SPATIOTEMPORAL SERIES ANALYSIS AND FORECASTING: NEW DEEP LEARNING ARCHITECTURES ON WEATHER AND CRIME FORECASTING

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Selim Furkan Tekin

August 2022

Spatiotemporal Series Analysis and Forecasting: New Deep Learning Architectures on Weather and Crime Forecasting
By Selim Furkan Tekin
August 2022

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Süleyman Serdar Kozat(Advisor)

Sinan Gezici

Abdullah Aydın Alatan

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

SPATIOTEMPORAL SERIES ANALYSIS AND FORECASTING: NEW DEEP LEARNING ARCHITECTURES ON WEATHER AND CRIME FORECASTING

Selim Furkan Tekin
M.S. in electrical and electronics engineering
Advisor: Süleyman Serdar Kozat
August 2022

We investigate spatiotemporal series through weather and crime forecasting and introduce new successful deep learning architectures that are also applicable to other spatiotemporal domains. First, we present our weather model being an alternative to physical models by addressing the computational cost and the performance. In our weather model, we extend the *encoder-decoder* structure with the *Convolutional Long-short Term Memory* units and enhance the performance with attention and context matcher mechanisms. We perform experiments on high-scale, real-life, benchmark numerical weather datasets. We show that attention matrices model atmospheric circulations and successfully capture spatial and temporal relations. Our model obtains the best scores among the baseline deep learning models and comparable results with the physical models. Secondly, we study high-resolution crime prediction and introduce a new generative model with Graph Convolutional Gated Recurrent Units (Graph-ConvGRU) and multivariate Gaussian distributions. We introduce a subdivision algorithm and create a graph representation to tackle the sparsity and complexity problem in high-resolution spatiotemporal data. By leveraging the flexible structure of graph representation, we encode the relations to state vectors for each region. We then create a multivariate probability distribution from the state vectors and perform prediction at any resolution for the first time in the literature. Our model obtains the best score in our experiments on real-life and synthetic datasets compared to the state-of-the-art models. Hence our model is not only generative but also precise. We also provide the source code of our algorithms for reproducibility.

Keywords: spatiotemporal, weather forecasting, crime forecasting, encoder-decoder, attention, probabilistic graph models.

ÖZET

UZAY-ZAMANSAL SERİLERDE ANALİZ VE TAHMİNLEME: HAVA DURUMU VE SUÇ TAHMİNİNDE YENİ DERİN ÖĞRENME MİMARİLERİ

Selim Furkan Tekin
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Ağustos 2022

Uzay-zamansal serileri hava durumu ve suç tahmini üzerinden çalışmaktayız ve başka uzay-zamansal problemlerde de kullanılabilecek yeni başarılı derin öğrenme mimarileri tanıtmaktayız. İlk olarak, fiziksel modellerin işlem gücü gereksinimlerine ve performanslarına dikkat çekerek hava durumu modelimizi alternatif olarak sunmaktayız. Hava durumu modelimiz kodlayıcı-çözücü yapısını Evrişimli Uzun-Kısa Soluklu Belleklerle büyütmekte ve dikkat ve bağlam eşleştirici mekanizmalarla performansını artırmaktadır. Büyük çaplı, gerçek, literatürde kıstas olan sayısal hava durumu verileri üzerinde deneyler gerçekleştirmekteyiz. Dikkat matrislerinin atmosferik dalgalanmaları nasıl modellediğini ve uzay-zamansal bağıntıları başarılı bir şekilde öğrendiğini göstermekteyiz. Modelimiz, temel derin öğrenme modelleri arasından en iyi skoru ve fiziksel modellerle kıyaslanabilecek sonuçlar almaktadır. İkinci olarak, yüksek çözünürlüklü suç tahminini çalışmaktayız ve Çizge Evrişimli Geçitli Tekrarlayan Ünite ve Çoklu Gauss Dağılımlarını kullanarak yeni üretici model sunmaktayız. Yüksek çözünürlüklü uzay-zamansal verideki seyreklik ve karışıklık problemini çözmek için çizge gösterimini oluşturmaktayız ve parselleme algoritması sunmaktayız. Çizge gösteriminin kullanışlı yapısı sayesinde, bağıntıları bağlam vektörlerine kodlamaktayız ve Çoklu Gauss Dağılımını oluşturarak literatürde ilk olan istenilen her çözünürlükte tahmin üretmekteyiz. Modelimiz gerçek ve sentetik veride en son kullanılan modeller arasında en iyi performansı göstermektedir. Bu yüzden modelimiz sadece üretken değil aynı zamanda doğrudur. Ek olarak, kaynak kodlarımızı deneyleri tekrar gerçekleştirmek için sağlamaktayız.

Anahtar sözcükler: uzay-zamansal, hava durumu tahmini, suç tahmini, kodlayıcı-çözücü, dikkat, olasılıksal çizge modeli.

Acknowledgement

First of all, I would like to thank Prof. Süleyman Serdar Kozat for his wise supervision during my M.S. studies. From the day I walked in his door, I felt his consistent support and encouragement, and I feel truly fortunate to have worked with him. I have had fruitful graduate studies as I have been able to author studies for highly respected journals and conferences while working as a full-time Machine Learning Engineer facing real-life problems. I should particularly acknowledge that his pursuit of perfection with a fantastic work ethic was both motivational and contagious. Therefore, I do not doubt that he will continue to raise hardworking and successful researchers.

I would like to thank Prof. Sinan Gezici and Prof. Abdullah Aydın Alatan as my examining committee members. I also would like to thank my professors in Bilkent, Fazlı Can, Tolga Çukur, and Abdullah Ercüment Çiçek for all their support.

I would like to thank my fellow research mates in Prof. Kozat's group. To name a few, Fatih İlhan was always helpful, and he mentored me on how to be an excellent researcher with his incredible work ethic. I will never forget the cultural chats from music to movies we had during our challenging journeys. They were always refreshing and lively talks, which I would like to continue in person when I see him in Atlanta. Oğuzhan Karaahmetoğlu provided all the great advice, cheer, and happiness during my most stressful moments with his keen intelligence. My friend, I believe we have accomplished our walk to Mordor.

I would like to thank all my coworkers at Databoss, İsmail Balaban, Yunus Emre Özertaş, İlker Sıgırcı, Ertuğ Ufuk, Furkan Küçük, Mustafa Mert Keskin, Doğan Can Çiçek, and Arda Fazla. I have really enjoyed our memorable experiences and the opportunity to work closely with you. I appreciate your support and friendship that made Databoss a better workplace for me. I hope our paths will cross in the future once again. I wish Doğan Can Çiçek and Arda Fazla much success in their upcoming Ph.D. applications.

I would like to give my best regards to my friends during my master's. Mahmut Yurt, I enjoyed every minute we spent together. I wish we had met much earlier so I would have more of your vision and humor. Your constant support and guidance helped me achieve my goals, and I am very grateful. My dearest friends, Kemal Vatansever, Zeynep Şendil, Menar Ekizce, and Berk Alparslan thank you for all your supports. I want to give special thanks to my friend Furkan Şahinuç for answering all the questions I asked and for his hospitality. We started our data science journey together, and I wish luck for both of us in our Ph.Ds. Lastly, I would like to thank Mert Bozkurt, Arda Turgut, and Mert Şahin for their support. I wish them all the best.

Most importantly, I would like to express my deepest gratitude to my family, who have always supported and encouraged me throughout my journey. They always believed in me, trusted my decisions, and shared my feelings. Without their unconditional love and guidance, I could not have gone this far. Therefore, every success I achieve in my life is undoubtedly theirs.

Contents

List of Figures

List of Tables

List of Publications

This thesis includes content from following publications(s):

1. S. F. Tekin, O. Karaahmetoğlu, F. İlhan, İ. Balaban, and S. S. Kozat, "Spatio-temporal weather forecasting and attention mechanism on convolutional lstms," *arXiv preprint arXiv:2102.00696,* 2021.
2. S. F. Tekin and S. S. Kozat, "Crime prediction with graph neural networks and multivariate normal distributions," *Signal, Image and Video Processing*, 2022.

Chapter 1

Introduction

1.1 Preliminaries

A spatiotemporal series is a collection of series with temporal and spatial covariances, such as the temperature at numerous weather stations, crime events happening in various spots, or the traffic in cities. Thus, effective processing of spatiotemporal series plays a vital role in mitigating high-impact events. The distribution of the time series locations can be in order as a grid or unordered as a graph. For example, we can represent the traffic flow in the city by a graph where each node belongs to boulevards, and the weather of the city can be represented by a grid where nodes are equidistant from each other, forming a rectangular shape. In data science and computer vision, new successful deep learning models for spatiotemporal series forecasting are introduced for grid and graph representations. In this thesis, we study spatiotemporal series on weather and crime forecasting problems, where we present new deep learning architectures in grid and graph representations. We show that both architectures are practical and applicable to real-life problems. Thus, our thesis can guide researchers in applying our architectures to the different spatiotemporal domains.

In weather forecasting, we aim to create an accurate and interpretable model

that has comparable performance with the physical models. We show that our method brings significant developments in this field and can become an alternative to currently used short-term physical models, which are hardly accessible for data scientists. We perform experiments on real-life benchmark datasets and show that our model performs better than the baseline models.

In crime forecasting, our goal is to perform high-resolution crime predictions with a generative deep learning model. We overcome the sparsity issue of grid base representation with a subdivision algorithm, which allows us to use a graphical deep learning model. We show that the proposed architecture leads to high performance compared to baseline models and applies to different domains.

In both problems, we leveraged the high performance of deep learning, a subclass of machine learning. A learning algorithm contains three key elements defined in [2] and [3], the learning experience, E, with respect to a task, T, and performance measure, P. In each iteration of the learning algorithm, the performance P at tasks in T improves with experience E. For crime forecasting, our task is classification, where we classify given input according to the probability of a crime event happening, i.e., the learning algorithm learns the following mapping, $f : \mathbb{R}^N \to \{0, 1\}$. For weather forecasting, the task is the regression where we predict the temperature value given other weather features as input, i.e., we learn the following mapping $f : \mathbb{R}^N \to \mathbb{R}$. In both tasks, we use a performance measure to assess the learning algorithm, such as the accuracy and the squared error. The experience of machine learning algorithms can be categorized as unsupervised or supervised by their access to the label of the inputs. In supervised experience, we observe several examples of a random vector $\mathbf{x}$ and associated label $\mathbf{y}$, and then we aim to predict $\mathbf{y}$ from $\mathbf{x}$ by estimating $p(\mathbf{y}|\mathbf{x})$. In unsupervised experience, we attempt to learn $p(\mathbf{x})$ [2]. We perform supervised experience in both problems.

We can represent a supervised learning model with the following mapping, $y = f^*(\mathbf{x})$. One of the key elements of deep learning is Multilayer Perceptrons (MLPs) which allows us to approximate the function f^* with $y = f(\mathbf{x}, \theta)$ by learning the best θ parameters [2]. MLPs consist of chained functions that can

learn non-linear relationships between the input and the label. For example, in a three-layered MLP, we have $f(\mathbf{x}) = f^3(f^2(f^1(\mathbf{x})))$, where each f represents a layer of MLP containing their individual sets of parameters. We aim to match f with f^* by finding the best set of parameters. During the training, we compare each produced output with the label using performance measures and update the parameters from this experience. This process is called gradient-based training. The algorithm takes steps on parameter space toward minimizing the loss function.

The layers of MLP contain matrix multiplications representing each function, such as $f^1(\mathbf{x}, \theta) = f^1(\mathbf{x}; \mathbf{w}, b) = \mathbf{x}^T\mathbf{w} + b$, where $\mathbf{w}$ and b are the parameters we learn. We can also represent the second layer $f(\mathbf{h}; \mathbf{w}, b) = \mathbf{h}^T\mathbf{w} + b$, where $\mathbf{h}$ is the output of the first layer. However, altogether this representation is linear. Thus, we must use non-linear activation functions such as sigmoid, softmax, or tanh in each layer to describe the input. As the number of layers, i.e., the depth of the model, increases, the MLP model's capacity to learn non-linear relations within the input and output also increases. However, the high number of parameters causes the model to overfit, i.e., each layer memorizes what the output should be and fail to generalize. The high number of parameters also increases the cost and the time to train the model. Furthermore, the high number of parameters can cause the gradient-based algorithm to get stuck in the local minimum.

A special kind of neural network called Convolutional Neural Network (CNN) is introduced and performs better on generalizing 2D grid topologies such as images [4]. In place of general matrix multiplications, CNNs perform the convolution operation. The CNNs leverage three important ideas, sparse interactions, parameter sharing, and equivariant representation [2]. The sparse interaction means that CNN works locally on the input with a small kernel that can detect small, meaningful features, improving cost and statistical efficiency. Contrary to the matrix multiplication, in CNN, each output unit is not affected by each input unit creating a sparse connection. Since the kernel in the convolution operation travels along the image, the kernel parameters are shared, allowing less number of parameters. Parameter sharing also causes the layers to be equivariant to the

translations, such as shifting. For example, the model learns to detect a particular edge and can recognize that edge elsewhere in the entire image. With these properties, CNNs become one of the fundamental units in deep learning architectures. Furthermore, the authors of [5] generalized the CNNs to graph topologies to create Graph Convolutional Networks, which is the building block of our crime forecasting model.

In our problems, we work with sequences, our input is single feature vectors $\mathbf{x} \in \mathbb{R}^N$ in MLPs and CNNs, but now we have an ordered list of feature vectors $\mathbf{x}_1, \ldots, \mathbf{x}_T$, where $\mathbf{x}_t \in \mathbf{R}^N$. Similar to unsupervised density modeling, we aim to estimate the probability mass function of the input sequence $p(\mathbf{x}_1, \ldots, \mathbf{x}_T)$. One can use chain rule and conditional probabilities to represent the joint distribution:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \prod_{t=0}^{T-1} p(\mathbf{x}_{t+1} | \mathbf{x}_1, \ldots, \mathbf{x}_t).$$

One typical application of deep learning is to model this distribution via Recurrent Neural Networks (RNN) [6]. The output of an RNN at a single time step represents each $p(\mathbf{x}_{t+1}|\mathbf{x}_1, \ldots, \mathbf{x}_t)$, creating the latent variable model $p(\mathbf{x}_{t+1}|\mathbf{x}_1, \ldots, \mathbf{x}_t) \approx p(\mathbf{x}_{t+1}|\mathbf{h}_t)$. The $\mathbf{h}_t$ is the hidden state that stores the sequence information up to time step t. An RNN contains three different MLPs that take input and the former hidden-state vectors to produce the next hidden state. Following RNNs, new sequential processing deep learning units are introduced, such as Long-short Term Memory Units (LSTM) and Gated Recurrent Unit (GRU). These architectures showed high performance on sequence modeling, language-speech processing, time series forecasting, and auto-regressive models.

