ISSN:1738-6764, eISSN:2093-7504

Forecasting COVID-19 confirmed cases in South Korea using Spatio-Temporal Graph Neural Networks

Kien Mai Ngoc [1,2] and Minho Lee [1,2,*]

[1] University of Science and Technology; Student Researcher, kienmn@kisti.re.kr;
[2] Korea Institute of Science and Technology Information; Principal Researcher, cokeman@kisti.re.kr
* Correspondence

https://doi.org/10.5392/IJoC.2021.17.3.001

Manuscript Received; 23 June 2021 Received; 02 September 2021 Accepted 02 September 2021

Abstract: *Since the outbreak of the coronavirus disease 2019 (COVID-19) pandemic, a lot of efforts have been made in the field of data science to help combat against this disease. Among them, forecasting the number of cases of infection is a crucial problem to predict the development of the pandemic. Many deep learning-based models can be applied to solve this type of time series problem. In this research, we would like to take a step forward to incorporate spatial data (geography) with time series data to forecast the cases of region-level infection simultaneously. Specifically, we model a single spatio-temporal graph, in which nodes represent the geographic regions, spatial edges represent the distance between each pair of regions, and temporal edges indicate the node features through time. We evaluate this approach in COVID-19 in a Korean dataset, and we show a decrease of approximately 10% in both RMSE and MAE, and a significant boost to the training speed compared to the baseline models. Moreover, the training efficiency allows this approach to be extended for a large-scale spatio-temporal dataset.*

Keywords: COVID-19 forecasting; Graph neural networks; Deep learning.

1. Introduction

1.1. An overview of the study

On February 6th, 2021, COVID-19 has caused more than 105M confirmed cases and over 2.3M deaths globally [1]. To prepare, understand, and control the spread of the disease, researchers worldwide have come together in a collaborative effort to model and forecast COVID-19 [2]. In the state of pandemic, the ability to accurately forecast caseload is extremely important to help inform policymakers on how to provision limited healthcare resources, rapidly control outbreaks, and ensure the safety of the general public [2].

Forecasting COVID-19 cases is a time series prediction problem in which given M observations, we need to predict the most likely number of cases in H following time steps.

$$\hat{v}_{t+1}, \dots, \hat{v}_{t+H} = \underset{v_{t+1},\dots,v_{t+H}}{\operatorname{argmax}} \log P(v_{t+1}, \dots, v_{t+H} \mid v_{t-M+1}, \dots, v_t) \tag{1}$$

where v_t is an observation of the number of confirmed cases at time step t.

Time series learning approach is popular in forecasting the next number of cases, for example, applying curve-fitting [3], Autoregression (AR) [4], or deep learning [5-7] on time series data. These approaches rely on the information from the time series. They can learn some useful insights or hidden patterns from the time series and use those to forecast next items in the sequence.

However, the development of these time series as well as the epidemic in general is affected by many other factors, such as people movement, weather, geography, etc. Therefore, in this work, we aim to incorporate information from spatial data (geographic location) with temporal data (time series) to make prediction about the number of cases for all locations at the same time. It is very natural and suitable to model our data using the structure of a graph where nodes present regions, edges present the geographic distance among regions, and the temporal features are the features of node through time.

This work is inspired by spatio-temporal graph neural network which is successfully applied in traffic field, for example STGCN [5], DCRNN [6], and the work of Amol Kapoor et al. [2] in COVID-19 forecasting. However, we cannot employ complicated models from traffic field, in which the data is recorded by minutes instead of days like ours. Therefore, we aim to improve the performance from the work of Amol Kapoor et al. [2], with a simpler model than that in traffic [8,9].

We evaluate the effectiveness of the approach when predicting the daily infection cases using a real COVID-19 dataset in 17 provinces of Korea. Based on the previous works, we propose a spatio-temporal graph neural network that can learn dynamic features from both spatial and temporal domains. Subsequently, we use this model to forecast COVID-19 daily new cases for all provinces at once.

Through this work, our contributions are:

- We propose a graph neural networks-based model for COVID-19 forecasting which can incorporate both spatial and temporal data. The architecture allows the model to process all time series simultaneously. With its training efficiency, the model can be extended for large-scale dataset.
- Our proposed model is superior to several models, but not as complicated as the one used in traffic field.
- A case study when we apply our proposed method on a real COVID-19 dataset in Korea.

The remaining of this paper is organized as follows. We briefly introduce related works in the rest of this section, including time series forecasting and graph neural networks. In Section 2, we first present our approach, from an overview to details of each component. We then present about experimental setup, including dataset, data preprocessing, and baseline models. The results are shown in Section 3 and discussed in Section 4. Finally, we end up this paper with conclusions in Section 5.

1.2. COVID-19 and time series forecasting

In time series analysis, one of the most consolidated classical statistics-based methods is autoregressive integrated moving average (ARIMA) and its variants [6]. Some studies have been done to apply ARIMA to COVID-19 time series data [10-11]. However, this method is constrained by the stationary assumption of time series and is incapable of dealing with the spatial and temporal dependencies. It is also infeasible to apply a large model for multiple time series. Nowadays, classic statistical methods have been overwhelmed by machine learning models, especially neural networks as higher prediction accuracy and more complicated data representation can be obtained by these models. In this work, we will not consider ARIMA as a baseline model because our effort is not to optimize models for each time series but a model performing well for all time series.

In past few years, recurrent neural network (RNN) [7] shows its effectiveness in dealing with sequence data, such as connected handwriting recognition [8] or speech recognition [14-15]. Long short-term memory (LSTM) [16-17], an artificial RNN architecture, was proposed to process long sequence dependencies and to avoid vanishing or exploding gradients problem. LSTMs are the efficient algorithms to build a time series sequential model and can be applied for COVID-19 data [18-19].

Sequence to Sequence (Seq2Seq) [9] was originally invented for neural machine translation. Attention mechanism from Bahdanau's paper [10] and Luong's paper [11] helped to boost the performance of the Seq2Seq model. Because it can deal with sequence data, it can be adapted to process COVID-19 time series data [2].

1.3. Graph neural networks

Recently, graph neural network has drawn more attention because of the great expressive power of graphs [12]. There are two approaches have been proposed to generalize convolution neural networks (CNN) for graph structure [5]. Those are to apply convolution operation on certain grid forms of rearranged vertices or in spectral domain with graph Fourier transforms [13]. Several studies follow the latter approach to reduce the computational complexity of graph convolution using approximation [25-26].

Spatio-temporal graph neural network has been applied successfully in traffic, such as STGCN [5], DCRNN [6], etc. In traffic prediction on road graphs, the problem is to predict the most likely traffic measurements (e.g., traffic flow or speed) in the following time steps for n road segments. By defining the traffic network on a graph and focusing on structured traffic time series, they can predict for short and long terms with relatively low errors.

