Spatio-Temporal Mobile Traffic Forecasting

Dziugas Vysniauskas

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2020

Abstract

With the ever-growing mobile internet traffic, mobile operators are forced to find ways to optimize their cellular networks. Lately, the leading approaches are dynamically allocating computing resources required for the network functions, based on predictions of upcoming mobile traffic, generated by Machine Learning methods. However, this technique strongly depends on the accuracy of the predictor models. With the aim of improving the state of the art city-wide mobile traffic forecasting accuracy and efficiency, in this study, three original spatio-temporal Deep Learning models — sequence to sequence ConvLSTM, CNN-ConvLSTM, CNN-ConvLSTM+Attention — and a video prediction model, PredRNN++, are evaluated on a real-world mobile traffic data set. We find that PredRNN++ increases five hour-long prediction accuracy of the state of the art model by 5%, while being 22 times computationally more efficient. As an even more efficient alternative, sequence to sequence ConvLSTM achieves comparable performance for short-term (two hour-long) predictions. Both of these models make a case for being applied in a practical network optimization solution.

Acknowledgements

My studies in the Artificial Intelligence Master's program are coming to an end. They have challenged me to my absolute limit of discipline and mental capacity. More importantly, this year has been incredibly eye-opening, rewarding and has helped me grow immensely as a person. I would like to express my deepest gratitude to everyone who was a part of this journey.

Firstly, I am most grateful to my supervisors, Dr. Paul Patras and Chaoyun Zhang, Ph.D. candidate, whose invaluable insight and guidance has made this thesis possible. Secondly, I would like to thank all University of Edinburgh lecturers who have shown the way of understanding this complicated and diverse discipline. Finally, I could not be where I am today without the continuous support and love of those that are dearest to my heart: my girlfriend Urte, my brother Ignas and my parents.

Table of Contents

Chapter 1

Introduction

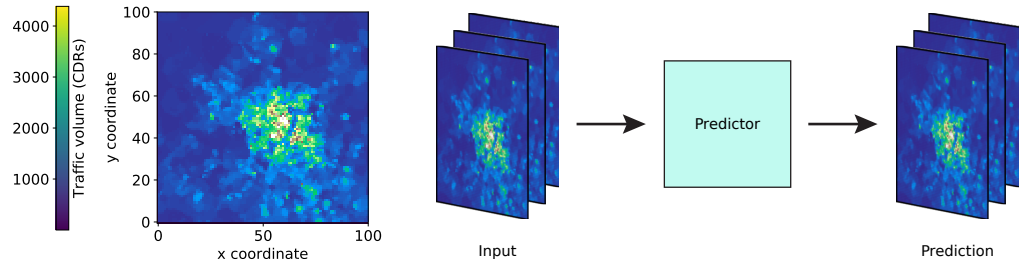
1.1 Motivation

Few technologies are growing as rapid as mobile internet. Although there are currently over nine billion mobile connections worldwide [24], it is estimated that mobile internet data usage will more than double and reach 77 exabytes (10^{18} bytes) per month by 2022 [23]. With the upcoming faster 5G networks and new types of connected devices, such as Internet of Things and autonomous vehicles, mobile internet providers face a huge challenge of maintaining fast and reliable cellular connections. They are forced to expand their infrastructures and employ larger amounts of computing resources. However, this increases the risk of inefficient utilization of the resources, leading to a loss of profits.

In order to make mobile networks efficient, a lot of effort has been put in designing flexible, demand-aware networks, capable of dynamically assigning computing resources [32]. Nevertheless, it is not always enough to react after the mobile traffic volume has already changed, as this can lead to high latency and unsatisfactory user experience or resource underutilization [2, 7]. A more innovative solution is to use mobile traffic forecasting to foresee the network load in the nearest future and allocate computing resources accordingly in advance [5]. However, it becomes extremely important to have these predictions as accurate as possible.

1.2 Problem statement

Mobile internet traffic forecasting is a task of predicting the volume of internet usage in a cellular network for the next n hours, given the measurements of the volumes



a) A measurement snapshot of mobile internet traffic at a specific time in the city of Milan. For CDR definition see section 3.1.

b) The general idea of spatio-temporal mobile traffic forecasting. Given a temporal sequence of spatial traffic usage measurements, the predicting method outputs the most likely future sequence of data usage.

Figure 1.1: Spatio-temporal mobile traffic forecasting.

during the previous k hours. This study focuses on spatio-temporal traffic forecasting, i.e. prediction where both the measurements and predictions are a temporal sequence of network activity volumes, measured (or predicted) in different geographical regions. For example, a measurement at a specific time can then be a two-dimensional grid, as shown in Figure 1.1a, where each grid cell has its measurement, which comes from the traffic generated by mobile devices in that area. As such, the spatio-temporal mobile traffic forecasting task can be depicted as in Figure 1.1b, where the predictor is a method capable of producing the predictions given the input measurements.

Formally, the problem can be expressed as:

$$\underset{X_{t+1},\ldots,X_{t+n}}{\operatorname{argmax}} \; p(X_{t+1},\ldots,X_{t+n}|X_{t-k+1},\ldots,X_t)$$

where X_t is a two-dimensional network traffic snapshot at time t, n and k are the prediction and input lengths, respectively.

1.3 Aim and objectives

Mobile traffic forecasting is a difficult task. Individual city areas experience a wide dynamic range of data usage caused by varying user engagement during work, leisure and night time as well as on different week days, resulting in complex temporal data patterns. In addition, spatial patterns are formed due to user mobility and diverse people concentrations in different areas. Finally, anomalies such as events and bank holidays further complicate accurate mobile traffic prediction.

Recently, Deep Learning methods have achieved unparalleled success in time series prediction involving audio [30], precipitation [54], video [50] and other data. Inspired by Deep Learning's ability to model a mapping from raw sequential data to predictions, there has been a number of studies employing it for mobile traffic forecasting (see section 2.2). Nevertheless, the current state of the art model (D-)STN [60] might be difficult to integrate into a production-ready cellular network optimization system as it requires a large amount of computation for the predictions (subsection 5.4.2). In addition, due to the sensitive nature of mobile network optimization, it is important to investigate whether the accuracy can be further enhanced.

The **aim** of this work is to improve spatio-temporal mobile traffic forecasting accuracy and efficiency of the state of the art (D-)STN model. To achieve it, three **objectives** were raised:

1. Acquire, explore and preprocess mobile internet traffic data, required for training and evaluating Deep Learning methods.

2. Design and implement Deep Learning models for mobile traffic prediction.

3. Evaluate the models on the data set, compare the prediction accuracy and efficiency with the state of the art.

1.4 Summary of results

We chose to use a real-world mobile traffic data set, recorded in a two month span, in the city of Milan [4], which was also used in the state of the art study [60]. Next, we designed three original Deep Learning models — sequence to sequence ConvLSTM, CNN-ConvLSTM, CNN-ConvLSTM+Attention — and evaluated them on the data set. All three models in different ways combine convolutional and recurrent neural network structures with the intent of modeling both spatial and temporal patterns of the network data. In addition, we employed a state of the art video prediction model, PredRNN++ [50], which was motivated by the similarities between video frames and mobile traffic spatial snapshots. To the best of our knowledge, this is the first time PredRNN++ was used for mobile traffic forecasting.

The proposed models were evaluated using the same metric as in [60], for an easy comparison with the state of the art. In addition, to get a broader understanding of the effectiveness of the models, they were compared to a sequence to sequence Long

Short-Term Memory [43] baseline and benchmarks, commonly encountered in mobile traffic forecasting: ARIMA [6] and Multilayer Perceptron (MLP) [15], which were also implemented and evaluated on the Milan data set. Moreover, the proposed models were compared to CNN-RNN [21] method, designed for mobile traffic forecasting.

We found that out of all the evaluated models, PredRNN++ outperforms other methods on short and long-term predictions in terms of forecasting accuracy. Importantly, while being 22 times computationally more efficient, PredRNN++ improves the state of the art five hour-long mobile traffic prediction accuracy by 5%. Moreover, a simpler sequence to sequence ConvLSTM model surpasses the baseline and benchmarks on two hour-long prediction accuracy, showing a potential to be used for scenarios where computation time is very limited.

1.5 Structure of the document

The remainder of this document is structured as follows: in chapter 2 the related work on mobile traffic forecasting is presented. Chapter 3 details the exploration and preprocessing of the data set used to train and evaluate the models. Chapter 4 describes the exact structures and design decisions of the baseline, the original models and PredRNN++. Chapter 5 follows up with detailed experiment descriptions and results. In chapter 6 practical considerations of applying the introduced methods and limitations of this study are discussed. Finally, the study is concluded and future work is discussed in chapter 7.

Chapter 2

Related work

Mobile traffic forecasting is a relatively new research area, existing for around a decade. Nevertheless, there is published work examining the spatio-temporal network data patterns and designing prediction models. Here, the existing publications are categorized by the methods used for the forecasting models. The review is split into two parts: classic time series analysis and Deep Learning methods.

2.1 Classic time series analysis

ARIMA As mobile traffic forecasting is foremostly a time series prediction problem, there have been a number of attempts in using standard time series analysis methods. One of them is the Autoregressive Integrated Moving Average (ARIMA) [6]. It is a linear model, which works by firstly transforming the series to stationary (with constant mean and variance) by differencing and then modeling the series as a sum of signal and noise components. ARIMA model can be expressed as:

$$\hat{x_t} = \mu + \phi_1 x_{t-1} + \ldots + \phi_p x_{t-p} - \theta_1 e_{t-1} - \ldots - \theta_q e_{t-q}$$

where $\hat{x_t}$ is the predicted (scalar) value, $x_{t'}$ is the series value at time t', $e_{t'}$ are the moving average error terms, computed as $e_{t'} = x_{t'} + \theta e_{t'-1}$. $\phi_{p'}, \theta_{q'}$ are learnable weights, p and q are model hyperparameters.

ARIMA was used to forecast mobile traffic in the cities of China [55, 18, 31, 38].

Holt-Winters method Another common time series analysis approach, applied to mobile traffic forecasting, is the Holt-Winters method (also known as Triple Exponential Smoothing) [52]. Similarly to ARIMA, it is a linear forecasting model, which

computes the prediction as a linear combination of three smoothed series values [10]:

$$
\begin{aligned}
s_t &= \alpha \frac{x_t}{i_{t-L}} + (1-\alpha)(s_{t-1} + b_{t-1}) && \text{Overall Smoothing} \\
b_t &= \beta (s_t - s_{t-1}) + (1-\beta) b_{t-1} && \text{Trend Smoothing} \\
i_t &= \gamma \frac{x_t}{s_t} + (1-\gamma) i_{t-L} && \text{Seasonal Smoothing} \\
\hat{x}_{t+m} &= (s_t + m b_t)\, i_{t-L+m} && \text{Prediction}
\end{aligned}
$$

where x_t are the series value at time t, $\hat{x}_{t+m}$ is the predicted value at time $t+m$, α, β, γ are learnable parameters.

Holt-Winters method has been used to forecast data traffic per mobile network slice (a virtual network running on top of a shared physical network infrastructure) [36]. The predictions were used to guide slicing and computing resource allocation for efficient infrastructure utilization. Similarly, it was used for short-term forecasts of GSM/GPRS network traffic to notify network operators about potential data usage spikes [44].

