Software project: Prediction of statewide confirmed cases using multiple input artificial neural network models for COVID-19 forecasting in India with long and short training period data

Rahaf Gharz Addien[*], **Suresh Kumar Choudhary**[*], **and Melanie Vogel**[*]

[*]Freie Universität Berlin, Computer Science department, Berlin, 14195, Germany

ABSTRACT

This project focuses on the implementation of several artificial neural network models on a dataset of COVID-19 cases from India. The main goal is to evaluate these methods on long and short term training periods to find one that can efficiently predict the numbers in both cases while retaining a high efficacy. The project is based on the study "Multiple-Input Deep CNN Model for COVID-19 Forecasting in China" by Huang et al.[1], that focuses on the need of an artificial neural network model capable of forecasting COVID-19 cases with only a short time period as training data at hand during the start of the COVID-19 pandemic. For this purpose, the artificial neural network methods Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Gate Recurrent Unit (GRU) were used for predicting the cumulative confirmed cases of a given day based on the data of the previous five days including six time sequences (features) that influence the confirmed cases. The methods were evaluated with the R^2 score, Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) metrics using multiple features as input. Like this, we also implemented the aforementioned methods using the Keras and Pytorch Python libraries and evaluated them respectively. The Indian dataset provides the same features and the models were trained and tested on data from five Indian states. Our experimental results revealed that the deep CNN model shows the best efficacy and the lowest error. We also explore the data from India in terms of health care, population density and interventions in the respective states that were most affected.

Introduction

COVID-19 arrived in Germany in early 2020. Since then, we all had to live with the restrictions due to the pandemic. Back then, the new disease began to spread in Wuhan City, China, where the first infection was discovered in early December 2019. The Coronavirus disease 2019, as it was named by the World Health Organization (WHO), began to spread extensively within China and after a few weeks also in other countries. Our seed paper, namely the study from Huang et al.[1], is aimed at the forecasting of cumulative confirmed COVID-19 cases (the confirmed number includes the deaths, recovered, and active cases) and focuses on deep learning techniques for time series forecasting for this purpose. Inspired by this study and motivated by finding a neural network architecture that would allow for precise forecasting of such cases, even with only a short time period of data available for the training of the model, we implemented all four methods during this project and trained them on a long as well as a short time period of data from India with the same features at hand. In this regard, the model of CNN model has the best results out of the four methods compared.

1 Related Work

Aiming with a better understanding of the used methods, we are interested in papers that made use of deep learning for time series forecasting generally and for predicting COVID-19 cases especially as well as papers with a focus on different structures of the used methods and factors that influence their performance.

Omran et al.[2] presented two deep learning methods GRU and LSTM for predicting confirmed and deaths COVID-19 cases in three countries: Egypt, Saudi Arabia, and Kuwait. LSTM has the best performance in predicting the confirmed cases whereas GRU is more efficient in predicting the death cases. For predicting confirmed, released, negative, and death COVID-19 cases, Dutta et al.[3] compared the performance of GRU, LSTM, and a hybrid model of GRU and LSTM that outperforms the other models whose performances differ based on the case of study. Arora et al.[4] predicted the new COVID-19 cases in India using different variants of LSTM and compared their performance in predicting the daily and weekly cases. Frausto-Solís et al.[5] used the CNN based method to predict the COVID-19 confirmed cases for the countries: United States, Mexico, Brazil, and Colombia. In this study, the CNN predictions adjusted by AutoRegressive Integrated Moving Average (ARIMA) and Exponential Smoothing (ES) methods. This hybrid approach surpassed the performance against the individual CNN, LSTM, ARIMA and ES in terms of Mean Absolute Percentage Error (MAPE) metric. Wang et al.[6] also used the CNN based method to detect the COVID-19 cases using the chest x-ray images instead of time series data. In this study, they explained the predictions as well to make sure that it was making decisions based on relevant information in x-ray images. Mohimont et al.[7] proposed a multiple CNN and temporal CNN approach to predict the cumulative confirmed cases, the number of hospitalizations, recoveries, and deaths in France. Nabi et al.[8] also found the CNN as outperforming method comparing to LSTM and GRU methods. They used data from three countries (Brazil, Russia, and the United Kingdom) for a long time period (training data of 291 days, test data of 40 days) to conclude their experiment.

2 Data

Huang et al.[1] chose the data from several most affected Chinese cities at the time of writing the study, because COVID-19 was most prominent in China back then and not as much data was available in comparison to today. The data was obtained from the Surging News Network and the WHO for the period between January 23, 2020 and March 2, 2020 and contains information on the cumulative confirmed, new confirmed, cumulative cured, new cured, cumulative deaths, and new deaths COVID-19 cases in seven Chinese cities. We adopted this idea and found a regularly updated data set from India on Github [*] with the same features as considered in the study, where we chose the five most affected states (relatively) instead of cities, namely Karnataka, Kerala, Maharashtra, Tamil Nadu and Uttar Pradesh. For the evaluation of the implemented models on a long time period for the training, all dates from March 10, 2020 till June 30, 2021 are considered. When evaluating on a short time period for the training, we only considered data from March 10, 2020 to June 18, 2020. An overview of the data statistics can be seen in Table 1.

Table 1. Data Overview

Dataset	Total cities/states	Number of features	Time period	Number of Data Points
China	7 cities	6	January 23, 2020 to March 2, 2020	40
India (short)	5 states	6	March 10, 2020 to June 18, 2020	101
India (long)	5 states	6	March 10, 2020 to June 30, 2021	476

3 Methods

3.1 Multilayer Perceptron - MLP

Multilayer Perceptrons (MLPs) are feed-forward Artificial Neural Networks (ANNs) that refer to networks composed of multiple layers of the so-called Perceptrons, including a threshold activation, allowing us to solve deep learning problems.

What is a Perceptron?

The perceptron algorithm was invented in 1958 at the Cornell Aeronautical Laboratory by Frank Rosenblatt[9], originally intended to be a hardware machine for image recognition. It is an algorithm for supervised learning of binary classifiers. Usually we have a linear classifier algorithm that classifies the input by separating two classes or categories with a straight line (Appendix A, Figure 14).

As a result, we get a single output that is calculated by forming a linear combination using the input weights, which can be mathematically described with equation (1):

$$y = \phi\left(\sum_{n}^{i=1} w_i x_i + b\right) = \phi\left(w^T x + b\right) \tag{1}$$

However, Perceptrons cannot be trained to recognise many classes of patterns. For example, if we take the logical operators AND (&&) and OR (||), we can easily see that they are linearly separable by looking at their truth tables (Appendix A, Figure 15) and since we can linearly separate the classes True and False, we can solve this with a single Perceptron (Appendix A, Figure 16).

As stated above, a Perceptron is not enough when we face a problem that is not solvable with linear separation, e.g. the logical operator XOR (Appendix A, Figure 17). In this case we obviously need more than one line to separate the values into

[*]https://github.com/covid19india/api

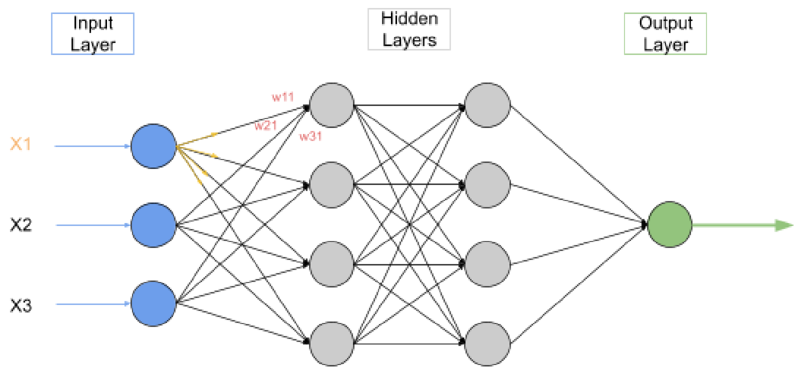


Figure 1. Multilayer Perceptron Model

The Input layer (blue) receives several inputs (X1, X2, X3), there are several hidden layers (grey) and the Output layer (green).

two categories, so we cannot solve the problem with only one layer. This is why we use a Multilayer Perceptron, where we can use a combination of the negated logical AND and the normal logical OR in order to calculate XOR (Appendix A, Figure 18).

Multilayer Perceptron Model

Multilayer Perceptrons (MLPs) are fully connected neural networks that are composed of more than one Perceptron. They consist of an input layer (receives the incoming signal), hidden layers for the calculations, and an output layer where a decision/prediction is made upon the input.

We speak of feedforward networks with a forward and backward pass.

Forward pass: The incoming signal gets directed from the beginning, i.e. the input layer that receives this signal, through the hidden layers that do the calculations, to the output layer, which makes a decision or prediction on this signal in the end.

Backward pass: Backpropagation is used in order to gain a gradient of error, where the parameters get adjusted as they move the model closer to the error minimum (e.g. via stochastic gradient descent). A common method to measure the error in MLP models is the Root Mean Square Error (RMSE).

