Korekcija kratkoročne vremenske prognoze radarskim snimcima i strojnim učenjem

Orlić, Sara

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 121

WEATHER NOWCASTING USING RADAR IMAGING AND MACHINE LEARNING

Sara Orlić

Zagreb, February 2025

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 121

WEATHER NOWCASTING USING RADAR IMAGING AND MACHINE LEARNING

Sara Orlić

Zagreb, February 2025

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 03 October 2024

MASTER THESIS ASSIGNMENT No. 121

Student:	**Sara Orlić (0036525955)**
Study:	Information and Communication Technology
Profile:	Control Systems and Robotics
Mentor:	assoc. prof. Vinko Lešić, PhD

Title: **Weather nowcasting using radar imaging and machine learning**

Description:

Weather forecasting today most commonly uses systems based on numerical weather prediction. Such systems enable forecasting of several days ahead but are not suitable for predictions in shorter time intervals. Nowcasting, as an immediate weather prediction, is defined as a detailed description of current weather conditions and the prediction of changes expected within a few hours in a specific area. The thesis task is to study machine learning approaches that can be applied to improve short-term weather forecasts using radar images. For selected weather variables, with emphasis on precipitation, it is required to perform training and testing on an available dataset and analyse the obtained results.

Submission date: 14 February 2025

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. listopada 2024.

DIPLOMSKI ZADATAK br. 121

Pristupnica:	**Sara Orlić (0036525955)**
Studij:	Informacijska i komunikacijska tehnologija
Profil:	Automatika i robotika
Mentor:	izv. prof. dr. sc. Vinko Lešić

Zadatak: **Korekcija kratkoročne vremenske prognoze radarskim snimcima i strojnim učenjem**

Opis zadatka:

Za vremensku prognozu danas se najčešće koriste sustavi bazirani na numeričkom predviđanju vremena. Takvi sustavi omogućavaju dobivanje prognoze za nekoliko dana unaprijed, ali nisu pogodni za prognoziranje u kraćim vremenskim intervalima. Prognoza neposrednog razvoja vremena (eng. nowcasting) definirana je kao detaljan opis trenutnih vremenskih prilika i predviđanje promjena koje se mogu očekivati unutar par sati na određenom području. U radu je potrebno proučiti pristupe strojnog učenja koji se mogu primijeniti za korekciju kratkoročne vremenske prognoze koristeći radarske snimke. Za odabrane vremenske varijable s naglaskom na padaline potrebno je provesti učenje i testiranje na dostupnom skupu podataka te analizirati dobivene rezultate.

Rok za predaju rada: 14. veljače 2025.

Contents

1 Introduction

In today's rapidly evolving world, the demand for precise weather forecasts has become increasingly critical. Due to this requirement nowcasting, or the short-term forecasting of weather conditions, has become more and more common. Nowcasting tries to provide highly accurate, localized information about precipitation intensity over the next few minutes or hours [1]. Such forecasts play a crucial role in enabling early warnings for regions tending to serious weather events, optimizing agricultural operations by helping farmers in rainfall preparation, and supporting planning in airports and other critical infrastructure. Additionally, accurate nowcasting contributes to disaster management, improves energy grid stability during damaging conditions, and supports urban mobility planning during sudden weather changes. As it is written in [2]: "According to a recent report from the WMO (World Meteorological Organization), over the past 50 years, more than 34% of all recorded disasters, 22% of related deaths (1.01 million) and 57% of related economic losses (US$ 2.84 trillion) were consequences of extreme-precipitation events." Given the need for high accuracy and precision in precipitation prediction, nowcasting is significantly different from standard weather forecasting tasks.

Traditionally, numerical weather prediction (NWP) models have been the primary tool for weather forecasting, including precipitation prediction. These models rely on solving complex systems of mathematical equations that simulate atmospheric processes. However, they are computationally intensive and less effective for short-term forecasting due to the fine temporal and spatial resolution required. This limitation highlights the need for alternative approaches that can deliver real-time, high-resolution predictions [3]. The rapid growth of artificial intelligence (AI) has significantly impacted many scientific fields, including weather and climate modeling. Machine learning (ML) and deep learning (DL) have become powerful tools for identifying patterns and relationships in at-

mospheric data. Unlike traditional weather prediction models, ML and DL methods use large datasets to uncover trends and dynamics in weather systems. Over the last decade, improvements in computing power and access to vast weather datasets have made these approaches increasingly popular, especially for precise tasks like nowcasting [4]. Radar imaging has showed as a great tool in this context, offering detailed spatiotemporal data on precipitation intensity and distribution. That is why radar images meet demands of nowcasting.

This study focuses on developing a deep learning-based model for short-term precipitation forecasting using radar imaging. Given the spatiotemporal complexity of radar data, a Convolutional Long Short-Term Memory (ConvLSTM) model with an encoder-decoder architecture was chosen. This structure combines convolutional layers, which capture spatial patterns, with LSTM units, which model temporal dependencies, making it well-suited for weather nowcasting tasks[4]. The model was trained on high-resolution radar composites provided by the Croatian Meteorological and Hydrological Service. Two different training approaches were explored: teacher forcing and autoregressive forecasting. In the teacher-forcing approach, the model learns by always using the correct past radar frame (ground truth) as input, ensuring stable and accurate predictions[5]. This method helps the model learn precipitation patterns effectively, but it is not practical for real-world forecasting, where future ground truth is unknown. On the other hand, in the autoregressive approach, the model generates predictions sequentially, using its own previous outputs as inputs for future frames. This method better represents real forecasting conditions but can suffer from error accumulation, where small mistakes grow over time, reducing accuracy in long-term predictions. A major challenge in training the model was data imbalance, as radar images contain large areas without precipitation, making it difficult for the model to focus on relevant rainfall patterns. To address this, a Weighted Mean Squared Error (MSE) loss function was used, giving more importance to precipitation areas. This adjustment helped improve the model's ability to predict both rainfall intensity and movement, making it more effective in handling real-world weather forecasting tasks.

To train the model effectively, the radar data goes through a preprocessing pipeline that includes filtering, normalization, and transformation. The data is first cleaned to

remove corrupt or missing frames, ensuring that only complete radar sequences are used. Next, background noise and non-relevant regions are filtered out, focusing the model on precipitation zones. The dataset is then rescaled and normalized, ensuring that pixel values are properly adjusted for deep learning. Given the high computational demands of training a ConvLSTM-based model on spatiotemporal data, training was made on the Supek supercomputer, using high-performance computing (HPC) resources to overcome memory and processing constraints. The use of multi-GPU acceleration allowed efficient batch processing, significantly reducing training time.

2 Radar Composites and Data Preparation

Radar composites are images obtained from meteorological radars. The basic principle of meteorological radar operation involves emitting short, directed pulses of electromagnetic radiation into the atmosphere and then detecting the energy reflected by potential targets. Depending on the wavelength of the emitted radiation and the properties of the target, reflection may or may not occur [6]. Radar composites are actually spatiotemporal maps of precipitation intensities determined from radar imaging. Their biggest advantage is real-time availability which can help in detecting disasters in short-term weather prediction.

Figure 2.1: Radar on a specially designed tower located at meteorological radar center Uljenje in Croatia [6]

2.1 Description of the Dataset

The radar images used in this research were obtained from the Croatian Meteorological and Hydrological Service (hrv. Državni hidrometeorološki zavod, DHMZ). These datasets were accessed and retrieved from DHMZ's online servers over a period from December 2022 to July 2024. The radar data was collected at five-minute intervals, providing high temporal resolution essential for the goals of this study. The Figure2.2 is example of one picture retrieved from DHMZ's webpage.

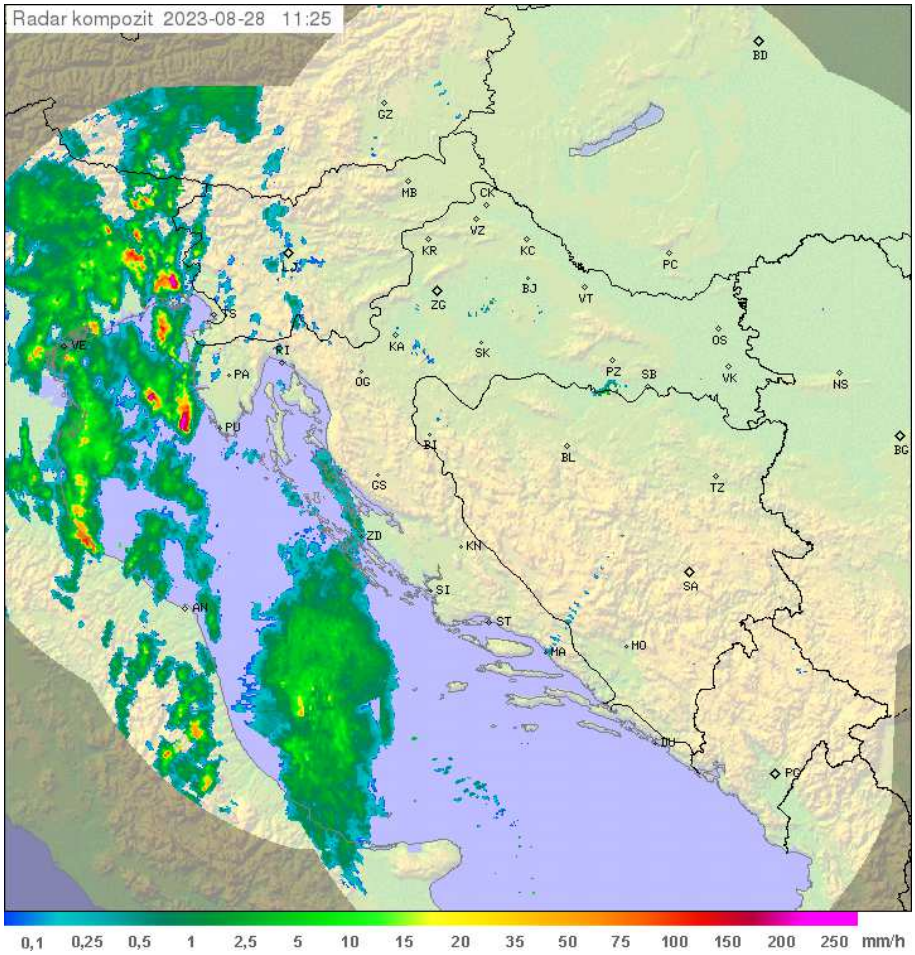

Figure 2.2: Example of radar composite retrieved from Croatian Meteorological and Hydrological Service (DHMZ) webpage [7]

The radar composites display precipitation intensities, which are represented in the legend located at the bottom of each image. The unit of measurement for these intensities is *mm/h*, indicating the estimated rainfall rate. The legend's scale is non-linear to cover a wide range of precipitation intensities, from lighter rain to heavy downpours, showing valid representation of varying weather conditions. Explanation of precipitation intensity is showed in Table2.1.

Table 2.1: Description of precipitation intensities

Precipitation intensity [mm/h]	Description
0.1 - 1	Insignificant to low
1 - 5	Low to medium
5 - 25	Medium to strong
25 - 100	Strong to very strong
> 100	Extreme, thunderstorm

The geographical coverage of the radar includes whole Croatia, extending to surrounding areas. However, due to external factors such as technical malfunctions, some portions of the data are corrupted or incomplete making them unable for further processing.

2.2 Data Preprocessing

The original radar composites have a resolution of 720*px* x 751*px*, which makes them too large for efficient processing. Hence, images are cropped to size 142*px* x 77*px*, reducing their dimensionality and making them more suitable for use in the deep learning model. The cropped images focus on the area around the city of Rijeka and its surroundings 2.3. Rijeka was chosen as the focus area because of its high annual rainfall, making it an ideal region for studying precipitation patterns and testing the model. The images are represented in RGB format, consisting of matrices with dimensions 142 x 77 x 3, where the first two dimensions correspond to the spatial resolution, and the third represents the three color channels (red, green, and blue).