The work related directly to this study is from Amol Kapoor et al. [2]. They try to incorporate mobility data into the graph. Mobility data, which is generated by GPS of portable devices under permission of people, to some extents, can describe the movement of people into and out of regions through time. This is a crucial information for modeling the disease related directly to human transmission. In this work, we cannot have access to the mobility data of people, hence, we replace it by the location of regions, which is similar to traffic field when using location of sensors.

1.4. Graph convolution network

Convolution on Graphs operator $*_{\mathcal{G}}$ in the spectral domain can be regarded as the multiplication of a signal $x \in \mathbb{R}^n$ (a scalar for every node) with a kernel Θ.

$$\Theta_{*_{\mathcal{G}}} x = \Theta(L)x = \Theta(U\Lambda U^T)x = U\Theta(\Lambda)U^T x \tag{2}$$

where graph Fourier basis $U \in \mathbb{R}^{n \times n}$ is the matrix of eigenvectors of the normalized graph Laplacian $L = I_n - D^{-\frac{1}{2}}WD^{-\frac{1}{2}} = U\Lambda U^T \in \mathbb{R}^{n \times n}$ (I_n is an identity matrix, $D \in \mathbb{R}^{n \times n}$ is the diagonal degree matrix with $D_{ii} = \sum_j W_{ij}$, W is the adjacency matrix) with a diagonal matrix of its eigenvalues $\Lambda \in \mathbb{R}^{n \times n}$; filter $\Theta(\Lambda)$ can be interpreted as a function of the eigenvalues of Λ, and is also a diagonal matrix [5]. By this definition, a graph signal x is filtered by a kernel Θ with multiplication between Θ and graph Fourier transform $U^T x$ [14].

Evaluating Eq. (2) is computationally expensive because of $O(n^2)$ multiplications with eigenvector matrix U (graph Fourier basis) and expense of eigen-decomposition of L for large graph. Thus, two approximation strategies were proposed to reduce the complexity.

Chebyshev Polynomials Approximation To avoid the problem, the kernel Θ can be well approximated by a truncated expansion up to Kth order in terms of Chebyshev polynomial $T_k(x)$, as $\Theta(\Lambda) \approx \sum_{k=0}^{K} \theta_k T_k(\widetilde{\Theta})$ with rescaled $\widetilde{\Theta} = \frac{2\Lambda}{\lambda_{max}} - I_n$ (λ_{max} is the largest eigenvalue of L), and $\theta \in \mathbb{R}^{K+1}$ is a vector of Chebyshev coefficients [15]. The graph convolution equation now becomes:

$$\Theta_{*_{\mathcal{G}}} x = \Theta(L)x \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{L})x \tag{3}$$

where $T_k(\tilde{L}) \in \mathbb{R}^{n \times n}$ is the Chebyshev polynomial of order k evaluated at the scaled Laplacian $\tilde{L} = \frac{2L}{\lambda_{max}} - I_n$. Now, the equation is K-localized as it is a Kth order polynomial in the Laplacian. K denotes the kernel size of graph convolution, which determines the maximum steps that nodes are away from the central node. By using this K-localized convolutions, evaluating Eq. (2) costs $O(K|\mathcal{E}|)$ (i.e., linear in the number of edges) as Eq. (3) shows [16].

1st-order Approximation We now can limit $K = 1$ in Eq. (3) to make the layer-wise convolution operation linear w.r.t L and therefore linear on the graph Laplacian spectrum. As a result, a deeper neural network-based model stacking these first-order approximation graph convolution layers can still recover spatial information without being limited to the explicit parameterization given by the Chebyshev polynomials [17]. Because neural networks are expected to adapt parameters to the change of scale during training, we can further approximate $\lambda_{max} = 2$. Under these approximations, the Eq. (3) is simply expressed as:

$$\Theta_{*_{\mathcal{G}}} x \approx \theta_0 x + \theta_1 \left(\frac{2}{\lambda_{max}} L - I_n\right) x \approx \theta_0 x - \theta_1 (D^{-\frac{1}{2}}WD^{-\frac{1}{2}})x \tag{4}$$

where θ_0 and θ_1 are two shared parameters of the kernel. To constrain the number of parameters and alleviate numerical instabilities, θ_0 and θ_1 can be replaced by a single parameter $\theta = \theta_0 = -\theta_1$; W and D

are renormalized by $\widetilde{W} = W + I_n$ and $\widetilde{D}_{ii} = \sum_j \widetilde{W}_{ij}$. This leads to the alternative expression of the graph convolution:

$$\Theta_{*_\mathcal{G}} x \approx \theta(I_n + D^{-\frac{1}{2}} W D^{-\frac{1}{2}}) x \approx \theta(\widetilde{D}^{-\frac{1}{2}} \widetilde{W} \widetilde{D}^{-\frac{1}{2}}) x \qquad (5)$$

Stacking 1st-order approximation of graph convolution layers effectively exploits the information from the K-order neighborhood of a node, where K is the number of successive filtering operations or convolutional layers in the model. Furthermore, the layer-wise linear formulation allows us to construct deeper models with a fixed computational budget, because the order of the approximation is limited to one [5], [17].

Generalization of Graph convolutions The graph convolution operator $*_\mathcal{G}$ defined on a signal $x \in \mathbb{R}^n$ can be generalized to a multi-dimensional signal $X \in \mathbb{R}^{n \times C_i}$ with C_i channels (i.e. a C_i-dimensional feature vector for every node) as follows:

$$y_j = \sum_{i=1}^{C_i} \Theta_{i,j}(L) x_i \in \mathbb{R}^n, 1 \le j \le C_o \qquad (6)$$

with the $C_i \times C_o$ vectors of Chebyshev coefficients $\Theta_{i,j} \in \mathbb{R}^K$ (C_i, C_o are the size of input and output of the feature maps, respectively). The graph convolution for 2-dimensional variables (2D) is denoted as $\Theta_{*_\mathcal{G}} X$ with $\Theta \in \mathbb{R}^{K \times C_i \times C_o}$ [5].

2. Materials and Methods

In this section, we present our proposed method and experimental setup. For the former part, we show an overview of our method and explain each component in detail in the following subsections. For the latter one, we present how we conduct experiments to evaluate our methods and other baseline models on a real dataset.

2.1. Proposed method

The general architecture of our approach is described in Figure 1. The input is time series of all regions/provinces, and the output is predictions for next time steps of corresponding provinces. In particular, we employ 1D convolution network (1D CNN or Conv1D) and graph convolution to extract temporal and spatial features, respectively. After convolution operators, the output is processed and passed through a fully connected layer to produce final prediction. In the following subsections, we will discuss more details about graph data structure modeling and prominent components of our model.

One special point of this design is to process and predict all time series simultaneously instead of building a model for each time series or iterating through each time series.