In Figure (1) we can see a schema of such a MLP, where we have the input values X1, X2 and X3. that are each connected with all the neurons of the first hidden layer. These connections are weighted. The assigned weights are either initially a random value between -1 and 1, or in some cases are chosen non-randomly as there are specific techniques for some models. During the training of the neural network we want to achieve the optimal weights at each edge of the model. We will call the weights w11, w21 and w31, as we have three inputs at this neuron. What we do as a calculation is a so-called weighted sum. For the first neuron in our case this means, we take the input values X1, X2 and X3, multiply them by the corresponding weight w11, w21 and w31, and add them together in the end. For the first neuron of the first hidden layer here, we can describe this with equation (2).

$$y = X1 * w11 + X2 * w21 + X3 * w31 \tag{2}$$

Once we have the weighted sums at each neuron of the first hidden layer, the activation function is calculated and the results get passed on through the connections to each neuron of the next hidden layer, and so on. Usually we store those weighted connections in a matrix and look at the input values as an array. In this case the cells of the matrix are filled with the weights (w_{ij}) and the input array is one column with X1, X2 and X3 as values. So what we do essentially for the weighted sum is matrix multiplication.

3.2 Convolutional Neural Network - CNN

CNNs are feedforward ANNs, inspired by the brain's visual cortex, and have been used in image recognition (two-dimensional data) since the 1980s. In contrast with the fully connected structures, where each neuron is associated with many different weights, CNNs use the concept of weight sharing, which enables faster computations and features extraction in an efficient manner[10]. 1D CNNs are a modified version of 2D CNNs that show high efficiency with data that has patterns along a single spatial dimension (time) in many applications such as time series forecasting, natural language processing tasks, and audio signals analysis.

Models of CNN consist basically of convolution layers and fully connected layers. As forward pass and using the training data, the input data passes through a series of 1D convolution layers with convolutional kernels (filters). This allows the network to concentrate on low-level features in the first hidden layer, then compile them into higher-level features in the next hidden layer, and so on. The output of the convolutional layer is given by equation (3):

$$\gamma(i,j) = X_{ij} * K \tag{3}$$

Where X_{ij} is the i_{th} and j_{th} matrix from the input matrix corresponding to the convolution kernel K, γ is the output, and the kernel slides over the input matrix in one direction (Appendix B, Figure 19).

The output of the convolutional layers is flattened and fed into the fully connected layer where the mathematical operations take place, as can be seen in equation (4):

$$o = f(\omega_f^T x + b) \tag{4}$$

Where **o** is a vector of the output values, **x** is the vector of the input values, ω_f^T is the vector of the weight values, **b** is the vector of the bias values, and **f** is the activation function[1].

Based on the output of the fully connected layer and the real target, the loss function and the gradient are calculated and the model parameters are updated in the backward pass in order to minimize the error. After the training, the established model is used for making predictions of the test data and the results are compared with the real values by the performance measures to evaluate the model.

3.3 Long short-term memory neural network - LSTM

The Long Short Term Memory (LSTM) is a special type of the recurrent neural network (RNN) architecture presented by Hochreiter et al.[11]. Since RNN has a problem with vanishing/exploding gradients, LSTM was developed to solve this issue. LSTM is composed of an input gate, a forget gate, and an output gate (Appendix C, Figure 20). The forget gate is used to decide which information we need to pass of cell state and which one we need to forget. It can be expressed by the equation (5), where W_f and b_f represent the weights and bias terms respectively. The concatenated form of previous hidden state (h_{t-1}) and current input(x_t) is supplied to a sigmoid function which decides to pass it to cell state or forget it. If the output of the sigmoid function is 0, it means that information needs to be forgotten and 1 means that the input needs to be remembered for further use.

$$f_t = sigmoid(W_f.(h_{t-1}||x_t) + b_f) \tag{5}$$

The input gate is used to update the cell state. The input gate can be understood by equation (6). It takes the input as the concatenated form of the previous hidden state and current input and returns the output using the sigmoid function. This output is multiplied point wise with the output of the tanh function (7), which regulates the network by compressing the input values to be between -1 and 1.

$$i_i = sigmoid(W_i.(h_{t-1}||x_t) + b_i) \tag{6}$$

$$\tilde{C_t} = sigmoid(W_c.(h_{t-1}||x_t) + b_c) \tag{7}$$

In cell state , first, we multiply the previous cell state (C_{t-1}) with the forget gate output point wise and keep the information based on the decision of the forget gate, then update the cell state using the input gate output. It can be expressed by the equation (8):

$$C_t = f_t.C_{t-1} + i_i.\tilde{C_t} \tag{8}$$

The output gate is used to make the predictions and to decide on the next hidden state. According to equation (9) and (10), it takes into account the updated cell state output by regulating it through the tanh function. It passes useful information from the previous hidden state and the input through the sigmoid function and decides what information needs to be carried.

$$O_t = sigmoid(W_o.(h_{t-1}||x_t) + b_o) \tag{9}$$

$$h_t = O_t.tanh(C_t) \tag{10}$$

3.4 Gate Recurrent Unit - GRU

The Gate recurrent unit (GRU) is a simplified version of the LSTM, as presented by Cho et al.[12]. It combines the input gate and the forget gate into the update gate, uses less parameters comparing to LSTM and is composed of the reset gate and the update gate. The basic unit architecture of each GRU cell can be seen in the supplementary material (Appendix D, Figure 21). The reset gate allows us to control how much we want the previous state to remember or forget. According to equation (11), it takes the input and the previous hidden state into account.

$$r_t = sigmoid(W_r.(h_{t-1}||x_t) + b_r) \tag{11}$$

The update gate allows us to control the previous state for the next level. It can be computed using the following equation (12):

$$z_t = sigmoid(W_z.(h_{t-1}||x_t) + b_z) \tag{12}$$

The hidden state or predictions can be computed using the equation (13) by taking into account the reset gate and the update gate.

$$h_t = (1 - z_t).h_{t-1} + z_t.\tilde{h_t} \tag{13}$$

where

$\tilde{h_t} = tanh(W_h(r_t * h_{t-1}||x_t) + b_z)$

4 Data Preprocessing

Finding a good dataset that covers information on the confirmed, death, and recovered COVID-19 cases for a long period and minimal missing values was our first challenge in this project. Since the available data for the Indian states has no missing values, only checking for such values was needed as data cleaning. Because the data has different start points of time in March 2020, we had to set the time period to be the same in order to enable a kind of comparison between the states. We chose to make predictions of the confirmed cases in the five most affected states by COVID-19, namely states which have the highest number of the confirmed cases: Maharashtra, Karnataka, Uttar-Pradesh, Kerala, and Tamil-Nadu. The new confirmed, recovered, and death cases for a given day were computed based on the difference between the total cases of this day and the day before.
The most important task in data preprocessing was converting the data of the six time sequences (total confirmed, total deaths, total recovered, new confirmed, new deaths, and new recovered) into a three dimensional matrix [samples, time step, the features] to be fed into the deep learning models. In other words, the data was split into samples where each sample consists of the data of the six features over n days (the time step), which means that the predictions will be made based on the data of the previous n days. For this purpose, we wrote a function that enables setting the time step to be compared. The previous steps were done basically using NumPy and Pandas.
In addition, we normalized the data using the MinMaxScaler from Sklearn. As variables measured at different scales do not contribute equally to the model fitting, it might end up creating a bias. Therefore, data normalization is required to improve the performance of the deep learning methods. The MinMax normalization process can be expressed by the equation (14):

$$x\_scaled = (x - x\_min)/(x\_max - x\_min) \tag{14}$$

5 Implementation

We implemented the deep neural networks with the help of Python packages, namely Pytorch, Scikit-learn and Keras. Here, we made use of the Python Matplotlib package for visualization tasks. The data was stored in our GitHub repository and the models were trained using google colab's GPU. The code, the dataset, and the results are available on our public GitHub repository [†].

In this section, we are going to discuss the experimental setup, the evaluation metrics, and the implemented architectures with their parameters, which are as follows:

5.1 Experimental Setup

Figure 2 illustrates the experimental setup. First, data of individual states was split into training data (80 %) and testing data (20%). The training data was used for training the models, then the established models made the predictions of the total confirmed cases of the test period. The predictions were compared to the true values by the evaluation metrics Mean Absolute Error (15) and Root Mean Squared Error (16):

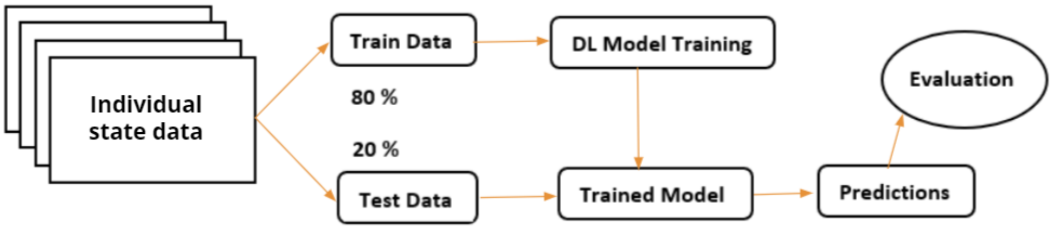


Figure 2. Experimental Setup

$$MAE = \frac{1}{N} \cdot \sum_{n=1}^{N} |y_n - \hat{y_n}| \tag{15}$$

$$RMSE = \sqrt{\frac{\sum_{n=1}^{N}(y_n - \hat{y_n})^2}{N}} \tag{16}$$

[†]https://github.com/sureshkuc/Data-Science-in-Life-Science-Project

where y_n and $\hat{y_n}$ are the true and the predicted values respectively.
To get the scale-free result and to check whether the selected model fits the data well or a further improvement scope is required, we used the coefficient of determination or R-squared score metric (17). The R^2 score is the proportion of the total variance explained by the model and the total variance. The value of the R^2 score will always be less than one and getting a larger value is desirable. R^2 can be calculated by equation 17:

$$R^2 Score = 1 - \frac{\sum_{n=1}^{N} (y_n - \hat{y_n})^2}{\sum_{n=1}^{N} (y_n - \bar{y_n})^2} \tag{17}$$

where $\bar{y_n}$ is the mean of all true values.