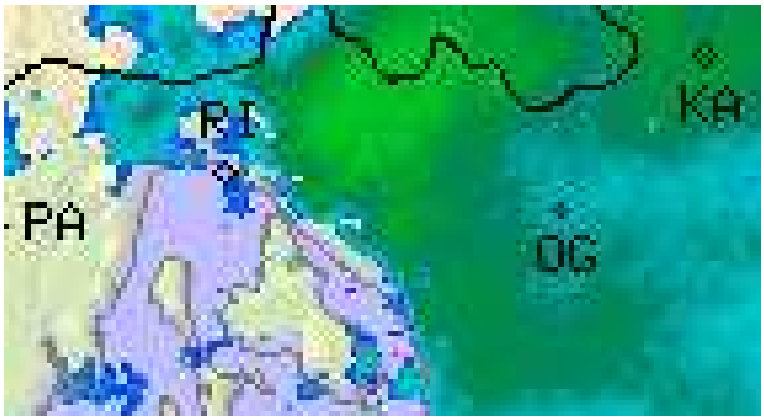

Figure 2.3: Cropped image showing area of Rijeka

For images to be compatible with further processing, they need to meet with criteria

for deep learning model. The primary goal of this preparation is to preserve the spatiotemporal dependencies within the data, as these relationships are essential for accurate weather predictions. The frames are divided into batches to preserve spatiotemporal dependencies. Each batch consists of 28 samples, with 14 used as input and 14 used for predictions. Determining the appropriate number of batches was challenging due to the limited timeframe of the dataset and the presence of days without precipitation. These dry periods also made it difficult to ensure that each batch contained consecutive frames with rainfall. To address this, a filtering process was applied. For each batch, the first and last frames were analyzed by comparing the colors in the legend to the precipitation regions on the radar images. A threshold for precipitation intensity was defined, and at least one of the two frames (first or last) had to meet this minimum threshold. If this condition was satisfied, the batch was considered valid and contained a sufficient percentage of precipitation for the task. Number of batches should be big enough for model to have data to work with and to show sufficient results. Therefore, number of batches is 5800 containing 28 frames per batch. But because of a lot batches not meeting the criteria for precipitation percentage that number of batches is 2553. That is 71484 images which is not sufficient number.

2.3 Data Shaping for the Model

2.3.1 Issues with Raw Data and Solutions

The main challenge with images, such as the one shown in Figure 2.3, was that they were not suitable for direct input into the model. This was due to the presence of background colors and other irrelevant details that could confuse the model. The goal was to focus entirely on the precipitation areas. To address this, all background colors, boundaries, and non-relevant parts of the images were replaced with black. As a result, the original three-dimensional RGB color channels were transformed into a single grayscale channel, where precipitation areas are highlighted, and the background remains black. To identify precipitation regions, we compared the colors on the legend with the colors on the radar images. A mapping was created between the colors in the legend and their corresponding precipitation intensities. Since there are more colors on the images than intensity levels in the legend, we introduced a tolerance range to match similar colors

to the correct intensity values. Precipitation intensities in the legend range from very low (0.1 *mm/h*) to very high (250 *mm/h*). These intensities were scaled to a range of [0.3, 1.2] to differentiate very low precipitation values from zero, which represents non-precipitation. This scaling ensures that the model can accurately interpret the intensity levels. The final preprocessed image used as input for the model is shown in Figure 2.4.

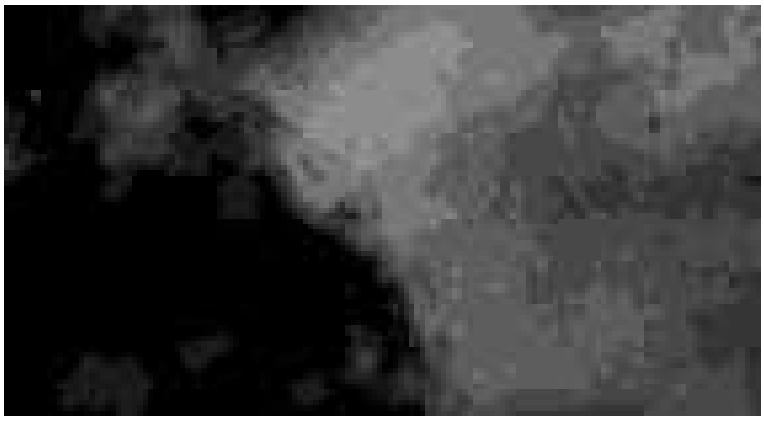

Figure 2.4: Final grayscaled image

The same image, but represented using the Viridis colormap from Matplotlib [8], is shown in Figure 2.5.

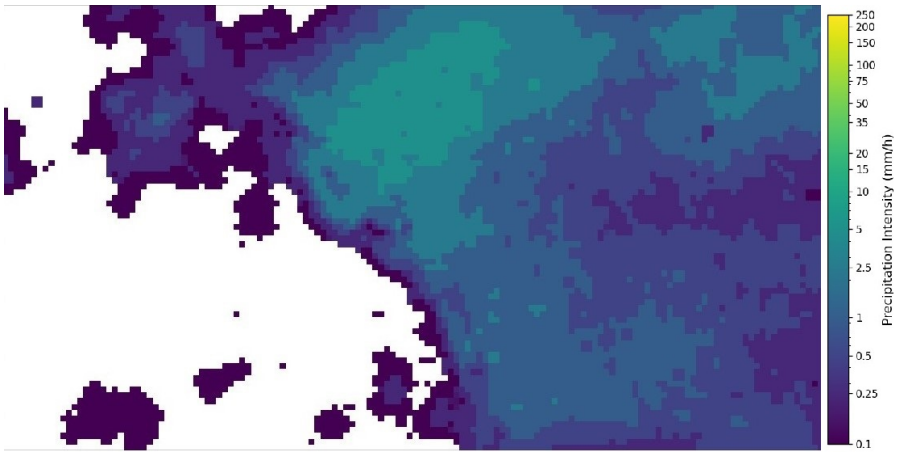

Figure 2.5: Image represented using the Viridis colormap, generated with Matplotlib

2.3.2 Input and output format

The dataset is then prepared for the model. The input dataset is structured with the shape [*batch size, number of input images, height, width*]. In this setup, the number of input images is 14. As mentioned before, each sequence contains 28 images which are split into two parts: the first 14 images are used as input, and the remaining 14 images are used

as the output for predictions. The output data represents the next 14 consecutive radar frames that follow the input sequence. The model's goal is to learn patterns from the input frames and use this knowledge to predict what the radar images will look like for the following time steps. In essence, the output data acts as the future that the model tries to forecast based on the given past. This setup allows the model to focus on predicting short-term changes in precipitation, which is crucial for accurate nowcasting.

3 Prediction model

In this chapter, we will look at how neural networks are used to work with radar images that change over time. Neural networks are powerful tools for finding patterns in complex data, like weather radar images. We will start by discussing what are basic neural networks, then discussing about Convolutional Neural Networks (CNNs), which are great at understanding the spatial details in images, and Long Short-Term Memory Networks (LSTMs), which are designed to handle changes over time in sequential data. Finally, we will focus on how these two methods are combined in the ConvLSTM model. This model is especially useful for weather predictions because it can capture both the spatial patterns and how they evolve over time.

3.1 Neural networks

Neural networks are subset of machine learning and a type of deep learning algorithm inspired by the structure and functioning of the human brain. Just as biological neurons in the brain process and transmit information through electrical signals, artificial neural networks consist of interconnected nodes, or neurons, that process and transmit data. This design mimics how the brain learns and adapts by forming connections between neurons, making neural networks capable of learning complex patterns from data [9].

A single node, or neuron, is the fundamental building block of a neural network. A visual representation of how a single node operates is shown in Figure 3.1. Each node takes in multiple inputs, processes them, and produces a single output. The inputs, often represented as X, come with associated weights (W) that determine their importance to the node. These inputs are combined into an intermediate value (H) through a weighted sum, adding an additional constant value called bias (B) for extra flexibility. Mathematically, this is represented as in equation(3.1). The intermediate value is then passed

through an activation function ($f()$), which determines whether the node should pass its output forward. The activation function introduces nonlinearity, enabling the network to handle more complex relationships in the data. The final output from the node (Y) is then used in further computations within the network.

$$H = \sum(W \cdot X) + B \tag{3.1}$$

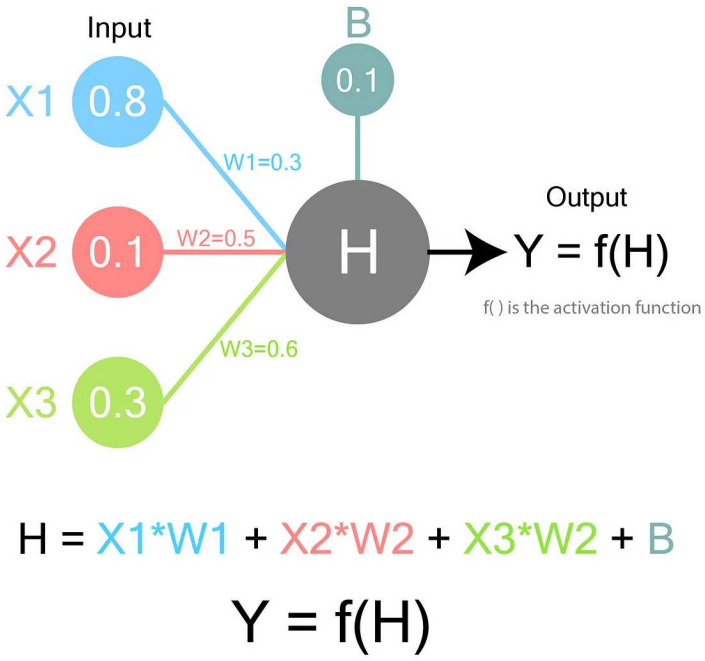

Figure 3.1: Single neuron functionality [10]

In Figure 3.2, we can see an example of a neural network with four layers. The first layer is the **input layer**, where we provide the data that the neural network will use to learn. This could be anything, like radar images or numerical data. The last layer is the output layer, which gives us the final result that the network is predicting or learning, such as a future weather map or a specific value. Between these are the **hidden layers**, which perform the actual computations needed to process the input data and transform it into the output. When we feed data into the input layer, the values are passed through the network layer by layer. Each node in a layer takes the values from the previous layer, multiplies them by their respective weights, adds a bias, and applies an activation function to produce its output. This process, called **forward propagation**, continues until we reach the output layer, where the network produces its final prediction.

The neural network learns by comparing its output to the actual target value or image using a loss function, which measures how far off the prediction is. Based on this error, the network adjusts its weights and biases in a process called **backpropagation**. During backpropagation, the error is propagated backward through the network, and the weights are updated using an optimization algorithm like gradient descent. This process is repeated many times over multiple rounds of training, allowing the neural network to improve its predictions and learn from the data.

Neural networks are particularly well-suited for handling spatiotemporal data, such as radar images used in weather prediction, due to their ability to capture both spatial and temporal patterns. The **output layer** of such a neural network generates a set of pixel values that represent the predicted radar image, arranging these pixels into the correct dimensions (height, width, and color channels) to form the image.

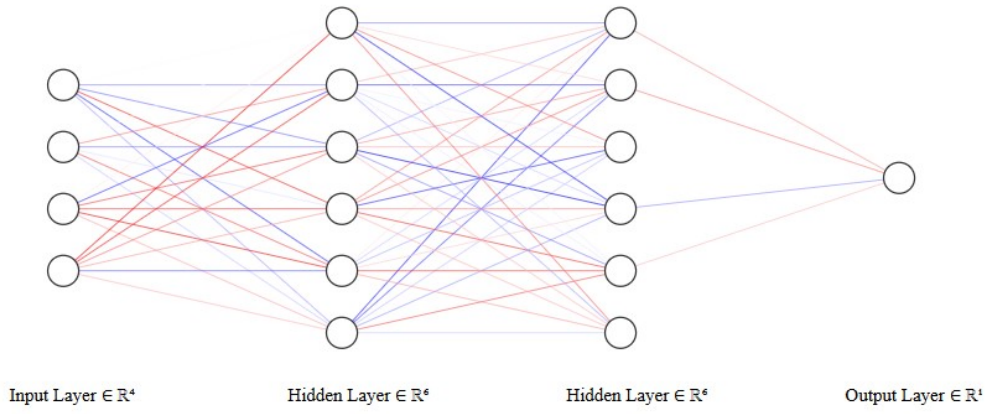

Figure 3.2: Representation of neural network drawn with [11]

3.1.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks, commonly called CNNs are specialized type of neural network design to work well with images. Convolutional layers are the fundamental building blocks of CNNs. These layers are designed to process data with a spatial structure, by applying a mathematical operation called *convolution*. We define convolution as the scalar product of one function with respect to the shifted and reflected second function[12]. The general equation for convolution in 2D is denoted as in equation(3.2). Where I is denoting input image with pixels values at position $(i+m, j+n)$. $K(m,n)$ is the kernel value at position (m,n). The ranges $m_{\min}, n_{\min}$ and $m_{\max}, n_{\max}$ are determined

by the values at which I and K are defined.

$$S(i,j) = \sum_{m=m_{min}}^{m_{max}} \sum_{n=n_{min}}^{n_{max}} I(i+m, j+n) \cdot K(m,n) \tag{3.2}$$

Convolution in images involves sliding a small matrix, known as a **kernel or filter**, over the input image to perform element-wise multiplication and summation. This operation produces a new matrix called a **feature map**, which highlights important features from the input. Kernel with size 3x3 going over initial image is shown in Figure3.3.