This thesis introduces two new deep learning architectures leveraging the latest developments on MLPs, CNNs, and RNNs and assessing their performance on two complex real-life problems.

1.2 Contributions

This thesis first addresses high-scale weather forecasting with a novel deep learning model. We introduce a novel attention-based recurrent network architecture called *Weather Model* for spatiotemporal series with convolution operations on attention layers and ConvLSTMs as memory units. This model can focus on the relevant cells in exogenous series to make accurate predictions and successfully learn spatial correlations. To the best of our knowledge, we apply an attention-based ConvLSTM network to spatiotemporal weather forecasting for the first time in the literature. This model architecture allows us to use multiple exogenous series and produce refined predictions with high interpretability. In addition, we can make hourly, daily, or in any temporal resolution forecasts. We compare our model with conventional deep learning and physical models using the benchmark metrics through extensive experiments over benchmark datasets and show that our model obtains higher performances than all the baseline models and some physical models.

Secondly, this thesis introduces a novel solution to high-resolution crime forecasting. We introduce a subdivision algorithm to create a graph from grid-based spatiotemporal data to reduce the detrimental effects of sparsity. We present a novel graphical architecture, *High Resolution Crime Forecaster*, with Multivariate Gaussian distributions to jointly train the micro and macro probabilities and model the actual data distribution. With the generative structure of our model, we produce highly accurate predictions even for high spatial resolution. Furthermore, we demonstrate significant performance gains through extensive experiments over real-life crime data.

1.3 Outline

In the remainder of this thesis, the organization is as follows. Chapter 2 describes in detail the weather model for high-scale weather forecasting. Chapter 3 then

introduces the high-resolution crime forecaster. Lastly, Chapter 4 portrays the conclusive remarks.

Chapter 2

Numerical Weather Forecasting by Convolutional-LSTMs with Attention Mechanism Using ERA5 Reanalysis Data

2.1 Introduction

The weather is a vital scientific research area by being the cornerstone of every real-life system, such as agriculture, energy production, tourism, transport, navigation, and climate [7]. Thus, accurate forecasts prevent life and economic losses and support emergency management by diminishing the effects of high-impact weather events and creating substantial financial revenue [7]. Foundations of modern-day weather forecasting were introduced in the early twentieth century when scientists solved a system of nonlinear differential equations by hand [8, 9]. Today, super-computers solve the equations for millions of points per time step to forecast weeks to months ahead [10]. Meteorologists call this problem *numerical weather prediction* (NWP), where the objective is to predict a numerical value of a weather feature such as temperature or wind speed for a location. However,

NWP models are getting more complex, and their demand for high computation power increases continuously [10]. Obtaining results from these models can take hours, limiting their ability to provide actionable predictions.

The NWP for multiple locations is a grid-based spatiotemporal series problem. Compared to physical models for NWP, deep learning models can provide results within the minutes of receiving data, exploit the big data aggregated in years, and make accurate predictions with less costly models [11]. The success of the spatiotemporal series forecasting with deep learning emerged with the introduction of Convolutional Long-short Term Memory (ConvLSTM) cells in [12], a modified version of Long-short Term Memory (LSTM) cells of [13] that is targeted to capture spatial patterns with convolution operations. Specifically, the ConvLSTMs focused on the precipitation nowcasting problem, where this task aims to give a precise and timely prediction of rainfall intensity in a local region over a relatively short period. Following this model, [12, 14] works introduce new approaches to model spatial and temporal patterns and show high performance in different tasks, such as weather, crime, and traffic density forecasting.

However, previous studies, [12, 14, 15], do not exploit the numerous data sets available for weather, such as satellite images, numerical grid values, and observations in meteorological stations. These works have focused on one input source, such as a sequence of radar images. The weather, however, is a chaotic system affected by many exogenous factors, e.g., vegetation, geographical contour, and human. Therefore, more data source is needed to model such chaotic behaviors. In this thesis, we call our target series as the endogenous variable, and the different data sources entering the forecasting model as the side-information or exogenous variable. As a solution, the side-information can be concatenated with the input series and directly given to the model. However, the side-information integration in previous works shows more effective ways [16–18]. Authors of [19] show that implementing Fully Connected Neural Networks (FC-NNs) before the memory units allowed the model to focus on the patterns of the exogenous series that are correlated with the endogenous series. This focusing process is known as attention and was introduced in [20, 21]. Even though FC-NNs are effective,

they cannot capture spatial information as well as Convolutional Neural Networks (CNNs). Thus, the authors of [22] introduced a convolutional attention mechanism for gesture recognition and showed the need for spatial attention in ConvLSTM structures.

Moreover, the authors of [23] showed the high performance of the U-net model in weather forecasting. The U-net contains multiple layers of CNNs with skip connections. Similar to [12,14,15], it follows the idea of encoding input spatiotemporal series to a small abstract representation and decoding this representation to generate an output sequence. The advantage of an encoder-decoder structure is that the length of the input and output sequence may differ. Thus, one can obtain any length of the forecast.

Recently, the authors of [1] released a benchmark dataset from the ERA5 archive and introduced evaluation metrics to make inter-comparison between studies. Furthermore, they provide baseline scores for deep learning and physical forecasting models. They show that their deep learning model makes accurate short-term predictions, yet its performance deteriorates rapidly as the output sequence length rises.

To this end, we introduce a novel deep learning architecture for the NWP by combining multiple spatiotemporal data sources with an attention mechanism and using ConvLSTM cells as building blocks to capture spatial and temporal correlations, as shown in Figure 2.1. Our model can select relevant cells in multiple spatiotemporal series to make long-term spatial predictions by preserving the long-term dependencies with attention and context matcher mechanisms. The encoder block consists of stacked ConvLSTM units, which encode the input sequence to hidden states. In the second stage, the context-matcher mechanism matches the decoder's hidden states by summing the encoder's hidden states across all time steps to carry out the long-term dependencies by increasing the length of gradient flows. Each ConvLSTM unit in the decoder block decodes the passed information from the hidden layers of the encoder. The output of the decoder then passes through the output convolutions to produce predictions. Altogether, the architecture focuses on different parts of spatiotemporal series

and successfully captures the long-term dependencies. We call the architecture as *Weather Model* (WM). We demonstrate significant performance gains through extensive experiments on two real-life datasets from the ERA5 archive and obtain predictions for hours and days ahead. We compare our model with conventional deep learning and physical models using the benchmark metrics and show that our Weather Model is accurate and easy to interpret.

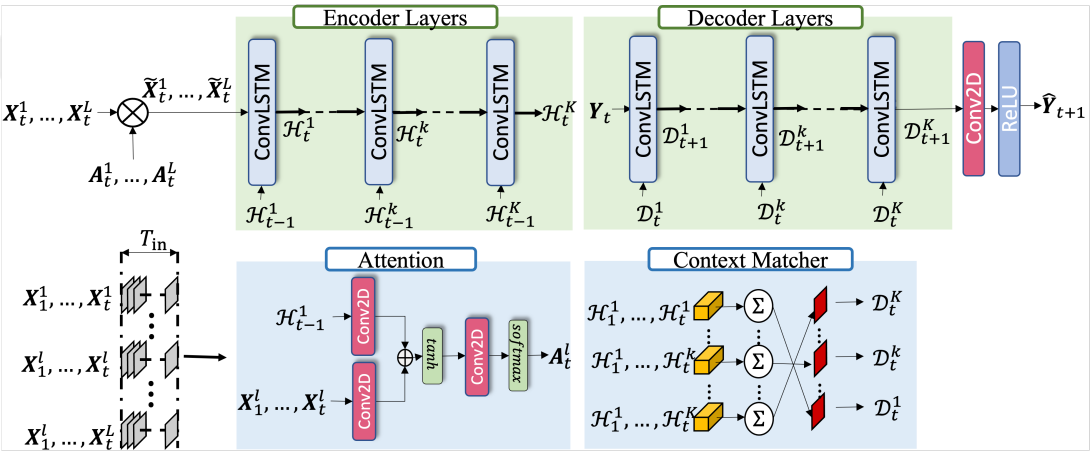


Figure 2.1: At each time step, t, we first multiply all the weather features, $\{\mathbf{X}_t^1, \cdots, \mathbf{X}_t^l, \cdots, \mathbf{X}_t^L\}$, with the attention matrices $\{\mathbf{A}_t^1, \cdots, \mathbf{A}_t^l, \cdots, \mathbf{A}_t^L\}$ to create the weighted input series $\{\tilde{\mathbf{X}}_t^1, \cdots, \tilde{\mathbf{X}}_t^l, \cdots, \tilde{\mathbf{X}}_t^L\}$, where $\tilde{\mathbf{X}}_t^l = \mathbf{X}_l^t \odot \mathbf{A}_t^l$. The attention mechanism computes the attention weights, $\mathbf{A}_t^l$, for each feature. The attention weights are conditioned on previous hidden state of the encoder's first layer, $\mathcal{H}_{t-1}^1$. The weighted inputs series enter to the first ConvLSTM unit in the encoder. After the input passes all the encoder layers, the encoder's hidden states, $\{\mathcal{H}_1^1, \cdots, \mathcal{H}_1^k, \cdots, \mathcal{H}_t^K\}$, enter to the context matcher, where k represents the layer index. The context matcher sums the hidden states in the time dimension and reverses the states layer order to create $\{\mathcal{D}_{t-1}^1, \cdots, \mathcal{D}_{t-1}^k, \cdots, \mathcal{D}_{t-1}^K\}$. Finally, the output of the decoder, $\mathcal{D}_t^K$ passes to output convolutions to create prediction $\hat{\mathbf{Y}}_{t+1}$.

2.1.1 Prior Art and Comparisions

Applying deep learning methods to weather forecasting is an extensively studied area. In [24], the authors leveraged a massive volume of weather datasets to train an Auto-encoder with multi-layered Neural Networks (NNs) to forecast hourly weather data. The authors of [25] improved this approach by introducing stacked

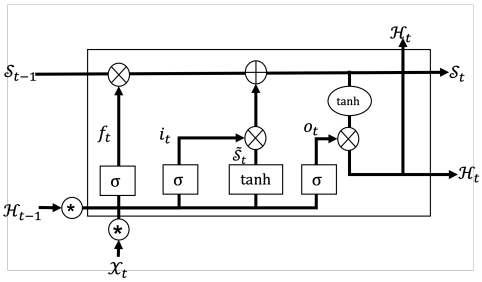


Figure 2.2: We show the inside of a ConvLSTM cell with the input i_t, cell s_t, forget f_t, and output o_t gates and the operations in each gate. Joining arrows represent the concatenation of the matrices. Note the convolution operations on $\mathcal{H}_{t-1}$ and $\mathcal{X}_t$, and there are two hidden states generated in each time step, $\mathcal{H}_t$ and $\mathcal{S}_t$.

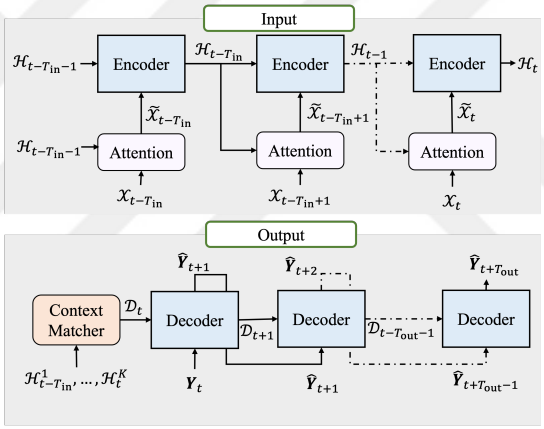


Figure 2.3: We show the unrolled view of the encoder and decoder structure. Input sequence first passes to the encoder, which encodes the input sequence to hidden states. Then, for the first prediction, the decoder takes $\mathbf{Y}_t$ and $\mathcal{D}_t$ to predict $\hat{\mathbf{Y}}_{t+1}$. Next, the decoder uses this prediction and the former state, $\mathcal{D}_{t+1}$, to predict the next output, $\hat{\mathbf{Y}}_{t+2}$. This recursive process continues until $t = T_{\text{out}}$.

auto-encoders to temperature forecasting. They feed corrupted inputs and reconstruct the clean data to obtain good representation. Further, [26] implemented a simple NN to forecast rainfall using lagged climate indices from various stations and historical rainfall data. These works showed the high performance of simple deep learning architectures and their potential when fed with a high volume of data. Moreover, authors of [27, 28] applied Recurrent Neural Networks (RNNs) and LSTMs to weather forecasting, and [29] focused on the vanishing gradient problem by implementing intensified LSTM architecture for rainfall forecasting. In [30], the authors introduced deep uncertainty quantification using RNN-based

architecture and obtained a single-value forecast. Recently, [31] used Temporal Convolutional Network (TCN) for multiple-output and single-output weather predictions and showed better results than a physical NWP model, The Weather Research and Forecasting model (WRF) [32]. However, this approach is costly, does not comprise multiple features and spatial covariances, and provides high performance on only multiple-inputs and single-output models.