All the models were trained with a learning rate of 0.001 and a momentum of 0.9 using Stochastic Gradient Descent (SGD) optimizer[13]. The loss was calculated at the output layer with the help of L1Loss[14] in order to update the model weights.

5.2 CNN

For building the model of CNN, the grid search (trial and test with a set of parameters) was used aiming at determining the best parameters setting. The proposed CNN architecture consists of an input layer, four 1D convolutional layers with 16, 32, 64, and 128 kernels respectively, in addition to a flatten layer, a fully connected layer, and an output layer. After the first and the third convolutional layers, we applied the tanh[15] activation function, whereas the elu activation function[16] was applied after the second and the fourth layers. The output of the convolutional block was then flattened (via the flatten layer) to be fed into the fully connected dense layer which has 128 neurons. These neurons are connected with the output layer of a single neuron. The parameters of the trained model are depicted in Table 2 and the proposed CNN architecture can be seen in Figure 3.

Table 2. CNN Model Parameters

Batch Size	Number of 1D convolution layers	Kernel Size	Epochs	Optimizer	Dense Layer Size
16	4	1x4	100	SGD	128

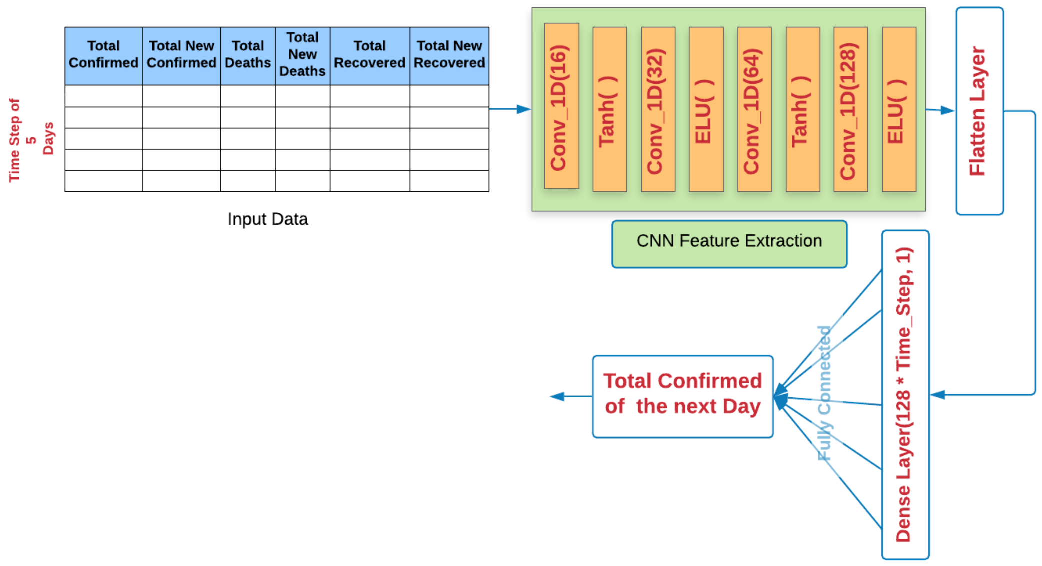


Figure 3. Proposed CNN architecture

5.3 MLP

To implement MLP, we used one input layer, one flatten layer, two hidden layers and one output layer. We tried to set the parameters to get the best model of MLP. The hidden layers (dense layers) were used with 16 neurons. We applied the tanh activation function on the first hidden layer output and the elu activation function on the second hidden layer output. Table 3 illustrates the parameters of the trained model and the implemented architecture is shown in Figure 4.

Table 3. MLP Model Parameters

Batch Size	Number of Hidden layers	Epochs	Optimizer	Hidden Layer Size
16	2	100	SGD	16

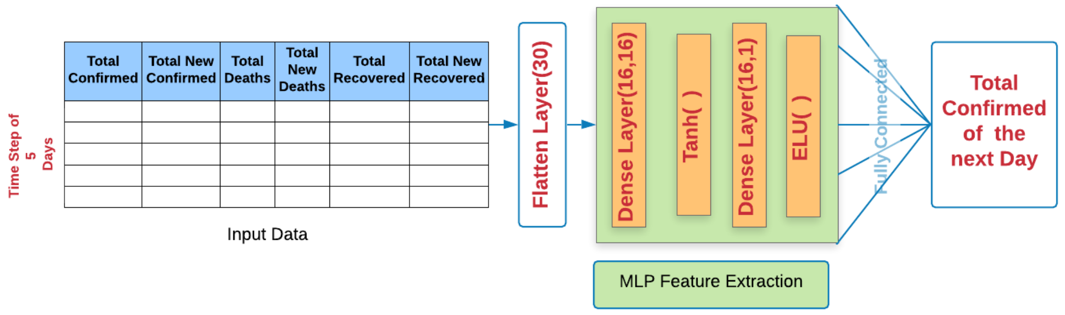


Figure 4. MLP Architecture

5.4 LSTM

Our implementation of the LSTM model has one input layer, one LSTM layer (16 hidden neurons), one fully connected dense layer (80 neurons) and one output layer (single neuron). On the output of the LSTM layer, relu activation function was applied. The parameters of the trained model are listed in Table 4 and the implemented architecture can be seen in Figure 5.

Table 4. LSTM Model Parameters

Batch Size	Number of LSTM Layer	Epochs	Optimizer	Hidden Node Size(LSTM)	Dense Layer Size
16	1	100	SGD	16	80

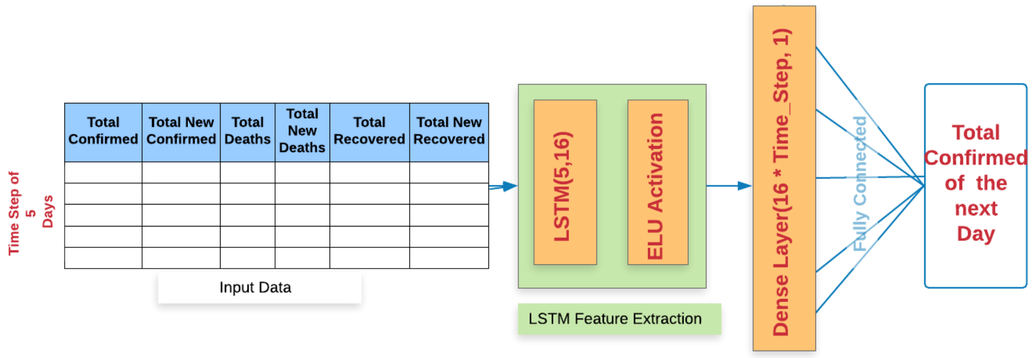


Figure 5. LSTM Architecture

5.5 GRU

In the GRU model, we have an input layer, a GRU layer (16 hidden neurons), a fully connected dense layer (16 neurons), and an output layer (single neuron). The relu[17] activation function was applied on the output of the GRU layer. The parameters of the trained model and the implemented GRU architecture can be seen in Table 5 and Figure 6 respectively.

Table 5. GRU Model Parameters

Batch Size	Number of GRU Layer	Epochs	Optimizer	Hidden Node Size(LSTM)	Dense Layer Size
16	1	100	SGD	16	16

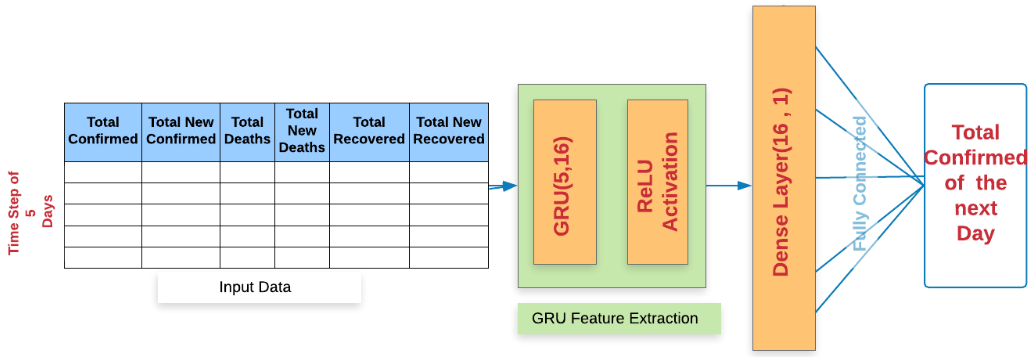


Figure 6. GRU Architecture

6 Experimental results

The data from March 10, 2020 to June 30, 2021 (Long Time Period) and from March 10,2020 to June 18, 2020 (Short Time Period) were used for training the deep neural network models (80% for training and 20% for testing). In this section, we are going to discuss the results of long time period and short time period individually.