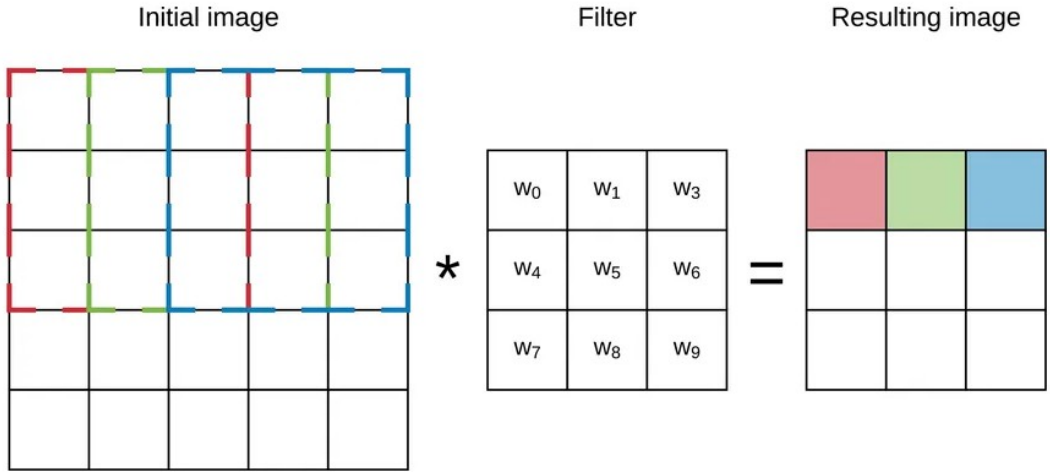

Figure 3.3: Example of the first 3 steps of a convolution with kernel size 3x3 over a 5x5 array of an image. The resulting pixel values correspond to the sum of the element-wise multiplication of the initial pixels-dashed lines and the kernel. [13]

The kernel is a small matrix of weights that is learned during training. Each kernel focuses on detecting specific patterns, such as edges, corners, or textures, within the input image. For example, one kernel might detect horizontal edges, while another focuses on vertical edges. By applying multiple kernels, convolutional layers extract various spatial features from the input data, which are then passed to the next layers for further processing.

In the context of radar images, which represent precipitation intensity across a geographical area, CNNs dominate at detecting spatial patterns by processing the images layer by layer. By stacking multiple convolutional layers, CNNs are able to identify increasingly complex features. The initial layers focus on basic patterns like edges, while deeper layers combine these features to recognize larger, more meaningful structures,

such as the shape of a storm system [14].

3.1.2 Recurrent Neural Networks(RNNs)

A recurrent neural network also called RNN is extension of traditional feedforward neural network. The RNN is having recurrent hidden state which allows it to handle variable-length sequence and it depends on activation time from previous state [15]. Recurrent Neural Networks (RNNs) can be understood as a series of connected copies of the same network, where each step passes information to the next. This architecture is designed to make use of the sequential nature of data. The term "recurrent" comes from the fact that these networks perform the same operation at each step of a sequence, with the output at one step depending on both the current input and the outputs of previous steps. This is achieved by looping the output of the network at time t back as input for the network at time $t + 1$. These loops allow the network to retain information from earlier steps in the sequence, effectively creating a memory that helps it understand patterns over time [16].

In Figure3.4 we can see representation of RNN unrolled structure. The network is expanded over time, showing its structure for a sequence of time steps. Each node corresponds to a specific point in time and represents the state of the network at that moment. In this architecture, the network can process a unique input (x_t) at each time step and produce a corresponding output (h_t) for that same time step. At the same time, the network maintains a memory state that carries information about everything that has happened in previous time steps up to the current point (t). This design allows the network to consider both the current input and past inputs, making it well-suited for sequential data.

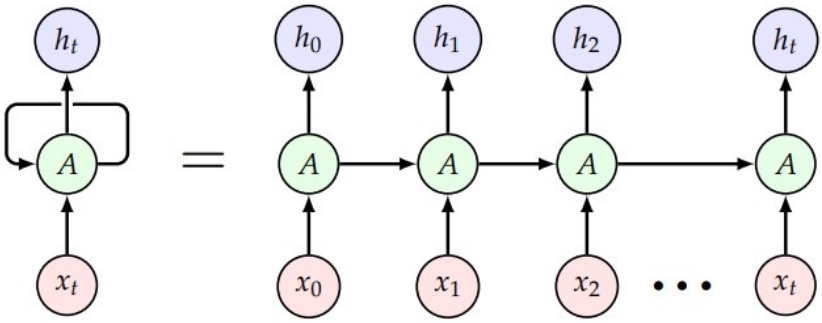

Figure 3.4: Unrolled RNN structure [16]

3.1.3 Long Short-Term Memory Networks (LSTMs)

Long Short-Term Memory Networks, known as LSTMs are a particular kind of RNN. LSTMs are designed to learn long-term dependencies thanks to their unique structure, which includes special components called gates that control the flow of information. Unlike regular neurons, the hidden layers in LSTMs are made up of **Memory Cells (MCs)**. Diagram that represents one LSTM memory cell is presented in Figure 3.5 These memory cells can either keep or forget information about past states of the network, depending on the instructions from the gates. Each gate is made up of a combination of a neuron with a sigmoidal activation function and a pointwise multiplication operation, which together decide how much information to keep or discard [17]. This structure allows each memory cell's output to consider the entire sequence of past states, making LSTMs especially effective for processing time-series data with long-term dependencies.

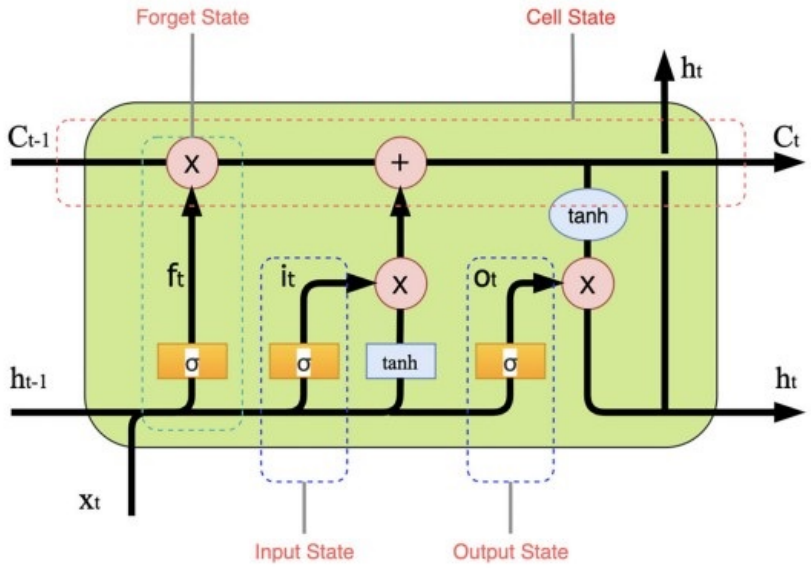

Figure 3.5: A single LSTM Memory Cell [18]

A single LSTM cell consists of: forget gate f_t, input gate i_t, cell candidate $\tilde{c}_t$, output gate o_t, cell state c_t and hidden state h_t:

- **Forget Gate** decides what information to discard from the previous cell state up until this point.
- **Input Gate** decides what new information to store in the cell state.
- **Cell Candidate** is the potential new cell state based on the current input and hid-

den state.

- **Output Gate** controls how much of the cell state will be used to compute the hidden state(output).
- **Cell State** is the memory of the LSTM, combining new information with the retained memory.
- **Hidden State** is encoding of the most recent time step $t-1$ and can be processed to obtain more meaningful data.

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f), \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i), \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c), \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o), \\
h_t &= o_t \circ \tanh(c_t).
\end{aligned}
\tag{3.3}
$$

In equation(3.3), $\sigma(\cdot)$ is the sigmoid function, W and b are respectively the weight matrix and bias of the gates or cell state. Operator "$\circ$" denotes element-wise multiplication or Hadamard product. The cells are managed by several gates. When a new input comes, input gate i_t decides whether information will be accepted to the cell or not. If forget gate f_t is activated, it decides if past cell status c_{t-1} will be "forgotten" or sent to further processing. The output gate o_t propagates whether latest cell state c_t will be sent to hidden state (final state) h_t [19].

One key advantage of using the memory cell and gates to regulate information flow is their ability to prevent gradients from vanishing too quickly, a significant issue in vanilla RNN models [20]. This mechanism, often referred to as constant error carousels [17], effectively traps the gradient within the cell, ensuring it persists over time and enhances the model's capacity to learn from long-term dependencies.

3.2 Convolutional LSTM Architecture

3.2.1 Comparing ConvLSTM with CNNs and LSTMs

A deep learning model called Convolutional Long Short-Term Memory (ConvLSTM) combines the advantages of Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNNs) into a single architecture. This model is particularly effective for processing spatiotemporal data, such as radar images, which require both spatial feature extraction and temporal sequence modeling. Unlike traditional CNNs, which analyze spatial structures in static images, or standard LSTMs, which process sequential data by flattening images instead of using their 2D structure, ConvLSTM is designed to preserve spatial information while capturing temporal dependencies, making it highly suitable for weather prediction tasks [21].

ConvLSTM achieves this by replacing the fully connected layers(dense layers) in standard LSTMs with convolutional layers, allowing it to process input data while maintaining its spatial structure. In contrast to traditional LSTMs, where state transitions rely on dense matrix multiplications, ConvLSTM performs convolution operations at each step of the LSTM cell as shown in Figure3.6. This means that instead of processing isolated numerical values, it operates on entire feature maps, enabling the network to analyze how precipitation evolves over time while maintaining spatial consistency.

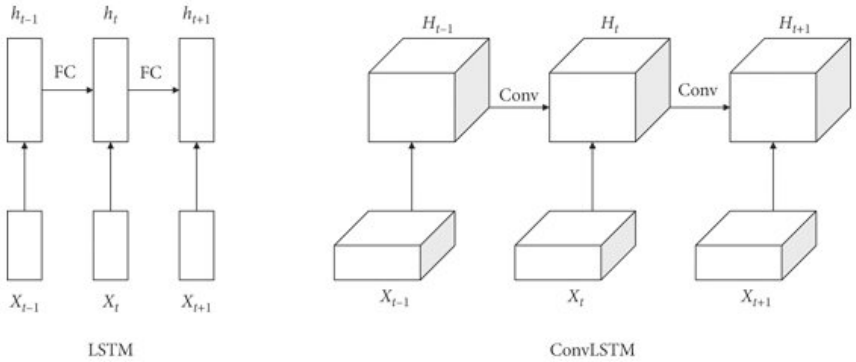

Figure 3.6: Structural differences between LSTM anf ConvLSTM [22]

As shown in Figure3.7, each gate in the ConvLSTM cell applies convolutional kernels over local neighborhoods in both the input data and the previous hidden state. This effectively acts like a sliding filter across the 2D spatial domain, rather than using dense transformations that discard spatial relationships.

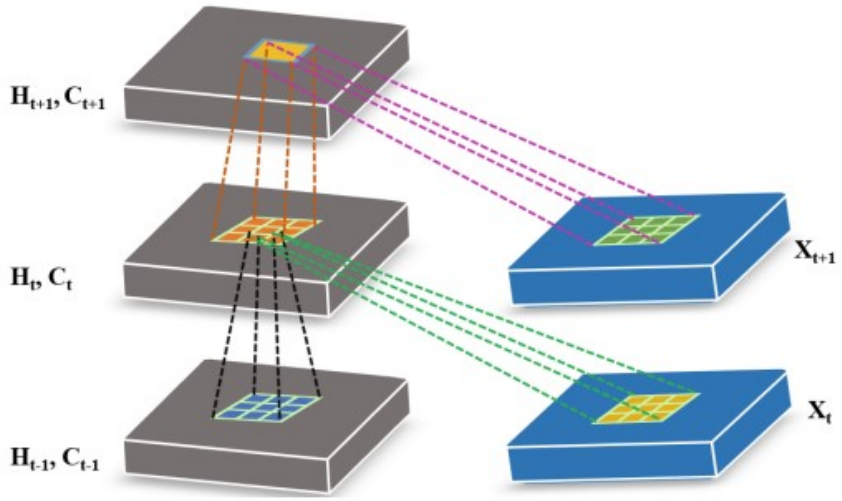

Figure 3.7: Inner structure of ConvLSTM [21]

The input-to-state and state-to-state transitions in ConvLSTM are handled through 3D tensors, where the last two dimensions represent spatial structure (rows and columns) and last one representing number of feature maps (channels/filters) (Figure3.8). That allows the model to keep critical spatial details that would otherwise be lost in a standard LSTM [19].