As we introduced in the introduction, the authors of [12] introduced the ConvLSTM and Convolutional Gated Recurrent Units (ConvGRU) with the forecasting network, which consists of an encoder and a decoder built by stacked ConvLSTMs or ConvGRUs. These models can capture the spatial-temporal covariances and showed high performance in the rainfall intensity prediction by using radar echo sequences for model training. The authors also showed that ConvLSTM is more accurate than the optical flow model [33] and better at capturing spatiotemporal correlations than the fully-connected LSTM. Later, [14] introduced Trajectory Gated Recurrent Unit (Traj-GRU) to learn location invariant movements between input radar images and showed slightly higher performance compared to ConvGRU. Although the ConvLSTM and TrajGRU offer high performance, the encoder-decoder structure has the limitation of producing an output with the same input dimension. Thus, [34] improved the encoder-decoder architecture with the downsampling and upsampling layers between TrajGRU units and omitted the problem of single-dimensional input with an output convolution layer. Furthermore, in [35], the authors introduced the recursive decoder prediction in the encoder-decoder architecture to enhance the long-term forecast. To further show the learning capacity of ConvLSTMs, the authors of [36] implemented Generative Adversarial Neural Network (GAN) using ConvLSTM units to extrapolate echos to a higher resolution by using adversarial loss. Moreover, U-Net showed high performance on precipitation nowcasting in [23], which is comparable with ConvLSTMs.

Attention mechanisms in the spatiotemporal domain are studied in various tasks. For human action recognition, [37] developed temporal and spatial attention mechanisms with fully connected NNs and LSTMs. Likewise, [19] implemented a dual attention mechanism in time-series forecasting. The authors

of [38] implemented a multi-level attention mechanism fusing the side information with multiple geo-sensory time-series data to forecast air quality. For the traffic prediction, [39] implemented an attention mechanism on the outputs of LSTM, which takes inputs from different past periods. Although these works successfully implement an attention mechanism, the input is single or multiple time-series rather than 3D tensors. The authors of [22] implemented an attention mechanism to improve ConvLSTM further, which took 3D tensors as input. However, the performance increase is limited. Recently, [40] compared CNN following LSTM (Conv + LSTM) with ConvLSTM in temporally unrolled architecture and temporally concatenated architecture. They observed the loss change according to different inputs to interpret the model decisions. Compared to Conv + LSTM, ConvLSTM showed higher performance. However, they obtained interpretable results only for the Conv + LSTM model, and their interpretation is on feature and location level rather than the activations of the model.

The authors of [41] recently adopted the self-attention idea and introduced transformers, a model that performs multiple attention mechanisms in parallel. The transformers show high performance in translation and text summarization. Following their work, [42] and [43] introduced pre-trained systems that one can finetune with just one output layer to create state-of-the-art models for a wide range of language processing tasks. The authors of [44] introduced Vision Transformer (ViT), which adopted the transformers to the sequence of image patches and obtained high performance on image classification tasks. In [45], the authors developed Swin Transformers to address domain differences between language and vision transformers. The Swin Transformers has the flexibility to model at various scales and has linear computational complexity. However, the vision transformers perform per image. Thus, the authors [46] introduced Video Swin Transformers and obtained benchmark results on the video recognition task. Even though transformers show high performance, they require high computational power to train and the pre-trained architectures are suitable for pixel-based models, unlike the spatiotemporal weather data.

Finally, the authors of [1] introduced a benchmark dataset, ERA5, to predict global weather patterns in advance. They provide baseline scores from simple

linear regression techniques, deep learning models, and purely physical forecasting models.

2.2 Problem Description

We represent all vectors as column vectors and denote them by bold lower case letters, such as $\mathbf{b}, \mathbf{h}$. We represent matrices as bold upper letters, such as $\mathbf{X} \in \mathbb{R}^2$, and we denote the tensors by uppercase letters with calligraphic font, such as $\mathcal{X}, \mathcal{Y}, \mathcal{W} \in \mathbb{R}^n$, where $n > 2$. The time index is given as subscript and superscript refers to other signals, for example, $\mathbf{X}_t^l$ shows k^l input signal's t^{th} time step. In addition, we show the entries of a matrix with lower case letters with subscript referring to its position, for example, $a_{i,j}$ shows the entry of matrix $\mathbf{A}$ at the i^{th} row and j^{th} column.

Our goal is to forecast the weather value of multiple locations using multiple features of historical numeric weather data. The input is the weather features with the grid representation creating a set of spatiotemporal series. We can show the weather values of a feature l at a time step t as a matrix, $\mathbf{X}_t^l = \{x_{1,1,l}^k, \ldots, x_{i,j,t}^k, \ldots, x_{M,N,t}^l\} \in \mathbb{R}^{M \times N}$, where M and N define the spatial dimensions and $x_{i,j,t}^l$ represents the cell value in the grid at the i^{th} row and j^{th} column. Following this definition, for a given time window T, $\mathcal{X}^l = \{\mathbf{X}_1^l, \ldots \mathbf{X}_T^l\} \in \mathbb{R}^{M \times N \times T}$ represents the spatiotemporal series for the weather feature l. Similarly, for the total number of L features, $\mathcal{X}_t = \{\mathbf{X}_t^1, \ldots \mathbf{X}_t^L\} \in \mathbb{R}^{M \times N \times L}$ represents all the weather features at time step t. Altogether, we represent the set of spatiotemporal series as $\mathcal{X} = \{\mathbf{X}_1^1, \cdots, \mathbf{X}_1^L, \mathbf{X}_2^L, \cdots, \mathbf{X}_T^L\} \in \mathbb{R}^{M \times N \times T \times L}$. Next, we define $\mathbf{Y}_t \in \mathbb{R}^{M \times N}$ as our endogenous variable and we represent the target series as $\mathcal{Y} = \{\mathbf{Y}_1, \cdots, \mathbf{Y}_T\} \in \mathbb{R}^{M \times N \times T}$. The target series is one of the series in the set of spatiotemporal series, i.e. $\mathcal{Y} \in \mathcal{X}$.

Here, our goal is to predict the next sequence of the target series given the previous input series. Mathematically, our objective is to learn the following

mapping:

$$\{\mathbf{X}_1^1, \ldots, \mathbf{X}_{T_{\text{in}}}^L\} \xrightarrow{\text{g(.)}} \{\mathbf{Y}_{T_{\text{in}}+1}, \cdots \mathbf{Y}_{T_{\text{out}}}\},$$

where $T_{\text{in}} < T$ and $T_{\text{out}} < T$ are the input and output sequence lengths, respectively. In numerical weather prediction the target value $\mathbf{Y}_t$ is continuous. Thus, our task is regression. Since we have targets to every input data, learning is supervised and we suffer mean squared error:

$$\mathcal{L} = \frac{1}{T_{\text{out}} MN} \sum_{t=1}^{T_{out}} \sum_{i=1}^{M} \sum_{j=1}^{N} (y_{i,j,t} - \hat{y}_{i,j,t})^2 \tag{2.1}$$

where $\hat{y}_{i,j,t}$ represents the predicted cell value and $y_{i,j,t}$ is the ground truth. We minimize the equation 2.1, the loss function, with Adam optimizer [47] using batches of the training data.

2.3 The Weather Model

In this section, we describe the developed model architecture by referring to every step and formulation shown in Figure 2.1.

2.3.1 Insights into Convolutional LSTMs

One of the critical temporal models in deep learning is the RNN [48]. An RNN stores the temporal information of the input sequence within the hidden state vectors, i.e., its memory. The ability to carry information through time in their memory makes the RNNs successful in the time series forecasting tasks. An RNN consists of a input sequence $\mathbf{x} = \{\mathbf{x}_1, \cdots, \mathbf{x}_T\}$, $\mathbf{x}_t \in \mathbb{R}^{L_{\text{in}}}$, a hidden state $\mathbf{h} \in \mathbb{R}^{L_{\text{hid}}}$, and an output $\mathbf{y} \in \mathbb{R}^{L_{\text{out}}}$, where $T, L_{\text{in}}, L_{\text{hid}}, L_{\text{out}}$ refer to sequence length, the input, hidden, and output dimensions, respectively. At each time

step t, the hidden state is updated by,

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t),$$

where f represents the RNN update equations, which are:

$$\mathbf{h}_t = f_h(\mathbf{W}_h \mathbf{x}_t + U_h \mathbf{h}_{t-1} + b_h),$$
$$\mathbf{y}_t = f_y(\mathbf{W}_y \mathbf{h}_t + b_y),$$

where $\mathbf{W}_y \in \mathbb{R}^{L_{\text{hid}} \times L_{\text{out}}}$, $\mathbf{W}_h \in \mathbb{R}^{L_{\text{in}} \times L_{\text{hid}}}$, b_h and b_y represent training weights of the network, where L_{hid}, L_{out} are the hidden and output dimensions. The functions, f_h and f_y are the non-linear activation functions, such as tanh or sigmoid.

However, the RNNs have only one hidden vector, $\mathbf{h}_t$, to store the temporal information. Thus, they have difficulty of capturing long term dependencies [13]. Moreover, RNNs suffer from vanishing gradients problem [19]. To remedy the problems of RNNs, the authors of [13] introduce LSTMs, which contain two hidden vectors and an additional memory vector to store the information. The LSTMs are more successful in modeling the long-term dependencies compared to RNNs.

Furthermore, the ConvLSTM improves the setup of the LSTMs by introducing the convolution operations to the input and the hidden state transitions and replacing its vectors with matrices. This architecture can take multiple time series as inputs to produce outputs for each series simultaneously. The convolution operations allow the ConvLSTM to capture the spatial correlations. Each ConvLSTM unit has inputs $\mathcal{X}_t = \{\mathbf{X}_t^1, \cdots, \mathbf{X}_t^{L_{\text{in}}}\} \in \mathbb{R}^{M \times N \times L_{\text{in}}}$, cell outputs $\mathcal{S}_t = \{\mathbf{S}_t^1, \cdots, \mathbf{S}_t^{L_{\text{hid}}}\} \in \mathbb{R}^{M \times N \times L_{\text{hid}}}$, hidden states $\mathcal{H}_t = \{\mathbf{H}_t^1, \cdots, \mathbf{H}_t^{L_{\text{hid}}}\} \in \mathbb{R}^{M \times N \times L_{\text{hid}}}$ and gates i_t, f_t, o_t in 3D tensor format, where L_{in}, L_{out}, and L_{hid} specify the input, output, and hidden dimension numbers, respectively. Here we show the gates in lowercase to be consistent with the original equations shown in [12]. $\mathcal{X}_t$ and $\mathcal{H}_t$ are the results of 2D convolutions in the input gates shown in Figure 2.2.

The update equations of the gates in ConvLSTM are as follows:

$$
\begin{aligned}
i_t &= \sigma(\mathcal{W}_{xi} * \mathcal{X}_t + \mathcal{W}_{hi} * \mathcal{H}_{t-1} + \mathcal{W}_{si} \odot \mathcal{S}_{t-1} + \mathbf{b}_i), \\
f_t &= \sigma(\mathcal{W}_{xf} * \mathcal{X}_t + \mathcal{W}_{hf} * \mathcal{H}_{t-1} + \mathcal{W}_{sf} \odot \mathcal{S}_{t-1} + \mathbf{b}_f), \\
\mathcal{S}_t &= f_t \odot \mathcal{S}_{t-1} + i_t \odot \tanh(\mathcal{W}_{xs} * \mathcal{X}_t + \mathcal{W}_{hs} * \mathcal{H}_{t-1} + \mathbf{b}_s), \\
o_t &= \sigma(\mathcal{W}_{xo} * \mathcal{X}_t + \mathcal{W}_{ho} * \mathcal{H}_{t-1} + \mathcal{W}_{so} \odot \mathcal{S}_t + \mathbf{b}_o), \\
\mathcal{H}_t &= o_t \odot \tanh(\mathcal{S}_t),
\end{aligned}
$$

where $*$ and $\odot$ are the convolution and element-wise multiplication operators respectively. $\mathcal{W}_{x\sim} \in \mathbb{R}^{I \times J \times L_{\text{in}} \times L_{\text{hid}}}$, $\mathcal{W}_{h\sim} \in \mathbb{R}^{I \times J \times L_{\text{out}} \times L_{\text{hid}}}$, $\mathcal{W}_{s\sim} \in \mathbb{R}^{I \times J \times L_{\text{hid}} \times L_{\text{hid}}}$, and $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^{L_{\text{hid}}}$ are the parameters to learn and I, J are the height and width of the filters respectively. The states in the ConvLSTM are the hidden representation of moving objects, where the filters of ConvLSTM capture any movement pattern [12]. We observe the movement patterns in the weather features as well, e.g. temperature, wind, humidity due to the generic atmospheric circulation. ConvLSTM learns the flow movement of atmosphere and passes this information to the next states. As shown in Figure 2.1, we use stacked ConvLSTM units to deepen the model's learning capacity.

2.3.2 Encoder and Attention Mechanism

The authors of [49, 50] introduced the encoder-decoder structure using RNNs as building blocks and showed high performance in the language translation task, where the model learns the meaningful representations of linguistic phrases. Apart from its learning capability, the essential advantage of the encoder-decoder structure is that the length of the input and output sequence may differ, making it suitable for translation and time-series forecasting tasks. The model aims to encode the input sequence to a latent representation and decode it into the target sequence. An encoder with multiple layers of RNNs encodes the input sequence into a context vector by referring to the previous hidden state. For example, an input sequence with the length of T_{in}, $\{\mathbf{x}_1, \cdots, \mathbf{x}_{T_{\text{in}}}\}$ enters the model, and the

encoder block summarizes the input sequence into a fixed-length context vector:

$$\mathbf{h}_t = f_{\text{enc}}(\mathbf{x}_t, \mathbf{h}_{t-1}),$$
$$\mathbf{c} = \mathbf{h}_{T_{\text{in}}},$$

where $\mathbf{h}_t$ is the hidden state of the encoder at time step t. The last hidden state, $\mathbf{h}_{T_{\text{in}}}$, is selected as the context vector, $\mathbf{c}$ [51]. The f_{enc} represents a non-linear function such as RNN, LSTM, or GRU [52]. Further, the decoder produces the target sequence with the length of T_{out}, $\{\mathbf{y}_1, \cdots, \mathbf{y}_{T_{\text{out}}}\}$ using the context vector and the previous steps of the target sequence as follows:

$$\mathbf{d}_1 = f_{\text{dec}}(\mathbf{y_1}, \mathbf{c}), \tag{2.2}$$
$$\mathbf{d}_t = f_{\text{dec}}(\mathbf{y_t}, \mathbf{d}_{t-1}), \tag{2.3}$$

where $\mathbf{d}_1$ is the first output of the decoder and $\mathbf{d}_t$ represents the subsequent outputs of the decoder. The output sequence can directly be the decoder states or further operations can be performed such as softmax. Note that in the architecture, there are two different non-linear functions representing the encoder and decoder learning units.

