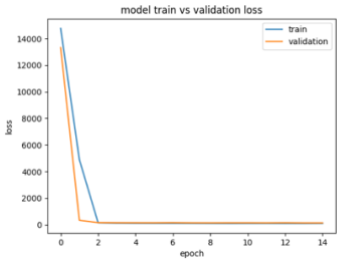
vanilla LSTM



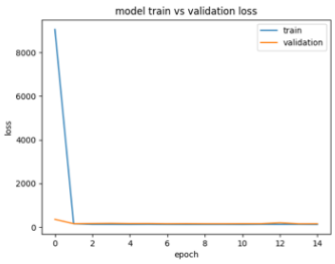

stacked LSTM



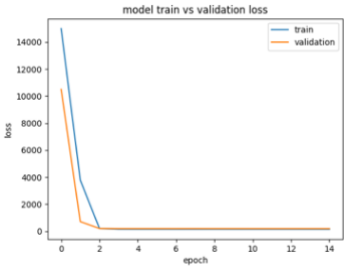

bidirectional LSTM




CNN LSTM

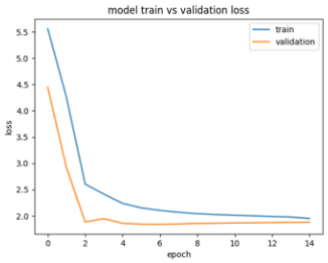



ConvLSTM

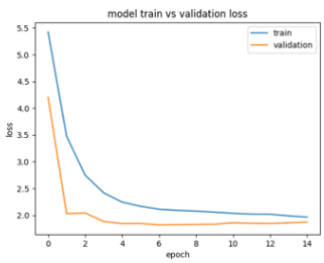


9. deaths

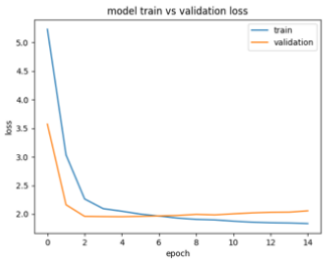
vanilla LSTM



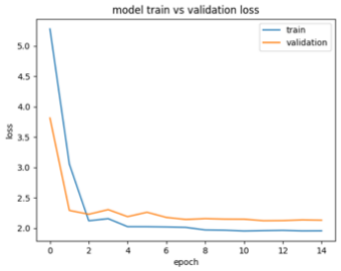

stacked LSTM



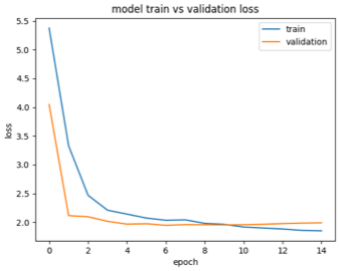

bidirectional LSTM




CNN LSTM

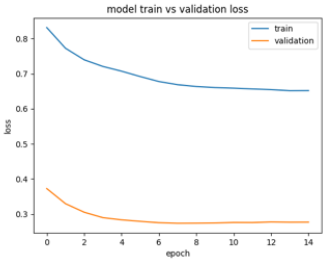



ConvLSTM

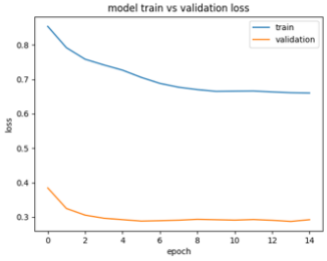



10. COVnoRes.csv

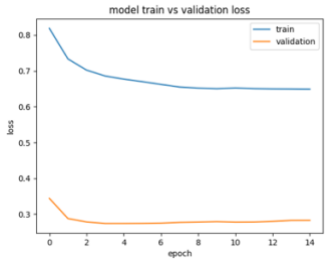
vanilla LSTM



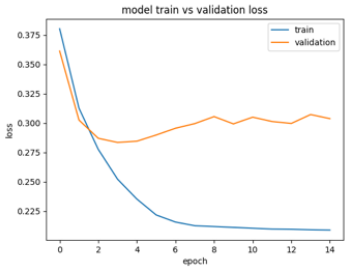

stacked LSTM



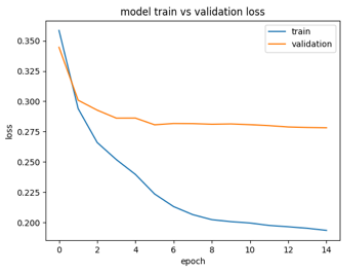

bidirectional LSTM




CNN LSTM