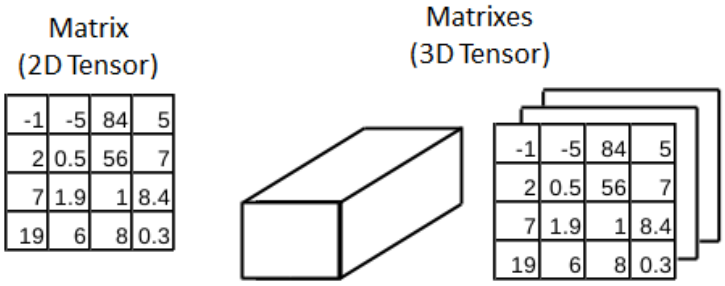

Figure 3.8: Representation of 2D and 3D tensor in imaging [23]

The key equations that are similar to standard LSTM but with convolution operator are shown in equation(3.4)

$$
\begin{aligned}
f_t &= \sigma(W_f * x_t + U_f * h_{t-1} + b_f), \\
i_t &= \sigma(W_i * x_t + U_i * h_{t-1} + b_i), \\
\tilde{c}_t &= \tanh(W_c * x_t + U_c * h_{t-1} + b_c), \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, \\
o_t &= \sigma(W_o * x_t + U_o * h_{t-1} + b_o), \\
h_t &= o_t \circ \tanh(c_t).
\end{aligned}
\tag{3.4}
$$

3.2.2 Encoder-Decoder Structure

The encoder-decoder structure was first proposed in *Sequence to Sequence Learning with Neural Networks* [24] as a powerful framework for sequence-to-sequence tasks. It has been widely applied in areas such as machine translation, speech recognition, and time-series forecasting due to its ability to encode past information into a compressed representation and use it to generate future sequences. The fundamental idea behind this architecture is to use an encoder to process and extract meaningful features from an input sequence and a decoder to transform this information into a structured output sequence.

The encoder-decoder structure is a widely used deep learning architecture designed for tasks that require transforming an input sequence into an output sequence, making it ideal for precipitation nowcasting with ConvLSTM. This approach is particularly useful when dealing with spatiotemporal data, as it allows the model to compress important information from past radar images and use it to predict future frames.

In the context of precipitation nowcasting, the encoder-decoder structure is particularly useful for modeling spatiotemporal dependencies in radar images. The encoder processes a sequence of past radar images, capturing both spatial patterns (storm structures, cloud formations) and temporal dynamics (movement, intensity changes). The extracted information is stored in a latent representation, which serves as a summary of the input sequence. The decoder then takes this representation and generates the next sequence of radar images, predicting how precipitation is likely to evolve. This sequential processing allows the model to learn from past weather conditions and forecast future precipitation patterns with high accuracy.

4 The Model

The goal of precipitation forecasting is to predict future radar frames based on a fixed sequence of input frames. As previously mentioned, the focus area is the region around Rijeka, where precipitation occurs frequently throughout the year. To achieve accurate forecasting, we use a deep learning model based on the Encoder-Decoder ConvLSTM architecture, implemented using TensorFlow/Keras.

The model follows an encoder-decoder structure to process radar image sequences efficiently. The encoder consists of three ConvLSTM layers, which extract both spatial and temporal patterns from the input sequence and it compresses them. This compressed information is then passed to the decoder, which reconstructs future frames step by step. Instead of predicting all frames at once, the decoder generates one frame at a time, using its previous output as the next input. This iterative approach ensures better temporal consistency and smoother predictions. The final frame predictions are produced using a Conv2D layer, mapping the extracted features to a single-channel precipitation intensity map.

Below, there are described the key components of the model, including encoder-decoder structure, the ConvLSTM2D layers, batch normalization, the TimeDistributed layer, kernel sizes, filter configurations. The shape of data is (None, 14, 77, 142, 1). *None* represents flexible batch sizes in Keras architecture and it means it can be any value during training, *number of input samples* is 14, *height and width* are (77,142) and *number of channels* is 1 as grayscaled data has one channel. Full system architecture from beginning with preprocessing of data, Supek supercomputer usage and model structure is provided in Figure4.1.

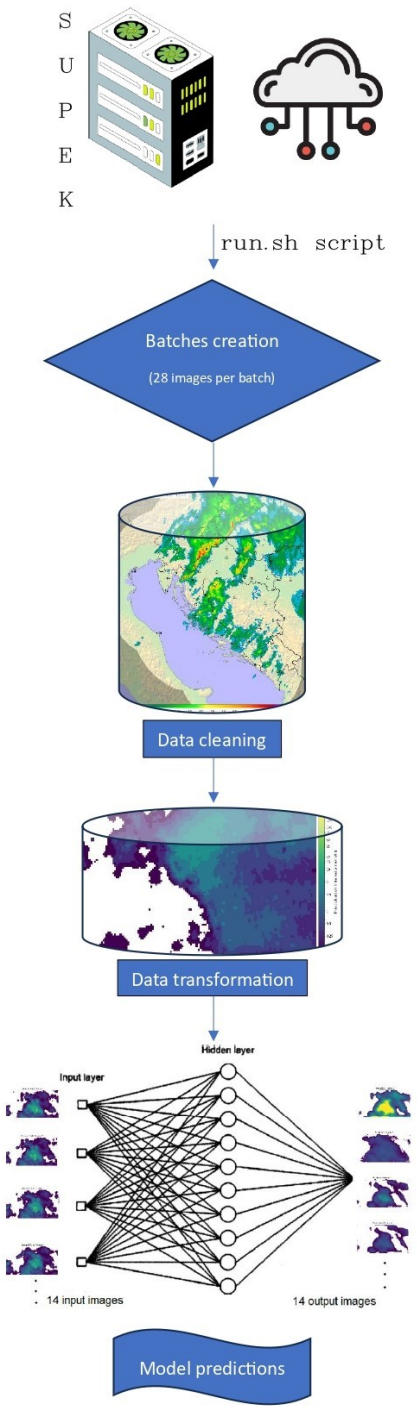

Figure 4.1: System architecture

4.1 Model Architecture

4.1.1 Encoder Decoder Architecture

The encoder-decoder architecture used in this model is designed to efficiently process sequences of radar images for precipitation forecasting. The model follows a two-step approach: the encoder compresses a sequence of past frames into a meaningful representation, and the decoder then generates future frames based on this learned representation.

In this study, two different training approaches were explored for the ConvLSTM-based Encoder-Decoder model in precipitation nowcasting: basic encoding-forecasting network also called autoregressive model and teacher forcing. Both approaches follow the same fundamental encoder-decoder structure, where the encoder processes past radar frames, extracting spatial and temporal dependencies, while the decoder generates future frames based on the learned representation. However, the key difference between these approaches lies in how the decoder receives input during training. The Figure 4.2 illustrates the autoregressive encoder-decoder model, where the encoder compresses past frames into hidden states and cell stats that are then passed to the decoder. Autoregressive models generate data sequentially, meaning each new output depends on the previous ones. In this approach, the decoder starts with an empty (zero) frame and generates each future frame step by step, using its own previous output as input for the next timestep. This approach closely represents real-world forecasting, as it simulates how precipitation develops over time without access to future frames. However, a major challenge of this method is error accumulation—if the model makes a small mistake in an early timestep, it may propagate through the sequence, affecting following predictions [25]. Additionally, this approach is computationally more expensive, as each frame must be generated sequentially.

The Figure4.3 demonstrates teacher forcing, an alternative training strategy where the decoder is provided with ground-truth future frames instead of its own predictions. In this approach, at timestep k, instead of using its previously predicted frame $k-1$ as input, the model is fed the actual ground-truth frame $k-1$ from the dataset. This method significantly improves training stability and convergence, allowing the model to learn

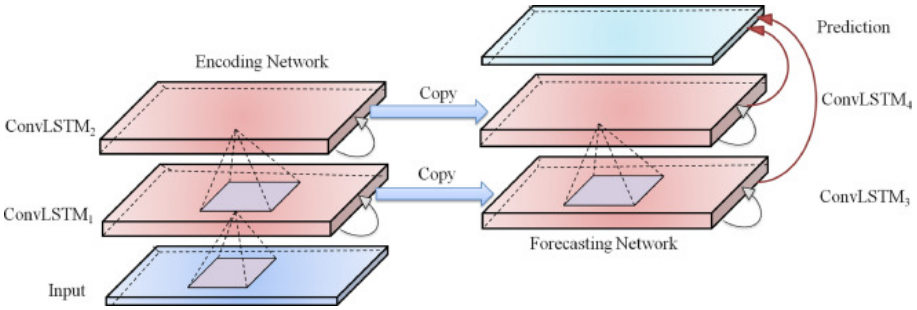

Figure 4.2: Autoregressive encoding-forecasting ConvLSTM network [26]

faster by always receiving the correct input. However, teacher forcing has never learned to rely on its own predictions. While teacher forcing accelerates training, it is not practical for real-time applications, as it does not reflect the actual conditions of precipitation forecasting.

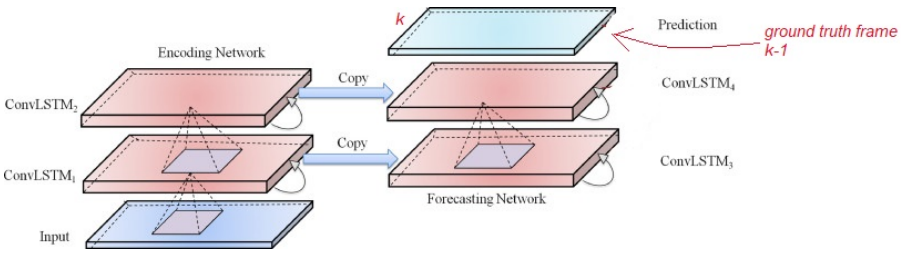

Figure 4.3: Basic encoding-forecasting ConvLSTM network

4.1.2 ConvLSTM2D Layers

We use ConvLSTM2D from *tensorflow.keras.layers*, which is an extension of LSTMs that incorporates convolutional operations inside its gates. The function's key parameters in the model include:

- **filters** – The number of feature maps learned at each ConvLSTM layer.
- **kernel_size** - The size of the convolutional filter applied at each step.
- **padding='same'** - Adds additional rows and columns of pixels around the edges of the input data so that the size of the output feature map is the same as the size of the input data. This is achieved by adding rows and columns of pixels with a value of zero around the edges of the input data before the convolution operation.
- **return_sequences=True** – Ensures that each ConvLSTM layer returns the entire sequence, allowing temporal dependencies to be preserved throughout the net-

work.

- **input_shape=(num_input_frames, 77, 142, 1)** – Defines the shape of the input sequence (time steps, height, width, channels).

Each **ConvLSTM2D** layer applies convolutional filters to extract spatial features. For first layer we use bigger kernel sizes like 7x7 or 5x5 because it attempts to capture wider spatial context. In precipitation data, storms can have large spatial coverage, so a bigger kernel can gather a broader region of rainfall patterns right away. As we go deeper, kernel sizes are decreasing in size like 3x3. With that size it is easier to capture small patterns. It is still good to capture local structures.

Filters represent how many distinct feature maps each layer can learn. Regarding number of filters, they are increasing in size as we go deeper into network. Smaller number of filters at beginning means computation cost is lower (16, 32). Later, network is learning more complex patterns, so number of filters is increasing (64, 128) . As well there are more abstract features, which might require more channels to represent distinct precipitation structures, movements, different intensities etc.

A 7x7 kernel is used for the first layer, which can capture broader spatial dependencies because first layer is not yet focused on fine details. Number of filters or feature maps for first layer is 32. Second layer uses kernel of size 5x5 as it can improve the features extracted by the first layer. Chosen number of filters for second layer is 64. Last two ConvLSTM2D layers use 3x3 kernel and number of feature maps 64, 128 respectively for last layer.

4.1.3 Batch Normalization

Batch Normalization (BatchNorm) is a technique used to keep the input values of each layer stable during training. It works by adjusting the distribution of activations (outputs of neurons) within each mini-batch. This is done by adding extra layers that modify the activation values so that their mean becomes zero and their variance becomes one. After this normalization step, the values are usually scaled and shifted using trainable parameters, allowing the model to maintain its flexibility in learning patterns. The normalization process takes place before the non-linearity of the previous layer and it

is applied to reduce big weights [27]. Independently for each feature in the batch it is computed :

$$\hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}. \tag{4.1}$$

In equation(4.1) μ represent mean of inputs and σ^2 represents variance of inputs. ϵ is added for numerical stability.

After normalization, the inputs are scaled and shifted using learnable parameters γ and β, as shown in equation(4.2).

$$y_i = \gamma \cdot \hat{x_i} + \beta \tag{4.2}$$

Batch normalization is especially useful in ConvLSTM models, where activations can vary significantly due to the complex interactions between spatial and temporal dependencies.

4.1.4 TimeDistributed Layer

In teacher-forcing model decoder processes a sequence of frames at once, so we use **TimeDistributed** from *tensorflow.keras.layers*, to apply the same Conv2D operation across all time steps. This ensures that each predicted frame in the output sequence goes through the same transformation. Output at the end has one output channel (filters=1), as we are predicting grayscale precipitation maps. The kernel size is 1×1, meaning the layer acts as a final mapping function rather than learning spatial features. In contrast, in the auto-regressive model, you generate one frame at a time in a loop. At each iteration you only have a single frame (no extra time dimension) so you can apply a standard Conv2D directly without wrapping it in TimeDistributed.