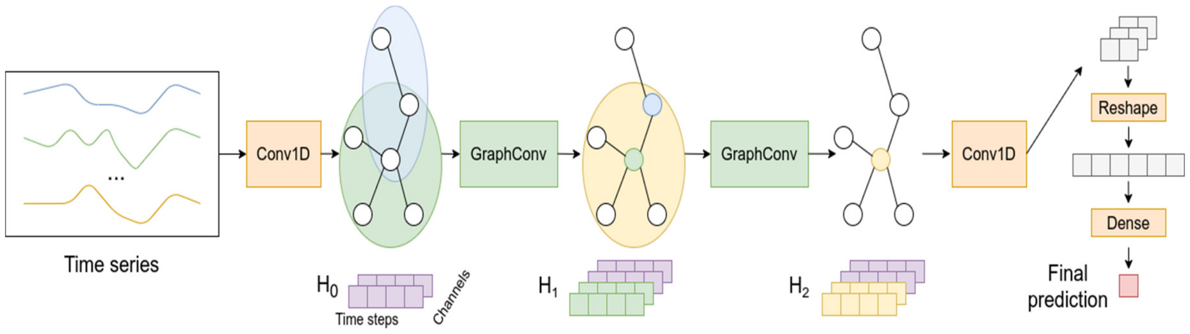

Figure 1. A visualization of our approach's architecture.

For clarity, only process for center node is presented, and the same mechanism is applied for all other nodes. Conv1D is used to extract temporal features. At each graph convolution layer, the output of the first Conv1D (H_0, represented in purple) is concatenated and propagated to the next layer. In a graph convolution layer, a node aggregates information from its immediate neighbors (nodes in same color region) to process in the next layer. The output of the last convolution is passed through a fully connected layer to produce final prediction.

2.1.1. Modeling the graph

Before explaining about components of our model, we present how to model the data structure, which is the input to our model.

In this problem, we have multiple time series data which indicate the number of daily infection cases in each location (province). In other words, v_t in the Eq. (1) can be regarded as a vector (or a matrix with one dimension is 1) containing infection cases for all provinces at day t. It is natural to model locations as nodes in a spatial graph where an individual node can have connections to others. Time series data can be incorporated as temporal features or temporal edges of these nodes.

In order to model spatial and temporal dependencies, we create a graph with two types of edge. In the spatial domain, edges represent distance between each pair of locations based on coordinates. In the temporal domain, edges simply represent connection to previous days. Temporal edges can be interpreted as sets of connected temporal features through time. The graph-structured data is visualized in Figure 2. Note that, graph notion in Figure 1 is the graph representation in hidden layers, and that in Figure 2 is spatial graph data in each time step.

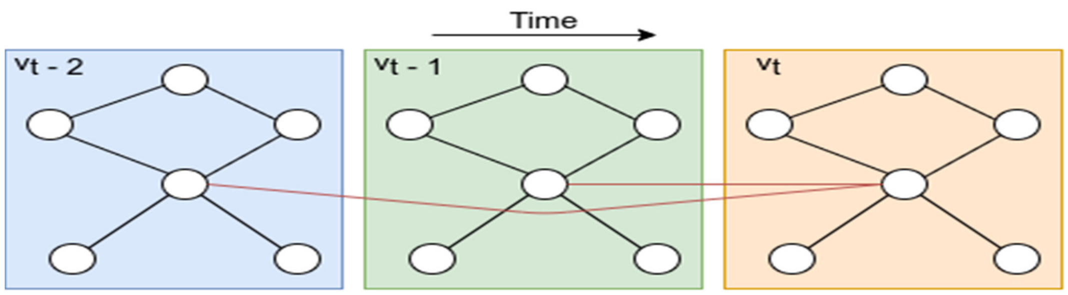

Figure 2. Graph data structure showing spatial and temporal edges (black and red, respectively).

Each node (province) has spatial edges to others in the same time step (same bounding box) and has temporal edges to itself in previous days (previous bounding box). At time step t, v_t indicates the number of cases in all provinces. For clarity, only temporal edges to the center node are shown. In practice, every node in the graph has direct temporal edges to itself in $M-1$ previous days. When processing, a graph at a time step will be passed through all steps in Figure 1.

Spatial dependencies are built from the coordinates of provinces. We first computed a matrix of distance which contain pairwise distance from provinces' coordinates. It is then used to compute adjacency matrix W in Eq. (5), as same as the work in [6].

$$W_{ij} = \begin{cases} \exp\left(-\frac{dist(a_i,a_j)^2}{\sigma^2}\right) & \text{if } \exp\left(-\frac{dist(a_i,a_j)^2}{\sigma^2}\right) \geq \mathcal{K} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where $dist(a_i, a_j)$ is the distance between two nodes a_i and a_j, σ is the standard deviation of distances and $\mathcal{K}$ is a threshold for sparsity. We set $\mathcal{K} = 0.1$ for our dataset.

The spatial graph of provinces in Korea and the unnormalized adjacency matrix is shown in Figure 3 and 4, respectively. The normalized adjacency matrix is used for graph convolution and is computed as from Eq. (5).

$$\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{W}\tilde{D}^{-\frac{1}{2}} \tag{8}$$

To build temporal dependencies, time series data is grouped by M days (containing $M-1$ previous days) for each node in each time step. 1D CNN is used to extract temporal features from these M-step time series.

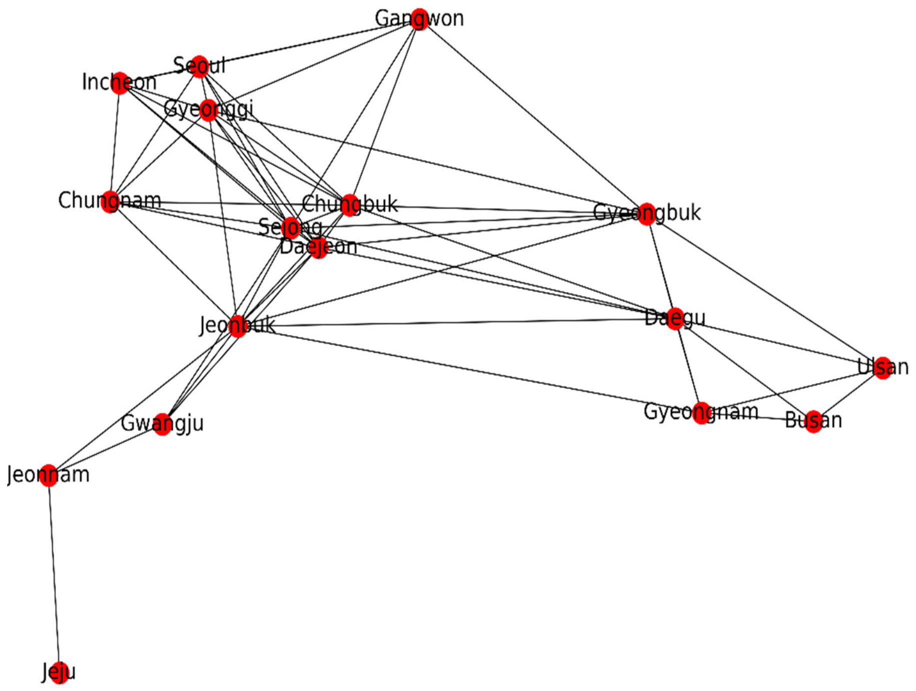

Figure 3. Korea spatial graph map build from coordinates of each province.