α-stable probabilistic model and Dictionary learning were employed to analyze mobile network traffic patterns and create a forecasting model [26]. The predictions of the probabilistic model were constrained to be close to some sparse vector by the Dictionary learning. Authors argue that this constraint reflects the sparsity of the network traffic and thus helps to model spatial correlations.

Entropy theory While not designing a forecasting model, [63] used Entropy theory to analyze the patterns of mobile internet, text and voice data traffic. Importantly, it was shown that data of different service types as well as adjacent measurement grid cells are closely correlated. Additionally, it was reasoned that traffic data might be predicted with high accuracy, having up to 15 hours of previous measurements. However, internet traffic was found harder to predict than voice or text.

Although aforementioned classic methods have been applied to mobile traffic forecasting due to their simplicity and efficiency, they lack the ability to learn spatial patterns and are generally weak in modeling temporal patterns. As a result, classic methods achieve predictions with limited precision. As the objective of this project is to design a highly accurate predictor, it is focused on Deep Learning models.

2.2 Deep Learning methods

This section presents published Deep Learning work on mobile traffic forecasting, excluding those that do not work with spatio-temporal data.

Recurrent neural networks (see section 4.1 for description) lead the most successful Deep Learning researches involving sequential data (e.g., natural language

translation [43], audio synthesis [58], video representation [41]). Their strong ability to model temporal patterns makes them effective in mobile traffic forecasting as well.

In two investigations ([2, 7]) LSTM (see section 4.1) networks were designed to predict mobile internet traffic. In [2] the model guides allocation of computing resources for virtualized network functions. Similarly, in [7] the predictions help manage the computing load of individual base station traffic. Both publications show an improvement of network latency and efficiency when compared to a threshold based solution in [2] and no forecasting or ARIMA predictor in [7].

Encoder-decoder type models are models consisting of two main parts: an encoder E which transforms the input $\mathbf{x}$ to a latent representation: $\mathbf{z} = E(\mathbf{x})$ and decoder D which computes the output $\mathbf{y}$ from $\mathbf{z}$: $\mathbf{y} = D(\mathbf{z})$. Autoencoders [45] are a special type of encoder-decoder type networks, where the decoder attempts to reconstruct the initial input: $D(\mathbf{z}) \approx \mathbf{x}$ and $\mathbf{z}$ is constrained to be low-dimensional or sparse.

Encoder-decoder type models have been explored in mobile traffic forecasting as they are known to achieve good generalization performance. A model combining neural network autoencoders and an LSTM has been used for spatio-temporal mobile traffic prediction [48]. Here, autoencoders extract spatial features, whereas LSTM models the temporal patterns. This architecture was proven to be effective and achieved a 28.4% lower Mean Absolute Error than ARIMA.

In [13] an architecture consisting of a feature extractor and a sequence to sequence LSTM with attention (see section 4.2 and section 4.5) was employed for cellular traffic demand forecasting. The feature extractor combines traffic of the grid cell being predicted with the traffic of neighbouring areas and external features such as day of the week, hour of the day and similar. The model was shown to outperform ARIMA, GRU recurrent neural network [8] and support vector regressor [11].

Convolutional neural networks (CNNs) Although CNNs [15] have originally been designed for computer vision tasks, they have shown prospect in spatio-temporal mobile traffic forecasting due to their efficient processing of spatial data. [61] employed Densely connected CNNs [22] (CNNs with residual connections) for aggregated SMS and call data traffic prediction. Their solution is more accurate than ARIMA and LSTM, showing that CNNs can also be effective for the task.

Convolutional recurrent neural networks Advanced mobile traffic forecasting solutions have shown success by combining convolutional and recurrent neural networks. [21] built a multi-task predictor, called CNN-RNN, combining CNNs and LSTMs. CNNs firstly extract spatial features of the mobile traffic data. LSTMs then

learn temporal patterns from the latent features. Finally, the model outputs maximum, minimum and average usage predictions for the next hour. Authors have found that this approach achieves better accuracy than an LSTM or a three-dimensional CNN model.

State of the art results in mobile traffic forecasting have been achieved by using a model combining three-dimensional convolutional and ConvLSTM (see subsection 4.3.1) layers [60]. In addition, long-term predictions (up to 10 hours) in this work were achieved by combining the model's output with historical average at the prediction time (week day and time).

Graph neural networks As mobile traffic measurements are often collected at individual base stations, which are scattered in a region, the data can be accurately represented in a graph structure. As a result, there have been two investigations using graph based methods for traffic prediction.

In [49] a Graph Neural Network [35] architecture was adapted to cellular network traffic forecasting. It models each base station as a graph node v and connects neighbouring stations with edges. Values of the nodes then represent traffic in each base station, whereas the edge values — traffic of devices moving between base station areas. In addition, each graph node iteratively computes a hidden state:

$$h(v) = f^*\left(x(v), x(NBR(v)), x(CO(v)), h(NBR(v))\right)$$

where $h(v)$ and $h(NBR(v))$ are hidden states of station v and its neighbours, $x(v)$, $x(NBR(v))$ and $x(CO(v))$ are values of v, its neighbors and edges connected to v, respectively. The predicted traffic at station v is then computed as: $o(v) = g(h(v), x(v))$. In [49] f^* is implemented by an LSTM network and g — by a MLP.

Despite its flexibility, this approach requires a separate model to be trained for each base station (which number can reach 6000 in a city), making it computationally infeasible for any Deep Learning architecture with a considerable number of parameters.

The second study [12] designed a model with a modified LSTM structure, where matrix multiplications were replaced by graph convolutions. A graph convolution is expressed as: $g_\theta(L) * x$, where L is the graph's Laplacian matrix, $g_\theta(L)$ is a convolutional filter, with respect to the graph topology, and x is the input. This structure models spatial and temporal patterns at the same time and was shown to be a flexible solution.

While it is evident that there has been a range of Deep Learning approaches employed for the task of mobile traffic forecasting, some of them ([60, 49, 13]) are computationally demanding, while others do not reach a satisfying prediction accuracy. Thus, it is significant to investigate further possible solutions.

Chapter 3

Data Analysis & Preprocessing

This chapter presents the data set used for training and evaluating the prediction models. In addition, scale, shape and spatial correlation properties of the data are analyzed and discussed. Finally, the preprocessing steps are listed.

3.1 Data set

A reliable, sufficient size data set is an essential part of training and evaluating Machine Learning models. In this work, it was chosen to use publicly available data, collected by mobile service provider Telecom Italia [4]. The data set contains mobile internet traffic measurements in the city of Milan during the months of November and December in 2013. The measurements are given in the number of Call Detail Records (CDRs) aggregated over every 10 minutes. A single CDR measurement is generated when a mobile user starts or ends an internet connection, or the internet connection lasted more than 15 minutes, or five megabytes of data were used. In essence, the number of CDRs measures the level of interaction of the users with the network, i.e. higher CDR count relates to a more intense internet usage.

In addition, for each time step (measurement every 10 minutes) the data is given in a two-dimensional, 100×100 grid of values. The grid divides the city into 10 000 cells (see Figure 3.1), which are 235×235 meters in size. As a result, the whole data set can be described as a three-dimensional tensor with dimensions $(8928 \times 100 \times 100)$, where 8928 is the number of measurements for 62 days every 10 minutes.

Reasons for choosing this data set are threefold: 1. It was used by other researchers [60, 2, 7, 21], which allows for comparison of results. 2. The data is provided in a grid structure, helping to conveniently model spatial features by using Machine Learning

structures designed for processing spatial data. 3. It is of sufficient size to train Deep Learning models.

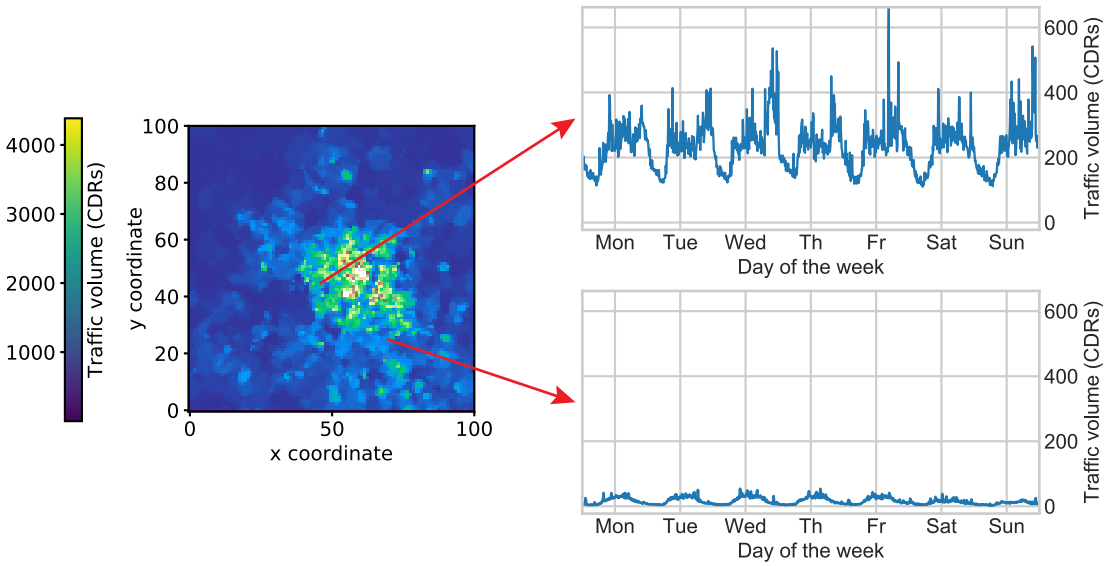


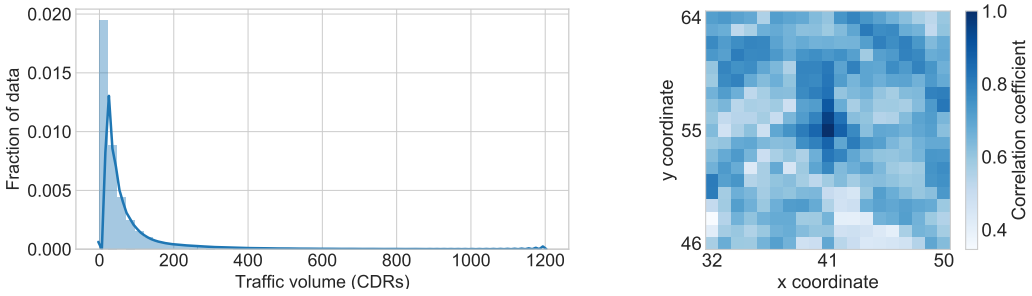
Figure 3.1: A snapshot of mobile traffic activity in the city of Milan (left) and week-long measurements of two grid cells (right). Traffic volume ranges are drastically different in highly populated areas compared to city outskirts.

3.2 Exploratory data analysis

In order to familiarize with the data and have insight when designing prediction models, an exploratory data analysis was undertaken. To begin with, the scale and shape of the data was analyzed. Traffic measurements take values from 0 to 8044 and the range varies significantly for different grid cells, as it is illustrated in Figure 3.1. This poses two issues: the data is non negative and has a huge, varying range. However, this is addressed during data preprocessing (section 3.3).