6.1 Short time period

The scores of R^2, MAE and RMSE were calculated for each state on the respective test set and listed in Table 6. In comparison to the other models, We found out that the CNN is the best model with the lowest error. The R^2 score resulted 0.981, 0.992, 0.994, and 0.931 for the states Uttar-Pradesh, Tamil-Nadu, Maharashtra, Kerala, and Karnataka respectively. Since the R^2 score of CNN is above 0.90 for all states, it shows that the model of CNN fits the data very well. It has also the lowest MAE and RMSE error scores for all the states comparing to the other models. By looking at the radar charts in Figure 7, we found the

Table 6. R^2 SCORE,AVERAGE MEAN ABSOLUTE ERROR (MAE) AND ROOT MEAN SQUARED ERROR (RMSE) OF THE DEEP NEURAL NETWORKS

	R^2 score				MAE				RMSE			
State	**CNN**	MLP	LSTM	GRU	**CNN**	MLP	LSTM	GRU	**CNN**	MLP	LSTM	GRU
Uttar-Pradesh	**0.981**	0.891	0.977	0.899	**0.013**	0.037	0.015	0.025	**0.017**	0.040	0.018	0.034
Tamil-Nadu	**0.992**	0.945	0.992	0.978	**0.011**	0.026	0.012	0.016	**0.012**	0.034	0.013	0.019
Maharashtra	**0.990**	0.923	0.990	0.741	**0.009**	0.026	0.010	0.040	**0.010**	0.030	0.011	0.049
Kerala	**0.994**	0.661	0.833	0.308	**0.008**	0.060	0.043	0.076	**0.009**	0.077	0.045	0.091
Karnataka	**0.931**	0.930	0.892	0.907	**0.024**	0.033	0.030	0.038	**0.029**	0.036	0.036	0.042

best models which are the innermost for the MAE and RMSE and outermost for the R^2 score. So, CNN is the best model comparing to the other models with data of the short time period.

6.2 Long time period

By applying the same proposed structures with the data of the long time period, the results did not differ that much. CNN showed also a high efficiency based on the scores of R^2, MAE, and RMSE, which can be seen in Table 7. However, GRU performed well in this case with data of Maharashtra, and small differences between the error scores of CNN and GRU were found.

We plotted the radar charts for MAE, RMSE and R^2 score by taking all the models and states into account, which can be seen in Figure 8. Based on the radar charts for the MAE and RMSE, we can conclude that the innermost one is the best model among all models, which is CNN and based on the radar chart for the R^2 score, the outermost is the best model, which is also CNN.

The predictions of the CNN model as well as the actual test data are compared in Figure 9.

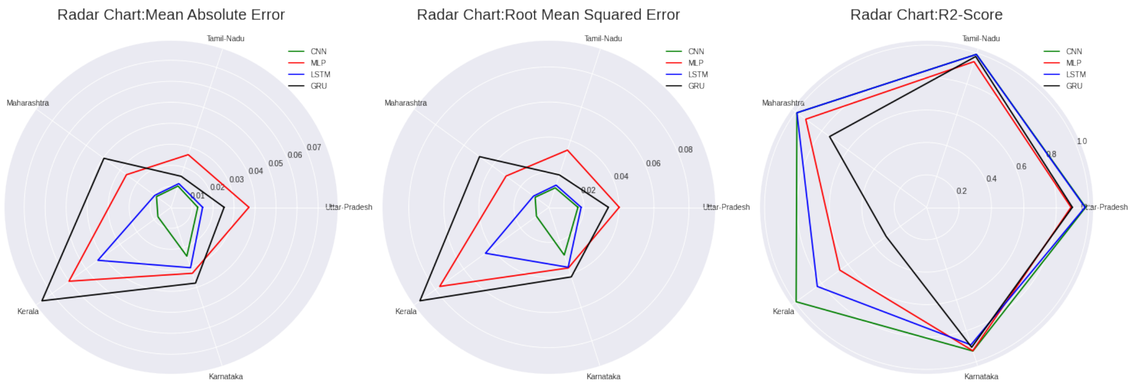


Figure 7. Model Results of a short time period data between 10 March 2020 and 18 June 2020

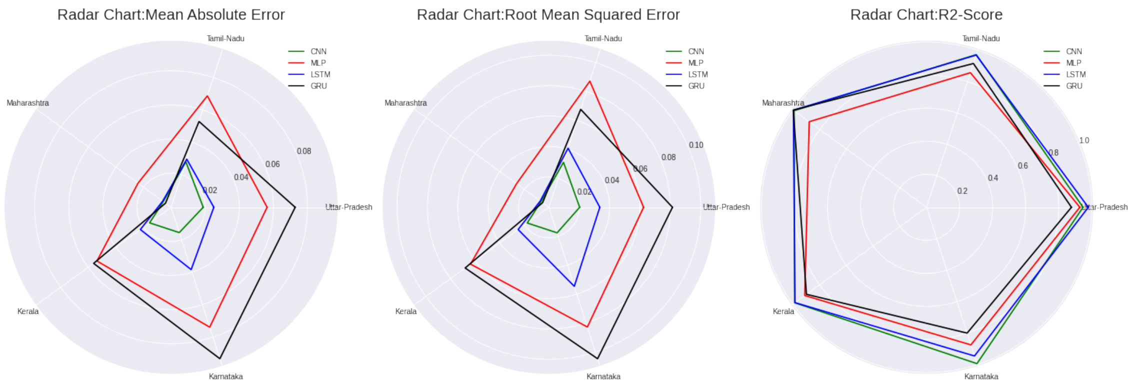


Figure 8. Model Results of a long time period data between 10 March 2020 and 30 June 2021