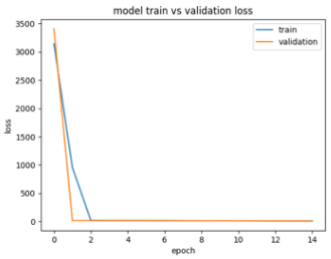



ConvLSTM

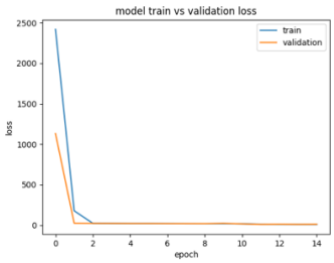


11. COVwithRES.csv

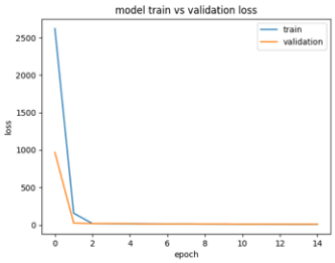
vanilla LSTM



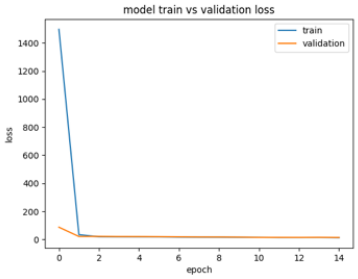

stacked LSTM



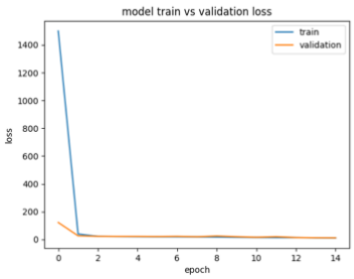

bidirectional LSTM




CNN LSTM

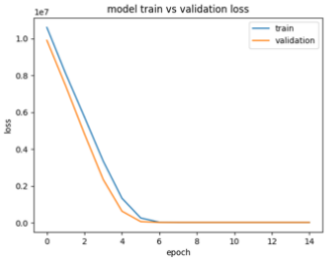



ConvLSTM

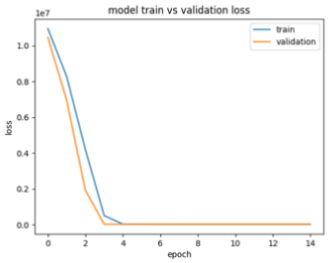



12. allBeds.csv

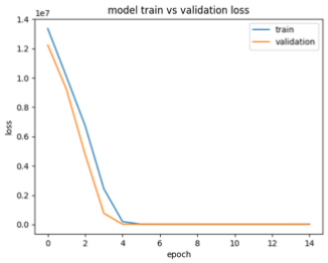
vanilla LSTM



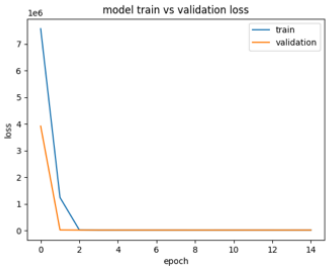

stacked LSTM



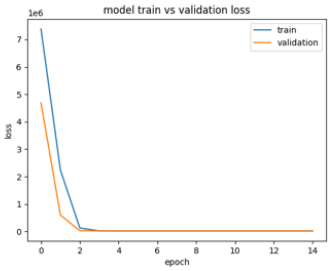

bidirectional LSTM




CNN LSTM

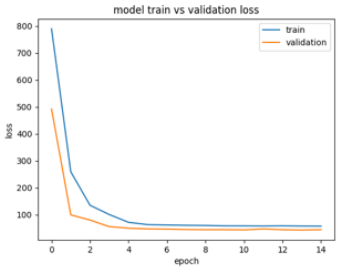



ConvLSTM

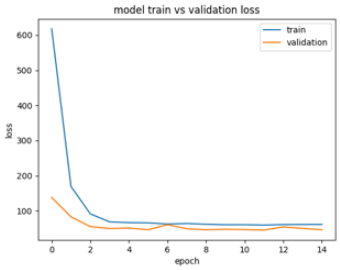


13. noRESpatients.csv

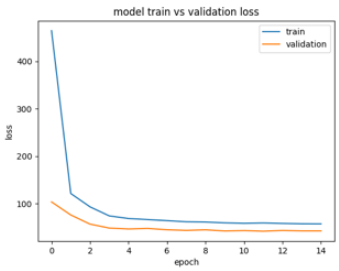
vanilla LSTM