Following this architecture, we use the encoder-decoder structure to model the spatiotemporal sequences, as shown in Figure 2.1. For the L number of spatiotemporal input sequences $\tilde{\mathcal{X}}_t = \{\tilde{\mathbf{X}}_t^1, \cdots, \tilde{\mathbf{X}}_t^L\}$ the encoder learns the following mapping:

$$\mathcal{H}_t, \mathcal{S}_t = f(\tilde{\mathcal{X}}_t, \mathcal{H}_{t-1}, \mathcal{S}_{t-1}), \tag{2.4}$$

where $\mathcal{H}_t, \mathcal{S}_t \in \mathbb{R}^{M \times N \times L_{\text{hid}}}$ are the hidden states of the encoder at time t and $\tilde{\mathcal{X}}_t$ is the output of the attention mechanism. Note that, we do not show the $\mathcal{S}_t$ in Figure2.1 and in Figure2.3 for the encoder and the decoder. $\mathcal{H}_t$ and $\mathcal{D}_t$ represent both hidden states of the encoder and the decoder. The f in equation 2.4 represents the stacked ConvLSTM units of the encoder. Note that the encoder recursively performs the mapping using the output states of the previous time step. We show the unrolled view of the encoder in Figure 2.3 by showing operations on each time step. In each layer, ConvLSTM units encode the input

spatiotemporal series by referring to the hidden state on the previous time step. As shown in Figure 2.1, after the input layer, the subsequent layers take the previous layer's outputs as input to their ConvLSTM unit. Each layer has unique hidden states $\mathcal{H}_t^k, \mathcal{S}_t^k$ where k represents the layer index. After every sample of the input sequence passed the encoder layers i.e. $t > T_{\text{in}}$, the hidden states in every layer, $\mathcal{H}_t^k, \mathcal{S}_t^k$ pass to the context matcher mechanism, and the encoding operation finishes.

We adapt the attention mechanism in [19] and [22] to the spatiotemporal domain. This mechanism aims to extract relevant spatiotemporal series at each time step by referring to the previous encoder's hidden state. As shown in Figure 2.1, for each time step t, the attention mechanism takes the input features of the spatiotemporal time series $\mathcal{X}_t = \{\mathbf{X}_t^1, \cdots, \mathbf{X}_t^l, \cdots, \mathbf{X}_t^L\}$, and the encoder's first layer of the previous hidden state $\mathcal{H}_{t-1}^1$ to calculate the attention weights for each feature l as follows:

$$\mathbf{E}_t^l = \mathcal{V} * \tanh(\mathcal{W}_E * \mathcal{H}_{t-1}^1 + \mathcal{U} * \mathbf{X}_t^l + \mathbf{b}_E), \tag{2.5}$$

where $\mathcal{W}_E \in \mathbb{R}^{I \times J \times L_{\text{hid}} \times L_{\text{att}}}$, $\mathcal{U} \in \mathbb{R}^{I \times J \times L_{\text{in}} \times L_{\text{att}}}$, $\mathcal{V} \in \mathbb{R}^{I \times J \times L_{\text{att}} \times 1}$, and $\mathbf{b}_E \in \mathbb{R}^{L_{\text{att}}}$ are the parameters to learn, L_{att} specify the attention dimension size, and I, J are the height and width of the filters. Note that, the output dimension of the equation 2.5 is $M \times N \times 1$ and $\mathbf{E}_t^l = \{e_{t,1,1}^l, \cdots, e_{t,i,j}^l, \cdots e_{t,M,N}^l\}$ is called the energy matrix for the feature l at time step t. We perform softmax operation on energy matrices of each feature to obtain the attention matrix $\mathbf{A}_t^l = \{a_{t,1,1}^l, \cdots, a_{t,i,j}^l, \cdots a_{t,M,N}^l\}$. We show the softmax operation at each entry of the attention matrix as follows:

$$a_{t,i,j}^l = \frac{\exp(e_{t,i,j}^l)}{\sum_l^L \exp(e_{t,i,j}^l)}.$$

Then, we obtain the weighted input spatiotemporal series by taking Hadamard product of each feature matrix with corresponding attention matrix:

$$\tilde{\mathbf{X}}_t = \{\mathbf{A}_t^1 \odot \mathbf{X}_t^1, \cdots, \mathbf{A}_t^L \odot \mathbf{X}_t^L\},$$

where $\mathbf{A}_t^l, \tilde{\mathbf{X}}_t^l \in \mathbb{R}^{M \times N}$. Each cell in the $\mathbf{A}_t^l$ represents the attention amount given to each of that particular cell's features. With the attention mechanism, the encoder can selectively focus on the different spatiotemporal series at each time step by referring to the previous encoder's hidden state. Moreover, we used the convolutional operations in the mechanism because the fully-connected methods require taking 2D inputs. If we flattened the input tensor, we would have lost the spatial covariances, which would be much more costly than the convolutions.

2.3.3 Decoder and Context Matcher Mechanism

The purpose of the decoder is to decode the encoded information inside the context vector. As shown in Figure 2.1, the decoder consists of stacked ConvLSTM units same as the encoder. To prevent performance loss on long length inputs, we introduce a context matcher mechanism. This mechanism sums the hidden states, $\mathcal{H}_1^k + \cdots + \mathcal{H}_T^k$ of the encoder's layers. This way, we can increase the length of gradient flow up to the first time step. As observed in [14, 15], the reversed matching of the hidden states of the encoder's layers increased the performance. As shown in Figure 2.1, we used that symmetry in our architecture, and we set the number of layers in the encoder and the decoder the same. The encoder's hidden state dimensions decrease as the number of layers increases, naturally vice-versa for the decoder. We can describe context matcher mechanism as

$$\mathcal{D}_t^{K-k+1} = \sum_{i=t-T_{\text{in}}}^{t} \mathcal{H}_i^k$$

where $\mathcal{D}_t^{K-k+1} \in \mathbb{R}^{M \times N \times L_{\text{hid}}}$ is the decoder hidden state, K is the total number of layers, and $k \in \{1, \cdots K\}$. As shown in Figure 2.1, the context matcher sums the hidden states in the time dimension and reverses the order of the states. Next, these states, $\mathcal{D}_{t-1}^1$ pass to the decoder's corresponding layers, and the input, $\mathbf{Y}_t$, passes the first layer of the decoder, as shown in Figure2.1. The first layer produces the next hidden state and the input, $\mathcal{D}_{t+1}^1$, to the next layer. The decoder's final layer produces the inputs to the output convolutions, $\mathcal{D}_{t+1}^K$, as shown in Figure 2.1. The output convolutions produce the prediction, $\hat{\mathbf{Y}}_{t+1}$, for

the time step $t+1$, and allow us to use higher dimensions in the first layer of the encoder. With this layer, we can produce predictions with the desired number of dimensions.

We show the recursive prediction of the decoder in Figure 2.3. The decoder uses the predictions produced in the previous time step. Thus, we can make any sequence length of predictions with this set-up. We observed that this architecture prevents overfit and increases generalization.

2.3.4 Evaluation Metrics

In our evaluation, we use the metrics of the benchmark models in [1]. Following their application, we also weight our evaluation metrics with a latitude weighting factor. Since the distance between longitudes decreases as we move towards the poles, the area of each cell in the grid is not equal. The cells that are close to the poles have less area than the cells that are close to Ecuador. Thus, we weight our metrics according to the cell sizes. For the j^{th} latitude, the weighting factor is:

$$l(j) = \frac{cos(lat(j))}{\frac{1}{N}\sum_{j}^{N} cos(lat(j))},$$

where N represents the number of latitudes and $lat(j)$ represents the j^{th} latitude. Our primary metric is the root mean square error (RMSE) since it reflects our loss function, equation 2.1, and minimizes the effect of outliers. We define the latitude weighted RMSE as follows:

$$\text{RMSE} = \frac{1}{T_{\text{out}}} \sum_{t}^{T_{\text{out}}} \sqrt{\frac{1}{MN} \sum_{i}^{M} \sum_{j}^{N} l(j)(\hat{y}_{i,j,t} - y_{i,j,t})^2}, \tag{2.6}$$

where T_{out} represents the output sequence length. We use latitude weighted mean absolute error (MAE) as our second evaluation metric, where we take the absolute value of the error instead of taking the square in equation 2.1, and perform the weighting in the same way shown in equation 2.6. We choose the latitude weighted

anomaly correlation coefficient (ACC) as our final metric, which is defined as:

$$\text{ACC} = \frac{\sum_{i,j,t} l(j) y'_{i,j,t} \hat{y}'_{i,j,t}}{\sqrt{\sum_{i,j,t} l(j)(y'_{i,j,t})^2 \sum_{i,j,t} l(j)(\hat{y}'_{i,j,t})^2}}, \tag{2.7}$$

where the prime $'$ denotes the difference to the climatology, $y'_{i,j,t} = y_{i,j,t} - \bar{y}_{i,j}$, where $\bar{y}_{i,j}$ represents the climatology and is defined as:

$$\bar{y}_{i,j} = \frac{1}{T_{\text{out}}} \sum_{t} y_{i,j,t}.$$

The ACC metric takes values between +1 and -1, where positive values show that the forecast anomalies have represented the observed anomalies successfully.

2.3.5 The Prediction Methods

As shown in Section 2, we aim to forecast multiple time steps ahead of the endogenous series. For example, if the temporal resolution is hourly and we want to forecast for the three-day interval, we need to predict for 72 steps. We can obtain these predictions in three methods. The first method is *sequential*, where we set the output length, $T_{\text{out}} = 72$ in our model, then the decoder produces the output for each time step $t > T_{\text{in}}$. The second method is called *iterative*, where we set the output length $T_{\text{out}} = 6$, then use the predictions as input for the next 6 hours, and repeat this process for 12 iterations. However, if the model uses four exogenous series as input, it needs to predict the four series at each iteration because we assume that we also do not know the future values of the exogenous series. The third method is *direct*, where we directly perform forecasts for the n^{th} time step. For example, if we want to predict only the 24^{th} step of the endogenous series, we set the temporal resolution to 24 hours and predict for one step, i.e., $T_{\text{out}} = 1$. Note that we can also predict the 72^{nd} step sequentially by setting the output length $T_{\text{out}} = 3$. We use all the prediction methods of the weather model and compare their performance on two different datasets. In the next section, we show our datasets and the results.

2.4 Experiments

We perform two sets of experiments using two different ERA5 reanalysis datasets. In the first set-up, we asses our model on a high spatial resolution dataset covering Turkey by comparing it with the state-of-the-art baseline models. In the second set-up, we compare our model performance with the baseline and a physical model on a benchmark dataset called WeatherBench [1].

2.4.1 Data Description

2.4.1.1 The Weather Dataset

Our first dataset is *ERA5 hourly data on pressure levels* [53], where the temporal resolution of the data is 3 hours, the spatial resolution is 30 kilometers, and the data belongs to the 100 hPa atm pressure level. The grid dimensions of the data are (61×121). Our data belongs to dates between 2000 and 2001 and the spatial range is 30° - 45° in latitude and 20° - 50° longitude covering the Mediterranean and Black Sea Coasts of Turkey. We show the weather features of this dataset in Table 2.1. Among these features, we select the temperature as our endogenous and the others as exogenous series. Thus, our task is to find the next temperature values using the previous values of the weather features. We split the dataset into train, validation, and test sets, with 0.8, 0.1, and 0.1 ratios

2.4.1.2 The Benchmark Dataset

Our second dataset is WeatherBench, where the dataset details are in [1]. The grid size of the data is (32×64), and the temporal resolution is hourly. In the paper [1], the authors used 850 hPa temperature as their endogenous series and obtained direct and iterative forecasts for 3-5 days of lead time. The authors selected their test data from the years 2017 and 2018. Thus, our test data has the same interval. For the training and validation, we use the data between 2015

Long Name	Short Name	Description	Unit
geopotential	z	The gravitational potential energy of a unit mass	m^2s^{-2}
potential vorticity	pv	The measure of the capacity for air to rotate in the atmosphere	$Km^2kg^{-1}s^{-1}$
relative humidity	r	The water vapour pressure as a percentage of the value at which the air becomes saturated	%
specific humidity	t	The mass of water vapour per kilogram of moist air	$kg\,kg^{-1}$
temperature	t	Temperature	K
u-wind	u	The eastward component of the wind	ms^{-1}
v-wind	v	The northward component of the wind	ms^{-1}
vertical velocity	w	The speed of air motion in the upward or downward direction	$Pa\,s^{-1}$

Table 2.1: The description and corresponding units of the features in ERA5 dataset.

and the end of 2016. Following their approach, we use a total of 19 features. [1]

2.4.2 The Baseline Models

We select five deep learning models to compare weather model performance.

- *ConvLSTM*:We follow the forecasting network proposed in [12], which has the encoder-decoder structure. This model is our control experiment. We aim to compare the performance increase we obtained in the weather model by implementing attention, context matcher, and output convolutions to this architecture. The encoder consists of three layers of ConvLSTM units. $1, 16, 32$ and $32, 16, 1$ are the number of hidden dimensions in the encoder

[1]You can find all of our codes to perform analysis and experiments at github.com/sftekin/spatio-temporal-weather-forecasting

and the decoder layers, respectively. We set the hidden states to zero at the beginning of each epoch, and the decoder takes zero tensors as input.