4.2 Model Training and Optimization

4.2.1 Splitting the data

Dataset is split into **train and validation sets**. Train set is used for learning the patterns in data. The model adjusts its internal parameters during this phase to minimize the prediction error. Validation set is used to determine how well model is preforming and based on validation loss weights are being updated to improve generalization. Train set consists of 80% of data and remaining 20% represents validation set. Initially dataset was split into train, validation and test set but there were too little data to work with.

4.2.2 Loss Function

The main problem with dataset is its extreme imbalance because most pixels contain no precipitation (zeros), forming the majority of the dataset. Only a small portion of pixels represent actual precipitation, which is crucial for accurate forecasting.

The MSE loss function handles equally the error raised by the samples across whole dataset, this happens to be true for problems with balanced dataset. However, the model being developed is biased toward the majority values when it comes to the unbalanced dataset [28]. For precipitation data, if we used standard MSE, the model would be biased towards minimizing errors on background pixels, potentially ignoring precipitation regions. That is why we are introducing **weighted Mean Squared Error** which is applying **higher weights** to precipitation pixels. For that reason the loss function forces the model to prioritize learning from precipitation areas, leading to improved forecasting accuracy.

The equation(4.3) represents described **weighted Mean Squared Error**:

$$\mathcal{L}_{weightedMSE} = \frac{1}{N}\sum_{i} w_i \cdot (y_{true,i} - y_{pred,i})^2. \tag{4.3}$$

Where w_i is the weight assigned to each pixel, defined as in equation(4.4):

$$w_i = \begin{cases} weight\_factor, & \text{if } y_{true,i} \geq 0.3 \text{ (precipitation pixel)} \\ 1, & \text{if } y_{true,i} = 0 \text{ (background pixel).} \end{cases} \tag{4.4}$$

The function modifies the standard MSE loss by introducing a weight factor that gives more importance to precipitation pixels than background pixels. Used weight factor was 3.0 making it not to large so the model is not predicting too much false positive. This is ensuring that error in precipitation areas contribute more to total loss, so the model tries not to miss precipitation. $\mathcal{L}_{weightedMSE}$ in equation(4.3) is the total weighted MSE loss. N is the total number of pixels in the batch. $y_{true,i}$ represents actual value(ground truth) of the i-th pixel.

4.2.3 Optimizer and Learning Rate

In deep learning, the **learning rate (η)** determines how large the steps are when updating the model's weights in order to minimize the loss function. This process is guided by an optimization algorithm, often Gradient Descent, which moves the model's parameters in the direction that reduces the error.

The learning rate controls how fast or slow this descent happens. If the learning rate is too high, the model might take large steps, overshooting the minimum and causing instability, making it difficult to converge. If the learning rate is too low, the model makes very small adjustments, leading to slow convergence, which can result in unnecessarily long training times[29].

To handle this issue, learning rate schedulers are often used to adjust the learning rate dynamically during training. In this study, we use *ReduceLROnPlateau* from *tensorflow.keras.callbacks*, which reduces the learning rate when the model's improvement slows down. This technique monitors the validation loss and, if it does not decrease for a set number of epochs, the learning rate is automatically reduced. By lowering the learning rate at the right time, this scheduler helps the model converge efficiently and stably, without requiring the learning rate to be manually tuned at every step. These schedules have to be defined in advance and because of that are unable to adapt to a dataset's characteristics [30]. In the model we set *patience* parameter to 5, which represent number of

epochs with no improvement after which learning rate will be reduced.

Adam (Adaptive Moment Estimation) is an extension of Stochastic Gradient Descent (SGD) that automatically adjusts the learning rate for each parameter during training. This makes it particularly effective for complex models like ConvLSTM, where optimal learning rates can vary across different weights. In this study, we use the Adam optimizer from *tensorflow.keras.optimizers*, which is one of the most commonly used optimization algorithms in deep learning. It performs well on non-stationary problems, such as precipitation forecasting, where data patterns change over time.

In this implementation, Adam is used with a learning rate scheduler (*ReduceLROnPlateau*) to dynamically reduce the learning rate when training improvements slow down, ensuring efficient and stable convergence.

4.2.4 Early Stopping and Regularization

Early Stopping is a form of regularization used to prevent overfitting. It monitors a specified performance metric, such as validation loss, during training. If this metric doesn't improve for a defined number of epochs (*patience*), training is stopped. This approach ensures the model doesn't continue to learn noise from the training data, which can degrade its performance on unseen data. [31].

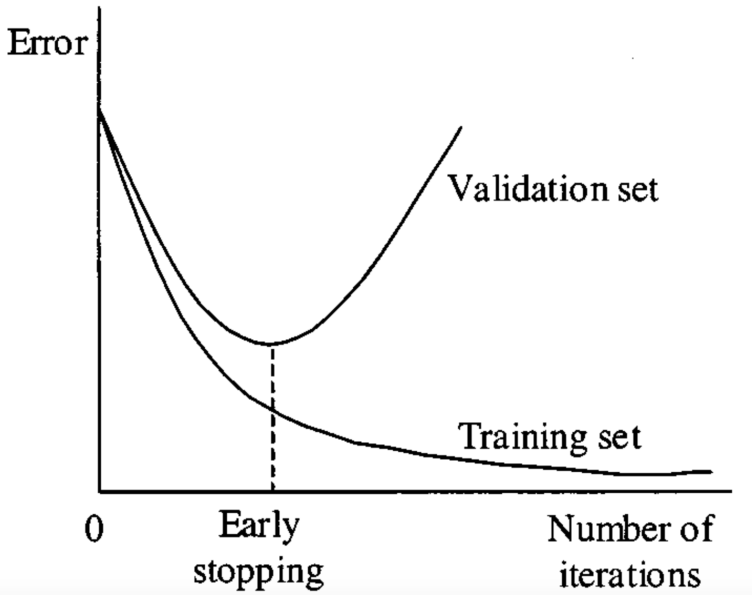

Figure 4.4: Early stopping halts the training after defined number of epochs [32]

In the model Early Stopping is implemented using *tensorflow.keras.callbacks.EarlyStopping*. *Early Stopping* and *ReduceLROnPlateau* combined make a training more efficient. With these two callbacks we are giving the model a chance to escape local minimum and prevent overfittitng.

4.2.5 Training Procedure and Hyperparameters

To get the best results when training a deep learning model, **batch size** and **number of epochs** must be carefully chosen. While epochs specify how many times the model sees the complete dataset during training, batch size dictates how many samples are processed prior to updating the model's weights. A larger batch size generally leads to more stable updates but requires more memory, while a smaller batch size allows for finer weight updates but may introduce more noise. Equally it is important to choose the appropriate number of epochs, too few epochs may lead to underfitting, while too many may cause overfitting, where the model learns patterns specific to the training data rather than generalizing to unseen data [33].

In this study, adjusting these parameters was particularly challenging due to the computational demands of the ConvLSTM model. Training required processing large sequences of high-resolution radar images while preserving spatial and temporal dependencies. Running this model on a local machine proved to be computationally expensive and impractical, as training iterations were extremely slow, and memory limitations restricted batch sizes. The model's complexity, together with the need for multiple hyperparameter adjustments, resulted in long training times, making it difficult to efficiently optimize the network.

While training deep learning models on standard hardware can be practical for smaller architectures, handling spatiotemporal radar data with ConvLSTM required access to more powerful computing resources.

4.3 Training on Supek Supercomputer

4.3.1 Motivation for Using High-Performance Computing (HPC)

As mentioned before, due to computational constraints, it was necessary to use the Supek supercomputer for training. The high-performance computing (HPC) environment provided the necessary GPU acceleration and memory capacity to process large batches and speed up training significantly [34]. This allowed for more efficient experimentation, enabling adjustments to batch size, learning rate, and other hyperparameters in a possible timeframe.

4.3.2 Configuring the Training Environment

The access to usage of supercomputer Supek was provided by Srce, Sveučilišni računski centar in Zagreb. For this study we used their service - Advanced computing (hrv. *Napredno računanje*).

The Supek supercomputer provides scientists with a cutting-edge computing environment for High-Performance Computing (HPC). HPC enables the execution of computationally intensive applications that require significant processing power and various computational resources, including CPU cores, accelerators such as GPUs, memory, storage, and high-speed networking. Supek is built using HPE Cray technology, featuring a total of 8,384 CPU cores and 81 GPUs, along with 32 TB of RAM. It delivers a computational power of 1.25 PFLOPS, making it the first petascale supercomputer in Croatia [35]. Figure4.5 is representing Supek which is located on campus Borongaj in Zagreb.

Job submission with PBS

Using the Supek supercomputer for training deep learning models involves a structured approach to job submission and resource management. The process is made through the **Portable Batch System (PBS)**, a job scheduler that efficiently allocates computational resources [37].

To execute a training task, you create a job script—named *run.sh*—which specifies the required resources and execution parameters. This script is then submitted to the PBS queue using the `qsub run.sh` command.

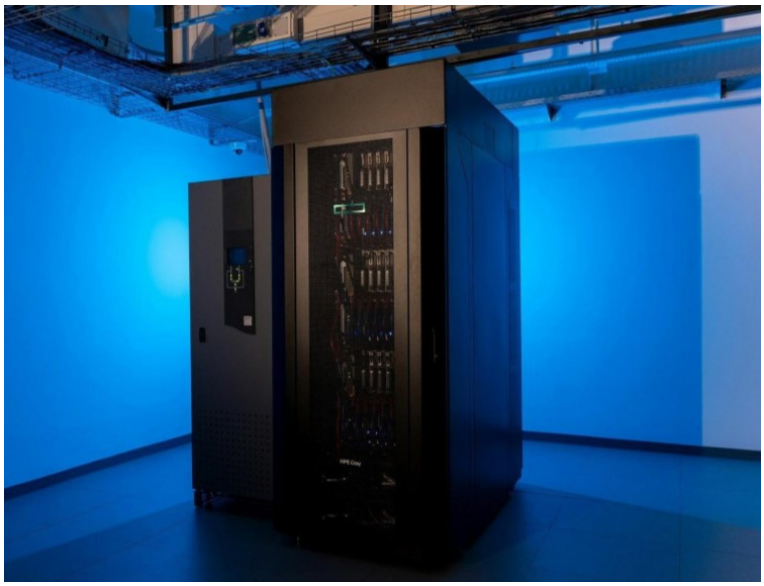

Figure 4.5: Supercomputer Supek based in Zagreb [36]

Listing 4.1: Example Job Script run.sh

```
#!/bin/bash

#PBS -q gpu
#PBS -l select=1:ncpus=16:ngpus=2:mem=32GB
#PBS -l walltime=06:00:00
#PBS -N precipitation_prediction
#PBS -o output.log
#PBS -e error.log
#PBS -V

# Load the TensorFlow module
module load scientific/tensorflow/2.10.1-ngc

# Execute the training script
run-singlenode.sh /user/project/model.py
```

With this script, it is requested one node with 16 CPU cores, 2 GPUs and 32GB of memory. It is also assigned walltime for maximum job duration of 6 hours, where to write output and error logs. Also it is important to note that it is needed to load *TensorFlow*

module version 2.10.1 optimized for NVIDIA GPUs as these are Supek's requirements.

TensorFlow Implementation on Supek

Supek provides optimized TensorFlow modules for efficient deep learning training. As already mentioned by loading the appropriate *TensorFlow module (scientific/tensorflow/2.10.1-ngc)*, we are ensuring compatibility with Supek's hardware and software environment. There is possibility to run *TensorFlow* scripts on single node, multiple GPUs on single node (like in `run.sh` script above) or mulitple GPUs on mulitple nodes. With that settings we need to be careful to adjust everything so the model works correctly on different number and sizes of GPUs. Efficient usage of RAM and storage is crucial as well. The `#PBS -l` command specifies the memory requirements, ensuring that the job is allocated sufficient RAM. For storage, it's important to manage input data and model checkpoints carefully, especially when dealing with large datasets [38] .

After Training on Supek

Because there is no direct terminal for real-time monitoring, standard output and error streams are automatically redirected to `output.log` and `error.log` files. As the training script executes on the supercomputer, any print statements, logs, or error messages appear in these log files instead of a normal console. Once the job finishes, the trained model and any visualization images are saved in the job's working directory on the HPC system. By inspecting the `output.log` file, we can verify that the training completed successfully, while the `error.log` file captures any runtime errors. After the job finishes, users can retrieve these logs and images from the HPC's file system to analyze the model's performance in detail.

5 Results and Discussion

In this study, a ConvLSTM-based Encoder-Decoder model was developed to predict short-term precipitation using radar images. The goal was to forecast future radar frames based on a fixed-length sequence of past frames. Two different training approaches were implemented: auto-regressive prediction and teacher forcing, each with its advantages and challenges in real-world forecasting.