Further investigation of the value distribution (Figure 3.2a) revealed that measurements are concentrated around smaller values, which come from the city's outskirt cells that outnumber populated city areas. However, some cells produce highly dynamic range of traffic volumes, which might be difficult to capture for simpler models.

Most of the publications presented in chapter 2 emphasize the importance of spatial features for mobile traffic forecasting [34, 63, 48, 13, 18, 21, 12, 49]. This is confirmed by inspecting spatial correlations in Figure 3.2b. The data shows complex spatial cor-



a) Histogram and Kernel density estimation of the traffic value distribution. Values larger than 1200 were removed for better visualization.

b) Correlations of each cell in regards to the cell in the middle. Correlation coefficients were computed over the whole data set sequences.

Figure 3.2: Properties of the data.

relation patterns, with a general tendency of stronger correlation in adjacent cells and weaker correlation in cells further apart. Coefficients were calculated using Pearson Correlation Coefficient:

$$\frac{\mathrm{cov}(\mathbf{x}_m, \mathbf{x}_i)}{\mathrm{std}(\mathbf{x}_m)\mathrm{std}(\mathbf{x}_i)}$$

where $\mathbf{x}_m$ is the sequence data of the middle cell, $\mathbf{x}_i$ is the data of the cell compared to, cov and std are sample covariance and standard deviation, respectively.

3.3 Preprocessing

Here, the performed preprocessing steps for the data set are listed. The data records are originally presented in (*time stamp - cell id - traffic volume*) format. Firstly, all the records where the traffic volume is not provided were removed. The remaining ones were mapped to a three-dimensional tensor structure: (time steps $\times$ grid width $\times$ grid height).

In addition, as pointed out in section 3.2, the data is non negative and covers a wide range of values. Fitting a prediction model on this data would require its outputs to be constrained in the positive domain. Moreover, gradient based training methods would require more time to adjust the model weights for the outputs to match large target values. To avoid these issues, centring and standardizing transformations were applied: $\tilde{x} = \frac{x-\mu}{\sigma}$, where x and $\tilde{x}$ are the original and final (scalar) data points, the mean μ and standard deviation σ were computed over all data points, irrespective to their gird locations.

Chapter 4

Methodology

This chapter presents the baseline and proposed Machine Learning methods. The first section describes recurrent neural networks, on which all of the Deep Learning models are based on. The rest of the chapter gives details about each of the models.

4.1 Recurrent Neural Networks

Recurrent neural networks (RNNs) are neural networks designed for tasks with sequential inputs (e.g. audio, stock prices, text). They differ from feed-forward neural networks by having connections along the temporal sequence between nodes (also called cells in RNN context) in the same network layer. In a RNN, a node receives a temporal point of the input sequence, computes its output and passes information (state) to the node processing the next temporal point. This structure allows the network to model sequential data dependencies.

Considerable number of researches [16, 34, 46, 47] have found success in using long short-term memory (LSTM) [20] RNN structure. LSTM extends the idea of RNNs by having two states: a hidden state and a cell state, as well as parameterized gates that allow the model to learn which information should be stored or removed from the cell state, used from the input and form the final output. This whole process

can be described by the following equations:

$$\begin{aligned} i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + b_i\right) \\ f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\ o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + b_o\right) \\ h_t &= o_t \odot \tanh\left(c_t\right) \end{aligned} \tag{4.1}$$

where σ and tanh are element-wise sigmoid and hyperbolic tangent functions, W_* and b_* are learnable model weights and biases, x_t — temporal point of the input sequence at time t, h_{t-1} and c_{t-1} — previous hidden and cell states, '$\odot$' denotes the Hadamard product.

Firstly, the values of input (i_t) and forget (f_t) gates are computed by applying sigmoid function to a weighted sum of the current input and previous hidden state. Then the cell state c_t is updated according to the gates, previous states and the input. The output gate is a sigmoid applied to a weighted sum of the input, hidden state and the cell's state. Finally, the hidden state, which is also the output of the cell, is computed as an element-wise product of the cell state and the output gate.

It was shown that this structure allows LSTMs to model long-term sequential dependencies [17, 42] and mitigate the vanishing [19] gradient problem. Vanishing gradients are gradients of parameters that tend to zero when backpropagated through time. LSTM mitigates this issue by having additive cell state updates instead of multiplicative, this way creating computational connections through time without exponentially decaying gradients.

4.2 Sequence to Sequence LSTM baseline

For the baseline model, sequence to sequence LSTM [43] (seq2seq LSTM) was chosen. Originally, the model was designed for natural language translation; however, its abilities to learn temporal patterns and predict arbitrary length sequences at once make it suitable for mobile traffic forecasting, as well. In addition, it has been shown to be effective for mobile traffic prediction by other studies [13].

The structure of the baseline model is illustrated in Figure 4.1. It consists of three main parts: a stack of LSTM layers that encode the input into hidden and cell state vectors (encoder), a stack of LSTM layers that decode the state vectors (decoder) and a fully connected layer which forms the final output vector. The first cells in the decoder

layers receive hidden and cell states from last cells in the respective encoding layers (denoted by "H, C" arrows in Figure 4.1). The full output of the encoder, however, is not used, following the original sequence to sequence model [43].

Due to the absence of an internal LSTM mechanism intended for spatial data, seq2seq LSTM cannot effectively process the full 100×100 data grid input or accurately predict a region of the target grid at once. To overcome this, following [60], the choice was made to use a 12-step (two hour) long sequence of flattened 11×11 regions as model's input. The output of the model was chosen to be a sequence of predicted mobile traffic consumption in the middle of the 11×11 region for the next n time steps. As such, seq2seq LSTM model can be described as a function $\mathcal{S}$, parameterized by θ:

$$v_{t+1}^{(j)}, v_{t+2}^{(j)}, ..., v_{t+n}^{(j)} = \mathcal{S}(\mathbf{r}_{t-k+1}^{(j)}, \mathbf{r}_{t-k+2}^{(j)}, ..., \mathbf{r}_{t}^{(j)}; \theta) \tag{4.2}$$

where $v_t^{(j)}$ is mobile traffic consumption at grid cell j, at time t; $\mathbf{r}_t^{(j)}$ is a vector containing the flattened values at the 11×11 region with cell j in the middle of it, at time t; k is the input sequence length (set to 12 in this study).

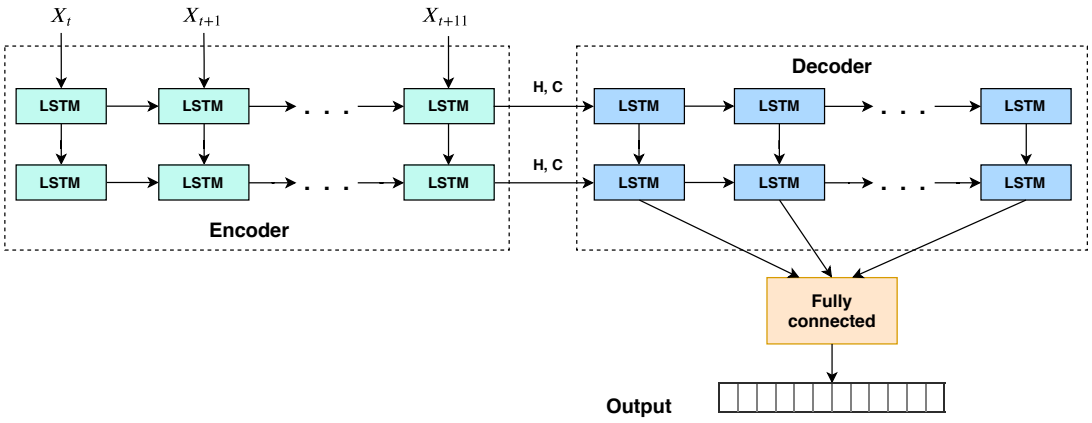


Figure 4.1: Sequence to sequence LSTM model used in this project. LSTM represents an LSTM cell, "Fully connected" — same-named layer. H denotes hidden state vector, C — state of the last cell in the layer.

4.3 Sequence to Sequence Convolutional LSTM

4.3.1 Convolutional LSTM

For spatio-temporal problems, where a temporal point in the input sequence is two-dimensional, LSTM models face similar problems as feed-forward neural networks

with spatial input: they cannot efficiently capture spatial correlations and generally have too many connections. To overcome these issues, convolutional LSTM (ConvLSTM) [54] structure was designed. It maintains the same structure as LSTM but replaces linear weight operations with convolutions. ConvLSTM can be described with the following equations, where '$*$' denotes two-dimensional convolution operation:

$$\begin{aligned} i_t &= \sigma\left(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \odot \mathcal{C}_{t-1} + b_i\right) \\ f_t &= \sigma\left(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \odot \mathcal{C}_{t-1} + b_f\right) \\ \mathcal{C}_t &= f_t \odot \mathcal{C}_{t-1} + i_t \odot \tanh\left(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c\right) \\ o_t &= \sigma\left(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \odot \mathcal{C}_t + b_o\right) \\ \mathcal{H}_t &= o_t \odot \tanh\left(\mathcal{C}_t\right) \end{aligned} \tag{4.3}$$

Note that in the case of ConvLSTM, inputs $\mathcal{X}_t$, hidden state $\mathcal{H}_t$ and cell state $\mathcal{C}_t$ are three-dimensional tensors with dimensions (height) $\times$ (width) $\times$ (number of channels).

It was shown that ConvLSTM outperforms LSTM models in video prediction [14] and precipitation nowcasting [54] by more successfully capturing spatio-temporal patterns. ConvLSTM is a strong building block for models solving spatio-temporal problems and is used throughout this work.

4.3.2 Sequence to Sequence ConvLSTM model

The first original model designed in this work for mobile traffic forecasting is called sequence to sequence ConvLSTM (seq2seq ConvLSTM). It draws inspiration from the seq2seq LSTM model's outstanding multistep prediction capabilities and ConvLSTM ability to learn spatio-temporal patterns. High level structure of the model can be seen in Figure 4.2.

Similarly to seq2seq LSTM, it has an encoder-decoder architecture, where the encoder and decoder are implemented by a stack of ConvLSTM layers. The encoder extracts all the relevant information from the input sequence and encodes it into the hidden state and cell state tensors. The decoder is then initialized with the encoded states and generates the prediction sequence. As in seq2seq LSTM, full output of the encoder is not used.