- *U-net*: After the success in weather forecasting, we implement the model architecture introduced in [54]. The model takes a sequence of spatiotemporal temperature series and predicts an output sequence. The architecture consists of 4 downsample and 4 upsample layers.
- *TrajGRU*: The TrajGRUs show slightly higher performances than ConvLSTMs in [14]. Thus, we want to compare this model architecture with the ConvLSTMs and Weather Model. The encoder-decoder structure is the same as baseline ConvLSTM, and we set the number of links to 1.
- *LSTM*: We also want to show the performance increase by ConvLSTMs instead fully connected LSTMs. We flatten the grid each time step to create (T, MN) input matrices to feed the LSTMs. We set the hidden dimension to 256. After the LSTM produces (T, 256) output matrices, we pass them to a Fully Connected Neural Network with (256, MN) matrix size. After reshaping the outputs to the grid size, we produce our spatiotemporal series.
- *Simple Moving Average (SMA)*: We want to compare our model with a simple but effective model called Simple Moving Average (SMA). In this model, we predict the value of the series at time t by taking the weighted average of spatiotemporal input series values with T window length [55]. The averaging can be performed with equal weights, or we can give a higher value to the recent values of the series, such as $t-1, ...t-7$. Much better, we implement an FC Neural Network to find the weights. At each epoch, we initialize the weights with a uniform random variable where the range is $[0, 1]$.

2.4.3 Hyper Parameter Selection

There are numerous parameters to set in the baseline models. To select the best performing parameters we implement grid search. We split the datasets into train, validation, and test sets. We train our model on the train set and validate it on the validation set to find the best collection of parameters. After we find the best

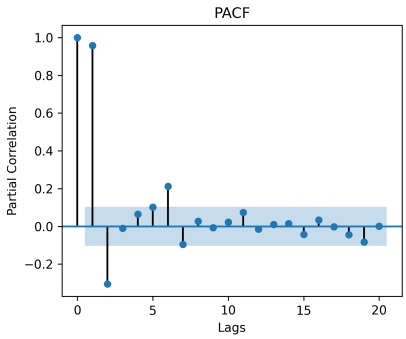


Figure 2.4: Partial autocorrelation function of temperature value for a randomly selected cell. Blue region shows the 95% confidence level and each lag represents three hours difference.

parameters, we train the model on the train and validation sets combined. Lastly, we test our model on the test data to determine its performance for comparison.

We use the Adam optimizer with 0 weight decay in each experiment and select the best performing learning rate from the $\{0.01, 0.001, 0.0005, 0.00001\}$. We set the layer size equal for each baseline model and choose the best performing hidden dimension size from the $\{16, 32, 64\}$.

We perform autocorrelation (ACF) and partial auto-correlation analysis (PACF) on temperature values of a randomly selected cells, as shown in Figure 2.4 and in Figure 2.4. We observe high correlation until lag 20 with the ACF. With the PACF we find the direct relationship between observation and its lagged values by removing correlations due to the terms at shorter lags [56]. We observe high correlation in first two lags and meaningful correlations until lag 10. To this end, we have select 10 time steps of input data with 3 hours of frequency and try to predict 10 time step ahead.

In addition, input series have different units and scales, which requires normalization. However, the dataset is massive, and we need high RAM capacity to normalize such a dataset. Thus, we apply min-max normalization to the batches in an adaptive manner.

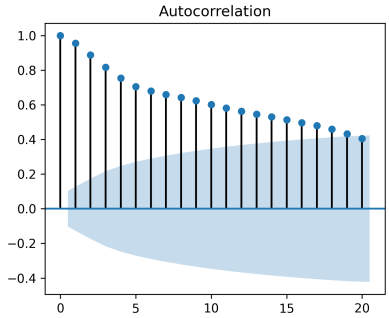


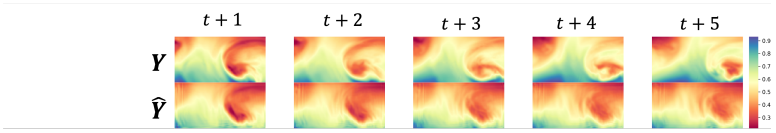
Figure 2.5: Autocorrelation function of temperature value for a randomly selected cell. Blue region shows the 95% confidence level and each lag represents three hours difference.

2.4.4 Performance Analysis and Results

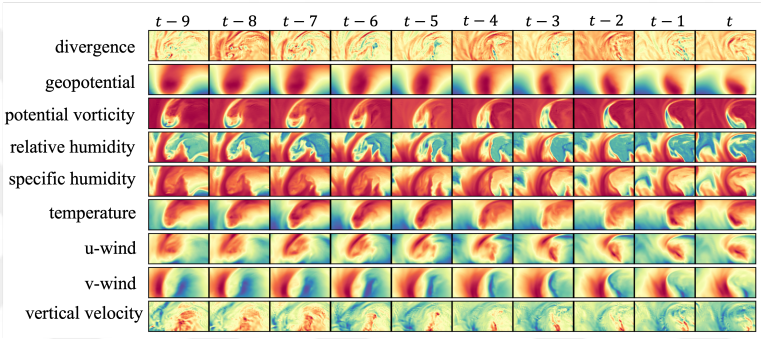
In the first set-up of experiments, we evaluate the performance of our model on the high-resolution dataset by comparing it with the baseline deep learning models. All the models produce the predictions using the sequential method, where the output sequence length is 5. We show numerical and visual results to assess the performances better.

We show the numerical results of the experiment in Table 2.2. We perform a two-tailed t-test with a p-value of 0.05 comparing ConvLSTM results with the Weather Model and found that the Weather Model significantly performs better in all metrics. Furthermore, the weather model outperforms all the baseline deep learning models, including TrajGRU and U-Net. When we compare the baseline methods, we observe that TrajGRU show close performance to ConvLSTM. Thus, this leads us to a further study where we replace the ConvLSTM units with the TrajGRU. Furthermore, we observe that the models including convolution operations, e.g., U-Net, ConvLSTM, and TrajGRU outperform fully connected models, e.g., LSTM and SMA. Thus, we say that the convolution operations are essential while modeling the weather movements.

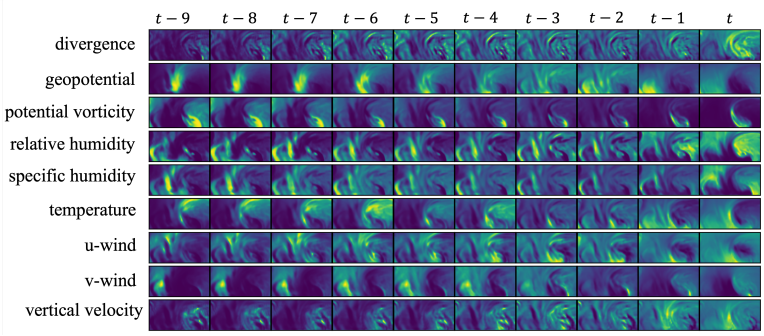
To visually analyze the spatial and temporal performance, we show the predicted sequence and ground truth in Figure 2.6a. Observing the changes between



(a) Predicted sequence and ground truth



(b) Input sequences to model



(c) Attention weights to input sequences at each time step

Figure 2.6: We show the Weather Model result with the output sequence (a), input sequence (b), and the attention weights (c) at each time step. In each step, the model pays attention to the cells on each feature that it finds significant to pass the next hidden state by referring to the input, which is the temperature value in the previous step. Observe that the model pays attention to the moving parts in each feature, e.g., potential vorticity and geopotential. When we compare the predicted and ground truth series in (a), we observe that the model is successful in the first step. After the first step, the model can forecast only the stationary parts, such as the regions that stay cold or hot.

Model Name	RMSE	MAE	MAPE
Weather Model	**1.22115**	**0.88872**	**0.00394**
U-net	1.33032	0.97636	0.00434
Convlstm	1.59211	1.18715	0.00527
TrajGRU	1.60205	1.21569	0.00540
SMA	1.90273	1.49958	0.00666
LSTM	1.99607	1.62078	0.0072

Table 2.2: We show model's test scores on the high resolution dataset. Bold scores are the best.

Model Name	RMSE	MAE	ACC
Convlstm (sequential)	4.43542	3.46913	0.63598
TrajGRU (sequential)	3.5988	2.78978	0.75716
U-net (sequential)	**3.05466**	**2.07594**	**0.82028**
WM (sequential)	3.97452	3.25712	0.6909
Convlstm (iterative)	3.16472	2.22504	0.80173
TrajGRU (iterative)	3.89876	3.1951	0.73826
U-net (iterative)	4.13038	3.26531	0.68035
CNN (iterative)	4.48	3.49	0.72
WM (iterative)	**2.91701**	**2.08413**	**0.83786**
WM (direct)	3.07334	2.2027	0.82357
CNN (direct)	**2.87**	**2.02**	**0.85**
IFS T42	3.09	1.99	0.86
IFS T63	1.85	1.30	0.94
Operational IFS	**1.36**	**0.93**	**0.97**

Table 2.3: We show model's scores on the WeatherBench dataset. Bold scores are the best. The IFS models are the physical models and with CNN models their scores are taken from [1].

the prediction steps, the model successfully generates the temperature fluctuations up to three steps. However, the performance of the model deteriorates as the prediction interval length increases. With the help of attention weights shown in Figure 2.6c and the input series shown in Figure 2.6b, we show that the model attends to the movements in the input features and produces the next hidden state by referring to the temperature at the previous step.

In the second set-up of experiments, we evaluate the performance of our model on the WeatherBench dataset by comparing it with the physical and baseline models with the benchmark metrics. Additionally, we compare our results with

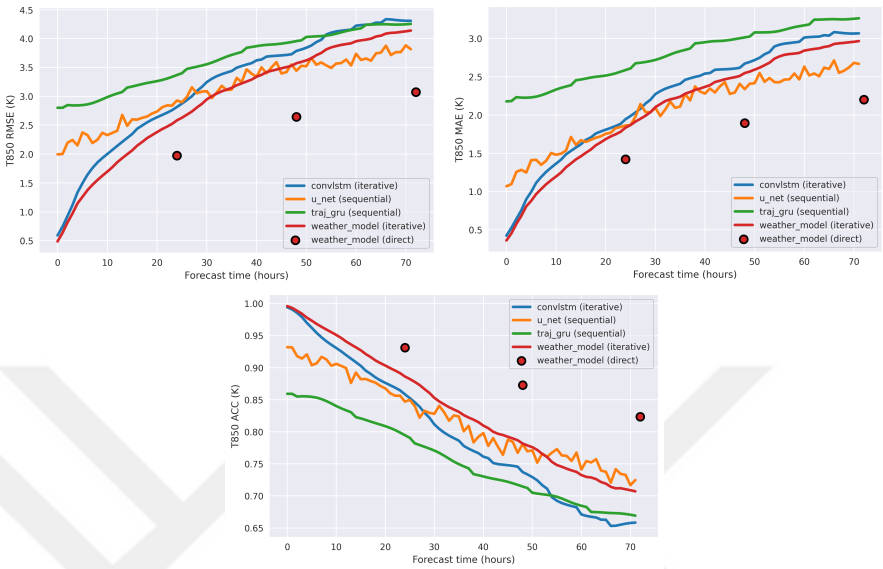


Figure 2.7: Evaluation metrics comparison for baseline models for 72 hours of forecasts. We show the best performing method for each model. We also show the direct Weather Model for the 3 days of forecast.

the CNN model shown in WeatherBench. We obtain two different results for each baseline model, one for the sequential and the other for the iterative method. In both methods, our goal is to forecast 72 hours. Thus, we set the output sequence length to 72 and 6 in sequential and iterative methods, respectively. We show the results of this experiment set-up in Table 2.3. Despite the U-net model obtaining the best scores in the sequential method, the Weather Model trained iteratively obtains the best scores among all deep learning models. We show the hourly performance of the models in Figure 2.7. As shown in Fig 2.7, we say that the growth is logarithmic in RMSE, MAE, and linear in ACC metrics. On the other hand, the error growth of the CNN model in [1] is exponential. Thus, we claim that WM solves the exponential error growth problem.

We also perform a direct forecast with the weather model and compare its performance with CNN direct model. We set the temporal resolution to 24 hours and output sequence length to 3. The CNN model performs better than the Weather Model in the direct method. To further increase the performance of the direct WM, we set the temporal resolution to 72 hours and predicted one step. However, this performed worse than the previous method. After we observe the

input sequence in both direct set-ups, we conclude that the spatial movements are lost with the significant temporal gaps. Thus, we say that our model heavily depends on the spatial movements in the data.

Even though WM performs better than IFS T42 in the RMSE metric, the physical models significantly outperform the deep learning models. The reason is that the physical models can simulate the atmosphere better for the long forecasts, while the WM shows high performance only in shorter periods, and its performance deteriorates in the long forecasts.

Chapter 3

Crime Prediction with Graph Neural Networks and Multivariate Normal Distributions

3.1 Introduction

Crimes are unlawful acts endangering public safety, leading to colossal life and economic losses if governments do not take the necessary precautions. Understanding the latent patterns of crimes in a geographical region of a city is highly important for preventing upcoming crimes events. With accurate and reliable crime prediction, the police force can be at the right place at the right time.

However, the problem of crime prediction is a highly complex and challenging task since it contains spatial, temporal, and categorical complexities with inherent interrelations [57–61]. In general, prior works on crime prediction can be classified in two main categories. The prior works in the first category [62–65] divide city area into a grid, where each cell contains the crime information for each crime

category that happened in the past. The works in the second category [18,66,67], divide the city area into unordered regions and create a graph structure that captures the correlations among the regions with corresponding edge weights and node features. However, the regions in both of these approaches correspond to extremely wide sections of the cities. To the best of our knowledge, the minimum size of a region in grid-based methods has $4km^2$ area. For the graph-based approaches, the regions correspond to the districts of a city and the range of sizes changes from $2km^2$ to $35km^2$. Thus, assigning a probability value to such a broad area is not expedient and decreases the feasibility in real-life situations. Note that one can increase the number of regions to enhance the detailing. However, this leads to severe problems in both of these two approaches. For the grid-based approach, as the grid size increases, the complexity and sparsity exponentially grow. On the other hand, it is difficult to define a rule to add new regions to the graph in graph-based methods.