To evaluate the model's performance, Weighted Mean Squared Error (Weighted MSE) loss was used as the primary metric to ensure an evaluation of precipitation forecasts. This approach gives higher importance to precipitation regions, addressing the data imbalance issue, where most pixels contain no rainfall. To monitor the model's training process, loss curves for both training and validation loss were plotted, providing insights into convergence behavior, overfitting, and generalization performance. Also to better present model performance, couple of forecasting metrics were presented - critical success index (CSI), false alarm rate (FAR), probability of detection (POD). They are commonly used in weather nowcasting to show model behavior.

The model was trained and tested using high-performance computing resources on the Supek supercomputer, allowing faster computation and better preformance.

5.0.1 Training and validation loss

For model that was trained with **teacher-forcing method**, number of epochs were set to 30 and it was used both Early Stopping and reducing of learning rate. Figure5.1 shows training and validation loss in a model trained using teacher forcing. The initial learning rate was set to 0.0001, and it was automatically reduced twice using the *ReduceLROnPlateau* strategy, which monitors validation loss and decreases the learning rate by a factor of 0.5 when no improvement is observed for four consecutive epochs. The first

reduction occurred at around epoch 11, lowering the learning rate to 0.00005, followed by another reduction at epoch 22, bringing it down to 0.000025. At the start of training, both the training and validation loss dropped quickly, meaning the model was learning useful patterns from the data. As training continued, the loss kept decreasing but at a slower pace, and small ups and downs started appearing. This suggests that the model was fine-tuning its predictions rather than making big improvements. Since the training and validation loss followed a similar pattern, it shows that the model was not overfitting. This shows that teacher approach is not experiencing unseen data because it was learning from ground truth and that model does not have high peeks in learning curve.

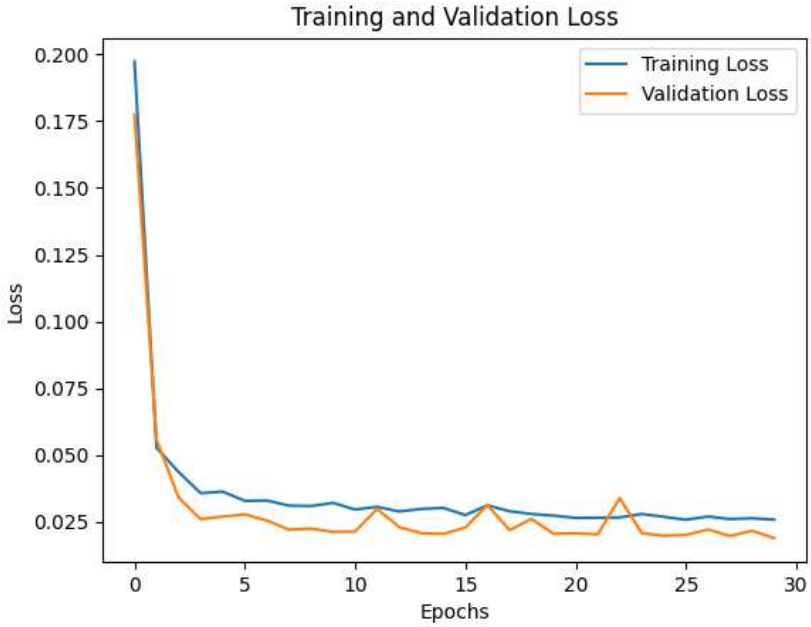

Figure 5.1: Plot of Training and Validation loss for model with teacher-forcing

The loss curve for the **autoregressive ConvLSTM model** with an encoder-decoder architecture is very unstable and shows a lot of ups and downs, which means the model had trouble learning properly and making good predictions shown in Figure5.2. Unlike the teacher-forcing method, where the model always learns from the correct answers at each step, the autoregressive model makes predictions based on its own previous guesses. This creates a problem called error accumulation—if the model makes a small mistake early on, that mistake keeps growing as it continues predicting, leading to worse results over time[39]. The big jumps in the validation loss suggest that the model struggled to stay consistent when making predictions. Also, early stopping happened quite early in

training, meaning the model couldn't improve much and even started overfitting (memorizing the training data instead of learning useful patterns) or making predictions that got worse instead of better. Unlike the autoregressive model, the teacher-forcing model

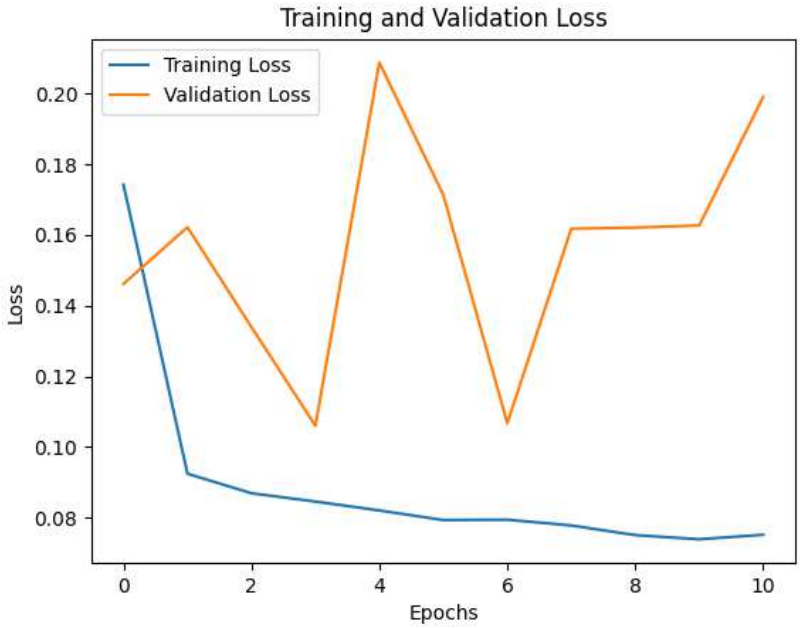

Figure 5.2: Plot of Training and Validation loss for autoregressive model

had a much smoother and more stable loss curve, with both training and validation loss gradually decreasing in a similar way. This means the model was learning in a steady and controlled manner without sudden changes. Because it always used the correct answers (ground truth) during training, it avoided the problem of errors building up over time, which helped it make better predictions on new data. While the autoregressive model struggled to stay accurate when making future predictions, the teacher-forcing model performed more consistently throughout training. Both models used early stopping and learning rate adjustments, but they had different outcomes. The teacher-forcing model improved its predictions step by step, while the autoregressive model remained unstable, with unpredictable changes in validation loss.

5.0.2 Precipitation forecast

Teacher-forcing approach

To see how model is behaving on already seen data - train set there is Figure5.3 showing 14 input frames used for precipitation forecasting on the left side of the image, which represent the initial conditions for the model. These frames have been denormalized, meaning they reflect real-world precipitation intensity values. They are not represented in scaled version of data that was fed to the model. Middle and right frames provide a comparison between the ground truth (actual precipitation frames) and the model's predicted frames. Ground truth frames from 1 to 14 are actually frames from 14 to 28 that are following input frames.

To evaluate how the model generalizes to unseen data, Figure5.4 presents predictions on the validation dataset. Similar to the training set visualization, the left side of the image displays the 14 input frames that serve as initial conditions for forecasting and middle and right shows ground truth and predictions, respectively. Unlike the training case, where the model had already seen similar patterns during training, these validation predictions assess how well the model can generalize to new sequences without prior exposure.

With the teacher-forcing approach, the model makes predictions that look very similar to the actual ground truth images. The predicted precipitation patterns match well with the expected ones, showing that the model correctly captures how rainfall moves and changes over time. This happens because, during training, the model always uses the correct previous frame as input, helping it learn more accurately. The visualization clearly shows that the predictions remain stable and follow the real patterns closely. The results are well-organized and easy to understand, making it simple to check how accurate the forecast is. This proves that training with teacher forcing helps the model learn effectively, allowing it to make high-quality predictions without big mistakes building up over time.

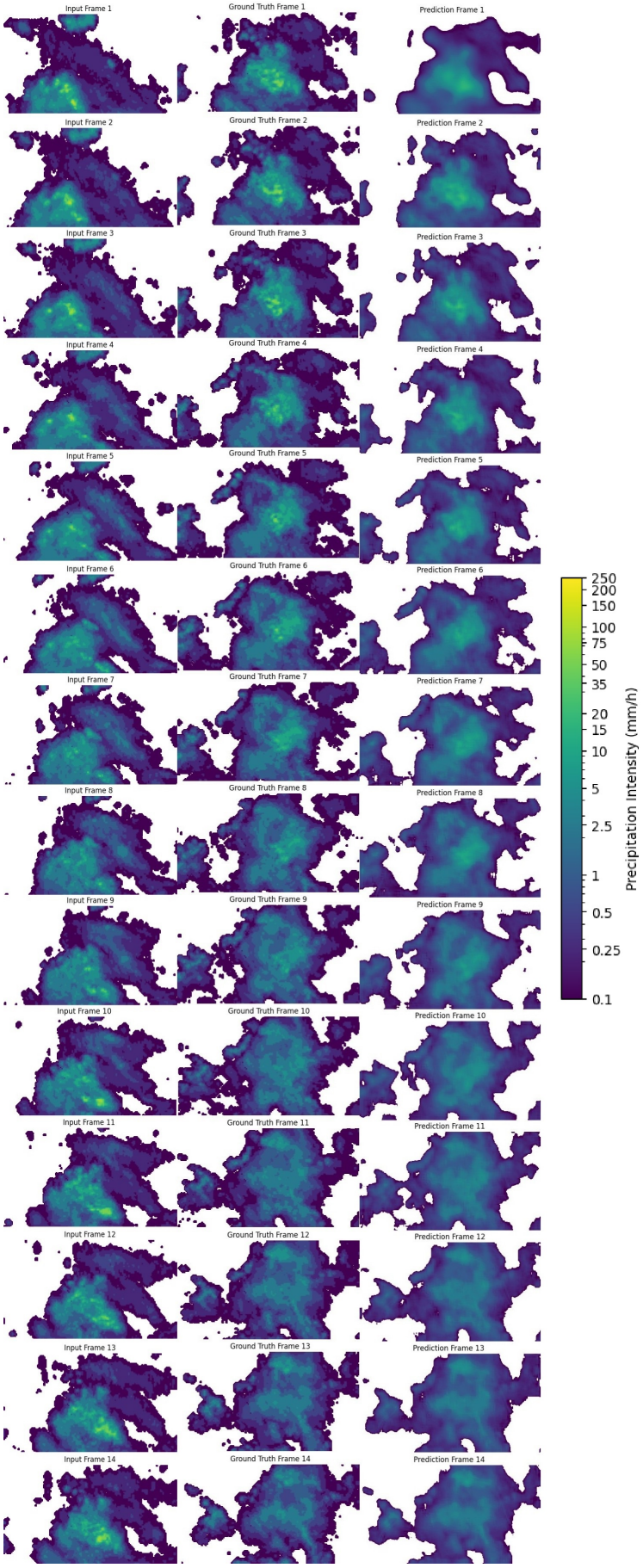

Figure 5.3: Teacher forcing approach for train set. On the left first 14 input frames that were fed to model, middle frames are representing next 14 frames of ground truth, on the right there are predictions that should resemble ground truth

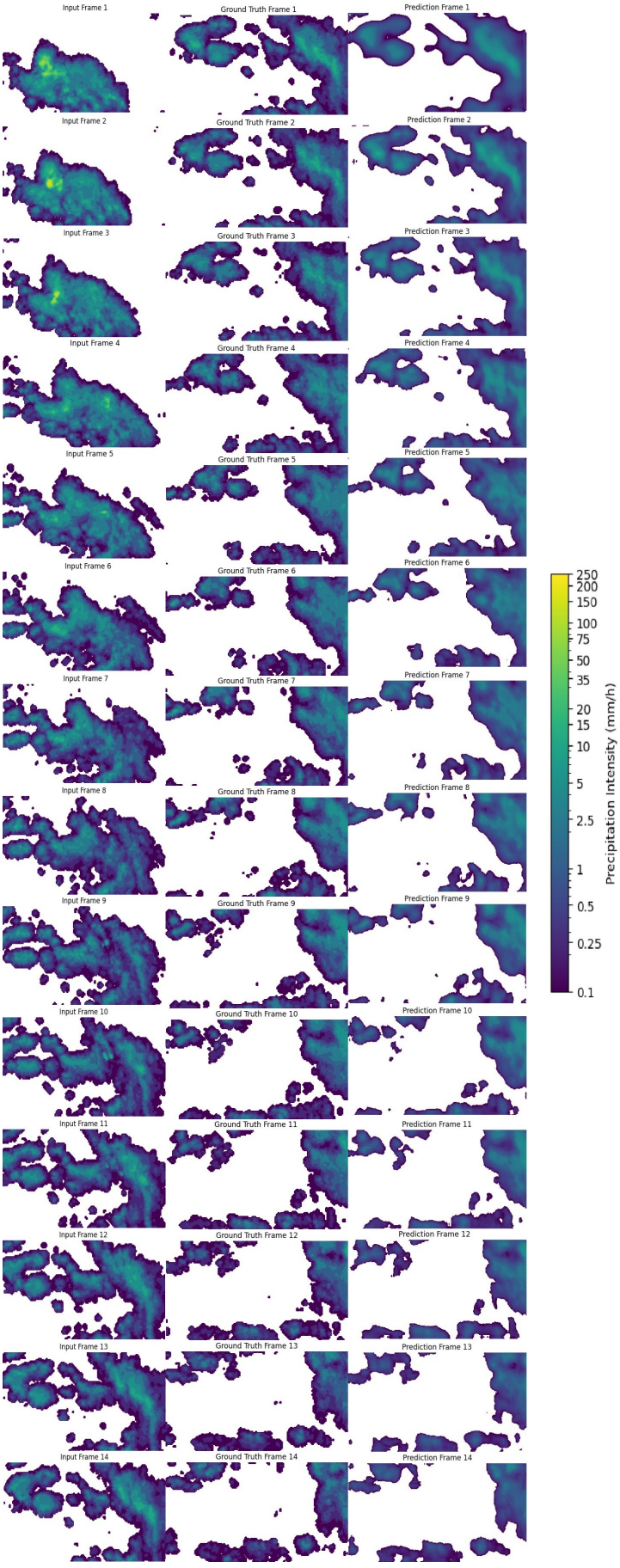

Figure 5.4: Teacher forcing approach for validation set. On the left first 14 input frames that were fed to model, middle frames are representing next 14 frames of ground truth, on the right there are predictions that should resemble ground truth