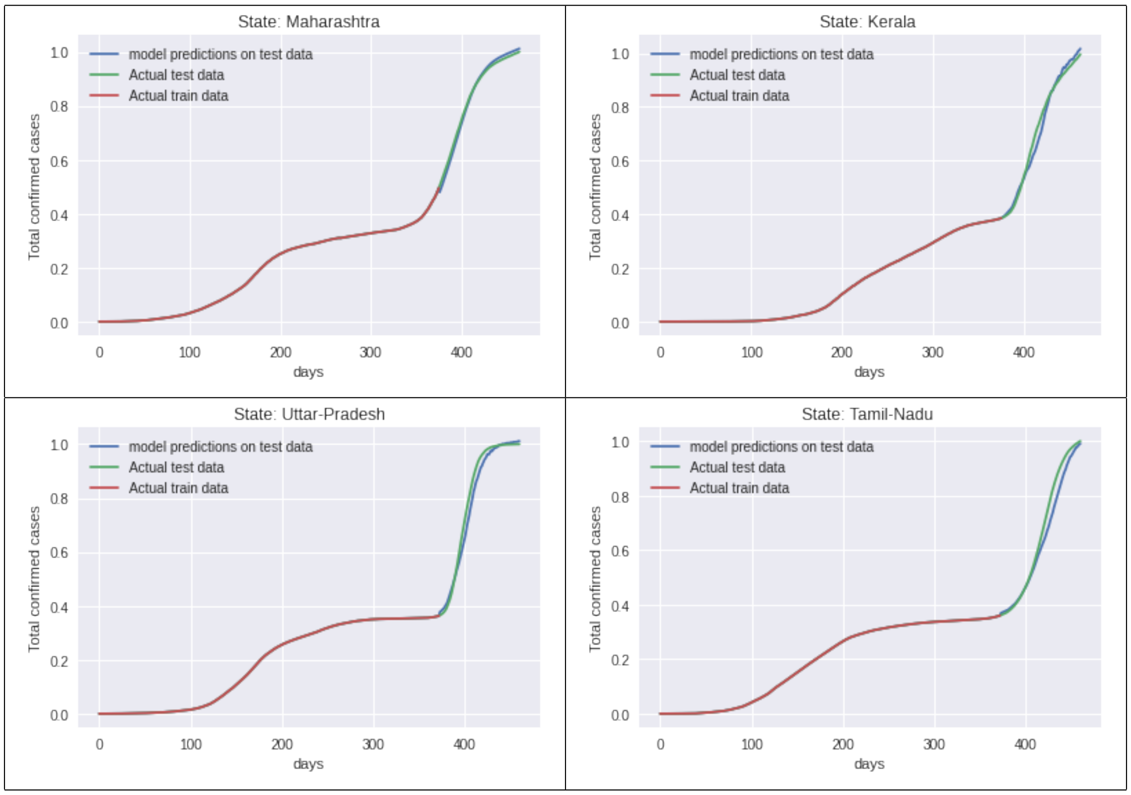


Figure 9. Proposed Model(CNN) Predictions

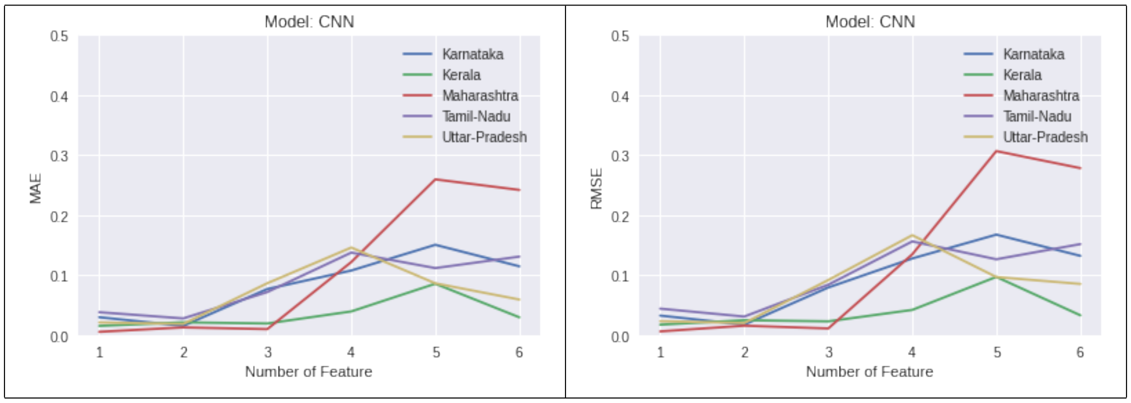


Figure 10. Proposed Model(CNN): number of features vs error evaluation

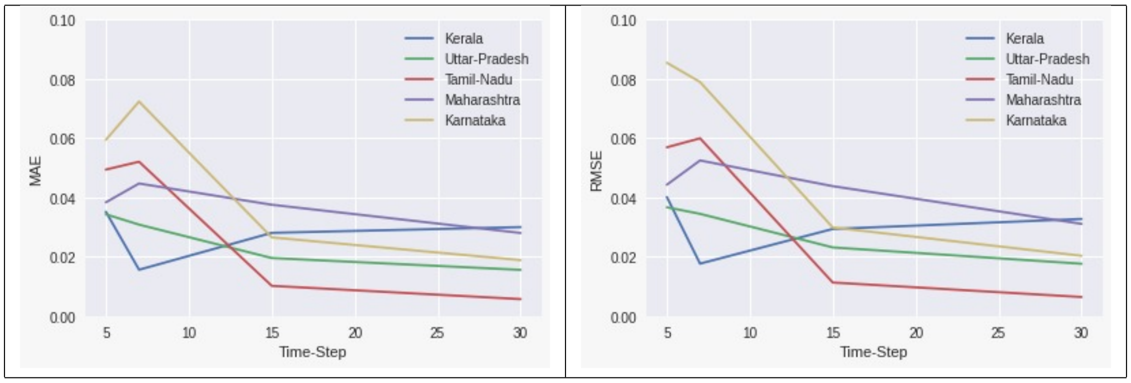


Figure 11. Proposed Model(CNN): timesteps vs error evaluation

Table 7. R^2 SCORE,AVERAGE MEAN ABSOLUTE ERROR (MAE) AND ROOT MEAN SQUARED ERROR (RMSE) OF THE DEEP NEURAL NETWORKS

	R^2 score				MAE				RMSE			
State	**CNN**	MLP	LSTM	GRU	**CNN**	MLP	LSTM	GRU	**CNN**	MLP	LSTM	GRU
Uttar-Pradesh	0.947	0.928	**0.979**	0.877	**0.019**	0.056	0.025	0.073	**0.020**	0.062	0.034	0.081
Tamil-Nadu	**0.972**	0.857	0.969	0.915	**0.028**	0.069	0.030	0.053	**0.031**	0.087	0.041	0.068
Maharashtra	0.993	0.880	0.998	**0.999**	0.006	0.024	0.006	**0.004**	0.006	0.026	0.007	**0.005**
Kerala	**0.986**	0.911	0.984	0.898	**0.016**	0.054	0.022	0.056	**0.018**	0.064	0.025	0.068
Karnataka	**0.994**	0.875	0.946	0.800	**0.016**	0.074	0.038	0.093	**0.018**	0.083	0.055	0.105

7 Discussion I

In this section, we broach the previous methods in the light of the different studies that have recently proposed deep learning methods for predicting COVID-19 cases in several countries and depending on the results of the seed paper as well as the experimental results of our implementation.

For this project, we compared the performance of the four methods using data of different time periods (short vs. long) as we mentioned, as univariate and multivariate time series forecasting, and for different time steps, namely predictions based on the data of the previous five days vs. predictions based on the data of the previous seven days. In terms of our evaluation metrics, the results revealed that CNN has the best performance based on averaging the error scores for both short and long time periods, whereas increasing the time step improved the accuracy of CNN (Figure 11) and the other models for most of the states (Appendix F, Figure 27. Furthermore, increasing the number of the input features decreased the efficiency of the models, which can be seen in an increase of the MAE and RMSE and a later decrease of the errors for most of the states. For CNN this can be seen in Figure 10 and the impact on the other models can be seen in the supplementary material (Appendix F, Figure 26).

In the study by Nabi et al.[8], CNN outperformed the other methods as well. The same results were presented also in the seed paper[1] where the authors made use of a new activation function proposed by Wang et al.[18]. We could not apply the same activation function since it is not among the activation functions supplied by Keras or Pytorch. Even though, we could implement a CNN model with four convolutional layers, exceed the problem of gradient disappearance, and achieve the expected high accuracy.

On the other hand, the influence of the data itself is worthy to be mentioned, for instance with data of the long time period, GRU has the best performance for data of Maharashtra, and LSTM showed high efficiency in this case. With data of the short time period, the performance of MLP improved significantly in comparison to the performance of GRU with the same data and its performance with data of the long time period. In this regard, in the seed paper also in the study of Omran et al.[2] where data of relatively short time period was used, the results of LSTM and GRU varied between the cities to confirm again the role of data. One other important factor to be considered is the proposed structure including the number of layers, the activation function, and the number of the neural units.

It is also notable that using models of hybrid methods as in Dutta et al.[3] and Frausto-Solís et al.[5] can significantly improve the results. Therefore and as a further work, this study can be extended by implementing hybrid architectures.

To conclude the discussion, CNNs are generally one of the best candidate methods for time series forecasting. However, building a deep learning model that returns a high accuracy is a sensitive task affected by many factors, basically, choosing the appropriate method and the appropriate structure including setting the number of the layers and the parameters. Tuning the hyperparameters for deep neural networks is the most difficult and critical part because it is slow to train and there are numerous parameters to configure. Furthermore, the impact of the data nature, its fluctuation, length, and the correlation between the features on the results. Thereby, tuning the hyperparameters and comparing different methods are strongly recommended in practice aiming at the best performance.

8 Data Exploration

The first look at the dataset included a comparison of the different features per state and between the states. We visualized the confirmed cases for each state as a comparison in Figure 12. Here we can see that Maharashtra has the highest numbers overall and especially a high increase in April 2021. What we can also mention here is the fact that the numbers of all states are steadily increasing since the beginning of 2021. We also took a look at the recovered and death cases comparison between the states, where you can find the respective figures in Appendix E. The sudden increase in the curves around April and May 2021 is also visible in the recovered cases (Appendix E, Figure 22). In the deaths comparison between the states (Appendix E, Figure 23) it is notable that Kerala has a rather flat curve and the lowest death rate overall. We also made a comparison of the new confirmed cases instead of just the total number that you can see in Figure 13. Here again it is visible that Maharashtra has

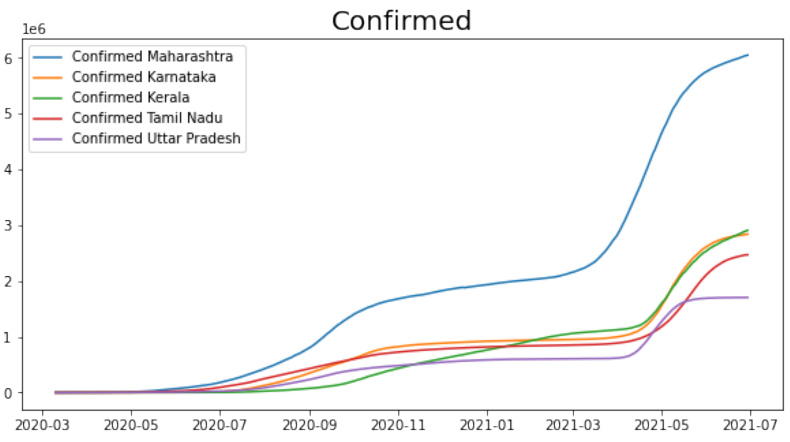


Figure 12. Comparison of the confirmed cases per state.