We introduce a probabilistic graph-based model called *High-Resolution Crime Forecaster* (HRCF) that performs grid-based crime prediction to remedy these problems. We learn a multivariate probability distribution instead of assigning one probability value to each node in a graph and model the underlying dynamics of each region with the objective of minimizing the cross entropy. We build our graph on spatiotemporal data using a divide-and-conquer algorithm to up weight the minority class. The architecture contains graph convolutional neural networks as memory units, which capture the spatial and temporal correlations. In each time step, the model predicts Multivariate Gaussian distributions for all the regions conditioned on the previous observations, allowing us to generate predictions for any resolutions. We demonstrate significant performance gains in real-life data through an extensive set of experiments compared to the state-of-the-art methods and show that our model is not only generative but also more precise.

3.1.1 Prior Art and Comparisons

Deep learning structures are extensively used in the signal processing literature to model crowd movements and provide high performance. For example, in [62], authors implemented deep learning based prediction model for spatiotemporal data (DeepST), where they model the crowd flow with the grid-based forecasting system. The authors of [64] adapted their work to predict crime distribution over the Los Angeles dataset and achieved high scores but with a very high computational cost. The authors of [65] designed multi-model units capturing spatial, temporal, and semantic information and producing predictions for each crime type for a 11km$\times$11km grid size but the model performance decreases as the spatial resolution increases. Due to the high sparsity and complexity of grid-based approaches, graphical models are recently used [68,69] to remedy these problems. In crime prediction, [18] formed the graph structure by using independent Hawkes processes in each node and obtained edge weights and performed predictions for 50 regions in the Chicago crime dataset. Another study [66] implemented Gated Recurrent Network with Diffusion Convolution modules following a Multi-Layer Perceptron (MLP). Recently, [67] introduced a homophily-aware constraint on the loss function so that neighboring region nodes share similar crime patterns. Nevertheless, these works are not generative and work on the districts, which causes the complexity and sparsity problems.

3.2 Problem Definition

Let $A = [lat_1, lat_2]$ and $B = [lon_1, lon_2]$ define the bounds of the city region that we divide into $I \times J$ disjointed cells, which creates the set of cells $M = \{m_{1,1}, \ldots, m_{i,j}, \ldots, m_{I,J}\}$. The cells contain the longitude, a, and latitude, b, information, which is given by $m_{i,j} = (a_i, b_i)$, and $a_i \in A$ and $b_j \in B$. For a given time window T, we also split T into non-overlapping and consequent time slots $T = \{t_1, \ldots, t_K\}$. In addition, $L = \{c_1, \ldots, c_L\}$ represents the categories of crimes. Following these definitions of sets, we define appearance of a crime, x,

with category c_l that belongs to $m_{i,j}$ at time step t_k as $x^l_{i,j,k} \in \{0, 1\}$. Next, we define an event matrix that shows all the events with category c_l happened at t_k in all cells as $\mathbf{X}^l_k = \{x^l_{1,1,k}, \ldots, x^l_{I,1,k}, x^l_{I,2,k}, \ldots, x^l_{I,J,k}\} \in \mathbb{R}^{I \times J}$. We also define an event matrix representing any crime event regardless of the crime type that happened in all the cells at time t_k as $\mathbf{X}_k \in \mathbb{R}^{I \times J}$ by dropping the l superscript.

Here, our primary goal is to predict the next event matrix representing any crime event given the previous event matrices with different categories. Thus, our goal is to learn the following mapping, $\{\mathbf{X}^1_1, \ldots, \mathbf{X}^L_K\} \xrightarrow{\text{f(.)}} \{\mathbf{X}_{K+1}\}$. Even though we produce grid-based predictions, we use a graph-based approach to model the inter-relations in the data, as shown in Section 3. Furthermore, our model is generic so that we can generate predictions even for high resolutions without retraining our model, i.e., $\mathbf{X}_\mathrm{K}$ can be $\mathbb{R}^{I' \times J'}$ where $I' \geq I$ and $J' \geq J$ so that we mitigate sparsity issues.

3.3 Methodology

3.3.1 Subdivision Algorithm and Graph Representation

To observe the distribution of events in a region, we sum all the event matrices in a period with length T to create the $\mathbf{Q}$ matrix, $\mathbf{Q} = \sum_{k=1}^{T} \mathbf{X}_k$. An example $\mathbf{Q}$ matrix is shown in the left side of Figure 3.1. Observe that the number of cells that contain no events is the majority, i.e., 60.42% of the cells have zero event count. Our goal is to remedy this sparsity problem by introducing new regions that have equal or close event counts. As shown in Algorithm 1, our subdivision algorithm partitions the set M into N disjoint sets with the divide-and-conquer approach to create the region set $R = \{r_1, \ldots, r_N\}$, where each region contains arbitrary number of cells and $M = \cup_i r_i$ and $\cap_i r_i = \emptyset$. By this algorithm, all the resulting regions contain a total event count less than a threshold value, τ. Our base condition is to have a region larger than 2km $\times$ 2km or a total event count less than the threshold value. With this algorithm we drop the percentage of the

regions that have zero event count to 5.97%.

Algorithm 1 Algorithm for Subdivision

```
procedure DivideRegions(Q, τ, r, c)
    Q* ⟵ Q[r[0] : r[1], c[0] : c[1]]                    ▷ Select the sub-region
    if sum(Q*) < τ OR Q*.shape < (2, 2) then
        return [[r, c]]                                   ▷ base case
    else
        I ← SplitRegions(Q*, r, c)                        ▷ divide 4
        R ← [ ]
        for r, c ∈ I do
            R_new ← DivideRegions(Q*, τ, r, c)
            R.append(R_new)
        end for
        return R
    end if
end procedure
```

After the subdivision process, we build our graph with the nodes representing the centers of each region and the edges representing the distance between the neighboring nodes. We form the region graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, where $\mathcal{V}$ represents the set of nodes with $\mid \mathcal{V} \mid = N$ where N is the number of regions and $\mathcal{E}$ is the set of undirected edges between region nodes, $\mathbf{A}$ is the weight matrix. Let $v_i \in \mathcal{V}$ denotes a node and $\epsilon_{ij} = (v_i, v_j) \in \mathcal{E}$ denotes an edge. The weight matrix $\mathbf{A}$ has dimension $N \times N$ with $A_{i,j} = (v_i - v_j)^2$ if $\epsilon_{i,j} \in \mathcal{E}$ and $A_{i,j} = 0$ if $\epsilon_{i,j} \notin \mathcal{E}$.

We define the input event in the graph with category l belonging r_i at time step t_k as $z^l_{i,k} \in \{0, 1\}$. Hence, $\mathbf{z}^l_k = \{z^l_{1,k}, \ldots, z^l_{N,k}\} \in \mathbb{R}^N$ represents events with type l happened in all the regions in this graph at time step t_k. Thus, the subdivision algorithm provides the following mapping $\{\mathbf{X}^l_1, \ldots, \mathbf{X}^L_K\} \xrightarrow{\text{s(.)}} \{\mathbf{z}^l_1, \ldots, \mathbf{z}^L_{\text{K}}\}$. Furthermore, we define the graph function, $g(.)$, which takes the output vectors of the subdivision algorithm and produces the state vectors, $\mathbf{h}_{i,k} \in \mathbb{R}^{d_h}$, for each graph node, where d_h is the hidden dimension. We formulate the graph function as: $\{\mathbf{z}^1_1, \ldots, \mathbf{z}^L_K, \mathcal{G}\} \xrightarrow{\text{g(.)}} \{\mathbf{h}_{1,\text{K+1}}, \ldots, \mathbf{h}_{\text{N,K+1}}\}$ where we generate the state vectors for the next time step for all the nodes. The details of this function is given in Section 3.3.

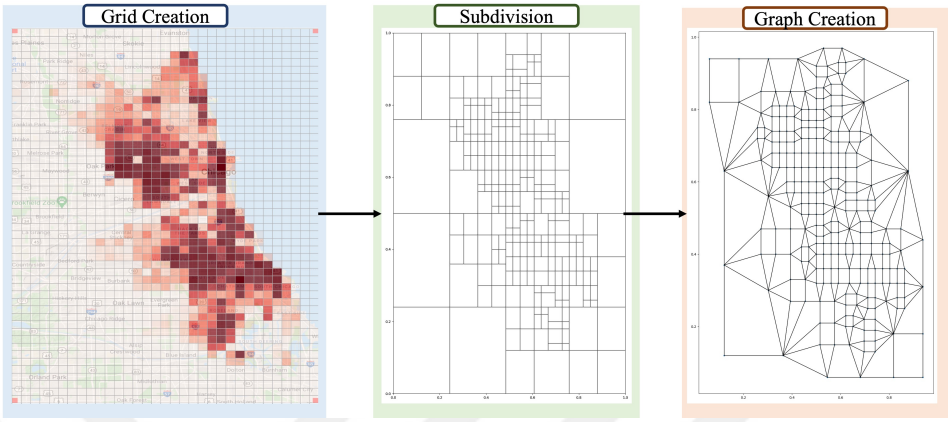


Figure 3.1: We show the phases of creating our graph. First, we create a grid by dividing the region into $I \times J$ cells, where each cell contains the total event count. Second, with the sampling algorithm that favors the crowded regions, we obtain graph regions. Third, we create the graph with nodes representing the centers of each region and edges representing the distances between nodes.

3.3.2 The Likelihood Model

Let $\mathcal{Y}$ model the random variable with the realization of $\mathbf{y}_{i,k} = \{y_1, y_2\} \in \mathbb{R}^2$, where y_1 and y_2 represent the longitude and latitude of any crime happening in r_i at time step t_k. Let $p(\mathbf{y}_{i,k})$ be the probability density function governing this random variable. Our goal is to learn the conditional probability function $p(\mathbf{y}_{i,k} \mid \mathbf{z}_1^1, \ldots, \mathbf{z}_{k-1}^L)$. We model this probability density function with a likelihood model, $l(y_{i,k}; \theta(\mathbf{h}_{i,k}))$, and parametrize it by the $\theta(.)$ function. This underlying model is chosen as a Multivariate Gaussian Distribution (MGD) since MGD can readily model even complex functions while avoiding overfitting [70]. Here, $\theta(.)$ corresponds to the mean vector and covariance matrix, $\theta = (\mu, \mathbf{\Sigma})$, where $\mu = \begin{bmatrix} \mu_1 & \mu_2 \end{bmatrix}$, and $\mathbf{\Sigma} = \begin{bmatrix} v_1^2 & 0 \\ 0 & v_2^2 \end{bmatrix}$. Note that cross terms are selected 0 to alleviate overfitting.

In training, we could directly maximize the log-likelihood (MLE) where we put the locations of each crime event into the likelihood model, $\mathcal{L} = \sum_{i=1}^{N} \log(l(\mathbf{y}_{i,k}; \theta(\mathbf{h}_{i,k})))$, and sum all the log-likelihoods in each region. However, the MLE loss does not directly suffer from negative predictions, which, as

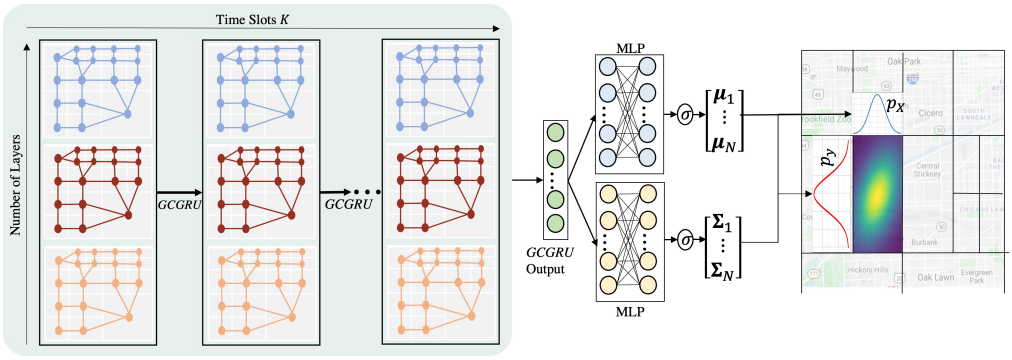


Figure 3.2: We show HRCF architecture. In each time slot t_k, we extract features with the Graph-ConvGRU operations. We pass these features to two different MLP heads, μ and Σ. Each head produces the μ_i and $\mathbf{\Sigma}_i$ parameters of distribution in each node.

we observe in our experiments, causes the model to get stuck on the local minima. Therefore, we minimize the Kullback-Leibler divergence [71], between the actual distribution and the likelihood model, which is equivalent to minimizing the cross-entropy $\mathcal{L} = -\sum_{i=1}^{N} \sum_{\mathbf{y} \in \mathcal{Y}_{i,k}} p(\mathbf{y}) l^*(\mathbf{y}; \theta(\mathbf{h}_{i,k}))$, where $l^* = \log(l(.))$, but the term $p(\mathbf{y})$ is still unknown and note that $\mathbf{y}$ contains continuous values. Recall that we quantize each region with the grid-based approach where each cell takes 1 if a crime happened, 0 otherwise. Hence, we can rewrite our loss as binary-cross entropy:

$$\mathcal{L} = -\frac{1}{IJ} \sum_{i=1}^{IJ} \sum_{\mathbf{y} \in \mathcal{Y}_{i,k}} q_i l^* + (1 - q_i) \log(1 - l), \tag{3.1}$$

where $q(.)$ maps the longitude and latitude to the crime event value:

$$q(y_1, y_2) = \begin{cases} 1, & \text{if an event is observed at } (y_1, y_2), \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

The parameters of the model is optimized by minimizing (3.1) with the Adam optimizer using batches of training data.