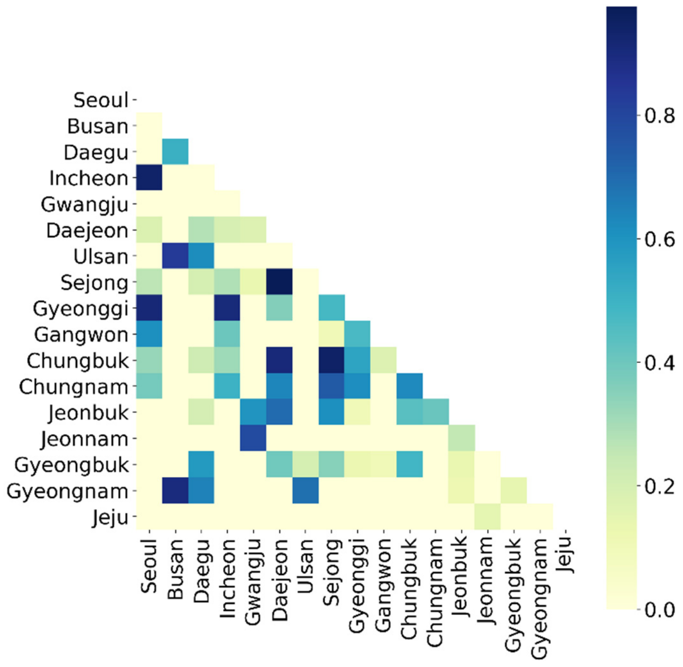

Figure 4. Unnormalized adjacency matrix for spatial graph of provinces in Korea.

2.1.2. CNN for temporal features extraction

Although RNN-based models are widely used for time series analysis, they still suffer from time-consuming and complicated gate mechanism. On the contrary, CNNs have the advantage of simple structure, fast training, and no dependency constraints to previous steps [5]. In this model, we employed 1D temporal convolution layer (1D CNN) with a kernel size of K_t.

For each node in the graph $\mathcal{G}$, the input of CNN can be regarded as a length-M sequence with C_i channels $Y \in \mathbb{R}^{M \times C_i}$ ($C_i = 1$ for our input time series data). In this scenario, the 1D CNN explores K_t neighbors in the input series without padding, which shortens the length of sequences by $K_t - 1$ units each time. Therefore, the convolution kernel $\Gamma \in \mathbb{R}^{K_t \times C_i \times C_o}$ maps the input Y to an output $Z \in \mathbb{R}^{(M-(K_t-1)) \times C_o}$. We apply a non-linear activation σ (ReLU in our case) to the output as follows.

$$\sigma(\Gamma *_{\mathcal{T}} Y) = \sigma(Z) \tag{9}$$

The CNN can be extended to 3D input by utilizing the same convolution kernel Γ to every node $\mathcal{Y}_i \in \mathbb{R}^{M \times C_i}$ (i.e., provinces) in $\mathcal{G}$, noted as $\Gamma *_{\mathcal{T}} \mathcal{Y}$ with $\mathcal{Y} \in \mathbb{R}^{M \times n \times C_i}$ where n is the number of nodes.

2.1.3. Graph convolution network for spatial features extraction

In our model, we use Graph convolution network proposed by Kipf and Welling [17] for extracting spatial features. Feature of each node contains multiple channels for each time step. Hence, we utilize generalization of graph convolutions to aggregate information from neighbor nodes for each channel and in each time step, as shown in Figure 1. Stacking two layers of graph convolution allows a node to aggregate information from its 2-hop neighbor nodes.

In particular, the input of case prediction contains M time steps of province graphs as Figure 2 shows. Each time step v_t can be regarded as a matrix $X \in \mathbb{R}^{n \times C_i}$, i.e., a matrix of n C_i-dimensional vectors for n nodes (in this case, $C_i = 1$ for the input data). For each time step t of M, the same kernel Θ of graph convolution operation is applied on $X_t \in \mathbb{R}^{n \times C_i}$ in parallel. Hence, the graph convolution can be further extended for 3D signals, denoted as $\Theta_{*_{\mathcal{G}}}\mathcal{X}$ with $\mathcal{X} \in \mathbb{R}^{M \times n \times C_i}$.

In our model, graph convolution is not applied directly to the input $\mathcal{X}$ but to the processed data $\widetilde{\mathcal{X}} \in \mathbb{R}^{M-(K_t-1) \times n \times C_o}$ which is the output of the K_t-kernel CNN. However, the calculation is unchanged as the meaning of each dimension is similar.

2.1.4. Skip-connection model

Inspired by the work of Kapoor et al. [2], we utilize skip-connections between layers to avoid diluting the self-node feature state. In particular, the output of each graph convolution layer is concatenated with the output of the first temporal convolution as supplemental information from temporal node features. Our proposed model can be formulated as:

$$\boldsymbol{H}_0 = \sigma\big(\Gamma_{*_{\mathcal{T}}}\mathcal{X}\big) = \sigma(\Gamma_{*_{\mathcal{T}}}(v_t|v_{t-1}| \dots |v_{t-M+1})) \tag{10}$$

$$\boldsymbol{H}_{l+1} = \sigma(\Theta_{*_{\mathcal{G}}}\boldsymbol{H}_l)|\boldsymbol{H}_0 \tag{11}$$

$$\boldsymbol{H}_o = \sigma(\Gamma_{*_{\mathcal{T}}}\boldsymbol{H}_s) \tag{12}$$

$$\widehat{\boldsymbol{H}}_o = \boldsymbol{H}_o.\text{reshape}(\boldsymbol{H}_o.\text{shape}[0], \boldsymbol{H}_o.\text{shape}[1], -1) \tag{13}$$

$$\boldsymbol{P} = FC(\widehat{\boldsymbol{H}}_o) \tag{14}$$

where $\mathcal{X}$ represents for the input data containing of temporal features through M time steps. The shape of tensor $\mathcal{X}$ is (batch_size, n_nodes, n_timesteps, n_features) in which batch_size is batch size, n_nodes is the number of nodes (provinces) in the graph, n_timesteps is the number of time steps (M for the input), n_features is the number of features of the data (1 for the input). $\boldsymbol{H}$ represents for hidden state at layer l with shape of (batch_size, n_nodes, n_timesteps, n_features). Note that, n_timesteps and n_features are different after each layer of our model. $\Gamma_{*_{\mathcal{T}}}$ and $\Theta_{*_{\mathcal{G}}}$ are convolution operators for extracting temporal features and spatial features (on graph), respectively. $|$ is concatenation and σ is non-linear function (ReLU for our case).