In this model, both the inputs and outputs are full grid mobile traffic measurements (or predictions). To achieve this, padding is used for convolutional ConvLSTM operations to maintain the data dimensionality. This makes the training process and inference easier as the model needs only to compute a single output for a single time

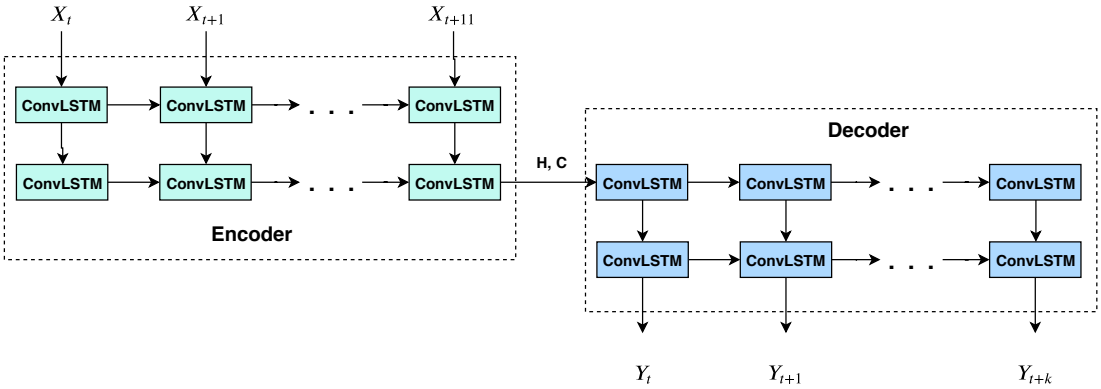


Figure 4.2: Sequence to sequence ConvLSTM model used in this project. "ConvLSTM" box represents a ConvLSTM unit. H denotes hidden state tensor, C — state of the last cell in the layer. Here, seq2seq ConvLSTM is depicted with two encoder and two decoder layers.

step prediction as opposed to 10000 outputs in the case of seq2seq LSTM.

Seq2seq ConvLSTM model, can be expressed as function $\mathcal{F}$:

$$Y_{t+1}, Y_{t+2}, ..., Y_{t+n} = \mathcal{F}(X_{t-k+1}, X_{t-k+2}, ..., X_t; \theta) \tag{4.4}$$

where X_t, Y_t are full input and predicted mobile traffic grids at time t, n is the prediction length, k is the input length (set to 12 in this study), θ are the network weights.

4.4 CNN-ConvLSTM model

Although seq2seq ConvLSTM model has mechanisms designed to process both spatial and temporal data, its computations are not efficient as there is no dimensionality reduction in the computation graph, forcing all the operations to be done on high-dimensional grids. CNNs, however, are known to extract abstract features in higher layers with the help of dimensionality reduction [59, 39]. Our next proposed model, CNN-ConvLSTM, builds on these observations.

The high-level structure of CNN-ConvLSTM can be seen in Figure 4.3. In essence, this model extends seq2seq ConvLSTM model by applying Convolutional and Average Pooling layers to extract spatial features of the input and a Multilayer Perceptron (MLP) [15] to form the final outputs. Here, convolutional and MLP layers are applied in a time distributed manner, i.e. each temporal point in the sequence is passed through the same layers independently.

CNN-ConvLSTM was designed with a hypothesis that the CNN layers can learn spatial correlations and seq2seq ConvLSTM can then model temporal patterns, this way achieving more accurate predictions.

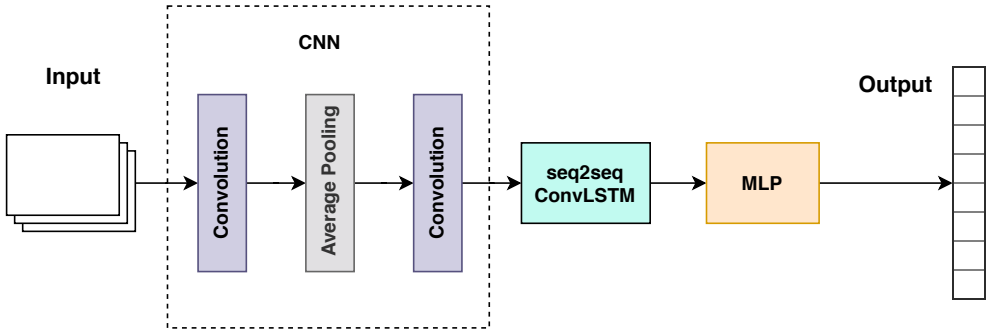


Figure 4.3: CNN-ConvLSTM model used in this project. "Convolution" and "Average Pooling" represent same-named, time distributed layers, "seq2seq ConvLSTM" — seq2seq ConvLSTM model illustrated in Figure 4.2; "MLP" — Multilayer Perceptron. Here, CNN-ConvLSTM is depicted with a single average pooling layer.

In principle, CNN-ConvLSTM could be used to predict the whole data grid sequences by employing deconvolution and unpooling [28] layers instead of the MLP. However, pilot experiments using such architecture showed that it is too difficult for the model to convert low-dimensional latent variables to full grid predictions without losing accuracy. Thus, it was chosen to simplify the task by using the input and output form identical to the seq2seq LSTM baseline. As such, the CNN-ConvLSTM model can be expressed as a function $\mathcal{C}$, parameterized by θ:

$$v_{t+1}^{(j)}, v_{t+2}^{(j)}, ..., v_{t+n}^{(j)} = \mathcal{C}(R_{t-k+1}^{(j)}, R_{t-k+2}^{(j)}, ..., R_{t}^{(j)}; \theta) \tag{4.5}$$

where $R_t^{(j)}$ is a matrix of 11×11 region grid values with cell j in the middle of it, at time t. Other notation follows Equation 4.2.

4.5 CNN-ConvLSTM+Attention model

In an attempt to further improve CNN-ConvLSTM model, inspired by the success of attention in natural language translation [3] and its application to ConvLSTM networks [62], experiments with CNN-ConvLSTM augmented with attention were performed.

In the context of encoder-decoder RNNs, attention is a structure that enables the decoder to shift its focus to specific encoder outputs, imitating the attention observed in human cognition. This allows the decoder to more flexibly interpret the encoded information and thus, potentially, make better predictions.

Attention is achieved by computing attention weights (called "alignment weights" in [3]) for each decoder cell, which are then applied to the encoder outputs to form specific input for the cell. In this work, attention weights are computed as:

$$Z_j^{(t)} = W_z * \tanh\left(W_e * E_j + W_h * H_{t-1} + b\right)$$
$$A_j^{(t)} = \text{softmax}(Z_j^{(t)})$$

where $A_j^{(t)}$ is a (three-dimensional) weight for the j^{th} encoder output E_j, computed for decoder's t^{th} cell. H_{t-1} is the hidden state of the decoder, W_z, W_e, W_h and b are learnable weights and a bias, and $Z_j^{(t)}$ is the unnormalized weight. The softmax function is applied on the temporal axis, so that separate weight elements would sum to one over all encoder outputs.

Having the weights, input for decoder's cell t is computed as:

$$I^{(t)} = \sum_{j=1}^{L} A_j^{(t)} \odot E_j$$

where L is the length of the encoder output sequence. Hence, the input for the decoder cell is a weighted sum of the encoder outputs, where the weights can be learned by the model and are adapted depending on the current state. Therefore, this allows the model to focus on the most relevant encoder outputs when computing the prediction.

In this work, attention mechanism is applied to the CNN-ConvLSTM model, more specifically, to compute the inputs for the ConvLSTM decoder. This solution is called CNN-ConvLSTM+Attention.

4.6 Predictive Recurrent Neural Network

The final model used in this work is called Predictive Recurrent Neural Network (PredRNN++) [50]. It is an architecture specifically designed for spatio-temporal predictive learning, which has achieved state of the art results in video prediction. As mobile traffic forecasting is a task of similar structure as video prediction, this work investigates PredRNN++ effectiveness in mobile traffic domain.

An overview of PredRNN++ architecture is seen in Figure 4.4. PredRNN++ introduces an RNN cell called Causal LSTM (Figure 4.4a, top). It extends ConvLSTM

by having an additional state $\mathcal{M}$, called "spatial memory", and increasing the number of nonlinear computations. Causal LSTM can be expressed using the following equations:

$$\begin{aligned}
\begin{pmatrix} g_t \\ i_t \\ f_t \end{pmatrix} &= \begin{pmatrix} \tanh \\ \sigma \\ \sigma \end{pmatrix} W_1 * \left[\mathcal{X}_t, \mathcal{H}_{t-1}^k, \mathcal{C}_{t-1}^k\right] \\
\mathcal{C}_t^k &= f_t \odot \mathcal{C}_{t-1}^k + i_t \odot g_t \\
\begin{pmatrix} g_t' \\ i_t' \\ f_t' \end{pmatrix} &= \begin{pmatrix} \tanh \\ \sigma \\ \sigma \end{pmatrix} W_2 * \left[\mathcal{X}_t, \mathcal{C}_t^k, \mathcal{M}_t^{k-1}\right] \\
\mathcal{M}_t^k &= f_t' \odot \tanh\left(W_3 * \mathcal{M}_t^{k-1}\right) + i_t' \odot g_t' \\
o_t &= \tanh\left(W_4 * \left[\mathcal{X}_t, \mathcal{C}_t^k, \mathcal{M}_t^k\right]\right) \\
\mathcal{H}_t^k &= o_t \odot \tanh\left(W_5 * \left[\mathcal{C}_t^k, \mathcal{M}_t^k\right]\right)
\end{aligned} \tag{4.6}$$

where $g_t, i_t, f_t, g_t', i_t', f_t'$ are gates, W_* — learnable kernel weights, $\mathcal{H}_{t-1}^k$ — hidden state, $\mathcal{C}_{t-1}^k$ — cell state (called temporal memory in [50]), square brackets denote concatenation. Superscript k indicates the index of the layer. Other notation follows Equations 4.1 and 4.3.

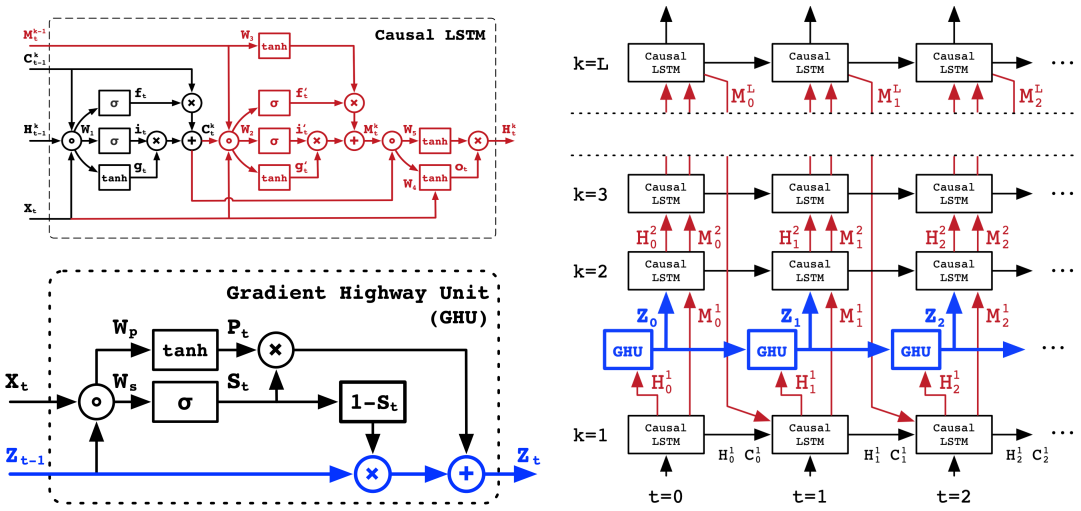
Importantly, PredRNN++ shares the spatial memory $\mathcal{M}$ between different layers and time steps by passing it in a zigzag manner through all the nodes (see Figure 4.4b and notice the superscript of $\mathcal{M}_t^{k-1}$ in Equation 4.6). As a result, the computational depth of temporal transitions is increased significantly. Combined with additional computations in the Causal LSTM cell, PredRNN++ is stronger and more flexible in modelling spatio-temporal patterns, when compared to seq2seq ConvLSTM.