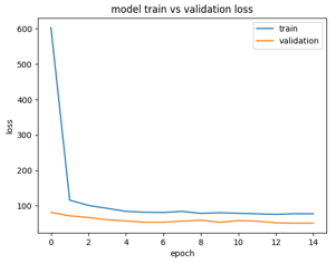

stacked LSTM



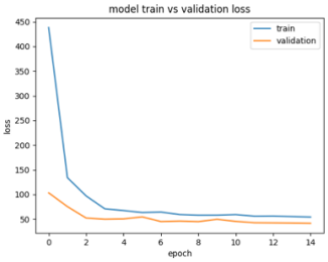

bidirectional LSTM




CNN LSTM

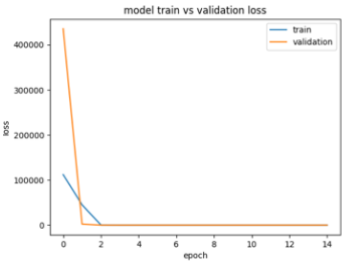



ConvLSTM

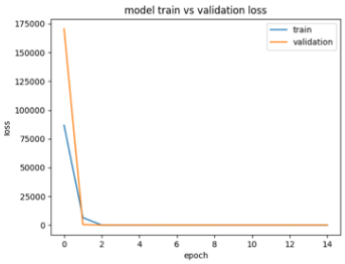



14. accumulatedDeaths.csv

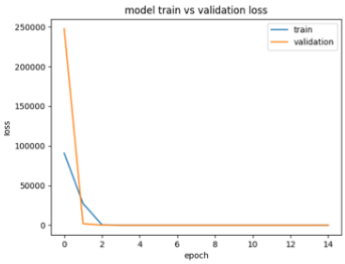
vanilla LSTM



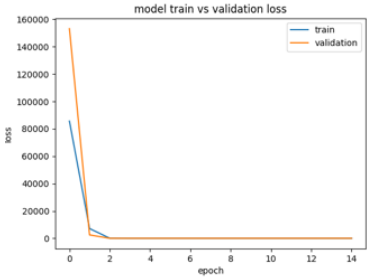

stacked LSTM



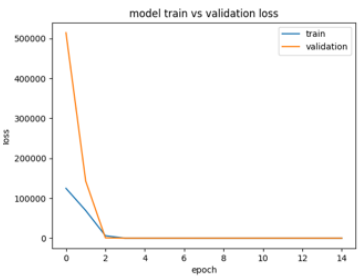

bidirectional LSTM




CNN LSTM




ConvLSTM



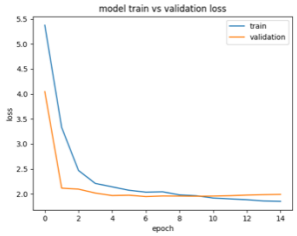
As a result, it is visible that almost all the different LSTM models are underfitting, resulting in very high validation and training losses, meaning that a larger and more complex training dataset would be needed. Another solution would be adding more layers to the models, increasing their complexity.