According to Equations 10 – 14, the first hidden state $\boldsymbol{H}_0$ is the output of a CNN on temporal input $\mathcal{X}$. This hidden state is then concatenated with the output of each graph convolution to create skip connections. After s graph convolution, the hidden state $\boldsymbol{H}_s$ is passed through another CNN to extract temporal information and produce hidden state $\boldsymbol{H}_o$. The number of time steps is decreased by $2 * (K_t - 1)$ (K_t is the kernel of CNN) because there are two CNN layers. The hidden state $\boldsymbol{H}_o$ is reshaped to flatten the time and feature dimensions. The final prediction $\boldsymbol{P}$ having the shape of (batch_size, n_nodes, n_predicted_timesteps) is the output of a fully connected layer (FC) applied on the last hidden state $\widehat{\boldsymbol{H}}_o$ (n_predicted_timesteps is the number of time steps ahead to forecast). A visualization of these equations is presented in Figure 1.

2.2. Experimental setup

In this section, we present preparations to conduct experiments, including dataset, data preprocessing, and architecture of models.

2.2.1. Dataset

We verify our model on a real COVID-19 in Korea dataset which is reported by Statistic Korea using API provided by the public data portal site [31-32]. It includes the number of daily infection cases from April 1st, 2020 to January 12th, 2021 in all 17 provinces of Korea. We trained all models on 70% of data (until October 20th) and evaluate the performance on the other 30%.

In addition, we also utilize the geographical location (coordinates) of all provinces in Korea extracted from COVID-19 in Korea dataset to construct the spatial dependency among nodes in our graph. This dataset is available on Kaggle [33].

2.2.2. Data preprocessing

In our time series dataset, we first transform our data into stationary time series. Although many deep learning models do not need to assume a stationary input data, the empirical results of this COVID-19 in Korea dataset suggest transforming it into a stationary form. In this case, the trend can be removed from the series, and added back to forecasts later to return the predictions to the original scale and calculate comparable metrics.

A standard way to discard non-stationarity in the data is by differencing. The difference value is the result of subtraction between the observation of the current time step t and the previous observation $t - 1$. The transformation gives us a different series, in other words, the changes to the observations from one time step to the next.

Subsequently, we re-scale the stationary data using standard scaler of scikit-learn, which has the formulation

$$z = \frac{x - \mu}{\sigma} \tag{15}$$

where μ is the mean of the training samples, and σ is the standard deviation of the training samples.

2.2.3. Baselines

For our experiments, we try to forecast one time step ahead using $M = 7$ previous observations.

Persistence model (PM) considers that the number of cases at $t + 1$ is equal to that at t. It is also called the naive predictor.

LSTM baseline model contains a stack of two LSTM layers with 8 units each and a final dense layer. The LSTM layers learn sequential information from the input through recurrent neural network. The fully connected layer takes the final output from the second LSTM layer and produce a vector of size 17, which is equal to the number of time series (or provinces) in the data.

Seq2Seq model has an encoder-decoder architecture. The encoder comprises of a fully connected layer (with 4 units) and a GRU layer [18] that can learn from a sequential input and return the full output sequence and the last state. The decoder has an inverse structure of the encoder. It takes the output sequence and last state from the encoder and applies Bahdanau attention [10] for each decoding step to make predictions for target sequences. The number of units for both GRU layers is 4 and the decoder's final dense layer is 1 because we are dealing with a regression problem. We use a single Seq2Seq for all time series in our data.

For both LSTM and Seq2Seq models, we use MSE loss, Adam optimizer [30] with a learning rate of 0.01, and a droprate of 0.2 for training. We implement these models using Tensorflow framework.

As mentioned earlier in Section 1.2, we will not consider ARIMA as a baseline model. We will also not consider STGCN [5] here because this model is more complicated and requires much more data.

2.2.4. Proposed method setup

For our proposed model, thanks to the parallel and combination of extended CNN and graph convolution, our model is able to process multi-channel input and arbitrary batch size. The first 1D CNN turns 1-channel time series into 32-channel graph features. The next two graph convolution layers have 16 output channels each. The second 1D CNN also has 16 output channels and the final dense layer has one unit which is equal to number of time step ahead to predict. We train the model for 30 epochs with a batch size of 16, Adam optimizer [30] with a learning rate of 0.001, MSE loss, and a droprate of 0.2. We implement this model using PyTorch framework for flexibility.

We also implement STGNN model from the work of Kapoor et al. [2]. There are two main differences between our model and STGNN. Firstly, instead of 1D CNN layers in Figure 1, STGNN simply uses multilayer perceptron (MLP) for the first and the last layer with 4 and 1 unit, respectively. Secondly, STGNN utilizes implementation of graph convolution layer from Kifp and Welling [17], which can only process 1-channel signal and one graph at a time. For two graph convolution layers, the graph output features (before concatenation) are 1-channel signal with 4 output units each. The model is trained for 30 epochs with Adam optimizer [30], a learning rate of 0.001, MSE loss, and a droprate of 0.2.

A summary of several hyperparameters is presented in Table 1. The hyperparameters were carefully chosen for each model. Note that, arbitrary batch size is only for Seq2Seq and our model. In LSTM model, we use stateful LSTM which requires to define batch size in advance. Thus, the batch size of 1 guarantees the model to work with any size of training and test set. Meanwhile, as mentioned above, STGNN can only process 1 graph at a time.

Table 1. Summary of hyperparameters.

	LSTM	**Seq2Seq**	**STGNN**	**Our model**
Loss	MSE	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam	Adam
Learning rate	0.01	0.01	0.001	0.001
Batch size	1	16	1	16
Drop rate	0.2	0.2	0.2	0.2
Epochs	10	10	30	30

We run all the experiments for 5 times and report the mean and standard deviation of the metrics for each model. The metrics we use in this study are root mean squared error (RMSE) and mean absolute error (MAE). The experiments are conducted on a computer using a NVIDIA GeForce GTX 1050 graphic card. All source code is available on Github [34].

3. Results

In this section, we provide the results of the experiments for evaluation. Concretely, we report the metrics and running time, and visualize the prediction.

3.1. Metrics evaluation

Table 2 shows the performance of our proposed model and baselines in terms of RMSE and MAE on the test dataset. It shows the mean and standard deviation of the metrics through 5 runs. It can be seen that graph neural network models (GNN) outperform baselines, reducing the errors by around 10%. Between two GNN models, our model produces the lower RMSE and MAE error (17.8492 and 8.2180 in comparison to 18.1490 and 8.2579).

Table 2. Mean ± Std of RMSE and MAE through 5 runs of each model.