Nevertheless, this deep structure is prone to vanishing gradient problem due to the long temporal transitions. To overcome this, PredRNN++ introduces Gradient Highway Unit (GHU) [50] (Figure 4.4a, bottom). It is a simpler structure inserted after the first Causal LSTM layer, that provides a shorter route for the gradients along the temporal axis, this way helping the gradients to be backpropagated through time. GHU can be summarised using these equations:

$$\begin{aligned}
\mathcal{P}_t &= \tanh\left(W_{px} * \mathcal{X}_t + W_{pz} * \mathcal{Z}_{t-1}\right) \\
\mathcal{S}_t &= \sigma\left(W_{sx} * \mathcal{X}_t + W_{sz} * \mathcal{Z}_{t-1}\right) \\
\mathcal{Z}_t &= \mathcal{S}_t \odot \mathcal{P}_t + (1 - \mathcal{S}_t) \odot \mathcal{Z}_{t-1}
\end{aligned} \tag{4.7}$$

where $\mathcal{Z}_t$ is the output of GHU at time t, W_{**} are learnable kernel weights and other

notation is the same as above. It was shown [50] that GHU successfully overcomes vanishing gradients and contributions to the error from all time steps are appropriately reflected in the model weights update.



a) Causal LSTM and GHU structure. b) PredRNN+ network architecture.

Figure 4.4: Causal LSTM structure and PredRNN+ model. Figures taken from [50]. Here, concentric circles denote concatenation, $\times$ is the convolution operation. Other notation follows Equation 4.6 and 4.7.

In this work, PredRNN++ was modeled to take the full grid mobile traffic measurements as input and output the full grid predictions. Thus, the model can be expressed in an equivalent way as seq2seq ConvLSTM, using Equation 4.4.

Chapter 5

Evaluation

This chapter presents Machine Learning experiments undertaken in this work. Firstly, the details about how the data set was split and training samples were formed are given. Secondly, the overall training strategy is stated. Then follows specific comments about each of the models experiments. The chapter concludes with the obtained results.

5.1 Training and evaluation data

All of the models in this work were trained using the data set described in section 3.1. It was split into three parts: training (first 42 days — 68% of data), validation (the following 10 days — 16% of data) and test data (the last 10 days). The complete training procedure, which is known as Two Times Hold-out approach, is presented in Algorithm 1.

Algorithm 1 Two Times Hold-out Training & Evaluation Procedure

Input: model $\mathbf{m}$, training D_{tr}, validation D_{val}, test D_{test} data,
set of hyperparameter configurations $\mathcal{H}$
for h in $\mathcal{H}$ **do**
  $\mathbf{m_h} = Train(\mathbf{m}, \mathbf{h}, D_{tr})$ (Train model $\mathbf{m}$ with parameters $\mathbf{h}$ on D_{tr})
  $validationError = Evaluate(\mathbf{m_h}, D_{val})$
end for
Choose $\tilde{\mathbf{m}}$ from trained models with lowest $validationError$
$performance = Evaluate(\tilde{\mathbf{m}}, D_{test})$

When working with time series data, there are different techniques for generating training samples (input-target pairs). In this work a common, overlapping with a stride

size of one, approach is adopted. This yields the highest possible number of training samples and helps avoid overfitting. Formally, given a data sequence $X_1, X_2, ..., X_N$, the training samples are generated as:

(input: $X_1, ..., X_k$; target: $X_{k+1}, ..., X_{k+n}$),

(input: $X_2, ..., X_{k+1}$; target: $X_{k+2}, ..., X_{k+n+1}$),

(input: $X_3, ..., X_{k+2}$; target: $X_{k+3}, ..., X_{k+n+2}$),

...

where k, n are input and target sequence lengths, respectively. In this work, k was always chosen as 12. As for n, all models were initially trained and their parameters tuned with $n = 12$. However, models, which structure or prediction accuracy depends on the output length, were retrained with 30-step targets (see Table 5.1 for details). Exception being the ARIMA model, which is fit on the whole training data at once.

5.2 Training and evaluation parameters and details

All Neural Networks in this study were trained using Mini-Batch Stochastic Gradient Descent. In addition, backpropagation through time [51] was used for RNN layers, meaning that gradients obtained from the error at time t are propagated to all the previous steps. Moreover, in order to increase the convergence speed, a popular adaptive learning rate optimizer Adam [25] was used. It decreases the overall training time and improves convergence precision by normalizing the learning rate with a moving average of squared parameter gradients as well as adding momentum to the updates, preventing wrong direction, noisy updates. Default values were chosen for Adam parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$. The learning rate was uniquely chosen for each model by experimenting and inspecting the training loss, and was in the range of $[0.0005, 0.002]$.

All of the models were trained by minimizing Mean Squared Error (MSE) between model outputs and corresponding targets. The training was done for 100 epochs with early stopping, i.e. while training, the model is constantly evaluated on the validation set and the validation error is monitored, once the error has not decreased for 20 epochs, the training is stopped. In the end, weights that produced the lowest validation error are used.

The models were evaluated using Normalized Root Mean Squared Error (NRMSE), which can be expressed as: $NRMSE = \frac{1}{\bar{y}} \sqrt{\sum_{z=1}^{N} \frac{(\tilde{y}_z - y_z)^2}{N}}$, where N is the number of predicted points over space and time, $\tilde{y}_z$ is a predicted value, y_z is the corresponding target

and $\bar{y}$ is the mean of all targets. NRMSE was chosen following [60], which allows to compare this study results with the state of the art. The reason behind choosing different metrics for training loss (MSE) and evaluation (NRMSE) is that NRMSE cannot be used for negative data as the mean $\bar{y}$ can be negative, causing the gradient descent algorithm to have no lower bound and diverge towards $-\infty$. When evaluating using NRMSE (on validation and test data), the predictions were converted back to the initial (non-negative) data range, i.e. the centring and standardizing transformations were reversed.

All of the training was done on a GPU cluster with 30 nodes, equipped with up to four Geforce GTX (different exact models are used) GPUs.

5.3 Experiments

This section describes the parameter tuning and training details of each model.

All of the software required to train and evaluate the models was written using Python [33] programming language. Seq2seq LSTM, seq2seq ConvLSTM, CNN-ConvLSTM, CNN-ConvLSTM+Attention and benchmarks — MLP, CNN-RNN — were implemented using Keras library [9] with Tensorflow [1] backend. For PredRNN++, the code released from paper [50][1] (written in Tensorflow) was adapted. Finally, ARIMA model was created using statsmodels [37] library.

5.3.1 Sequence to Sequence LSTM

For seq2seq LSTM model (section 4.2) the most important hyperparameters are the number of layers n_l in the encoder and decoder LSTM stacks and the size of hidden state h in the LSTM cells. To reduce the search space, it was chosen to use the same h for all LSTM layers. n_l was searched in $\{2,3,4\}$ and h in $\{50,75,100\}$. The experiments showed that seq2seq LSTM is not sensitive to different parameter settings. The best validation result was achieved with $h = 100$, $n_l = 2$ (i.e. two layered encoder and two layered decoder).

5.3.2 Sequence to Sequence ConvLSTM

Similarly to seq2seq LSTM, for seq2seq ConvLSTM model (subsection 4.3.2) the number of layers n_l and the size of the hidden state needs to be tuned. However, for

[1] https://github.com/Yunbo426/predrnn-pp

ConvLSTM the hidden state size is determined by the number of filters f in the structure. Thus, f is the hyperparameter in question. In addition, there is a possibility to use different ConvLSTM filter sizes s_f. A larger filter has a wider receptive field, allowing the convolution operation to gather information from a larger region. Alternatively, a smaller filter focuses on the neighboring cell correlations.

ConvLSTM was trained using full grid inputs, which meant that there were 10000 less samples than for the models taking a region of the grid as input and outputting a single point per time step. This leads to a much faster training, however, it also makes the model more prone to overfitting. In order to avoid it, Dropout [40] regularization technique was used. As the name suggests, a Dropout layer drops (sets to zero) an input unit with probability r. This adds noise to the training and prevents the weights from overfitting specific inputs. In this model, dropout (with small r) was applied before each ConvLSTM layer.

The hyperparameters were chosen from: $n_l \in \{2,3,4\}$, $f \in \{25,32,50,64,100,120\}$ (number of filters was not necessarily the same in all the layers — different combinations were explored), $r \in \{0,0.05,0.1\}$, $s_f \in \{3,5\}$. The lowest validation error was achieved with $n_l = 2$, $f = 50$ (for all the layers), $r = 0$.

5.3.3 CNN-ConvLSTM

CNN-ConvLSTM (section 4.4) also requires setting the number of layers for the encoder and decoder (n_l), number of filters (f) and their size (s_f) in ConvLSTM cells. In addition, the structure of CNN (number of layers (cnn_l), filter sizes (cnn_{s_f}), number of filters (cnn_f), activation function (cnn_a)) and MLP (number of layers (m_l), number of units (m_u)) is needed to be specified. As grid search is time-wise infeasible for such a large number of parameters, the parameters were adjusted one at a time, observing the impact to the validation error. Experimented values were: $n_l \in \{2,3,4\}$, $f \in \{32,50,64\}$ (higher number of filters was prevented by the training time), $s_f \in \{3,5\}$, $cnn_l \in \{1,2\}$, cnn_{s_f} in {3, 5}, $cnn_f \in \{25,32,50,64,100\}$, $cnn_a \in \{\text{ReLu}, \text{Tanh}\}$[2], $m_l \in \{1,2\}$, $m_u \in \{50,75,100\}$.

The CNN-ConvLSTM structure which achieved the lowest validation error is listed in Figure 5.1. Although, it is possible that the performance of this model could be improved by having a larger number of layers or filters, the increased complexity prolongs training time and thus, complicates convergence. However, the convergence of CNN-

[2]Here, ReLu and Tanh are the Rectified Linear Unit and hyperbolic tangent activation functions, respectively

Layer	# Fl/Un	Fl size	Act
conv1	25	3	Tanh
pool1	-	2	-
conv2	50	3	Tanh
pool2	-	2	-
conv3	50	3	Tanh
clstm(e)	50	3	Tanh
clstm(e)	50	3	Tanh
clstm(e)	50	3	Tanh
clstm(d)	50	3	Tanh
clstm(d)	50	3	Tanh
clstm(d)	50	3	Tanh
FC	1	-	None

CNN-ConvLSTM

Layer	# Fl/Un	Fl size	Act
conv1	20	3	Tanh
pool1	-	2	-
conv2	40	3	Tanh
pool2	-	2	-
conv3	40	3	Tanh
clstm(e)	40	3	Tanh
clstm(e)	40	3	Tanh
clstm(a)	40	3	Tanh
clstm(d)	40	3	Tanh
FC	100	-	ReLu
FC	1	-	ReLu

CNN-ConvLSTM+Attention

Figure 5.1: Final configurations of CNN-ConvLSTM and CNN-ConvLSTM+Attention models. "# Fl/Un" denotes number of filters or units (output dimensionality), "Fl size" — filter size, "Act" — activation function, "conv" and "pool" — convolution and average pooling layers, "clstm(e)" — "ConvLSTM layer in the encoder", "clstm(d)" — "ConvLSTM layer in the decoder", "clstm(a)" — "ConvLSTM decoder layer with attention", "FC" — fully connected layer.