3.3.3 High Resolution Crime Forecaster (HRCF)

A crime event correlates with abnormal events in nearby locations and an after-effect of a crime can trigger another crime in nearby areas. We use graph convolution to learn such relations due to its success in modelling spatialtemporal series [69]. The graph convolution is similar to the traditional 2D convolution. An image pixel is similar to a node in a graph, where the size of a filter in 2D convolution determines the number of neighbors. The 2D convolution takes the weighted average of pixel values of the central node along with its neighbors. Similarly, graph convolution takes the weighted average of the central node along with its unordered and variable in size neighbors. Hence, each node uses the information coming from its neighbors. Therefore, we use convolution operations to propagate crime information at each node to the other nearby nodes. We assume that structurally similar nodes will show close behavior. Since graph convolutional networks allow parameter sharing across the nodes in the graph, we can capture the spatial dependencies for different patterns observed in other locations. To capture the temporal relations, we use graph convolutions with deep memory units [72]. Graph Convolutional Gated Recurrent Unit (Graph-ConvGRU) generalizes the Convolutional Gated Recurrent Unit (ConvGRU) model [73] to graphs by replacing the Euclidian 2D convolution $*$ by the graph convolution $*_{\mathcal{G}}$:

$$
\begin{aligned}
\mathbf{i} &= \sigma(\mathbf{W}_{zi} *_{\mathcal{G}} \mathbf{z}_k + \mathbf{W}_{hi} *_{\mathcal{G}} \mathbf{h}_{k-1}), \\
\mathbf{r} &= \sigma(\mathbf{W}_{zr} *_{\mathcal{G}} \mathbf{z}_k + \mathbf{W}_{hr} *_{\mathcal{G}} \mathbf{h}_{k-1}), \\
\tilde{\mathbf{h}} &= \tanh(\mathbf{W}_{zh} *_{\mathcal{G}} + \mathbf{W}_{hh} *_{\mathcal{G}} (\mathbf{r} \odot \mathbf{h}_{k-1})), \\
\mathbf{h}_k &= \mathbf{i} \odot \mathbf{h}_{k-1} + (1 - \mathbf{i}) \odot \tilde{\mathbf{h}},
\end{aligned}
$$

where $\mathbf{W}_{h.} \in \mathbb{R}^{K \times d_h \times d_h}$ and $\mathbf{W}_{z.} \in \mathbb{R}^{K \times d_h \times d_x}$ are the parameters that we learn. The hidden dimension, d_h, the input dimension, d_x, and K determine the number of parameters, which is independent of the number of nodes N. The gates $\mathbf{i}$, $\mathbf{r}$, and $\tilde{\mathbf{h}}$ enable resetting and updating the stored information.

As show in Figure 3.2, we stack Graph-ConvGRU units and use as an encoder to encode the input sequence. At each time slot t_k, each unit takes the previous hidden state, $\mathbf{h}_{k-1}$ and input node feature, $\mathbf{z}_k$, to produce the next state $\mathbf{h}_k$. Note

that we apply these operations for every node on the graph. Thus, in each time step, t_k, we feed the Graph-ConvGRU unit at each layer with N different node features, $\mathbf{z}_{i,k}$, where the subscript i denotes the node index. Similarly, at each time step, t_k, Graph-ConvGRU unit outputs the hidden state, $\mathbf{h}_{i,k}$, for each node. As given in the Section 2, $\mathbf{z}_{i,k} \in \mathbb{R}^L$ are the event vectors and $\mathbf{h}_{i,k} \in \mathbb{R}^{d_h}$ are the state vectors that parametrize the likelihood model, $l(\mathbf{y}_{i,k}; \theta(\mathbf{h_{i,k}}))$. Each Graph-ConvGRU output, $\mathbf{h}_{i,k}$, passes to the MLP heads as shown in Figure 3.2. Each MLP head is responsible for generating Multivariate Gaussian parameters for each node, $\mu_1 = \sigma(\mathbf{W}_\mu \mathbf{h}_{1,k} + b_\mu), \ldots, \mu_\mathrm{N} = \sigma(\mathbf{W}_\mu \mathbf{h}_{\mathrm{N},k} + b_\mu)$ and $\mathbf{v}_1 = \sigma(\mathbf{W}_v \mathbf{h}_{1,k} + b_v), \ldots, \mathbf{v}_\mathrm{N} = \sigma(\mathbf{W}_v \mathbf{h}_{\mathrm{N},k} + b_v)$, where $\mathbf{W}_\mu \in \mathbb{R}^{d_h \times 2}$, $\mathbf{W}_v \in \mathbb{R}^{d_h \times 2}$ and $b_\cdot$ are the weights we learn, $\sigma(.)$ is the sigmoid activation, and $\mu_i = [\mu_{i,1}, \mu_{i,2}]$, $\mathbf{v}_i = [v_{i,1}, v_{i,2}]$ are the parameter vectors. Since the multivariate distribution is uncorrelated, we write the likelihood model as follows:

$$l(y_1, y_2 \mid \mu, \mathbf{\Sigma}) = \frac{\exp(\frac{-1}{2}(\frac{(y_1-\mu_1)^2}{v_1} + \frac{(y_2-\mu_2)^2}{v_2}))}{2\pi(v_1 v_2)^{\frac{1}{2}}},$$

which allows us to generate likelihoods for any location under the distribution, as we show in the right side of Figure 3.2.

3.4 Experiments

This section compares our model performance with the baseline models on a real life and a synthetic datasets. We first describe the datasets then the models, and finally, discuss the results.

3.4.1 Data Description

3.4.1.1 Synthetic Dataset

First, we form an AR(1) process to produce correlated event counts in each time step. Second, we create multiple Multivariate Gaussian distributions belonging

to 4 different categories, and each category has 10 different distributions. We randomly select the distributions centers between $[0.2, 0.8]$ and set the distribution variance values from $0.02, 0.01, 0.005, 0.003$. We sample the crime events, where the AR(1) process determines the total number of samples in each time step. Third, to create self and cross relations, we form 4 different temporal and categorical patterns and assign them to each distribution. At even time steps, distributions in the first category can generate events. Likewise, the distributions in the second category can generate at odd time steps. At time steps divisible by 4, the distributions in the third category can generate events, and the fourth category is the opposite of the third. Moreover, after the 10 time steps, we switch the behaviors of the categories, where the first category behaves like the second and vice-versa. Hence, our data has both temporal, spatial, and categorical interrelationships. An example data is shown in Figure 3.3a.

3.4.1.2 The Chicago Crime Dataset

We collect the Chicago crime data between 2015 and 2019 [74]. The working area has a longitude range of $[41.60, 42.05]$ and a latitude range of $[-87.9, -87.5]$, creating a rectangle with height 50km and width 33km. The rectangle is non-Euclidean due to the shape of the Earth, yet, we assume an Euclidean rectangle. We partition the region into 50×33 disjoint geographical regions yielding 1km $\times$ 1km size squares. As described in Section 2, we map crime events to each square region. In Figure 3.3b, we show all the crimes on these squares as a heatmap without considering crime type, where the dense red color represents high event count. We also select the time resolution as 24 hours. Thus, our target is to predict the locations of the crime events regardless of their type in the partitioned regions for the next 24 hours. Moreover, we perform an exploratory data analysis [1] to investigate spatial, temporal, and categorical covariance. We filter the data by selecting crime types that forms the %90 percent of the crime data. We perform 4 experiments where each experiment consists of 1 year of data. We split the data into 10 months of training, 1 month of validation, and 1 month of testing

[1] You can find all of our codes to perform analysis and experiments at github.com/sftekin/high-res-crime-forecasting

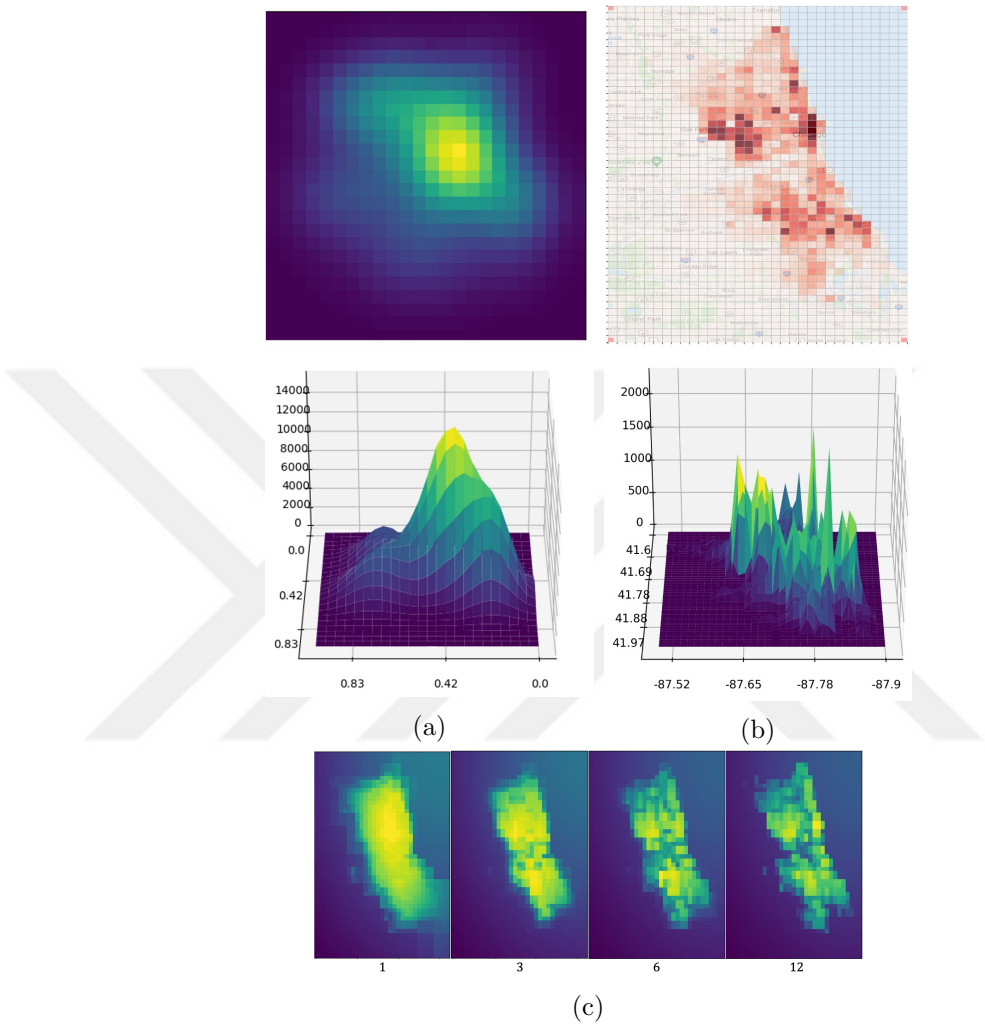


Figure 3.3: We show the distribution of events for the Chicago crime data (a) and the synthetic data (b). We show the predictions of HRCF model in different epochs at (c).

in each experiment. We report the validation and test scores in Section 4.3.

3.4.2 HRCF Configurations

For the subdivision algorithm, we select the threshold value as 1000 and minimum region size as 2km $\times$ 2km and obtain 203 regions, which is also the number of nodes in our graph. Our model uses 16 input features total, 8 features representing the crime counts for each crime type, 2 features for the locations of the nodes, and 6 features for the calendar features such as weekends and holidays. We set

Validation Scores for Chicago Dataset												
	HRCF		ConvLSTM		ARIMA		RF		SVR		GPR	
Date	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc
2015	**0.724**	**0.895**	0.705	0.865	0.708	0.866	0.700	0.860	0.701	0.872	0.695	0.868
2016	**0.728**	**0.891**	0.714	0.864	0.717	0.865	0.727	0.873	0.716	0.875	0.703	0.865
2017	**0.731**	**0.895**	0.709	0.865	0.716	0.868	0.713	0.865	0.718	0.879	0.703	0.869
2018	**0.720**	**0.894**	0.704	0.868	0.705	0.867	0.707	0.869	0.698	0.873	0.692	0.869
Test Scores for Chicago Dataset												
	HRCF		ConvLSTM		ARIMA		RF		SVR		GPR	
Date	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc
2015	**0.723**	**0.888**	0.713	0.871	0.705	0.863	0.718	0.874	0.709	0.876	0.694	0.867
2016	**0.709**	**0.881**	0.698	0.863	0.718	0.876	0.693	0.855	0.697	0.870	0.684	0.864
2017	**0.708**	**0.877**	0.699	0.865	0.686	0.853	0.697	0.862	0.697	0.872	0.686	0.866
2018	**0.720**	**0.884**	0.708	0.870	0.712	0.869	0.716	0.872	0.708	0.876	0.693	0.869
Scores for the Synthetic Dataset												
	HRCF		ConvLSTM		ARIMA		RF		SVR		GPR	
Set	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc
Val	**0.484**	**0.775**	0.470	0.738	0.344	0.611	0.387	0.677	0.341	0.784	0.300	0.660
Test	**0.491**	**0.781**	0.475	0.742	0.345	0.585	0.398	0.692	0.356	0.780	0.308	0.679

Table 3.1: We show model's validation and test scores on Chicago crime and synthetic dataset. Bold scores are the best. Date column shows the years the experiment belongs.

the Graph-ConvGRU layers as 3, where hidden dimensions of each layers are, $50, 20, 10$, respectively. The K value is 3 in each layer, and we set the bias flag "true" with the symmetric normalization. The number of layers in the MLP is 2, and the hidden layer contains 64 nodes. We set the input sequence length as 10 with a batch size of 5. For the training, we use a 0.003 learning rate, with an early stop tolerance of 5. Lastly, we use 0.001 as L2 regularization to regularize the model parameters.

3.4.3 Baseline Models

We compare our algorithm with the well known sequential learning models: Autoregressive Integrated Moving Average (ARIMA), Support Vector Regression (SVR), Random Forest (RF), Gaussian Process Regression (GPR) and ConvLSTM [73]. Note that ARIMA, SVR, RF, GPR are regression models; however, we make classification whether a crime event happens or not. Thus, after we provide the regression output, we select a threshold value for classification. We

select the threshold value that maximizes the difference between True Positive Rate (TPR) and False Positive Rate (FPR). Since ARIMA, SVR, and RF are only applicable to single time series, we create a separate model for each cell. For the ConvLSTM, SVR, GPR, and RF, we feed the previous 10 time step values of the target series as input features to be fair with the graph model.