Models	**RMSE**	**MAE**
PM	20.2611	9.1661
LSTM	20.2628 ± 0.0817	9.2138 ± 0.0299
Seq2Seq	20.5036 ± 0.0391	9.7839 ± 0.1099
STGNN	18.1490 ± 0.1150	8.2579 ± 0.0468
Our model	**17.8492 ± 0.1730**	**8.2180 ± 0.0359**

For a detailed view of the errors for each province, we report in Table 3 and 4. GNN models are able to reduce the errors in all time series prediction. Although the performance of 2 GNN models are relatively equivalent, our proposed model is likely to reduce the errors in some extreme cases in which the errors are extremely high.

Table 3. Mean ± Std of RMSE through 5 runs for each province prediction of each model.

	PM	**LSTM**	**Seq2Seq**	**STGNN**	**Our model**
Seoul	63.5511	63.5036 ± 0.3301	63.9791 ± 0.0872	56.4441 ± 0.5655	**54.9129 ± 0.6694**
Busan	12.0118	12.0593 ± 0.1465	12.2181 ± 0.0547	10.9910 ± 0.3565	**10.5160 ± 0.1881**
Daegu	7.6158	7.6909 ± 0.0774	7.8842 ± 0.0824	**6.6634 ± 0.2267**	6.6776 ± 0.0936
Incheon	14.2581	14.2790 ± 0.1586	14.5660 ± 0.0537	**11.4192 ± 0.2510**	11.8265 ± 0.0973
Gwangju	10.3543	10.3539 ± 0.1561	10.3750 ± 0.0590	9.4024 ± 0.1062	**9.2534 ± 0.1119**
Daejeon	7.4935	7.5052 ± 0.0491	7.6419 ± 0.0338	6.3342 ± 0.0952	**6.1003 ± 0.0707**
Ulsan	12.7090	12.6365 ± 0.0762	13.1233 ± 0.0811	11.2403 ± 0.0905	**11.1525 ± 0.0544**
Sejong	1.7047	1.7089 ± 0.0063	3.5590 ± 0.1849	1.4532 ± 0.0226	**1.4278 ± 0.0157**
Gyeonggi	36.6409	36.7668 ± 0.2652	36.6879 ± 0.0246	34.5582 ± 0.3343	**34.4536 ± 0.5350**
Gangwon	10.7057	10.7263 ± 0.0248	11.4689 ± 0.0738	**9.2273 ± 0.0803**	9.3150 ± 0.0753
Chungbuk	14.8771	14.8455 ± 0.0221	15.2735 ± 0.0529	**11.7576 ± 0.0809**	11.9461 ± 0.1804
Chungnam	10.8156	10.8347 ± 0.0650	11.3531 ± 0.0360	11.4302 ± 0.1059	**11.1253 ± 0.1728**
Jeonbuk	12.2758	12.2363 ± 0.0445	12.3899 ± 0.0600	**9.5665 ± 0.0674**	9.7965 ± 0.0564
Jeonnam	4.4800	4.5067 ± 0.0651	4.6905 ± 0.1938	**3.9139 ± 0.0856**	3.9505 ± 0.0403
Gyeongbuk	9.2380	9.1812 ± 0.0417	10.0681 ± 0.1808	**8.5691 ± 0.0913**	8.7161 ± 0.1145
Gyeongnam	11.7343	11.6796 ± 0.0607	11.8406 ± 0.1072	10.8008 ± 0.0914	**10.5990 ± 0.1404**
Jeju	4.1671	4.1307 ± 0.0211	4.6902 ± 0.1688	**3.9224 ± 0.1401**	3.9845 ± 0.0380

Table 4. Mean ± Std of MAE through 5 runs for each province prediction of each model.

	PM	**LSTM**	**Seq2Seq**	**STGNN**	**Our model**
Seoul	39.4706	39.4846 ± 0.4074	40.0710 ± 0.1108	36.3597 ± 0.3166	**35.5140 ± 0.4108**
Busan	8.0000	8.1274 ± 0.1194	8.1865 ± 0.0892	7.3760 ± 0.2389	**7.0705 ± 0.1320**
Daegu	4.8706	4.9644 ± 0.0526	5.1375 ± 0.0664	**4.1714 ± 0.1184**	4.3193 ± 0.0831
Incheon	9.2706	9.3014 ± 0.0640	10.0981 ± 0.1888	**7.5604 ± 0.2083**	7.7949 ± 0.1429
Gwangju	6.1529	6.2686 ± 0.1498	6.7823 ± 0.0983	5.6858 ± 0.1690	**5.6629 ± 0.1612**
Daejeon	4.4118	4.4894 ± 0.0348	4.9723 ± 0.1330	3.6891 ± 0.0919	**3.6536 ± 0.0562**
Ulsan	6.8588	6.8848 ± 0.0384	7.8574 ± 0.1728	6.1551 ± 0.0420	**6.1449 ± 0.0577**
Sejong	1.0941	1.1240 ± 0.0129	2.3987 ± 0.1884	0.9413 ± 0.0318	**0.9350 ± 0.0125**
Gyeonggi	27.9412	28.0970 ± 0.2535	27.9170 ± 0.0458	26.4018 ± 0.2213	**26.3796 ± 0.3693**
Gangwon	7.3412	7.3820 ± 0.0462	7.9261 ± 0.0866	**6.2015 ± 0.0623**	6.2606 ± 0.0841
Chungbuk	7.2353	7.2290 ± 0.0179	8.0714 ± 0.0883	**6.0588 ± 0.1039**	6.0696 ± 0.1041
Chungnam	6.9765	7.0361 ± 0.0732	7.7015 ± 0.0639	7.1593 ± 0.0869	**7.1329 ± 0.1175**
Jeonbuk	6.8824	6.8854 ± 0.0264	7.0949 ± 0.1349	**5.1970 ± 0.0892**	5.3656 ± 0.0598
Jeonnam	3.0588	3.1152 ± 0.0321	3.4256 ± 0.1801	**2.8438 ± 0.0578**	2.8566 ± 0.0664
Gyeongbuk	5.8118	5.7977 ± 0.0663	6.8917 ± 0.2320	**5.4013 ± 0.0774**	5.5025 ± 0.0371
Gyeongnam	8.1647	8.1249 ± 0.0428	8.4542 ± 0.1940	7.0898 ± 0.1064	**6.9739 ± 0.1218**
Jeju	2.2824	2.3229 ± 0.0267	3.3407 ± 0.2404	2.0926 ± 0.0756	**2.0688 ± 0.0407**

For visualization purpose, we select to plot time series of three provinces which have the 25-percentile, medium and 75-percentile of the errors. Figures 5 – 7 show the ground truth of daily number of infection cases and models' forecasts on the test set in Daegu, Jeonbuk, and Ulsan, respectively. According to the figures, Seq2Seq model cannot perform well when it even makes negative prediction. This is obvious because one single Seq2Seq model cannot deal with different patterns which are featured for each time series. Both LSTM and Seq2Seq predictions are about 1 day lagging the ground truth, thus, their performances are similar or worse than the PM model. Meanwhile, our model makes more stable predictions. It can follow the trend of the ground truth when the numbers are stable but lag a bit behind when the ground truth fluctuated wildly. Our model's numbers do not change as significantly as those of the others.