ConvLSTM (as well as CNN-ConvLSTM+attention) was found difficult as it can be seen in Figure 5.2, even with relatively small number of filters.

5.3.4 CNN-ConvLSTM+Attention

CNN-ConvLSTM+Attention (section 4.5) differs from CNN-ConvLSTM only by having an attention mechanism between the last encoder and the first decoder layers. Meaning, that all the same hyperparameters need to be tuned. For the attention model, similar parameter combinations were used for experiments as in CNN-ConvLSTM.

However, attention mechanism is a computationally costly operation, prolonging the training time. Combined with complicated training (Figure 5.2), it was found difficult to obtain a well converged, complex attention model. As a result, the lowest validation error was achieved by using a simpler configuration listed in Figure 5.1.

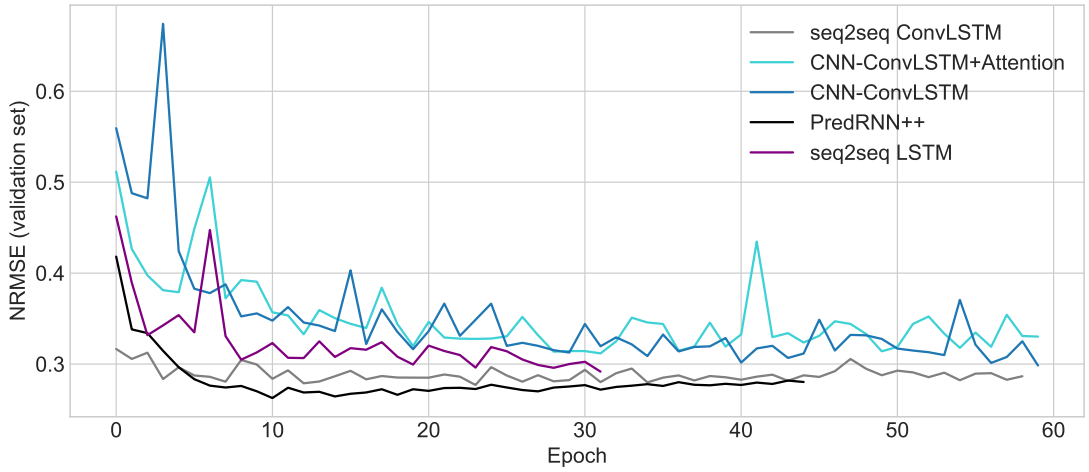


Figure 5.2: Validation NRMSE errors observed during training of the Deep Learning models.

5.3.5 PredRNN++

For PredRNN++ (section 4.6), the tunable hyperparameters are: number of layers (n_l), number of filters in each layer (f_1, f_2, ...) and filter size (s_f). The options suggested by the original PredRNN++ publication [50] are: $n_l = 4$, $f_1 = 128$, $f_2 = 64$, $f_3 = 64$, $f_4 = 64$, $s_f = 5$. However, PredRNN++ requires large amount of memory due to the concatenation operations in the Causal LSTM cells (Equation 4.6). With full grid inputs (100×100 size grids at each time step) and the suggested configuration, PredRNN++ exceeded GPU memory during runtime and it was not possible to train the model. As such, it was chosen to train simpler PredRNN++ models, where $n_l \in \{2,3,4\}$, $f_n \in \{20,30,40,50\}$, $s_f = 5$. In addition, similarly to seq2seq ConvLSTM, in order to avoid overfitting, Dropout before each Causal LSTM layer was explored, with Dropout rates $r \in \{0, 0.1, 0.2\}$.

Configuration $n_l = 3$, $f_1 = 30$, $f_2 = 30$, $f_3 = 30$, $s_f = 5, r = 0$ achieved the lowest validation error.

5.3.6 Benchmarks

In order to better assess the results obtained in this study, three benchmark models were evaluated: ARIMA (see section 2.1 for description), MLP and CNN-RNN [21] (see section 2.2).

5.3.6.1 ARIMA

ARIMA was chosen as, arguably, the most popular autoregressive time series prediction technique. The method works only with one-dimensional sequential data, thus to adapt it for spatio-temporal mobile data forecasting, it was needed to train an ARIMA model for each of the cell in the Milan grid. This results in 10000 ARIMA models, making the inference slower than for other models, despite the simplicity of the ARIMA structure.

ARIMA requires setting three hyperparameters: p, d, q (section 2.1). In this study, $p = 12$ was chosen as this corresponds to using 12 previous sequence points for calculating the prediction, matching the input of the Deep Learning models. Other parameters were searched in $d \in \{0,1\}$, $q \in \{0,2,4\}$. The lowest validation error was achieved with $p = 12$, $d = 1$, $q = 2$.

5.3.6.2 MLP

MLP is the most common Deep Learning model and its computations are more parallelizable and thus, faster than recurrent networks, which need to make computations one time step at a time. As such, MLP is a suitable choice for a benchmark model.

MLP is modeled to receive the 12 time step windowed input as a single 1452 length vector (equal to a flattened $12 \times 11 \times 11$ tensor) and output an n length vector, corresponding to predicted traffic measurements for the cell in the middle of 11×11 region, for the next n time steps.

The structure of the model was searched in: hidden layers $n_l \in \{2,3,4,5\}$, nodes per layer $h \in \{50,100,200,400,600,1000\}$. ReLU activation function was chosen.

The best validation accuracy was achieved with $n_l = 5$, $h = 600$.

5.3.6.3 CNN-RNN

CNN-RNN [21] was designed for mobile traffic forecasting and was also used for the Milan data set. As such, it is important to compare the findings of this work with the CNN-RNN model performance. The network was trained using the hyperparameters and input form (15×15 windowed regions) suggested in [21].

5.4 Results

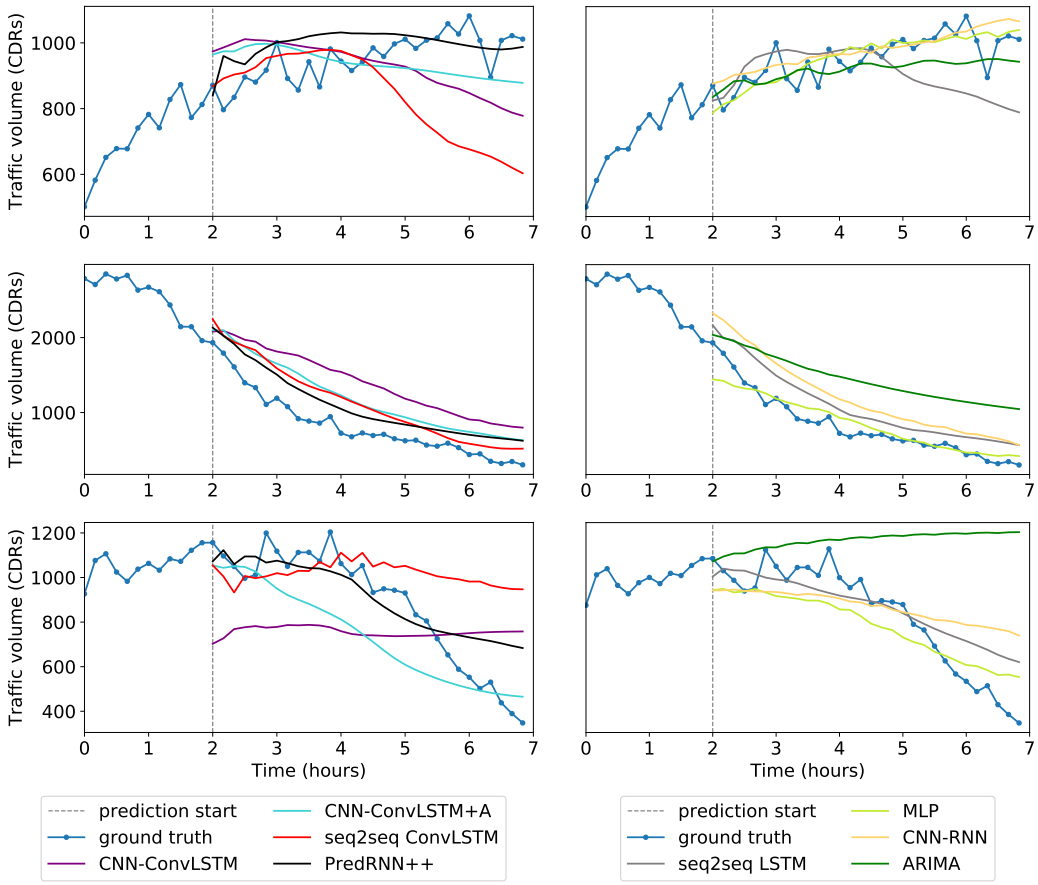
5.4.1 Forecasting accuracy

Table 5.1 presents the forecasting performance evaluation of all models analyzed in this work in terms of 10, 12 and 30 step NRMSE errors on test data. In addition, single grid cell prediction samples for three distinct traffic scenarios are shown in Figure 5.3. Finally, for an overall look at the model performances, city-wide predictions are demonstrated in Figure 5.4.

MODEL	TEST SET NRMSE			FLOP	PARAMS
	10-STEP	12-STEP	30-STEP		
SEQ2SEQ LSTM	$0.332 \pm 0.06^*$	0.361 ± 0.07	0.725 ± 0.24	201	0.33
SEQ2SEQ CONVLSTM	$0.327 \pm 0.05^*$	0.351 ± 0.06	0.644 ± 0.19	315	0.63
CNN-CONVLSTM	$0.342 \pm 0.06^*$	0.367 ± 0.07	0.754 ± 0.18	2520	1.12
CNN-CONVLSTM+A	$0.356 \pm 0.05^*$	0.383 ± 0.06	0.711 ± 0.16	1060	0.54
PREDRNN++	$\mathbf{0.304 \pm 0.05}^*$	$\mathbf{0.322 \pm 0.05}$	$\mathbf{0.480 \pm 0.11}^*$	3302	1.43
ARIMA	$0.329 \pm 0.08^*$	$0.358 \pm 0.10^*$	$0.599 \pm 0.22^*$	—	0.015
MLP	$0.330 \pm 0.05^*$	0.352 ± 0.05	0.629 ± 0.17	78	1.96
CNN-RNN	$0.378 \pm 0.07^*$	0.406 ± 0.08	0.744 ± 0.21	3551	0.41

Table 5.1: Prediction accuracy and computational complexity measurements of the models analyzed in this study. "Flop" denotes 10^9 floating-point operations for a single $(12 \times 100 \times 100)$ prediction, "params" — trainable parameters in millions, "CNN-ConvLSTM+A" — CNN-ConvLSTM+Attention model. '*' marks a prediction output length, for which the model was not specifically trained. NRMSE scores are given with a $\pm$ range of evaluation sample error standard deviation.