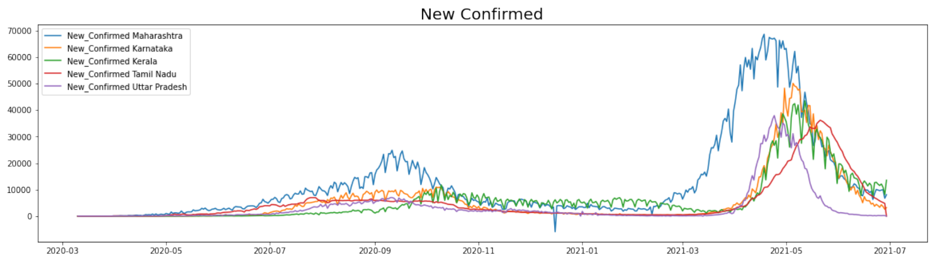


Figure 13. Comparison of the new confirmed cases per state.

the highest peaks of the curves and we have two time points with an overall increase in new confirmed cases, namely autumn 2020 and spring 2021. Uttar Pradesh has the lowest number of new confirmed cases at both peaks of the curves and is especially flat during the first wave. In Figure 24 of Appendix E all features for the state Maharashtra are visualized. As before we can see the bigger impact of the second wave in April/May 2021. There is also a peak of new deaths in June 2020. As a comparison we can also have a look at the overall features for the state Kerala in Figure 25 of Appendix E, where we can also clearly see the second wave.

The huge difference in numbers is due to the population per state, which is highest in Uttar Pradesh, followed by Maharashtra out of the five compared states. In the following Table 8 you can see the population numbers per state in a descending order.

Table 8. Population Overview

State	Population in 2021
Uttar Pradesh	241,066,874
Maharashtra	129,877,541
Tamil Nadu	82,722,262
Karnataka	70,462,375
Kerala	35,336,581

9 Discussion II

In this section, we want to emphasize the understanding of the data we explored beforehand. For that purpose we want to find out factors that partly influenced the numbers in each state, such as population number, population density, the location of the state and its health care system, as well as social events and government interventions. Our comparison can be found in the following Table 9.

Even though we saw in the previous section in Table 8 that Uttar Pradesh and Maharashtra are the states with the highest population number, Kerala is due to its forest areas the state with the highest population density. We thought that this would be

Table 9. States Overview

State	Location	Population density	Health care	Social Events	Government interventions
Kerala	South, Malabar coast	860 / km²	robust during 1st wave, can't keep up since 2nd wave, best testing ratio	Elections in April 2021	Contact tracing, targeted tests
Uttar Pradesh	North, Taj Mahal	828 / km²	Bad conditions in hospitals, vaccine shortages	Holi festival, Migrant workers crisis	Contact tracing, educational institutions, Lockdown
Tamil Nadu	South, Hindu temples	550 / km²	85 labs for tests, >75k beds	Religious congregation event	Contact tracing, testing, surveillance, medical supplies
Maharashtra	West/Central, Mumbai	370 / km²	High testing rate	Mobility, Religious events, mass flouting of covid norms	Contact tracing, targeted tests
Karnataka	South West, Arabian Sea	319 / km²	10k tests/day, 74 labs, >21k beds reserved	Religious events	Early containment efforts, Closure of borders with Kerala, Lockdown

an indicator for higher numbers in the new confirmed cases, because the disease would spread easier through people in dense space, but surprisingly Kerala is as mentioned before the state with a rather flat curve regarding the confirmed and death cases. This could be reasoned by their very early government interventions like contract tracing and targeted tests, but is also due to its rather good health care system and testing ratio. However, as elections took place in April 2021 and the robust health care system that lasted through the first wave couldn't keep up with the second wave anymore, the peak in the curves around spring 2021 can be well explained.

Like the elections in April 2021 in Kerala, there are also social events that can be reported from the other states, e.g. the Holi Festival in Uttar Pradesh, religious events and also general mobility and migration between the states, as well as mass flouting of covid norms. Together with the general second infection wave this influenced the numbers during spring 2021 greatly.

Overall, we can observe that in all five states the governments made intervention efforts quite early. Aside from contact tracing and targeted tests, educational and other institutions were closed, lockdowns were issued, borders with other states were closed and medical supplies were organized. Looking at the health care system Tamil Nadu and Karnataka have most of the labs in India, enabling further testing and a high testing ratio in these states. However, the hospital conditions are somehow bad, especially in Uttar Pradesh and also in other states since the second wave. Medical supplies are still missing and vaccine shortages were reported.

10 Project History

10.1 Working together and organization

During the semester we had several tasks to complete: working on our presentations and the report. In order to improve our communication and agreements, we wrote them down in Google documents that have shared access, so that any of us can look at them and edit them anytime, including a change history and comment section.

Like this, we also created the presentations with Google presentations, so that we can work on them simultaneously and comment on each other's part in order to discuss it in our group meetings. We met once or several times a week, depending on the workload, and used the Webex Meeting possibility for that. The code of the implementation was also shared since we worked on it using Google Colaboratory. There, you can make a python notebook, which makes it easier to understand, comment on the workflow later. and refine the things one has done so far. To save the dataset and the respective results we share the files through a Github repository. We also communicated and set the meeting via the messenger WhatsApp.

10.2 Difficulties, failures and bad experiences

The first thing we had to deal with was the fact that we live in different time zones. Germany and India are five hours apart, which means that meetings set at 6 pm here in Germany are at around 11 pm in India. We all had completely different schedules during the semester because we are in different study programs, different semesters, and chose different courses, which is why most of the meetings ended up in the evening after all the other courses or even on a Saturday evening.

At first, we were pretty sure that our organization was good since we wrote down the main points of our meetings and also the course meetings and comments on our presentation in a shared file. For instance, we also wrote down the states we chose in the beginning by looking at the data, the long and short time period we want to look at, the deadlines and also the tasks discussed in our group meetings, who the respective task is assigned to and what is expected from it. What we surely didn't expect was that such written assignments or agreements are not kept, but that is exactly what happened. So for one thing this was the bad experience we had to live through during this group work.

Also, if the group had a meeting on the finalized version of a presentation, the slides were changed two hours before the presentation on the same day, which stressed out the other group members, even though one should be able to rest and relax before a presentation to have a clear mind during it. Luckily for us, this happened only during the intermediate presentations, not the final one, so that we were able to improve and set clear deadlines already days before the actual presentation day with frequent meetings to ensure it doesn't happen again. Still, we would say that this was a failure that could have been avoided if all of us would have kept an eye more on the others while working on the tasks and more regular updates.

Speaking of which, another complexity was the work on the code and report. Initially, we were not on same line which package we should use for the method implementation. So, some members implemented it using keras and some used pytorch without discussing the prons and cons of both. All in all we finalized the model results and the visualizations with pytorch and only compared the overall results with our implementations in keras. However, in our opinion this situation was very difficult to handle due to the lack of communication.

On the other hand in context of report, agreeing on the parts each member was supposed to work on and setting a date to meet again, not all group members where finished on that day, which was only some time before the final deadline. Here again, the missing parts where only added in a haste, forcing the other group members to rewrite it. Errors in Latex produced by this hasty adding of text also caused the need for correction, which was also difficult on a level of communication, since the group member couldn't be reached well. However, we were able to manage it and completed it before the deadline.

Aside from also some misunderstandings during meetings (maybe language-related) and communication failures, we don't have any more bad experiences or failures to report.

10.3 Successes and good experiences

At first, we were worried about the dataset we were going to use for our implementation. We couldn't find the original dataset from our seed paper, but this wasn't the thing worrying us, since we wanted to adopt the idea of the study especially for a different case, i.e. a different country. The problem was that in the seed paper a dataset was used with data per city. This was difficult to find if the features were to be the same as well. We first found a dataset of the United States, but since there were features missing, we decided on the dataset on the Indian states and are happy with it, since it includes exactly the data we need and is even updated regularly, providing us with a long time period.

We are also relieved that we managed to implement all four methods from the seed paper since it takes time to grasp the theory behind them and work through tutorials in order to be able to implement them. In that regard but also in general, working in a group really eased the stress of having deadlines to keep and giving presentations.

Also, since we were free to use whatever programming language we wanted to, we chose Python, which has a wide variety of libraries and packages helping with artificial neural networks (like PyTorch and Keras) and the respective guides, manuals, and tutorials for a better and deeper understanding of it.

11 Conclusion

The challenge of countering COVID-19 highlighted many questions on the efficiency of the used AI tools, especially for time series forecasting that had wide reflections on the quality of the decision-making process by governments. It was most critical in the first months when few amounts of data were available. This motivated us, as a team, to detect factors that should be considered dealing with such a defying situation.

In this project, we focused on two dimensions. First, studying four deep learning methods that have been commonly used for predicting COVID-19 cases. We explained these methods, compared them, and discussed factors that influence their performance. We supported our discussion with the experimental results of our implementation with data of COVID-19 in India, in addition to studying the results of the related work on predicting COVID-19 cases in different countries via deep learning methods.

Since data analysis is not only about making predictions, the second dimension of this work is based on applying data exploration analysis to detect the whole picture of COVID-19 in India. We went further in interpreting the results and factors that have a significant impact on the rates of confirmed and death cases of COVID-19 like the location, the population density, the social events, the government´s interventions, and the health care in India have been broached.