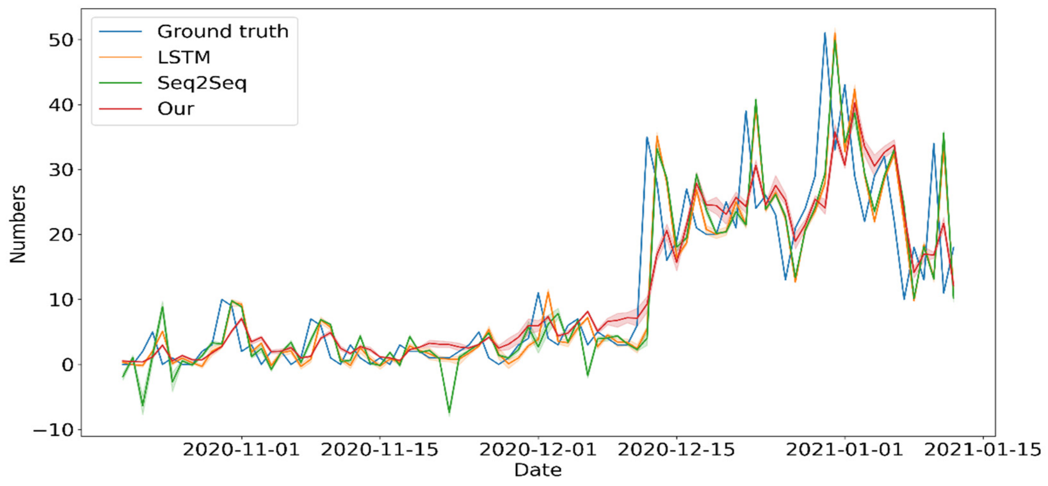

Figure 5. The ground truth and predictions (mean and 95% confidence interval) of the number of cases in Daegu.

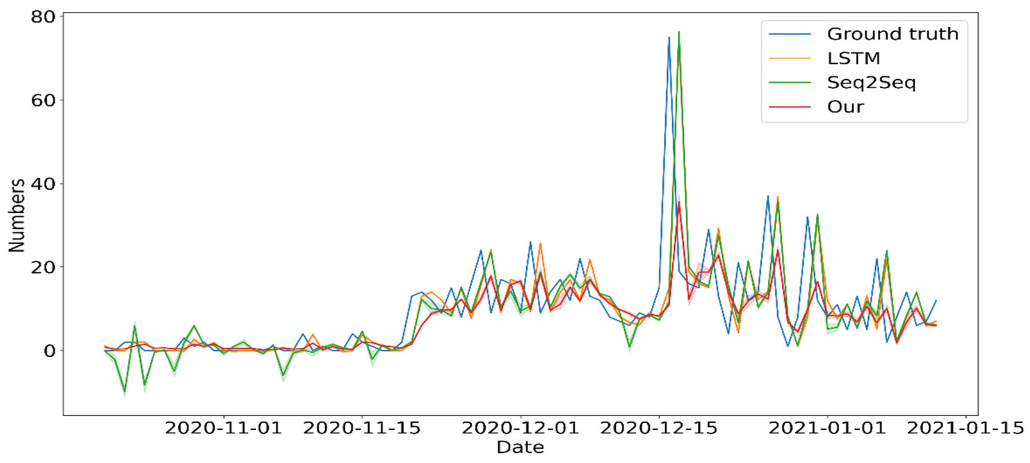

Figure 6. The ground truth and predictions (mean and 95% confidence interval) of the number of cases in Jeonbuk.

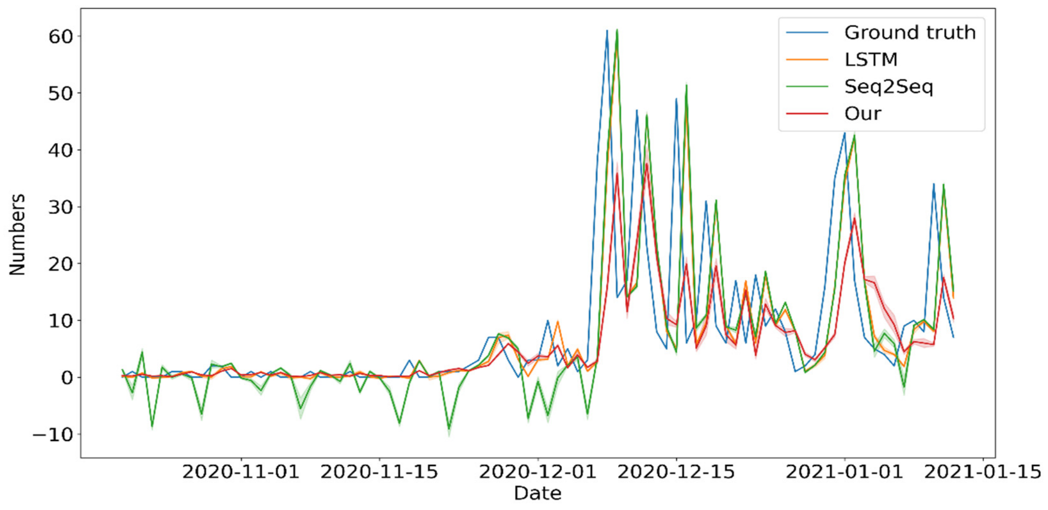

Figure 7. The ground truth and predictions (mean and 95% confidence interval) of the number of cases in Ulsan.

The last thing to note is that in December 2020, Korea witnessed the third wave of infection. It was more serious than ever before when the number of daily cases continuously peaked. That makes it harder for the models to provide accurate predictions.

3.2. Training efficiency

Table 5. Elapsed time for training of each model.

Models	**LSTM**	**Seq2Seq**	**Our model**
Elapsed time (s)/Epoch	4.5	5.3	0.04

Turning to traiing time in Table 5, it respectively takes LSTM and Seq2Seq approximately **4.5** and **5.3** seconds to finish training an epoch. For our model, only about **0.04** seconds is needed for an epoch with a batch size of 16, accelerating about **100** times.

4. Discussions

As shown in Table 2, our model and STGNN were able to reduce the errors comparing to other baselines. This means that, to some extents, the spatial data does impact on the behavior of the temporal data. Because other baseline models only relied on the time series data. Meanwhile, our model and STGNN also modeled distance relationship among provinces as spatial dependencies among these time series.

As a result, spatio-temporal graph is an effective method to combine temporal and spatial data together. In this research, we can utilize time series data of all provinces and the distance relationships among these provinces to forecast the following time steps. For further study, we can use different data to construct temporal and spatial dependencies, such as transportation, temperature or human activities which can have impacts on the spread of the disease. Moreover, this work can be also adapted for other spatio-temporal data if we can define spatial and temporal domains.