PredRNN++ surpasses other methods in all prediction step categories, giving strong evidence of the model's superiority in mobile traffic forecasting. It achieves at least 7% 10-step, 8% 12-step and 20% 30-step smaller NRMSE errors than other presented approaches. Furthermore, investigating single cell prediction samples (Figure 5.3), it is clear that due to the computationally deeper structure, PredRNN++ is more flexible and captures the temporal trends in more detail, allowing it to maintain accurate predictions for longer sequences. Meanwhile, other models (especially benchmarks) predict in a more linear manner and thus are limited in capturing the longer data trend. The effect of this difference is clearly visible in Figure 5.4, where only PredRNN++ predictions retain accurate dynamic range and spatial granularity. Other methods diverge



Proposed model predictions. Baseline and benchmark model predictions.

Figure 5.3: Sample predictions of the proposed models (left) and the baseline and benchmarks (right), analyzed in this study. Each graph represents predictions for a single cell (not the full grid data). The input is the first 12 points of the sequence, for the next 30 steps all the models are predicting without the knowledge of the ground truth. The shown prediction samples were triggered at different times and locations in the city.

strongly from the ground truth.

When compared to the state of the art model D-STN [60], PredRNN++ and D-STN both achieve 0.48 NRMSE error for 30-step prediction. However, D-STN uses historical data to improve long-term predictions. It is more fitting to compare PredRNN++ with STN [60], which is equivalent to D-STN without historical data. PredRNN++ achieves 5% smaller 30-step NRMSE error than STN. As such, **PredRNN++ achieves** 5% **improvement for 30-step (five hour-long) mobile traffic forecasting over the state of the art**.

Regarding other models, seq2seq ConvLSTM is also notable, achieving lower er-

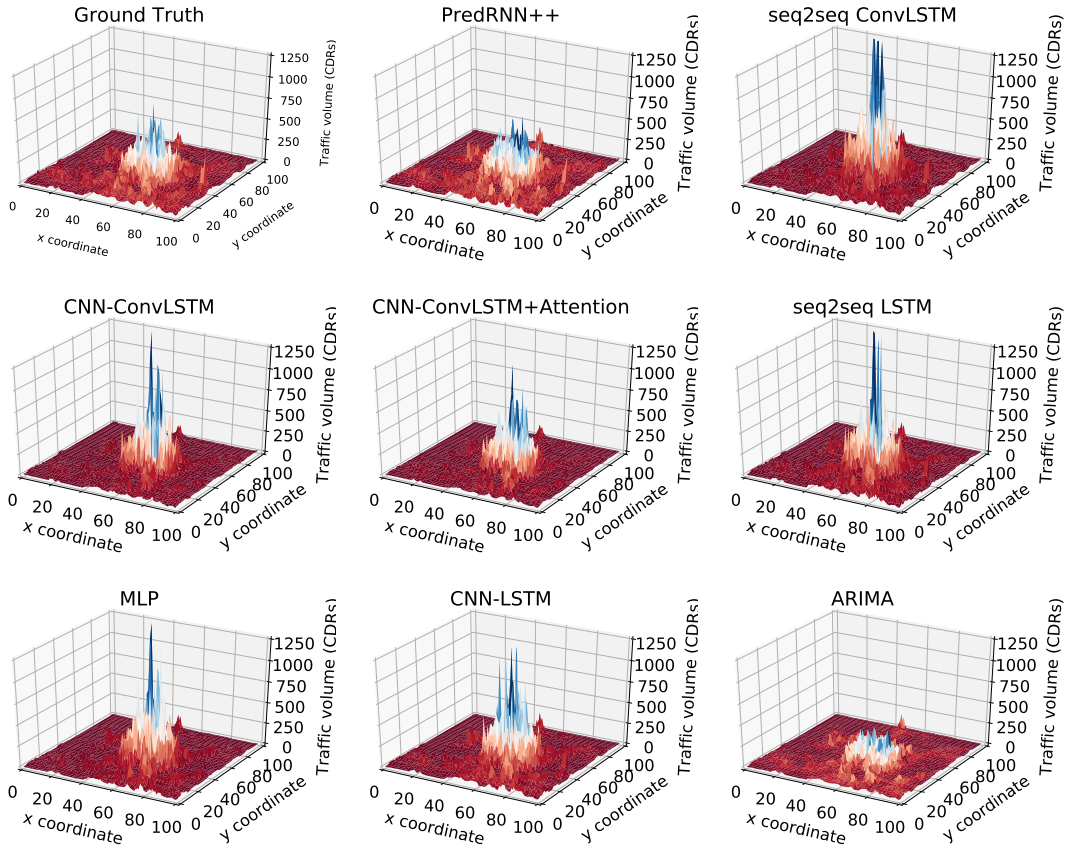


Figure 5.4: Comparison of city-wide ground truth and predictions made by the proposed models, benchmarks and the baseline, after 5 hours from the last observation. Some values were truncated to fit in the graph.

rors on 10 and 12-step predictions than any other model, excluding PredRNN++. Prediction samples reveal that seq2seq ConvLSTM is similarly flexible as PredRNN++, allowing it to accurately model shorter-term temporal patterns. Nevertheless, the network's predictions do not always follow later data trends, hinting that the model struggles to learn longer patterns. This might be the case due to the small network size (subsection 5.3.2), which does not have enough memory to remember longer data patterns. It is likely, that if the hyperprameters were tuned for 30-step prediction, the long-term accuracy would increase. However, this was outside the scope of this study.

CNN-ConvLSTM(+Attention) did not reach satisfactory forecasting accuracy. Both models perform worse than the baseline and benchmarks, except for CNN-RNN. Although it is difficult to confidently state the reason of this, it is possible that CNN dimensionality reduction loses relevant information and is a worse approach than passing raw data points to the ConvLSTM encoder-decoder. This, paired with slow training

and complicated convergence, which prevented exploring larger and stronger models, could be the cause of the inaccurate predictions.

Unexpectedly, MLP and ARIMA benchmarks achieved solid performances. As MLP models are faster to train than RNNs, it was possible to explore deep networks with large number of nodes. This allows MLP to remember more traffic patterns. However, prediction samples show that it is limited to near-linear predictions.

ARIMA success comes from having a separate model for each grid cell, allowing the model to easily adapt to dissimilar temporal patterns in different city regions. However, as mentioned in subsection 5.3.6.1, it is an inefficient approach. In addition, no possibility to exploit spatial features limits ARIMA's performance.

Finally, benchmark CNN-RNN achieves the worst results of all the models. Originally, CNN-RNN [21] was used to predict the Milan city traffic only in a small central area. However, results obtained in this study show that CNN-RNN approach does not generalize well to the city-wide prediction problem.

5.4.2 Computational complexity

In addition to forecasting accuracy, model computational complexities were analyzed. For this purpose, number of floating-point operations (FLOPs), required for a single $12 \times 100 \times 100$ size (two hour-long, city-wide) prediction, and the number of trainable parameters were measured for each model (see Table 5.1). FLOPs were measured using Profiler tool provided by Tensorflow library.

Although from the models, evaluated in this study, PredRNN++ is second in number of FLOPs, when compared to the state of the art model (D-)STN [3], it requires about 22 times less FLOPs for a single prediction. In addition, (D-)STN has around 100 times more trainable parameters than PredRNN++. These measurements show a significant PredRNN++ efficiency advantage over the state of the art method, which is of great value in practical applications, such as mobile network optimization.

It is also worth noting that albeit seq2seq ConvLSTM is not as accurate as PredRNN++, it is very efficient, requiring more than 10 times less FLOPs than PredRNN++ and having 2.3 times less trainable parameters. As such, seq2seq ConvLSTM could be considered in scenarios where quick predictions are needed or the model needs to be often retrained on new observed data.

[3] FLOP and trainable parameter counts for (D-)STN model were measured using the published (D-)STN source code: https://github.com/vyokky/Mobihoc-18-STN-mobile-traffic-forecasting.

Chapter 6

Discussion

6.1 Practical considerations

Both PredRNN++ and seq2seq ConvLSTM models, analyzed in this study, show prospect of being employed as mobile traffic forecasting methods in a real-time network optimization system. Potentially, predictions made by these models could lead resource allocation in virtualized network slices [5] or a Cloud Radio Access Network [7], where base stations share cloud computing resources. Accurate predictions of the models would help avoid costly resource underutilization as well as violations of Service-Level Agreements, which occur when the cellular network is not capable of processing the traffic load due to a lack of resources.

For the best results, the models would need to be trained on the specific traffic data observed in the network in question. In addition, the models should be retrained on new measurements, once the traffic data distribution changes significantly. One possible approach would be to monitor the accuracy on part of the model predictions. Once the accuracy falls below some chosen threshold, a retraining could be scheduled. Furthermore, if the traffic patterns are dependent on yearly seasons, it is possible to have separate model versions, trained on different month data, stored and ready for use.

PredRNN++, however, has a significant advantage over seq2seq ConvLSTM by being able to accurately forecast further into the future. Mobile traffic measurement collection is a demanding operation, taking up to 10 gigabytes of log data per Radio Network Controller [56], which need to be densely deployed geographically for fine-grained location measurements. On top of that, there is additional cost of sending and processing the collected data. Using PredRNN++ as a long-term predictor, measurements for network optimization could be collected in intervals. For example, if the

model is used for six hour predictions, a four hour break after each two hour-long measurement collection could be done, reducing the amount of collected data by a factor of three.

6.2 Limitations

This study has potential limitations, which might impact further research of the proposed Machine Learning models or their practical applications. First of all, the methods were evaluated using only a single data set. This limits the validity of the obtained results and might not accurately reflect the true mobile traffic forecasting performance of the models. Moreover, although a lot of effort in this study was put into improving the models and tuning their hyperparameters, due to limited time and computational resources, there still exists unexplored options. This is especially true for seq2seq ConvLSTM and CNN-ConvLSTM(+Attention) models, which potentially could benefit from a deeper and stronger (i.e. with more learnable parameters) structure. In addition, regularization apart from Dropout technique could be explored.

Finally, it is likely that a data set such as Milan mobile traffic does not accurately reflect the qualities of real-world production data. More precisely, it is often the case that due to some disruptions, mobile network traffic measurements become unavailable for certain regions. In order to investigate how well the proposed models predict upcoming traffic with missing data, PredRNN++ and seq2seq ConvLSTM were evaluated with 5% and 10% missing data for two hour-long predictions (Table 6.1). In these evaluations, 5% or 10% of cell measurements were set to zero over all input time steps. The target was retained unmodified.

From the results listed in Table 6.1, it is clear that both PredRNN++ and seq2seq ConvLSTM are very sensitive to missing data. With 5% missing data, PredRNN++ accuracy drops by 46% and by 83% with 10% missing data. For seq2seq ConvLSTM it is 36% and 65%, respectively.

One of the ways to mitigate this loss of accuracy is to use models trained with Dropout regularization, introduced in subsection 5.3.2. With Dropout, the model is trained with missing data and is forced to adapt its weights to accommodate for the missing information. However, this only partly resolves the problem as both PredRNN++ and seq2seq ConvLSTM with a 10% dropout rate still experience a significant decrease in performance when data is missing (Table 6.1).

Model	0%	5%	10%
PredRNN++	0.322 ± 0.05	0.470 ± 0.07	0.589 ± 0.10
PredRNN++ with dropout	0.362 ± 0.08	0.404 ± 0.07	0.453 ± 0.07
seq2seq ConvLSTM	0.351 ± 0.06	0.476 ± 0.07	0.581 ± 0.08
seq2seq ConvLSTM with dropout	0.352 ± 0.05	0.450 ± 0.07	0.543 ± 0.09

Table 6.1: PredRNN++ and seq2seq ConvLSTM test data 12-step (two hour-long prediction) NRMSE errors with different percentages of missing data. For models with dropout, a 10% dropout rate was used.