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:

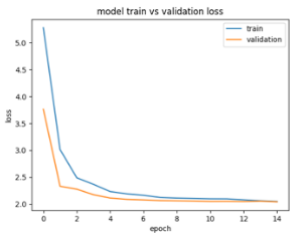
```
model.add(
    ConvLSTM2D(filters=64, kernel_size=(1, 2), activation='relu', input_shape=(n_seq, 1, n_steps, n_features),
               kernel_regularizer=regularizers.L1L2(l1=0.02, l2=0.05)))
```

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

no regularization




with regularization




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

## Conclusion

In conclusion, the use of LSTM models for COVID-19 forecasting in the Canary Islands showed promising results. By using appropriate data preprocessing techniques and hyperparameter tuning, the LSTM models were able to accurately predict the number of COVID-19 cases and deaths in the region. The choice of appropriate hyperparameters has been, though, critical for achieving an improved performance, and it has required an iterative process of training, evaluation, and adjustment. Moreover, the model's training and validation losses have been monitored to detect signs of overfitting or underfitting in the results.
Overall, the results suggest that LSTM models can be a valuable tool for predicting the spread of COVID-19 and other infectious diseases.

**Bibliography**


[1] Baeldung, W.by: *Training and validation loss in Deep Learning*, *Baeldung on Computer Science*. Accessed: February 10, 2023. https://www.baeldung.com/cs/training-validation-loss-deep-learning

[2] Hasty.ai*, Accuracy*. Accessed: February 10, 2023. https://hasty.ai/docs/mp-wiki/metrics/accuracy

[3] Elastic, *Loss functions for regression analyses: Machine learning in the Elastic Stack [master]*. Accessed: February 12, 2023. https://www.elastic.co/guide/en/machine-learning/master/dfa-regression-lossfunction

[4] Castillo, D. and Dianne CastilloDianne is a content marketing manager at Seldon (2023) *Machine learning optimisation - why is it so important?*, *Seldon*. Accessed: February 13, 2023. https://www.seldon.io/machine-learning-optimisation

[5] Brownlee, J. (2020) *How to use Timesteps in LSTM networks for time series forecasting*, *MachineLearningMastery.com*. Accessed: February 13, 2023. https://machinelearningmastery.com/use-timesteps-lstm-networks-time-series-forecasting/

[6] Nyuytiymbiy, K. (2022) *Parameters and hyperparameters in machine learning and Deep Learning*, *Medium*. Towards Data Science. Accessed: February 16, 2023. https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac

[7] Mic (2015) *Selecting the number of neurons in the hidden layer of a neural network: R-bloggers*, *R*. Accessed: February 16, 2023. https://www.r-bloggers.com/2015/09/selecting-the-number-of-neurons-in-the-hidden-layer-of-a-neural-network/

[8] Brownlee, J. (2019) *How to configure the learning rate when training deep learning neural networks*, *MachineLearningMastery.com*. Accessed: February 18, 2023. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[9] Brownlee, J. (2019) *How to use different batch sizes when training and predicting with lstms*, *MachineLearningMastery.com*. Accessed: February 18, 2023. https://machinelearningmastery.com/use-different-batch-sizes-training-predicting-python-keras/

[10] Gretel.ai - Incorporate generative AI into your data*, How many epochs should I train my model with?*. Accessed: February 21, 2023. https://gretel.ai/gretel-synthetics-faqs/how-many-epochs-should-i-train-my-model-with

[11] dishashree26 (2022) *Fundamentals of deep learning - activation functions and when to use them?*, *Analytics Vidhya*. Accessed: February 21, 2023. https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/

[12] Brownlee, J. (2020) *How to diagnose overfitting and underfitting of LSTM models*, *MachineLearningMastery.com*. Accessed: February 28, 2023. https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/