Between two GNN models, our model is superior to previous STGNN as it further reduces the errors Table 2. This probably comes from the temporal convolution layer (1D CNN or Conv1D). Particularly, we employ a CNN layer instead of a MLP because a CNN layer can explore surrounding time steps better than a normal fully connected layer. Another advantage of our model over STGNN is the processing capacity. Because of the parallel and combination of extended CNN and graph convolution, our model can process multi-channel input and arbitrary batch size. Therefore, we can also incorporate more input information by increasing the number of channels of the data.

As illustrated in Table 5, our model accelerates the training speed by approximately 100 times comparing to RNN based models due to couple of reasons. Although there are some differences in the implementation, we are also beneficial from the model design. First, we utilize convolution instead of recurrent neural network which is simpler and faster, as mentioned in subsection 2.1.2. The model design also allows us to train and predict all time series simultaneously, which save us from iterations. The training efficiency shows the potential scalability of our model for large scale dataset.

5. Conclusions

Forecasting daily COVID-19 infection cases for each region or province is one of crucial problems to help the combat with the pandemic. In this study, we developed a graph neural network-based approach for this forecasting problem in which we integrated both spatial and temporal data, rather than only time series as other RNN-based models. Specifically, we utilized both time series data of all provinces and the distance relationships among these provinces to construct spatial and temporal dependencies of the input data. We then applied our proposed model to learn both temporal and spatial features and forecast the following time step. The model's design allows us to process all time series simultaneously. We evaluated the performance of our model on the real COVID-19 dataset in Korea and showed that our model outperformed other RNN-based models, reducing the errors by 10% and cutting down the running time significantly. For future work, further processing can take more data into account, such as transportation, temperature or human activities which can have impacts on the spread of the disease. Moreover, the training efficiency opens potentials for our model to be extended for large scale spatio-temporal data or adapted for other fields.

Acknowledgments: This work was funded by KISTI grant ('Establishing a system for sharing and disseminating research data' - K-21-L01-C04-S01).

Conflicts of Interest: The authors declare no conflict of interest.

References

[1] E. Dong, H. Du, and L. Gardner, "An interactive web-based dashboard to track COVID-19 in real time," *ancet infectious diseases,* pp. 533-534, 2020, doi: https://doi.org/10.1016/s1473-3099(20)30120-1.

[2] A. Kapoor, X. Ben, L. Liu, B. Perozzi, M. Barnes, M. Blais, and S. O'Banion, "Examining covid-19 forecasting using spatio-temporal graph neural networks," *arXiv preprint arXiv:2007.03113,* 2020.

[3] I. COVID, C. J. Murray and others, "Forecasting COVID-19 impact on hospital bed-days, ICU-days, ventilator-days and deaths by US state in the next 4 months," *MedRxiv,* 2020, doi: https://doi.org/10.1101/2020.03.27.20043752.

[4] J. Durbin and S. J. Koopman, Time series analysis by state space methods, Oxford university press, 2012, doi: https://doi.org/10.1093/acprof:oso/9780199641178.001.0001.

[5] Z. Yang, Z. Zeng, K. Wang, S.-S. Wong, W. Liang, M. Zanin, P. Liu, X. Cao, Z. Gao, Z. Mai, and others, "Modified SEIR and AI prediction of the epidemics trend of COVID-19 in China under public health interventions," *Journal of thoracic disease,* vol. 12, p. 165, 2020, doi: https://doi.org/10.21037/jtd.2020.02.64.

[6] Q. Guo and Z. He, "Prediction of the confirmed cases and deaths of global COVID-19 using artificial intelligence," *Environmental Science and Pollution Research,* pp. 1-11, 2021, doi: https://doi.org/10.1007/s11356-020-11930-6

[7] L. Wang, A. Adiga, S. Venkatramanan, J. Chen, B. Lewis, and M. Marathe, "Examining Deep Learning Models with Multiple Data Sources for COVID-19 Forecasting," *arXiv preprint arXiv:2010.14491,* 2020.

[8] B. Yu, H. Yin, and Z. Zhu, "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018, doi: https://doi.org/10.24963/ijcai.2018/505.

[9] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting," in *International Conference on Learning Representations*, 2018.

[10] A. K. Sahai, N. Rath, V. Sood, and M. P. Singh, "ARIMA modelling and forecasting of COVID-19 in top five affected countries," *Diabetes and Metabolic Syndrome: Clinical Research and Reviews,* vol. 14, pp. 1419-1427, 2020, doi: https://doi.org/10.1016/j.dsx.2020.07.042

[11] A. Hernandez-Matamoros, H. Fujita, T. Hayashi, and H. Perez-Meana, "Forecasting of COVID19 per regions using ARIMA models and polynomial functions," *Applied Soft Computing,* vol. 96, p. 106610, 2020, doi: https://doi.org/10.1016/j.asoc.2020.106610.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature,* vol. 323, pp. 533-536, 1986, doi: https://doi.org/10.1038/323533a0.

[13] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 31, pp. 855-868, 2008, doi: https://doi.org/10.1109/tpami.2008.137.

[14] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," 2014.

[15] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, doi: https://doi.org/10.1109/icassp.2015.7178826.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation,* vol. 9, pp. 1735-1780, 1997, doi: https://doi.org/10.1162/neco.1997.9.8.1735.

[17] A. Graves, "Long short-term memory," in *Supervised sequence labelling with recurrent neural networks*, Springer, 2012, pp. 37-45, doi: https://doi.org/10.1007/978-3-642-24797-2_4.

[18] V. K. R. Chimmula and L. Zhang, "Time series forecasting of COVID-19 transmission in Canada using LSTM networks," *Chaos, Solitons & Fractals,* vol. 135, p. 109864, 2020, doi: https://doi.org/10.1016/j.chaos.2020.109864.

[19] F. Shahid, A. Zameer, and M. Muneeb, "Predictions for COVID-19 with deep learning models of LSTM, GRU and Bi-LSTM," *Chaos, Solitons & Fractals,* vol. 140, p. 110212, 2020, doi: https://doi.org/10.1016/j.chaos.2020.110212

[20] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215,* 2014.

[21] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473,* 2014.

[22] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025,* 2015.
[23] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434,* 2018, doi: https://doi.org/10.1016/j.aiopen.2021.01.001.
[24] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203,* 2013.
[25] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016.
[26] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv preprint arXiv:1609.02907,* 2016.
[27] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE signal processing magazine,* pp. 83-98, 2013, doi: https://doi.org/10.1109/msp.2012.2235192.
[28] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," Applied and Computational Harmonic Analysis*,* vol. 30, pp. 129-150, 2011, doi: https://doi.org/10.1016/j.acha.2010.04.005.
[29] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555,* 2014.
[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.
[31] https://kosis.kr/covid_eng/covid_index.do
[32] https://www.data.go.kr/data/15043378/openapi.do
[33] https://www.kaggle.com/kimjihoo/coronavirusdataset
[34] https://github.com/KienMN/STGNN-for-Covid-in-Korea