Chapter 7

Conclusion

Mobile traffic forecasting is becoming a key element in the optimization of cellular networks, challenged by the exponentially increasing usage of internet. With the aim of improving state of the art mobile traffic prediction accuracy, in this study, three original Deep Learning architectures — seq2seq ConvLSTM, CNN-ConvLSTM, CNN-ConvLSTM+Attention — and a video prediction model PredRNN++ were implemented and evaluated on a real-world mobile network data set. All of the approaches are specifically designed to model complex spatio-temporal patterns, allowing to accurately forecast mobile traffic, which is known to exhibit spatial and sequential features, created by user mobility and a wide dynamic range of internet usage during different parts of the day and the week.

We found that PredRNN++ achieves 5% more accurate five hour-long predictions than the state of the art STN method. In addition, by modeling city-wide (full grid) outputs, PredRNN++ is 22 times more computationally efficient than STN. Both advantages make PredRNN++ a superior model, when considered for mobile network optimization. In addition, although seq2seq ConvLSTM is not as accurate as the state of the art, the model could be considered for short-term predictions (e.g. two hour-long), when computational time or resources are limited. Nevertheless, PredRNN++ offers a great advantage of accurately predicting longer traffic sequences, potentially allowing mobile operators to store and process less traffic measurements and reduce the computational load for the infrastructure of the network.

However, both PredRNN++ and seq2seq ConvLSTM were found to be sensitive to missing data, which is a risk when considering practical applications.

7.1 Future work

Future research, based on this study, could involve applying PredRNN++ or seq2seq ConvLSTM in a network optimization strategy, measuring the costs of resource over-provisioning and violations of Service Level Agreements, comparing obtained results to other similar studies [5, 7, 2]. It is likely, that given the accurate predictions of PredRNN++ and seq2seq ConvLSTM, network optimization using these models would lead to a more efficient resource allocation than those achieved by previous studies.

Alternatively, supplementing the models, analyzed in this work, with historical network traffic data, similarly to [60], would possibly improve long-term prediction accuracy. It has been shown that mobile traffic exhibits periodicity in daily and weekly patterns [60, 29]. As such, the historical average of traffic volumes at specific times of the week could help the model avoid deviating from the empirical mean and thus, reduce uncertainty in long-term predictions. As an example, the outputs of PredRNN++ could be combined with the historical means by a simpler MLP network to form the final predictions. However, this makes the model reliant on the availability of the historical data and biased if the empirical mean does not equal the mean of currently observed traffic.

Lastly, an investigation on how to mitigate the impact of missing data could also yield important insight into the problem of mobile traffic forecasting. There are a number of probabilistic models, designed for imputing missing data [27]. However, majority of them rely on modelling a probability distribution in the form of: $p(\mathbf{x}_m|\mathbf{x}_o)$, where $\mathbf{x}_m$ is the missing part of a data sample and $\mathbf{x}_o$ is the observed part. In general, this requires fitting a different distribution for each possible $(\mathbf{x}_m, \mathbf{x}_o)$ combination. Given data such as mobile traffic measurement snapshots, this becomes computationally infeasible as there are close to $2^{100\times100}$ possible combinations of missing coordinates in a traffic snapshot, when the grid size is equal to 100×100. As such, for missing mobile traffic data it would be more promising to employ Deep Learning based imputation methods, such as General Adversarial Imputation Networks [57] or Denoising Autoencoders [45, 53].

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Imad Alawe, Adlen Ksentini, Yassine Hadjadj-Aoul, and Philippe Bertin. Improving traffic forecasting for 5g core network scalability: A machine learning approach. *IEEE Network*, 32(6):42–49, 2018.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Gianni Barlacchi, Marco De Nadai, Roberto Larcher, Antonio Casella, Cristiana Chitic, Giovanni Torrisi, Fabrizio Antonelli, Alessandro Vespignani, Alex Pentland, and Bruno Lepri. A multi-source dataset of urban life in the city of milan and the province of trentino. *Scientific data*, 2:150055, 2015.

[5] Dario Bega, Marco Gramaglia, Marco Fiore, Albert Banchs, and Xavier Costa-Perez. Deepcog: Cognitive network management in sliced 5g networks with deep learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 280–288. IEEE, 2019.

[6] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[7] Longbiao Chen, Dingqi Yang, Daqing Zhang, Cheng Wang, Jonathan Li, et al. Deep mobile traffic forecast and complementary base station clustering for c-ran optimization. *Journal of Network and Computer Applications*, 2018.

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[9] François Chollet et al. Keras. https://keras.io, 2015.

[10] C Croarkin, P Tobias, J Filliben, B Hembree, W Guthrie, L Trutna, and J Prins. e-handbook of statistical methods. nist/sematech,[online](2010).

[11] Harris Drucker, Christopher JC Burges, Linda Kaufman, Alex J Smola, and Vladimir Vapnik. Support vector regression machines. In *Advances in neural information processing systems*, pages 155–161, 1997.

[12] Luoyang Fang, Xiang Cheng, Haonan Wang, and Liuqing Yang. Mobile demand forecasting via deep graph-sequence spatiotemporal modeling in cellular networks. *IEEE Internet of Things Journal*, 5(4):3091–3101, 2018.

[13] Jie Feng, Xinlei Chen, Rundong Gao, Ming Zeng, and Yong Li. Deeptp: An end-to-end neural network for mobile cellular traffic prediction. *IEEE Network*, 32(6):108–115, 2018.

[14] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[16] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[17] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[18] Jia Guo, Yu Peng, Xiyuan Peng, Qiang Chen, Jiang Yu, and Yufeng Dai. Traffic forecasting for mobile networks with multiplicative seasonal arima models. In *2009 9th International Conference on Electronic Measurement & Instruments*, pages 3–377. IEEE, 2009.

[19] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Chih-Wei Huang, Chiu-Ti Chiang, and Qiuhui Li. A study of deep learning networks on mobile traffic forecasting. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6. IEEE, 2017.

[22] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[23] Cisco Visual Networking Index. Global mobile data traffic forecast update, 2017–2022. *Cisco white paper*, 2019.

[24] GSMA Intelligence. Definitive data and analysis for the mobile industry. https://www.gsmaintelligence.com/. Online; accessed 24 July, 2019.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Rongpeng Li, Zhifeng Zhao, Jianchao Zheng, Chengli Mei, Yueming Cai, and Honggang Zhang. The learning and prediction of application-level traffic data in cellular networks. *IEEE Transactions on Wireless Communications*, 2017.

[27] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

[28] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[29] Eunsung Oh and Bhaskar Krishnamachari. Energy savings through dynamic base station switching in cellular wireless access networks. In *GLOBECOM*, volume 2010, pages 1–5, 2010.

[30] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[31] Yu Peng, Miao Lei, Jun-Bao Li, and Xi-Yuan Peng. A novel hybridization of echo state networks and multiplicative seasonal arima model for mobile communication traffic series forecasting. *Neural Computing and Applications*, 24(3-4):883–890, 2014.

[32] Dario Pompili, Abolfazl Hajisami, and Hariharasudhan Viswanathan. Dynamic provisioning and allocation in cloud radio access networks (c-rans). *Ad Hoc Networks*, 30:128–143, 2015.

[33] Python Core Team. *Python: A dynamic, open source programming language*. Python Software Foundation, 2019.

[34] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*, 2014.

[35] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[36] Vincenzo Sciancalepore, Konstantinos Samdanis, Xavier Costa-Perez, Dario Bega, Marco Gramaglia, and Albert Banchs. Mobile traffic forecasting for maximizing 5g network slicing resource utilization. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[37] Skipper Seabold and Josef Perktold. Statsmodels: Econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference*, volume 57, page 61. Scipy, 2010.

[38] Yantai Shu, Minfang Yu, Oliver Yang, Jiakun Liu, and Huifang Feng. Wireless traffic modeling and prediction using seasonal arima models. *IEICE transactions on communications*, 88(10):3992–3999, 2005.

[39] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[41] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

[42] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[43] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[44] Denis Tikunov and Toshikazu Nishimura. Traffic prediction for mobile network using holt-winters exponential smoothing. In *2007 15th International Conference on Software, Telecommunications and Computer Networks*, pages 1–5. IEEE, 2007.

[45] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[46] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[47] Di Wang and Eric Nyberg. A long short-term memory model for answer sentence selection in question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 707–712, 2015.

[48] Jing Wang, Jian Tang, Zhiyuan Xu, Yanzhi Wang, Guoliang Xue, Xing Zhang, and Dejun Yang. Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[49] Xu Wang, Zimu Zhou, Fu Xiao, Kai Xing, Zheng Yang, Yunhao Liu, and Chunyi Peng. Spatio-temporal analysis and prediction of cellular traffic in metropolis. *IEEE Transactions on Mobile Computing*, 2018.

[50] Yunbo Wang, Zhifeng Gao, Mingsheng Long, Jianmin Wang, and Philip S Yu. Predrnn++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning. *arXiv preprint arXiv:1804.06300*, 2018.

[51] Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[52] Peter R Winters. Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342, 1960.

[53] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in neural information processing systems*, pages 341–349, 2012.

[54] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

[55] Fengli Xu, Yuyun Lin, Jiaxin Huang, Di Wu, Hongzhi Shi, Jeungeun Song, and Yong Li. Big data driven mobile traffic understanding and forecasting: A time series approach. *IEEE transactions on services computing*, 9(5):796–805, 2016.

[56] Qiang Xu, Alexandre Gerber, Zhuoqing Morley Mao, and Jeffrey Pang. Acculoc: practical localization of performance measurements in 3g networks. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 183–196. ACM, 2011.

[57] Jinsung Yoon, James Jordon, and Mihaela Van Der Schaar. Gain: Missing data imputation using generative adversarial nets. *arXiv preprint arXiv:1806.02920*, 2018.

[58] Heiga Ze, Andrew Senior, and Mike Schuster. Statistical parametric speech synthesis using deep neural networks. In *2013 ieee international conference on acoustics, speech and signal processing*, pages 7962–7966. IEEE, 2013.

[59] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[60] Chaoyun Zhang and Paul Patras. Long-term mobile traffic forecasting using deep spatio-temporal neural networks. In *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 231–240. ACM, 2018.

[61] Chuanting Zhang, Haixia Zhang, Dongfeng Yuan, and Minggao Zhang. Citywide cellular traffic prediction based on densely connected convolutional neural networks. *IEEE Communications Letters*, 22(8):1656–1659, 2018.

[62] Liang Zhang, Guangming Zhu, Lin Mei, Peiyi Shen, Syed Afaq Ali Shah, and Mohammed Bennamoun. Attention in convolutional lstm for gesture recognition. In *Advances in Neural Information Processing Systems*, pages 1953–1962, 2018.

[63] Xuan Zhou, Zhifeng Zhao, Rongpeng Li, Yifan Zhou, and Honggang Zhang. The predictability of cellular networks traffic. In *2012 International Symposium on Communications and Information Technologies (ISCIT)*, pages 973–978. IEEE, 2012.