Autoregressive model approach

Next, there are images showing results of the autoregressive model approach for precipitation forecasting. Figure5.5 shows frames for train data. It represents the 14 input frames on the left, while in the middle there is ground truth (actual precipitation frames) and model's predicted frames on the right. Unlike the teacher-forcing approach, the autoregressive model generates predictions based on its own previous outputs rather than using the actual ground truth at each step.

The Figure5.6 shows frames for validation data. Here we can see that error accumulated over time is giving worse outputs as time is progressing.

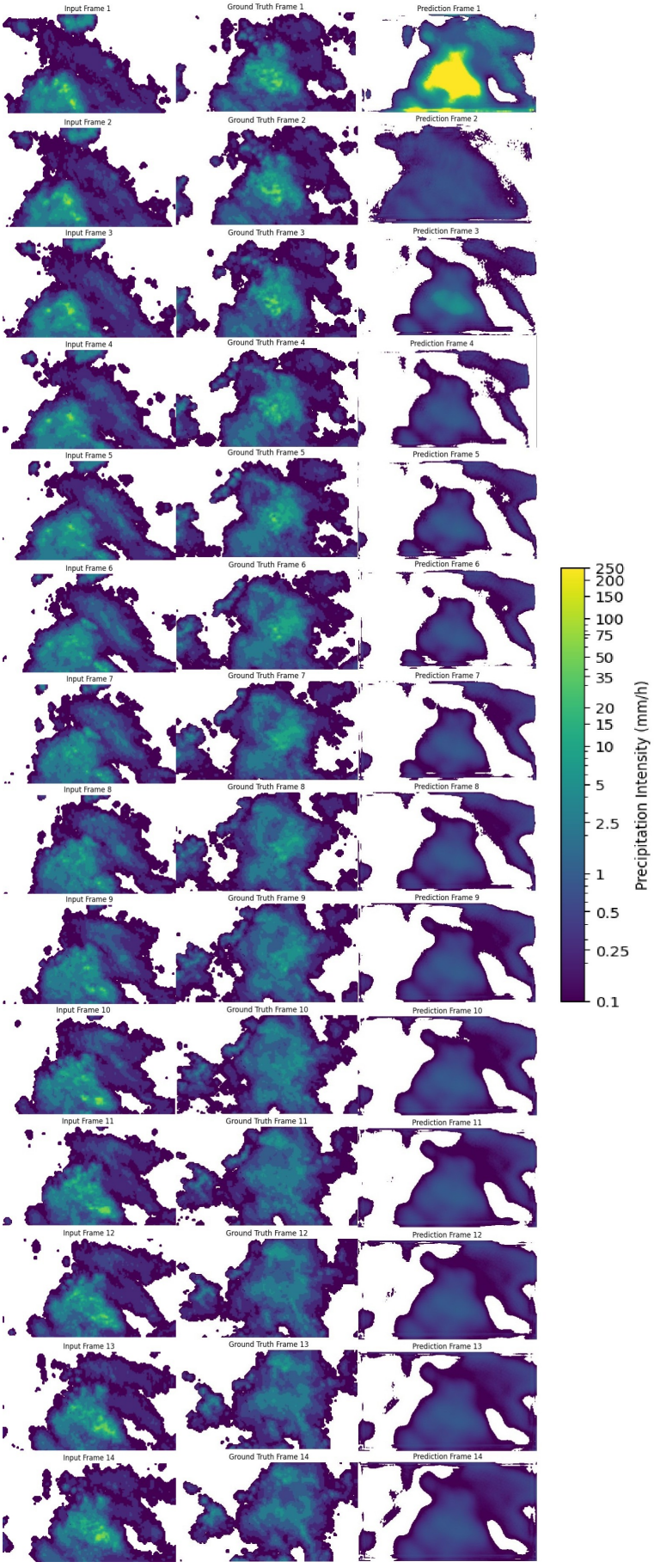

Figure 5.5: Autoregressive model approach for train set. On the left first 14 input frames that were fed to model, middle frames are representing next 14 frames of ground truth, on the right there are predictions that should resemble ground truth

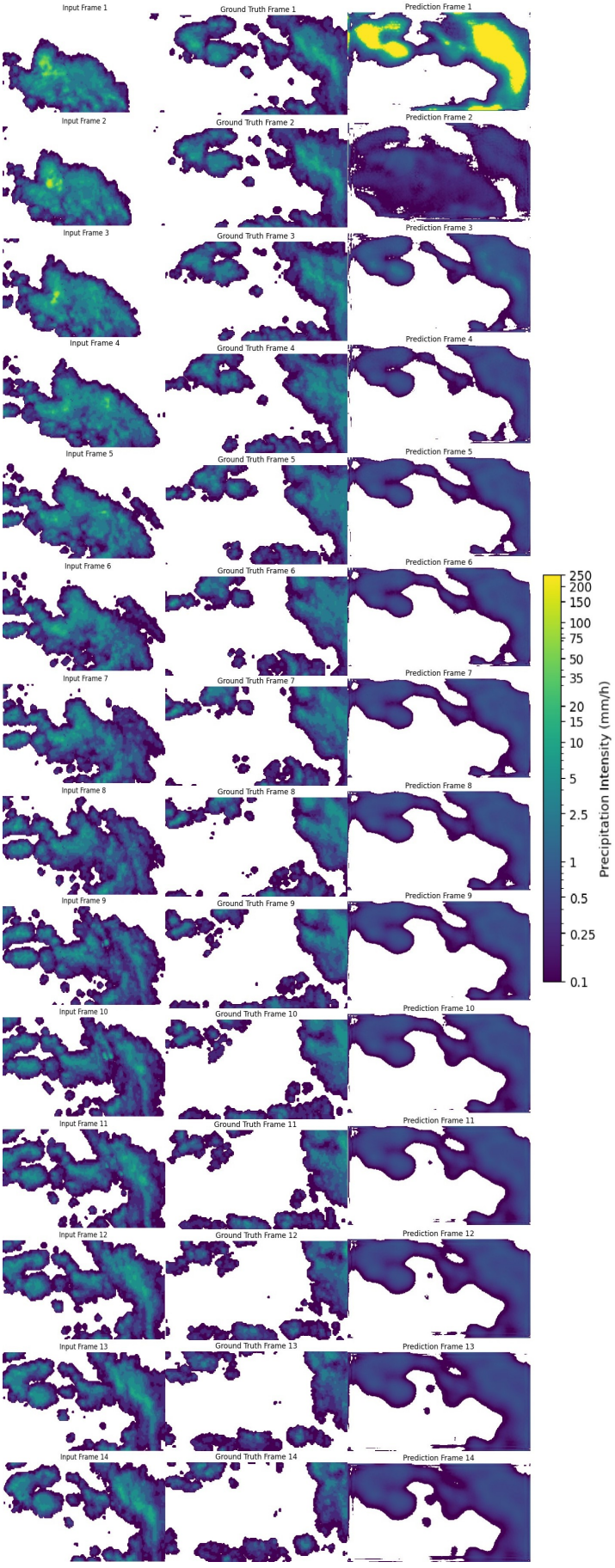

Figure 5.6: Autoregressive model approach for validation set. On the left first 14 input frames that were fed to model, middle frames are representing next 14 frames of ground truth, on the right there are predictions that should resemble ground truth

The predictions from the autoregressive model are noticeably less accurate compared to those from the teacher-forcing model. Initially, the first few predicted frames show some resemblance to the actual precipitation patterns, but as the prediction sequence continues, the model starts to lose important details, and the predicted precipitation areas fade or become deformed. This happens due to error accumulation—if the model makes a small mistake early on, that error carries over into the next step, which is then used as input for future predictions. As a result, the errors keep growing, leading to an increasing mismatch between predictions and actual precipitation patterns.

Unlike the teacher-forcing model, which produces stable and well-structured forecasts, the autoregressive model struggles with consistency and accuracy over longer sequences. The loss of detail and the breakdown of precipitation patterns in later frames suggest that the model has difficulty maintaining reliable predictions as it moves further into the future. One major reason for this is the limited amount of training data, especially the small number of samples containing heavy rainfall events. Since the model did not see enough examples of extreme weather conditions during training, it struggles to accurately predict them when they appear in the forecast.

Additionally, issues with the dataset itself contributed to these problems. Some images in the dataset were corrupted or contained errors, so it was hard to find batches with 28 consecutive frames. Because of that, number of batches and total images were very small. Another interesting issue is seen in the first predicted frame, where the model predicts too much heavy rain compared to the actual ground truth. This happens because the model was trained entirely in an autoregressive manner, meaning it always used its own past predictions during training. If the training data lacks enough diverse examples—especially for rare extreme rainfall events—the model can develop biases and misjudge the initial state, leading to false rainfall predictions. This error in the first frame sets off a chain reaction, causing the errors to grow as the model continues predicting future frames. The combination of insufficient diverse data, errors in the dataset, and the natural tendency of autoregressive models to accumulate mistakes contributed to the weaker predictions compared to teacher-forcing method. Improving the dataset by increasing the number of high-intensity rainfall events could help the model perform better in future experiments.

5.0.3 Precipitation nowcasting metrics

In precipitation nowcasting, we often convert predictions and ground-truth data into simple rain/no-rain maps using a chosen threshold. For this task it was taken 0.3 because below that threshold all pixels were treated as no rain and everything above that it was rain. Once we do this, we can count how many times the model correctly predicts rain (hits), how many times it predicts rain when it's actually dry (false alarms), and how many times it misses real rain (misses). Three useful metrics come from these counts: **the Probability of Detection (POD)**, which tells us how often the model successfully catches real rain events. **The False Alarm Ratio (FAR)**, which reveals how frequently the model predicts rain when there's none. **The Critical Success Index (CSI)**, a single score combining hits, misses, and false alarms into an overall measure of performance. Taken together, these metrics show how well the model identifies where and when rain will fall, helping us understand both its accuracy and its tendency to over- or under-predict rain events in real situations.

Equations(5.1) show how the POD, FAR and CSI are calculated.

$$\begin{aligned} POD &= \frac{\text{hits}}{\text{hits} + \text{misses}}, \\ FAR &= \frac{\text{false alarms}}{\text{hits} + \text{false alarms}}, \\ CSI &= \frac{\text{hits}}{\text{hits} + \text{misses} + \text{false alarms}}. \end{aligned} \tag{5.1}$$

Teacher-forcing approach

Table 5.1: Precipitation nowcasting metrics shown on teacher-forcing approach

Teacher-forcing approach	POD	FAR	CSI
train set	0.86	0.033	0.83
validation set	0.81	0.075	0.76

The teacher-forcing model performs exceptionally well on the training set, achieving a high CSI (0.83), meaning it correctly captures most rain/no-rain events. It has an extremely low FAR (0.033), so when it predicts rain, it's rarely wrong. Its POD (0.86) is strong, meaning it successfully identifies most precipitation events. On the validation set, the CSI remains high (0.76), showing good forecasting skill, but the false alarm rate

increases slightly (0.075), meaning it predicts rain incorrectly a bit more often. The POD remains solid (0.81), still detecting most rain events, though with slightly less confidence. This is expected because the model was trained by always seeing the correct previous step (teacher-forcing), which makes it highly accurate for familiar patterns.