References

1. Huang, C.-J. Multiple-input deep convolutional neural network model for covid-19 forecasting in china. *MedRxiv* (2020).
2. Omran. Applying deep learning methods on time-series data for forecasting covid-19 in egypt, kuwait, and saudi arabia. *Complexity* (2021).
3. Bandyopadhyay, S. K. & Dutta, S. Machine learning approach for confirmation of covid-19 cases: Positive, negative, death and release. *medRxiv* (2020).
4. Arora, P. Prediction and analysis of covid-19 positive cases using deep learning models: A descriptive case study of india. *Chaos, Solitons Fractals 139* (2020).
5. Frausto-Solís, J., Hernández-González, L. J., González-Barbosa, J. J., Sánchez-Hernández, J. P. & Román-Rangel, E. Convolutional neural network–component transformation (cnn–ct) for confirmed covid-19 cases. *Math. Comput. Appl.* **26**, 29 (2021).
6. Wang, L., Lin, Z. Q. & Wong, A. Covid-net: A tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images. *Sci. Reports* **10**, 1–12 (2020).
7. Mohimont, L., Chemchem, A., Alin, F., Krajecki, M. & Steffenel, L. A. Convolutional neural networks and temporal cnns for covid-19 forecasting in france. *Appl. Intell.* 1–26 (2021).
8. Nabi, K. N., Tahmid, M. T., Rafi, A., Kader, M. E. & Haider, M. A. Forecasting covid-19 cases: A comparative analysis between recurrent and convolutional neural networks. *Results Phys.* **24**, 104137 (2021).
9. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain, cornell aeronautical laboratory. *Cornell Aeronaut. Lab.* (1958).
10. Géron & Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems* (O'Reilly Media, 2019).
11. Hochreiter. Lstm can solve hard long time lag problems. *Adv. neural information processing systems* 473–479 (1997).
12. Cho. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).
13. Torch Contributors. Stochastic Gradient Descent in Pytorch. https://pytorch.org/docs/stable/optim.html (2021).
14. Torch Contributors. L1Loss. https://pytorch.org/docs/stable/generated/torch.nn.L1Loss.html (2021).
15. Torch Contributors. Tanh Function. https://pytorch.org/docs/stable/generated/torch.nn.Tanh.html (2021).
16. Torch Contributors. ELU Function. https://pytorch.org/docs/stable/generated/torch.nn.ELU.html (2021).
17. Torch Contributors. RELU Function. https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html (2021).
18. Wang, Y. The influence of the activation function in a convolution neural 419 network model of facial expression recognition. *Appl. Sci.* (2020).

Supplementary

Appendix A - Multilayer Perceptron

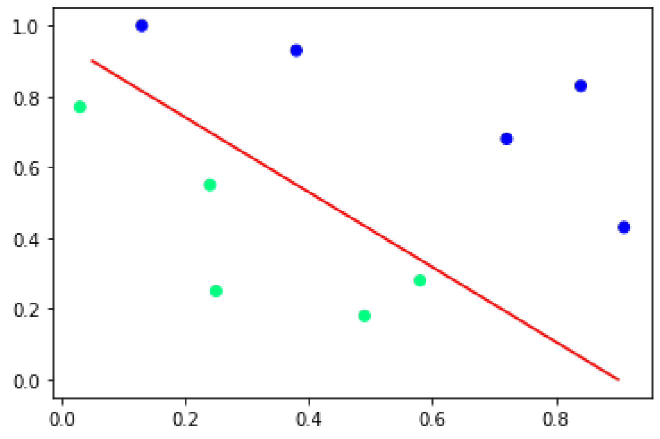
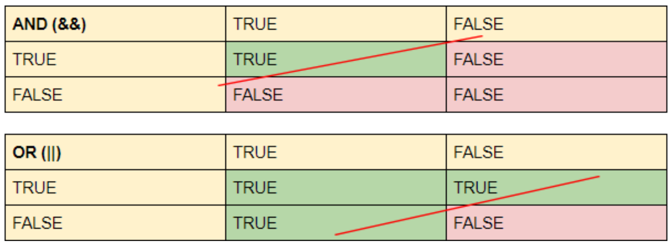


Figure 14. Linearly separable data set.



AND (&&)	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

OR (||)	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

Figure 15. Logical operators AND and OR are linearly separable.

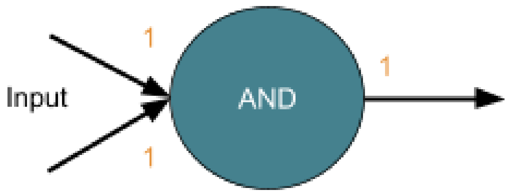
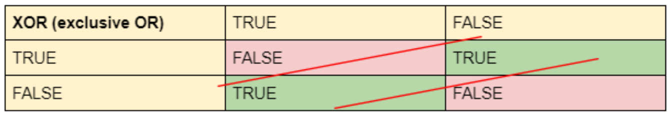


Figure 16. Single Perceptron Model

In case of the logical operator AND, we have our Perceptron, a single neuron which performs the AND operation (node), and one or more Inputs (black arrows). The inputs here have all the number 1, hence the output we receive from the Perceptron should be also the number 1.



XOR (exclusive OR)	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE

Figure 17. Exclusive OR: XOR

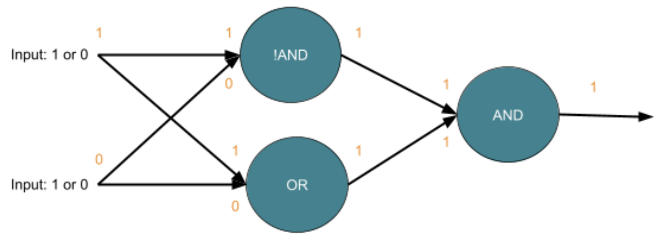
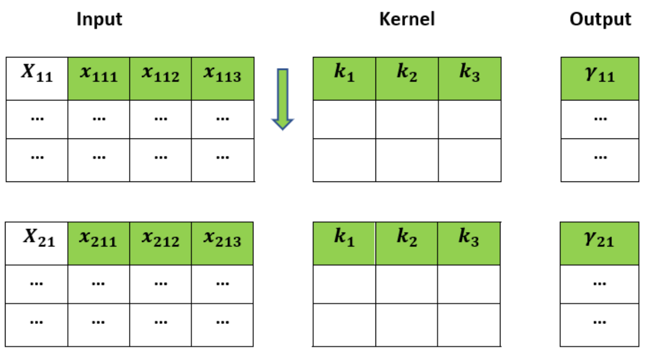


Figure 18. Model of the exclusive OR: XOR

We can calculate the exclusive OR by first calculating the negated AND and the OR and putting the respective results together with the AND operator in the end.

Appendix B - Convolutional Neural Network



$$y_{ij} = X_{ij} * K = x_{ij1} * k_1 + x_{ij2} * k_2 + x_{ij3} * k_3$$

Figure 19. The convolutional process

Appendix C - Long Short Term Memory

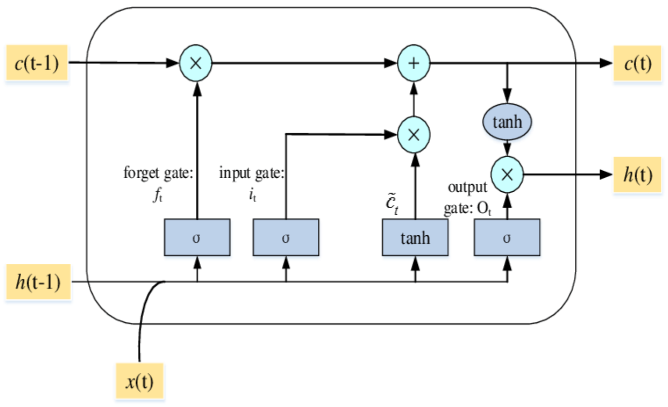


Figure 20. The structure of the Long Short Term Memory model.

Appendix D - Gate Recurrent Unit

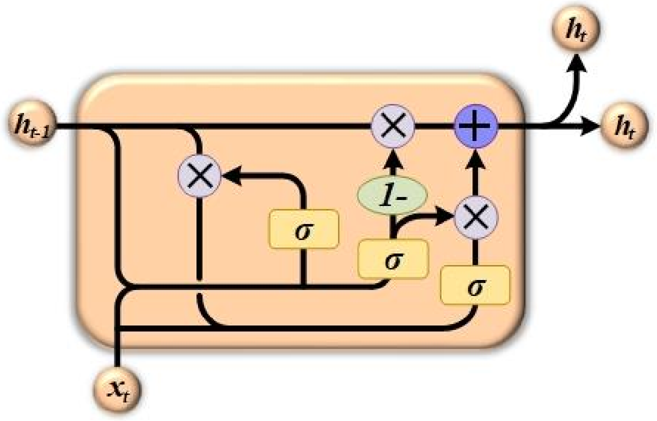


Figure 21. The structure of Gate Recurrent Unit[1]

Appendix E - Data Exploration

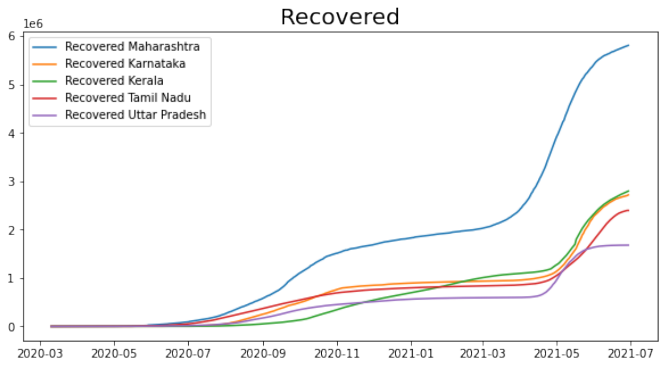


Figure 22. Comparison of the recovered cases per state.