3.4.4 Performance Analysis and Results

In Table 3.1, we show the validation and test scores of models for both real life and synthetic datasets. We use the F1 metric, which measures the overall model performance of how precise and sensitive a model is. We also present the accuracy of our models since it is essential in crime prediction. We observe that our model outperforms the baseline models up to 3.10% for both validation and test scores for both datasets. We perform a statistical significance test with a p-value of 0.05 comparing baseline results with the HRCF and observe that the HRCF performs significantly better in all metrics. HRCF is not only sensitive to crime events but also accurate. When we look at the score differences between years, we observe that HRCF is more adaptive to the changes in the dataset.

In the synthetic dataset results shown in Table 3.1, we expect to see higher performance from HRCF since we sampled the data from multiple Gaussian distributions. However, the performance of the GPR model is low, and GPR also assumes the data is coming from a Gaussian distribution. The reason is that the GPR model is not adaptive to temporal changes and can only capture the spatial covariates. From this perspective, we can say that HRCF behaves as a GPR model that can capture both spatial and temporal covariates.

In Figure 3.3c, we show how the output of HRCF evolves between epochs. The model learns generic bounds of crime locations with the close score values on the high crime rate regions in the first epochs. Then the score values on the predictions start to distinguish in the 6th epoch. After the 12th epoch, the model makes a detailed forecast. We observe that model binarizes the output predictions, which means the probability distribution generates likelihoods close

to 0 or 1. We expect this behavior since we use the BCE loss that makes negative predictions closer to 0 and positive predictions to 1. Furthermore, this causes the PDF to assign close values in responsible regions. To assess the generative property of our model on high resolution, we obtain predictions for a 100×66 resolution grid from the HRCF model trained on 50×33. We achieved 36.67 validation and 36.02 F1 scores on test sets, which are 4.77% relatively higher than the trained ConvLSTM model for the resolution of 100×66.

Chapter 4

Conclusion

Here, in this thesis, we first studied the problem of spatiotemporal weather forecasting and introduced a novel deep learning architecture. Our structure contains ConvLSTM units with the attention and the context matcher mechanisms to extend the classical encoder-decoder method. We evaluated our model on high-resolution and benchmark datasets, and the results show that our model achieves statistically significant performance gains over the state-of-the-art baseline models. We also showed that the error growth on the long forecasts is logarithmic while it is exponential in benchmark approach. However, when the input series lost its spatial and temporal correlations in higher temporal resolutions, the performance of our model decreased, and the direct CNN model showed higher performance than ours. In addition, we also compared our model with the physical models, and the physical models show much higher performance due to their accurate long forecasts. For future work, we plan to replace ConvLSTMs with the TrajGRU units and increase the training data size. We also provide the source code of our algorithm for reproducibility.

Secondly, we studied the problem of high-resolution crime forecasting and introduced a novel generative graph neural network architecture, HRCF. Our new subdivision algorithm performs balanced sampling on the data to create representative regions. We built our graph according to formed regions, parameterized

the problem using a likelihood model, and obtained probability density functions belonging to each region. We evaluated HRCF on a real-world and synthetic datasets, and results showed that our model achieves statistically significant performance gains over the state-of-the-art baseline models. For the future work, we plan to apply our model to any spatiotemporal data. We also provide the source code of our algorithm for reproducibility.

Bibliography

[1] S. Rasp, P. D. Dueben, S. Scher, J. A. Weyn, S. Mouatadid, and N. Thuerey, "WeatherBench: A Benchmark Data Set for Data-Driven Weather Forecasting," *Journal of Advances in Modeling Earth Systems*, vol. 12, no. 11, 2020.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[3] T. M. Mitchell and T. M. Mitchell, *Machine learning*, vol. 1. McGraw-hill New York, 1997.

[4] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[6] J. Bayer and C. Osendorfer, "Learning stochastic recurrent networks," *arXiv preprint arXiv:1411.7610*, 2014.

[7] P. Bauer, A. Thorpe, and G. Brunet, "The quiet revolution of numerical weather prediction," *Nature*, vol. 525, no. 7567, pp. 47–55, 2015.

[8] C. Abbe, "The physical basis of long-range weather forecasts," *Monthly Weather Review*, vol. 29, no. 12, pp. 551–561, 1901.

[9] P. Lynch, "The origins of computer weather prediction and climate modeling," *Journal of Computational Physics*, vol. 227, no. 7, pp. 3431–3444, 2008. Predicting weather, climate and extreme events.

[10] J. K. Lazo, R. E. Morss, and J. L. Demuth, "300 billion served: Sources, perceptions, uses, and values of weather forecasts," *Bulletin of the American Meteorological Society*, vol. 90, no. 6, pp. 785–798, 2009.

[11] E. R. Rodrigues, I. Oliveira, R. Cunha, and M. Netto, "DeepDownscale: A deep learning strategy for high-resolution weather forecast," *Proceedings - IEEE 14th International Conference on eScience, e-Science 2018*, pp. 415–422, 2018.

[12] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, pp. 802–810, 2015.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] X. Shi, Z. Gao, L. Lausen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Deep learning for precipitation nowcasting: A benchmark and a new model," in *Advances in neural information processing systems*, pp. 5617–5627, 2017.

[15] Q.-K. Tran and S.-k. Song, "Computer vision in precipitation nowcasting: Applying image quality assessment metrics for training deep neural networks," *Atmosphere*, vol. 10, no. 5, p. 244, 2019.

[16] H. De Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. Courville, "Modulating early visual processing by language," *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 6595–6605, 2017.

[17] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "FiLM: Visual reasoning with a general conditioning layer," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3942–3951, 2018.

[18] X. Wang, K. Yu, C. Dong, and C. Change Loy, "Recovering Realistic Texture in Image Super-Resolution by Deep Spatial Feature Transform," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 606–615, 2018.

[19] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," *arXiv preprint arXiv:1704.02971*, 2017.

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

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[22] L. Zhang, L. Mei, S. A. A. Shah, G. Zhu, P. Shen, and M. Bennamoun, "Attention in convolutional LSTM for gesture recognition," *Advances in Neural Information Processing Systems*, vol. 2018-December, no. NeurIPS, pp. 1953–1962, 2018.

[23] S. Agrawal, L. Barrington, C. Bromberg, J. Burge, C. Gazen, and J. Hickey, "Machine learning for precipitation nowcasting from radar images," *arXiv*, no. NeurIPS, pp. 1–6, 2019.

[24] J. N. Liu, Y. Hu, Y. He, P. W. Chan, and L. Lai, "Deep neural network modeling for big data weather forecasting," in *Information Granularity, Big Data, and Computational Intelligence*, pp. 389–408, Springer, 2015.

[25] M. Hossain, B. Rekabdar, S. J. Louis, and S. Dascalu, "Forecasting the weather of Nevada: A deep learning approach," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2015-September, pp. 2–7, 2015.

[26] J. Lee, C. G. Kim, J. E. Lee, N. W. Kim, and H. Kim, "Application of artificial neural networks to rainfall forecasting in the Geum River Basin, Korea," *Water (Switzerland)*, vol. 10, no. 10, 2018.

[27] B. Kanigoro and A. G. Salman, "Recurrent gradient descent adaptive learning rate and momentum neural network for rainfall forecasting," in *2016 international seminar on application for technology of information and communication (ISemantic)*, pp. 23–26, IEEE, 2016.

[28] M. A. Zaytar and C. El Amrani, "Sequence to sequence weather forecasting with long short-term memory recurrent neural networks," *International Journal of Computer Applications*, vol. 143, no. 11, pp. 7–11, 2016.

[29] S. Poornima and M. Pushpalatha, "Prediction of rainfall using intensified LSTM based recurrent Neural Network with Weighted Linear Units," *Atmosphere*, vol. 10, no. 11, 2019.

[30] B. Wang, H. Luo, J. Lu, T. Li, G. Zhang, Z. Yan, and Y. Zheng, "Deep uncertainty quantification: A machine learning approach for weather forecasting," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2087–2095, 2019.

[31] P. Hewage, M. Trovati, E. Pereira, and A. Behera, "Deep learning-based effective fine-grained weather forecasting model," *Pattern Analysis and Applications*, no. 0123456789, 2020.

[32] J. G. Powers, J. B. Klemp, W. C. Skamarock, C. A. Davis, J. Dudhia, D. O. Gill, J. L. Coen, D. J. Gochis, R. Ahmadov, S. E. Peckham, *et al.*, "The weather research and forecasting model: Overview, system efforts, and future directions," *Bulletin of the American Meteorological Society*, vol. 98, no. 8, pp. 1717–1737, 2017.

[33] W.-c. Woo and W.-k. Wong, "Operational application of optical flow techniques to radar-based rainfall nowcasting," *Atmosphere*, vol. 8, no. 3, p. 48, 2017.

[34] Q. K. Tran and S. K. Song, "Computer vision in precipitation nowcasting: Applying image quality assessment metrics for training deep neural networks," *Atmosphere*, vol. 10, no. 5, pp. 1–20, 2019.

[35] A. Heye, K. Venkatesan, and J. Cain, "Precipitation nowcasting: Leveraging deep recurrent convolutional neural networks," *Proceedings of the Cray User Group (CUG)*, vol. 2017, 2017.

[36] J. R. Jing, Q. Li, X. Y. Ding, N. L. Sun, R. Tang, and Y. L. Cai, "AENN: A GENERATIVE ADVERSARIAL NEURAL NETWORK for WEATHER

RADAR ECHO EXTRAPOLATION," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, vol. 42, no. 3/W9, pp. 89–94, 2019.

[37] S. Song, C. Lan, J. Xing, W. Zeng, and J. Liu, "An end-to-end spatio-temporal attention model for human action recognition from skeleton data," *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pp. 4263–4270, 2017.

[38] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "Geoman: Multi-level attention networks for geo-sensory time series prediction," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2018-July, pp. 3428–3434, 2018.

[39] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. Li, "Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction," *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, pp. 5668–5675, 2019.

[40] I. A. Abdellaoui and S. Mehrkanoon, "Deep multi-stations weather forecasting: explainable recurrent convolutional neural networks," 2020.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[42] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[43] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[44] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image

is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[45] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

[46] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, "Video swin transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3202–3211, 2022.

[47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[48] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[49] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[50] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[51] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep api learning," pp. 631–642, 05 2016.

[52] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[53] H. Hersbach, B. Bell, P. Berrisford, S. Hirahara, A. Horányi, J. Muñoz-Sabater, J. Nicolas, C. Peubey, R. Radu, D. Schepers, A. Simmons, C. Soci, S. Abdalla, X. Abellan, G. Balsamo, P. Bechtold, G. Biavati, J. Bidlot,

M. Bonavita, G. De Chiara, P. Dahlgren, D. Dee, M. Diamantakis, R. Dragani, J. Flemming, R. Forbes, M. Fuentes, A. Geer, L. Haimberger, S. Healy, R. J. Hogan, E. Hólm, M. Janisková, S. Keeley, P. Laloyaux, P. Lopez, C. Lupu, G. Radnoti, P. de Rosnay, I. Rozum, F. Vamborg, S. Villaume, and J. N. Thépaut, "The ERA5 global reanalysis," *Quarterly Journal of the Royal Meteorological Society*, 2020.

[54] O. Ronneberger, P. Fischer, and T. Brox, "Unet," *MICCAI2015*, 2015.

[55] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[56] A. V. Metcalfe and P. S. Cowpertwait, *Introductory time series with R*. Springer, 2009.

[57] H. Wang *et al.*, "Crime rate inference with big data," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 13-17-Augu, pp. 635–644, 2016.

[58] M. H. Khan *et al.*, "Spatiotemporal features of human motion for gait recognition," *Signal, Image and Video Processing*, vol. 13, no. 2, pp. 369–377, 2019.

[59] A. R. Patil *et al.*, "A spatiotemporal approach for vision-based hand gesture recognition using Hough transform and neural network," *Signal, Image and Video Processing*, vol. 13, no. 2, pp. 413–421, 2019.

[60] Q. Huang *et al.*, "View transform graph attention recurrent networks for skeleton-based action recognition," *Signal, Image and Video Processing*, vol. 15, no. 3, pp. 599–606, 2021.

[61] K. Deepak *et al.*, "Residual spatiotemporal autoencoder for unsupervised video anomaly detection," *Signal, Image and Video Processing*, vol. 15, no. 1, pp. 215–222, 2021.

[62] J. Zhang *et al.*, "Dnn-based prediction model for spatio-temporal data," SIGSPACIAL '16, (New York, NY, USA), Association for Computing Machinery, 2016.

[63] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pp. 1655–1661, 2017.

[64] B. Wang *et al.*, "Deep Learning for Real Time Crime Forecasting," pp. 33–36, 2017.

[65] C. Huang *et al.*, "MIST: A multiview and multimodal spatial-temporal learning framework for citywide abnormal event forecasting," *WWW 2019*, pp. 717–728, 2019.

[66] J. Sun *et al.*, "CrimeForecaster: Crime Prediction by Exploiting the Geographical Neighborhoods' Spatiotemporal Dependencies," *Lecture Notes in Computer Science*, vol. 12461 LNAI, pp. 52–67, 2021.

[67] C. Wang, Z. Lin, X. Yang, J. Sun, M. Yue, and C. Shahabi, "HAGEN: Homophily-Aware Graph Convolutional Recurrent Network for Crime Forecasting," 2021.

[68] Z. Wu *et al.*, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[69] A. Jain *et al.*, "Structural-RNN: Deep learning on spatio-temporal graphs," *CVPR*, vol. 2016-Decem, pp. 5308–5317, 2016.

[70] D. Salinas *et al.*, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting*, vol. 36, no. 3, pp. 1181–1191, 2020.

[71] S. Kullback, *Information theory and statistics*. Mineola, New York: Dover Publications Inc., 1968.

[72] Y. Seo *et al.*, "Structured sequence modeling with graph convolutional recurrent networks," *Lecture Notes in Computer Science*, vol. 11301 LNCS, no. 2013, pp. 362–373, 2018.

[73] X. Shi *et al.*, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," NIPS'15, (Cambridge, MA, USA), p. 802–810, MIT Press, 2015.

[74] C. I. P. Department, "Chicago crimes, 2001-2018," Tech. Rep. ICPSR37256 - v1, Inter-university Consortium for Political and Social Research, Ann Arbor , MI, 2019.