Autoregressive model approach

Table 5.2: Precipitation nowcasting metrics shown on autoregressive model approach

Autoregressive model approach	**POD**	**FAR**	**CSI**
train set	0.566	0.190	0.499
validation set	0.679	0.419	0.456

The auto-regressive model feed its own predictions back as input, mimicking real-world forecasting. It has a moderate CSI (0.45–0.50), indicating decent but not perfect performance. On the validation set, the false alarm rate (FAR) rises significantly (0.42 versus 0.19 in training), meaning it falsely predicts rain more often on new sequences. However, it also detects actual rain events more aggressively, with a higher POD (0.68 on validation versus 0.57 in training). This suggests that the model is better at catching real precipitation but does so at the cost of more false alarms. This trade-off is common in precipitation forecasting, going for higher detection can lead to more incorrect predictions, impacting overall accuracy [19].

Comparing the two, the teacher-forcing model excels in controlled training conditions because it always gets the correct previous step as input, making its predictions highly accurate. The auto-regressive model, while less accurate overall, is more realistic because it learns to handle its own predictions over multiple steps. As a result, it performs better in real-world forecasting, where errors can accumulate over time.

6 Conclusion

This study explored the use of deep learning models for precipitation forecasting using satellite image sequences. Two main approaches were tested: the teacher-forcing model and the autoregressive model. The results showed that the teacher-forcing model provided more stable and structured predictions, while the autoregressive model struggled with errors building up over time, leading to less accurate forecasts. The teacher-forcing approach performed well because it always used the correct past frames during training, allowing it to make reliable predictions. In contrast, the autoregressive model predicted each new frame based on its own past predictions, which caused small mistakes to grow and made long-term forecasts less reliable. Another issue in the autoregressive model was overestimating precipitation in the first predicted frame, likely due to biases in the training data and the small number of extreme weather events in the dataset. Additionally, the dataset contained some corrupted images and inconsistencies, which may have negatively affected the model's learning process.

To improve future models, several key changes should be made. First, expanding and improving the dataset by including more extreme rainfall events would help the model learn better patterns. Second, instead of fully autoregressive training, a hybrid approach could be used, where the model gradually shifts from using real past frames to using its own predictions, making it more stable. Third, better loss functions and regularization techniques could be introduced. Additionally, exploring newer architectures like Transformers. Also problem can be viewed from other angle - not having another inputs like course of wind, terrain of the surrounding area on the map, sea influence and a lot of other variables that significantly contribute to model predictions.

Despite its challenges, this research highlights the potential of deep learning models in precipitation forecasting. While the teacher-forcing model provided better short-term

accuracy, the autoregressive model faced difficulties in long-term predictions. By addressing these issues and applying the suggested improvements, future models could provide more accurate and reliable weather forecasts.

References

[1] F. Schmid, Y. Wang, and A. Harou. Nowcasting guidelines – a summary. [Online]. Available: https://wmo.int/media/magazine-article/nowcasting-guidelines-summary

[2] Y. Zhang, M. Long, K. Chen, L. Xing, R. Jin, M. I. Jordan, and J. Wang, "Skilful nowcasting of extreme precipitation with nowcastnet," *Nature*, vol. 619, no. 7970, pp. 526–532, 2023. https://doi.org/10.1038/s41586-023-06184-4

[3] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W. chun Woo, "Deep learning for precipitation nowcasting: A benchmark and a new model," 2017, available at https://arxiv.org/abs/1706.03458.

[4] S. Chen, G. Long, J. Jiang, D. Liu, and C. Zhang, "Foundation models for weather and climate data understanding: A comprehensive survey," 2023. [Online]. Available: https://arxiv.org/abs/2312.03014

[5] J. Brownlee. (2021) What is teacher forcing for recurrent neural networks? Accessed: January 2025. [Online]. Available: https://machinelearningmastery.com/teacher-forcing-for-recurrent-neural-networks/

[6] Daljinska meteorološka mjerenja - radari. [Online]. Available: https://meteo.hr/infrastruktura.php?section=mreze_postaja&param=dmm

[7] Dhmz radar composites. Accessed: January 2025. [Online]. Available: https://meteo.hr/naslovnica_radarska-slika.php?tab=radari&idr=kompozit&param=stat

[8] Colormap reference. Accessed: January 2025. [Online]. Available: https://matplotlib.org/stable/gallery/color/colormap_reference.html

[9] What is a neural network? Accessed: January 2025. [Online]. Available: https://www.ibm.com/think/topics/neural-networks

[10] H. Bommana. (2019) Single node representation. Accessed: January 2025. [Online]. Available: https://miro.medium.com/v2/resize:fit:1400/format:webp/1*SaQMHTLi4C7MIA4IzjAXJw.png

[11] A. LeNail. (2023) Nn-svg: Neural network svg generator. Accessed: January 2025. [Online]. Available: https://alexlenail.me/NN-SVG/index.html

[12] J. Krapac and S. Šegvić. Konvolucijski modeli. Accessed: January 2025. [Online]. Available: https://www.zemris.fer.hr/~ssegvic/du/du2convnet.pdf

[13] J. Padarian, A. B. Mcbratney, and B. Minasny. Representation of kernel with size of 3x3 and corresponding feature map. Accessed: January 2025. [Online]. Available: https://www.researchgate.net/figure/Example-of-the-first-3-steps-of-a-convolution-of-a-3x3-filter-over-a-5x5-array-image_fig1_329241581

[14] J. Gao, B. Deng, Y. Qin, H. Wang, and X. Li, "Enhanced radar imaging using a complex-valued convolutional neural network," 2018. [Online]. Available: https://arxiv.org/abs/1712.10096

[15] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. [Online]. Available: https://arxiv.org/abs/1412.3555

[16] H. D. Trinh, "Data analytics for mobile traffic in 5g networks using machine learning techniques," 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:230701926

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[18] J. Luo and X. Zhang, "Convolutional neural network based on attention mechanism and bi-lstm for bearing remaining life prediction," vol. 52, no. 1, pp. 1076–1091, 2022. https://doi.org/10.1007/s10489-021-02503-2

[19] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W. kin Wong, and W. chun Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," 2015. [Online]. Available: https://arxiv.org/abs/1506.04214

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

[21] G. Farooque, L. Xiao, J. Yang, and A. B. Sargana, "Hyperspectral image classification via a novel spectral–spatial 3d convlstm-cnn," *Remote Sensing*, vol. 13, p. 4348, 10 2021. https://doi.org/10.3390/rs13214348

[22] L. Li and N. Sun, "Attention-based dsc-convlstm for multiclass motor imagery classification," *Computational Intelligence and Neuroscience*, vol. 2022, pp. 1–13, 05 2022. https://doi.org/10.1155/2022/8187009

[23] R. Behera. Numerical algorithms and tensor learning laboratory. Accessed: January 2025. [Online]. Available: https://cds.iisc.ac.in/faculty/ratikanta/lab.html

[24] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014. [Online]. Available: https://arxiv.org/abs/1409.3215

[25] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," 2016. [Online]. Available: https://arxiv.org/abs/1610.09038

[26] S. Shastri, K. Singh, S. Kumar, P. Kour, and V. Mansotra, "Time series forecasting of covid-19 using deep learning models: India-usa comparative case study," *Chaos, Solitons and Fractals*, vol. 140, p. 110227, 2020. https://doi.org/https://doi.org/10.1016/j.chaos.2020.110227

[27] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" 2019. [Online]. Available: https://arxiv.org/abs/1805.11604

[28] L. Raguram and V. S, "A weighted mean square error technique to train deep belief networks for imbalanced data," *International journal of simulation: systems, science and technology*, vol. 19, 02 2019. https://doi.org/10.5013/IJSSST.a.19.06.14

[29] S. Ruder, “An overview of gradient descent optimization algorithms,” 2017. [Online]. Available: https://arxiv.org/abs/1609.04747

[30] “Neural networks for signal processing ii. proceedings of the ieee-sp workshop (cat. no.92th0430-9),” in *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, 1992. https://doi.org/10.1109/NNSP.1992.253714

[31] L. Prechelt, *Early Stopping — But When?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67. https://doi.org/10.1007/978-3-642-35289-8_5

[32] R. Gençay. Early stopping. Accessed: January 2025. [Online]. Available: https://www.researchgate.net/figure/Early-stopping-based-on-cross-validation_fig1_3302948

[33] Epochs, batch size, iterations - how are they important to training ai and deep learning models. Accessed: January 2025. [Online]. Available: https://www.sabrepc.com/blog/Deep-Learning-and-AI/Epochs-Batch-Size-Iterations

[34] P. Imperatore, A. Pepe, and E. Sansosti, “High performance computing in satellite sar interferometry: A critical perspective,” *Remote Sensing*, vol. 13, no. 23, 2021. https://doi.org/10.3390/rs13234756

[35] Napredno računanje. Accessed: January 2025. [Online]. Available: https://www.srce.unizg.hr/napredno-racunanje

[36] Supercomputer supek. Accessed: January 2025. [Online]. Available: https://www.srce.unizg.hr/vijesti/izdvajamo-iz-sn-96-usporedba-superracunala-supek-s-vrhunskim-europskim-superracunalima/879

[37] Pokretanje i upravljanje poslovima(supek). Accessed: January 2025. [Online]. Available: https://wiki.srce.hr/spaces/NR/pages/121966084/Pokretanje+i+upravljanje+poslovima+Supek

[38] Tensorflow supek. Accessed: January 2025. [Online]. Available: https://wiki.srce.hr/spaces/NR/pages/121964613/TensorFlow

[39] R. Parthipan, M. Anand, H. M. Christensen, J. S. Hosking, and D. J. Wischik, “Defining error accumulation in ml atmospheric simulators,” 2024. [Online]. Available: https://arxiv.org/abs/2405.14714

Abstract

Weather nowcasting using radar imaging and machine learning

Sara Orlić

This study investigates the use of Convolutional Long Short-Term Memory (ConvLSTM) networks for precipitation nowcasting using radar images. The model integrates Convolutional Neural Networks (CNNs) for spatial feature extraction and Long Short-Term Memory (LSTM) networks for capturing temporal dependencies. The model is based on an encoder-decoder architecture. A key focus of this study is the comparison between two forecasting approaches: teacher-forcing and autoregressive prediction. The teacher-forcing model receives the actual ground truth as input at each step during training. In contrast, the autoregressive model generates forecasts by feeding its own previous predictions as input, mimicking real-world conditions. A key challenge is the data imbalance, where precipitation covers only a small portion of radar images. To address this, a Weighted Mean Squared Error (MSE) loss function is applied. Training was performed on the Supek supercomputer, using high-performance computing to overcome standard hardware limitations. The results show that ConvLSTM effectively predicts short-term precipitation, achieving high spatial and temporal accuracy. Techniques like adaptive learning rate scheduling and early stopping further optimize training efficiency, demonstrating the potential of deep learning for improving weather nowcasting.

Keywords: deep learning; weather prediction; machine learning; ConvLSTM; high-performance computing

Sažetak

Korekcija kratkoročne vremenske prognoze radarskim snimcima i strojnim učenjem

Sara Orlić

Ovaj rad istražuje primjenu konvolucijskih mreža s dugom kratkoročnom memorijom (ConvLSTM) za kratkoročnu prognozu oborina korištenjem radarskih snimaka. Model objedinjuje konvolucijske neuronske mreže (CNN) za izdvajanje prostornih značajki i mreže s dugom kratkoročnom memorijom (LSTM) koja je sposobna uočiti dugoročne vremenske ovisnosti. Model je temeljen na arhitekturi enkoder-dekoder. Poseban naglasak u radu stavljen je na usporedbu dvaju pristupa prognoziranju: *teacher-forcing* i autoregresivno predviđanje. *Teacher-forcing* model tijekom treniranja u svakom vremenskom koraku kao ulaz prima stvarne vrijednosti iz skupa podataka. Nasuprot tome, autoregresivni model koristi vlastita prethodno predviđena stanja kao ulaz za sljedeći vremenski korak, oponašajući stvarne uvjete prognoze. Glavni izazov je neuravnoteženost podataka, jer oborine zauzimaju samo mali dio radarskih snimaka. Kako bi se to riješilo, primjenjuje se funkcija gubitka Weighted Mean Squared Error (MSE). Treniranje podataka se provodilo na superračunalu Supek, uz visoke računalne performanse koje omogućuju zaobilaženje standardnih hardverskih ograničenja. Rezultati pokazuju da ConvLSTM učinkovito predviđa kratkoročne oborine, postižući veliku prostornu i vremensku točnost. Tehnike poput *adaptive learning rate scheduling* i *early stopping* dodatno poboljšavaju učinkovitost treniranja, čime dokazujemo da je duboko učenje dobar pristup za poboljšanje vremenske prognoze za kratkoročno predviđanje vremena (*nowcasting*).

Ključne riječi: duboko učenje; vremenska prognoza; strojno učenje; ConvLSTM; na-

predno računanje