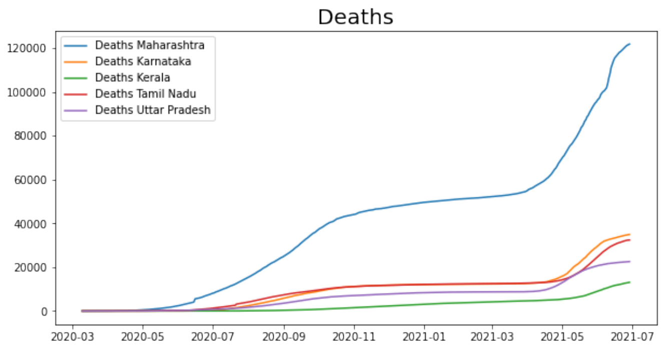


Figure 23. Comparison of the deaths per state.

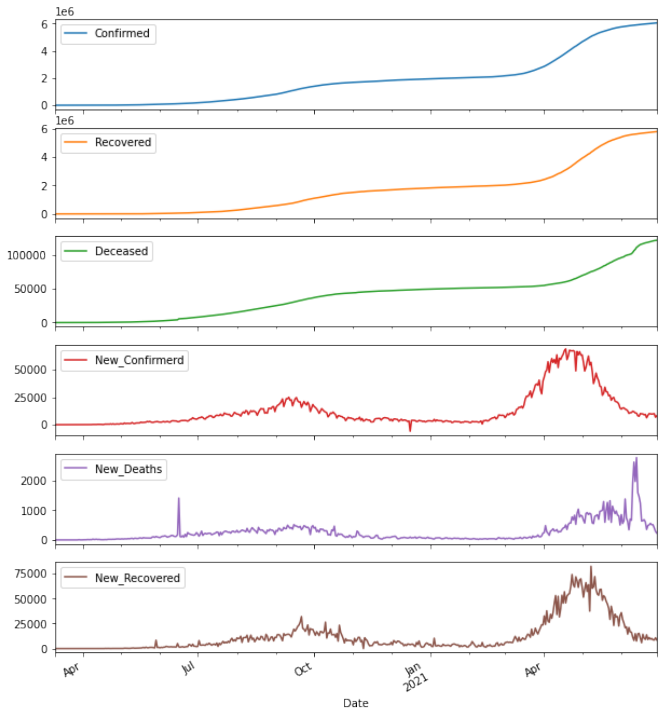


Figure 24. Comparison of the different features in Maharashtra.

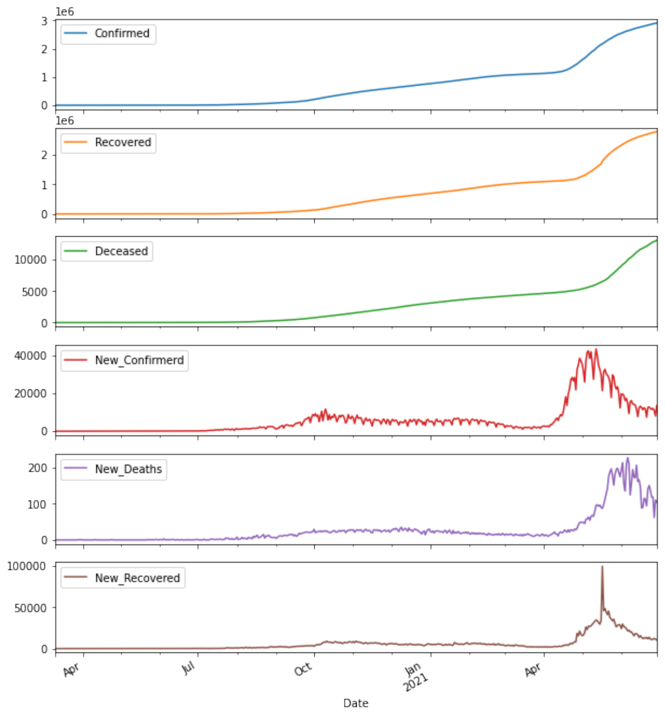


Figure 25. Comparison of the different features in Kerala.

Appendix F - Model evaluation on features and timesteps used

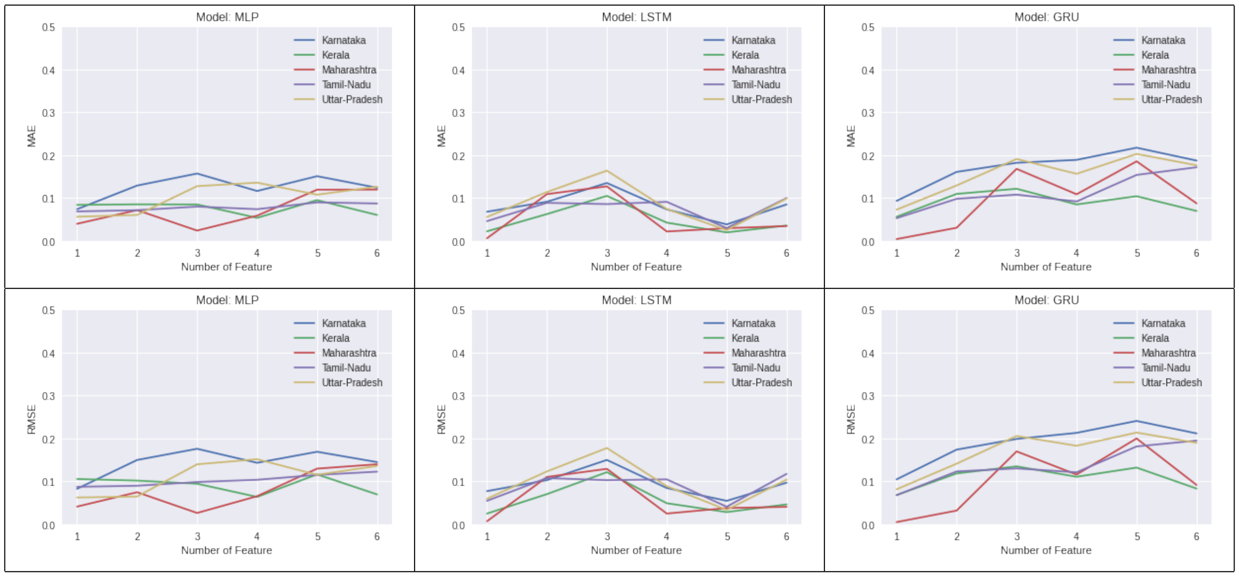


Figure 26. Number of Features vs Error Evaluation for the Methods MLP, LSTM and GRU

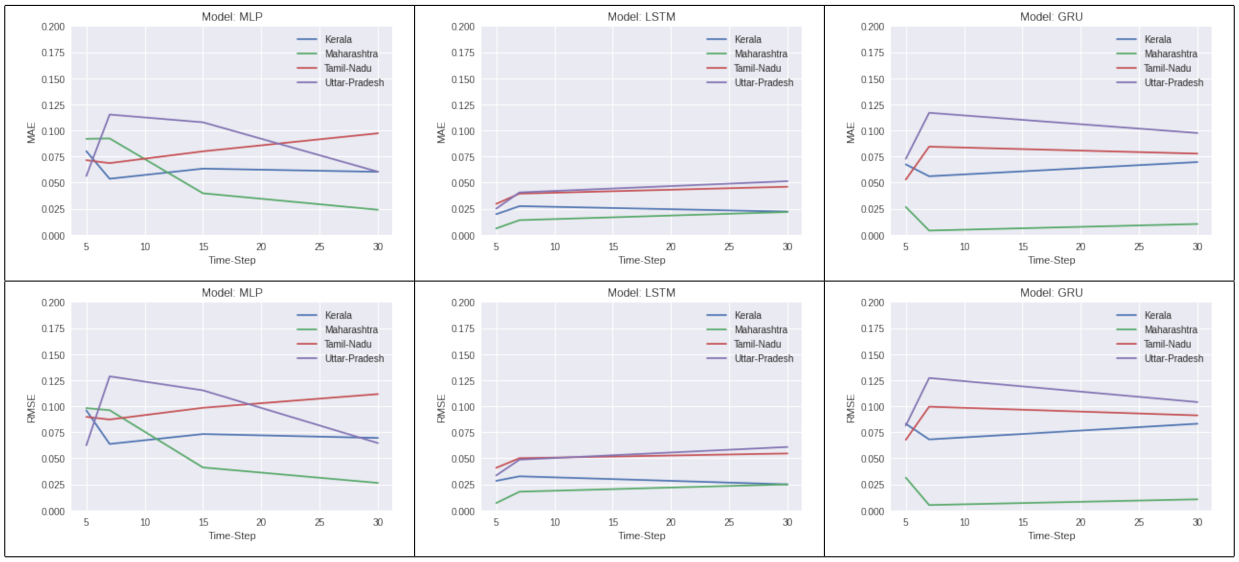


Figure 27. Time Step vs Error Evaluation for the Methods MLP, LSTM and GRU